博　士　論　文

Sensor Data Management and Transportation over Unreliable Networks

（センサシステム構成法と非信頼性ネットワークにおけるデータ配送法）

東京大学大学院 情報理工学系研究科

江崎研究室

落 合　秀 也

Abstract

Communication networks have enabled environmental monitoring and remote control with sensors and actuators. It has been applied to building automation and facility management for the last one decade and shown that they are promising technologies for *Green by ICT*. The differences of building automation and facility management are their targets and scopes. The target of building automation network is to autonomously control actuators based on observed events with sensors. The target of facility management network is to analyze the usage of facilities in buildings so as to improve the efficiency of energy usage.

The recent rise of green-awareness in global society has attracted attentions to these technologies. However implementing them into actual buildings is not an easy mission but requires huge efforts. Only a few people or companies afford to pay their huge time and cost for this. The workload of system maintenance or update is also heavy. The system owners have to operate it for at least several decades: i.e., for the lifetime of the building. Programs of the system should be changed according to the changes of floors or other configurations though it is not always simple. Some system components become out-of-order in several years and need to be replaced. Those system components are not always available in the next one decade.

The current development of a facility management system involves proprietary design and implementation. Engineers, with identifying the use cases, design the schema of databases and user interfaces, select suitable communication protocols, and implement software based on the designs and selections. The system components developed in the project cannot be directly used at other projects. The maintenance of the system involves the engineering of the software (e.g., internal design, programming and debugging) for the proprietarily specified data schema and communication protocols.

There is another issue at the installation of building automation systems. We must install communication cables in a building to connect sensors and actuators. Recently, radio devices have been also considered for this purpose. However, wireless networks are still unreliable; they frequently become disconnected and lose packets, and to provide network connectivity for sensors and actuators over multiple network relays is challenging. Thus, currently, we only let sensors and actuators

be wireless and few relays are used in the practical deployment. Wireless access range is limited, and cables must be installed in buildings to provide network connectivity.

Based on those issues, we have setup the following two research challenges, in this thesis.

- Development of a common protocol stack for composing data-centric sensor actuator networks: i.e., facility management system. Data-centric sensor actuator networks focus on sensor data management rather than networking of sensors and actuators. They archive the historical records of sensors. The component that organizes data-centric sensor actuator networks must have the same protocol stack so that we can integrate them by component-and-flow programming style. This structure reduces engineering cost at the system integration phase.

- Development of data transportation scheme for unreliable networks. We must enable data transportation over intermittently-connected networks. Even if nodes are fixed, wireless links frequently become disconnected. Delay or disruption tolerant networking (DTN) has theoretically enabled message transportation over such links, but widely-studied message routing algorithms are still targeted at specific mobility models. The challenge is to design a routing algorithm that can be applied to both stable networks and totally random networks.

In this thesis, we present our contributions in two parts. In the first part, we propose component and flow programming model for sensor data management, and design facility information access protocol (FIAP) and central controller-based device management (CCDM) architecture. In the latter part, we propose potential-based entropy adaptive routing (PEAR) and delay tolerant IP networking (DTIPN) as data transportation schemes over unreliable networks.

In the research of FIAP, we identify (1) design pitfalls in data-centric sensor actuator networks, (2) challenges in the design of common protocol stack for implementing essential functional components, and (3) policies and algorithms for FIAP specification. We have developed FIAP protocol stack, implemented such components, and integrated into a data-centric sensor actuator network in Engineering Bldg.2 in the University of Tokyo. This experiment has shown that FIAP allows lightweight integration for wide-varieties of applications of facility management systems.

In the research of CCDM, we propose a centralized architecture for managing hundreds or thousands of sensor and actuator devices, and their dataflows. CCDM allows the integration and configuration of large number of components in a centralized terminal (or window). This also helps the integration of FIAP components into a facility management system. In our experiment, CCDM has shown its capability

of computing moderately complex placement algorithms in the traffic optimization case. We consider that this capability makes other optimizations feasible, such as delivery latency, load-balancing, and fault-tolerance.

In the research of PEAR, we propose a reliable communication framework for wide varieties of mobility models from stable networks to totally random networks. PEAR changes message delivery patterns according to the complexity of the topology changes. If the contact patterns among node are somehow related, PEAR chooses almost the best path. If the contact patterns become complex and estimation of contacts become meaningless, PEAR replicates messages in the network and maintains delivery probability. We carried out both simulation-based and prototype-based experiments to validate the behavior of PEAR. We evaluated delivery probability, latency, total transmissions and buffer usage. Though the delay increased according to the hop count, PEAR could achieve 100% message delivery in sensor data transportation. This result indicates that PEAR is quite adaptive to various mobility models and provides reliable communication framework compared to previously proposed routing schemes.

We also demonstrate in this thesis that wireless links are totally intermittent even if nodes are physically stable. We carried out experiments with 50 wireless nodes in Engineering Bldg. 2 and Hongo campus, in the University of Tokyo. It has also shown that hop-by-hop transfer and parallel message propagation, which PEAR takes, enables scalable message propagation in hop count and increases message propagation speed.

In the research of DTIPN, we propose an alternative architecture of DTN that seamlessly connects intermittently-connected networks to the existing Internet. IP packets can be delayed not only for seconds but also for hours and days. By making use of this fact, we demonstrate that sensor nodes in isolated networks can still send message to the Internet host with the Internet protocol.

Contents

Part I

Introduction

Chapter 1

Introduction

Communication networks have enabled environmental monitoring and remote control with sensors and actuators. It has been applied to building automation and facility management for the last one decade. The recent rise of green-awareness has strongly attracted attentions to these technologies in the context of *Green by ICT*.

Green by ICT is frequently referred to as the application area of building automation and facility management systems. The most simple and clear use case of building automation is to control lights based on the existence of people; i.e., it switches off lights if no body stay in the room and saves the electric energy. It can be also applied to HVAC (Heating, Ventilating, and Air Conditioning) system to appropriately control the air. It can also reduce peak power usage by switching the whole building into power-saving mode especially in summer, when the building most heavily uses electricity.

The differences of building automation and facility management are their targets and scopes. The target of building automation network is to autonomously control actuators based on observed events with sensors. The scope is often smaller and shorter; i.e., it covers only one room or one floor, the observed sensor data disappears from the network soon. The target of facility management network is to analyze the usage of facilities in buildings; e.g., how the electricity has been consumed, how a room has been used, and how efficiently control has been made. The scope is larger and longer; the network usually covers whole the building or sometimes several buildings, it archives the historical records of sensor data or control signals for several years.

Although, these technologies are promising, implementing them into a building is actually a big project. Only a few people or companies afford to pay their huge time and cost for implementing this. Furthermore, maintenance and update of the system is also heavy. After the installation, the system owners have to operate it for at least several decades: i.e., for the lifetime of the building. The floor plan or deployment plan of tables, computers, and other objects would be changed. The user interface for system owners should be re-configured according to the change

though it is not always simple. Some system components become out-of-order in several years and need to be replaced. Those system components are not always available in the next one decade.

1.1 Problem Description

We here identify the issues that increase engineering cost for developing and maintaining such systems.

1.1.1 Proprietarily Integrated Facility Management Systems

The development of facility management systems frequently involves proprietary design and implementation. Engineers must identify the requirement or use cases for the operation of the building, then select suitable product-based components for the identified use cases, design the schema of databases, and implement software based on the selections and schema designs. They frequently use web service technologies and database management systems in order to organize such facility management systems. They sometimes have to work with multiple computer languages to integrate different platforms, or they have to design communication protocols and data formats if no suitable protocols were found in their platforms.

We can develop a facility management system in this way. However, the system components developed in the project cannot be directly used at other projects. The maintenance of the system still involves heavy engineering of software (e.g., internal design, programming and debugging) because it stands on the proprietarily specified data schema and communication protocols.

1.1.2 Cabling for Reliable Communication Platform

The conventional solution for networking physically deployed sensors and actuators is to install communication cables. Radio devices have been also considered for connecting them. However, wireless networks are still unreliable; they frequently become disconnected and lose packets. Actually, multiple network relays that try to extend connectivity for sensors and actuators sharply lose packets. Thus, currently, wireless sensors and actuators are mostly directly connected to sink nodes, and only few relays are used in the practical deployment. Because wireless access range is limited, cables must be still installed to provide network connectivity.

We recognize that tons of researches have been made for wireless sensor networking in the past. However, most of them do not focus on the reliability of data transportation over multiple relays.

1.2 Technical Challenges

1.2.1 Development of Interoperable Component Models for Data-Centric Sensor Actuator Networks

In the first part of this thesis, we design facility management networks as data-centric sensor actuator networks. The term *data-centric* indicates that we focus on data management rather than networking of sensors and actuators. Data-centric sensor actuator networks must archive the historical records, and the archived data are expected to be used for analysis.

The components for data-centric sensor actuator networks include gateways, storages, and user-interface terminals (UI-terminal). A gateway has a data bridge between a field-level bus (traditional building automation systems) and FIAP data model. A storage has a huge capacity of buffers to archive the history records of data. An UI-terminal has a data bridge between user interfaces and FIAP data model. If we could make those functional components interoperable, the system integration phase becomes much simpler. Although some application-specific management is required, the integrators just need to draw component-and-flow diagrams to develop a facility management system.

The challenge is to design the interoperable model that can connect gateways, storages, and UI-terminals. Since it is *data-centric* the protocol model must be different from the traditional building automation systems. We try to identify design policies for developing such interoperable model between those components. The design itself should be finally summarized as a protocol specification of interfaces, data structures and data exchange procedures.

1.2.2 Extension of Reliable Communication Framework to Unwired Hundreds of Devices

In the second part of this thesis, we try to develop a reliable communication framework with multiple wireless relays for remote sensors and actuators. Our experiment with 50 wireless nodes has identified that wireless links are intermittent; i.e., even if nodes are physically stable, wireless links frequently disrupt and become disconnected. Traveling packets are frequently lost. Network topology highly and dynamically changes. Routing of messages without losses in such networks is totally challenging.

We recognize that the emergence of delay tolerant networks (DTN)[1], which was originally proposed for inter-planetary communication, provided great impact on the intermittently-connected networks. They proposed hop-by-hop reliable transfer scheme (with enhanced store-and-forward mechanism). Even if end-to-end synchronized communication paths do not exist, DTN allows application message delivery by taking hop-by-hop reliable transfer.

DTN has opened up routing issues. If the availability of links are deterministic

(e.g., inter-planetary case), we can program and schedule message forwarding processes at each node. However, if they are opportunistic, we cannot predict the best message propagation path to the destination. Some researchers have proposed DTN routing schemes under specific mobility models, such as random way point[2], levy walk[3] and bus traces[4, 5]. However, they have not shown generic propagation framework, which can be applied to any mobility model (including physically-stable networks).

The challenge is to develop a routing method which can be applied to wide varieties of mobility models from stable networks to totally random networks. The communication framework that implements hop-by-hop transfer and such routing schemes can be used as a reliable communication framework for sensor and actuator networking. Although nodes move only in few building automation scenarios, sensor data gathering using vehicles is also an application area of such reliable communication framework.

1.3 Contributions

This thesis is composed of two parts: (1) component-and-flow programming model and (2) data transportation over intermittently connected networks.

The first part proposes the design of protocol stack for data-centric sensor actuator networks, and component-and-flow programming for the integration of functional components implemented on the protocol stack. It has two chapters. The first chapter presents the design policies of the protocol stack in the context of facility information access protocol (FIAP). The second chapter presents the programming and integration model in the context of central controller-based device management (CCDM) architecture.

The second part proposes the reliable message delivery schemes for unreliable and intermittently-connected networks. The second part is composed of three chapters. The first chapter presents the routing method, called potential-based entropy adaptive routing (PEAR), that can be applied to wide varieties of mobility models from stable networks to totally random networks. The second chapter shows with our 50 wireless nodes that wireless links are highly dynamic even if nodes are physically stable, and that PEAR works efficiently for delivering of messages in the network. The third chapter presents delay tolerant IP network (DTIPN), an alternative architecture for the well-known DTN, which seems more practically useful for IP-based sensor actuator networking.

1.3.1 Facility Information Access Protocol

In chapter 2, we identify (1) design pitfalls in data-centric sensor actuator networks, (2) challenges in the design of common protocol stack for implementing gateways, storages and UI-terminals in order to allow component-and-flow programming, and

(3) policies and algorithms for FIAP specification. Those are related to the definition of data schema, interface design and data exchange method. The data schema should be generic enough to allow definition of application-specific schema at the integration phase. The interface should be designed by general operations. Data exchange should be scalable because they sometimes transfer very large data sequence from a component to another. We present how we defined FIAP.

We provide the specification of FIAP in Appendix A.

1.3.2 Central Controller-Based Device Management

In chapter 3, we propose CCDM, a centralized architecture for managing hundreds or thousands of sensor and actuator devices, and their dataflows. CCDM allows the integration and configuration of large number of components in a centralized terminal (or window). This also helps the integration of FIAP components into a facility management system. This chapter also demonstrates that CCDM can perform placement optimizations even with moderately complex algorithms.

1.3.3 Potential-Based Entropy Adaptive Routing

In chapter 4, we propose PEAR as a reliable communication framework for wide varieties of mobility models from stable networks to totally random networks. It changes message delivery patterns according to the complexity of the topology changes. If the contact patterns among node are somehow related, PEAR chooses almost the best path. If the contact patterns become complex and estimation of contacts become meaningless, PEAR replicate messages in the network and maintain delivery probability.

1.3.4 Intermittently Connected Mesh Networks

In chapter 5, we present our experiment report on wireless mesh networks and the application of PEAR. We have developed 50 wireless nodes as a testbed, and carried out experiments by deploying them to a campus and a building of the University of Tokyo. We study wireless link availability and connectivity patterns with the testbed. Then, we apply PEAR and evaluate the delivery patterns, latencies and other properties.

1.3.5 Delay Tolerant IP Networking

In chapter 6, we propose delay tolerant IP networking (DTIPN) architecture to seamlessly connect intermittently-connected networks to the existing Internet. We focus on the fact that IP packets can be delayed not only for seconds, but also for hours, days and weeks, and propose hop-by-hop IP packet delivery at the link layer and forward-error correction (FEC) at the transport layer. We demonstrate that

sensor nodes in isolated networks can send messages with the Internet protocol and that those messages can arrive at the destination host in the Internet space.

Part II

Component and Flow Programming Model

Chapter 2

Facility Information Access Protocol

2.1 Introduction

The data sequences produced by sensors are historical records, which have not been primarily used for actuation controls in the traditional building automation systems. However, we are getting realized that such historical records are useful for analysing and planning control strategy to make the building more intelligent. By comparing the history of motion detection with HVAC (Heating, Ventilation, and Air Conditioning) working statuses and power usages, we can analyze how people use electricity, find wastes, and make plans for improvement. Intelligent building systems are getting "data-centric" (Fig. 2.1) for this purpose.

The widely-acknowledged solution for managing such historical records is to integrate web services, databases and many other technologies into a single building management system. There are various choices in the design of architecture, interfaces and data models that implement the identified requirements and use cases. However, if they are designed in a wrong way, the maintenance of the system becomes burden for several decades: i.e., for the lifetime of the building. Some system components become out-of-order. Requirements and use cases change according to the reconfiguration of floor plans. We must be able to re-assemble the system for these types of accidents and changes with moderate engineering cost.

This chapter presents facility information access protocol (FIAP) and demonstrates that FIAP-based system integration avoids design pitfalls that potentially multiply such engineering cost. The design pitfalls, which we identify in this work, are about (1) definition of data schema, (2) interface design, and (3) data exchange method.

FIAP-based system integration avoids these pitfalls by taking the following design principles: (1) use simple data structure as the common data model, and allow definition of application-specific schema at the system assembly-phase, (2)

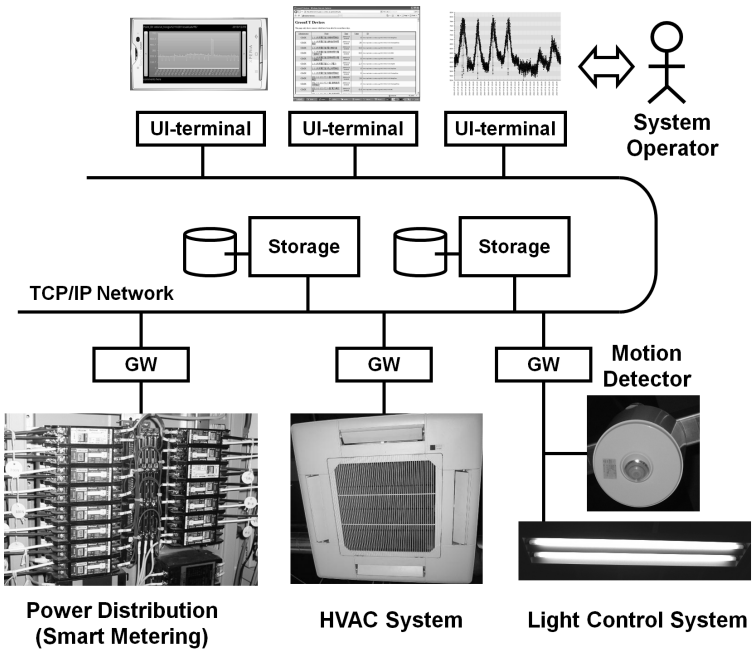

Figure 2.1: A data-centric sensor actuator network. Storage servers archive the historical records of power consumption, HVAC status, light control signals, detected motions and any other information. System operator analyzes the habitats, and plans the control strategies to make the building more intelligent.

generalize interfaces into a single interface, and develop system components (e.g., gateways, storages, and user interface terminals) under the common interface, (3) allow scalable data exchange on remote procedure calls (RPC).

This chapter is organized as follows. In section 2.2, we identify the architecture, the pitfalls and the challenges on data-centric sensor actuator networks. In section 2.3, we present the design principles of FIAP. We show FIAP-based system integration in section 2.4. Section 2.5 demonstrates our application. Section 2.6 addresses related works. We conclude this chapter in section 2.7.

2.2 Data-Centric Sensor Actuator Networks

2.2.1 Architecture

Fig. 2.1 shows typical system architecture for data-centric sensor actuator networks. The gateways translate data between field-level buses and Internet-side storages or user interface(UI) terminals. The field-level buses could be ZigBee[1],

[1]ZigBee provides tiny nodes for wireless sensor networks. http://www.zigbee.org/

Lonworks[2], BACnet[3] , 1-Wire[4] and any other sensor actuator networks. The data storages archive the history of data generated by those sensors and actuators. System operators access such storages and gateways from their UI-terminals and (1) obtain the historical records, (2) obtain the current snapshot and (3) set control schedules.

Some systems calculate statistics of historical records at gateway-side or storage-side. Daily usage of a room can be summarized at the gateway-side from the ON/OFF records of the corresponding motion detector. It can be also summarized at the storage-side.

Intelligent buildings must archive raw data with enough time-granularity. We sometimes need to compare the status of motion detector, door monitor and HVAC. We cannot make such analysis from aggregated trend data (e.g., daily average). This analysis basically involves very large dataset transfer mainly from storages to UI-terminals.

2.2.2 Conventional Solution

The widely-applied or conventional solution is to make use of web services and databases, and to integrate them for their identified use cases. Gateways with web service interfaces are available on the market (e.g., BACnetWS[5], oBIX[6], i.Lon SmartServer[7]). Any database management systems (DBMS) can be customized to archive historical records under their identified data schema. Some batch jobs can run behind the database to generate statistics of historical records. Any platforms can be used for developing UI-terminals if it provides the access interface to the database. Integrated and packaged products for building energy management system (BEMS) are also available: e.g., Exaquantum[8].

The problem of this solution comes from its proprietariness. A data-centric sensor actuator network can be developed with system integrators at the first phase. However, we have to maintain and reconfigure the system for several decades after the deployment. Especially if the design falls into the following pitfalls, such maintenance becomes a huge burden for the system owner for several decades.

[2]Lonworks: Local operating networks for building automations. http://www.lonmark.org/

[3]BACnet: a data communication protocol for building automation and control networks. http://www.bacnet.org/

[4]1-Wire. http://www.1wire.org/

[5]BACnet web services. http://www.bacnet.org/

[6]oBIX: Open Building Information Exchange. http://www.obix.org/

[7]i.Lon Smart Server: Embedded Internet server provided by Echelon. http://www.echelon.com/

[8]Exaquantum: a packaged product for energy management provided by Yokogawa Corporation

2.2.3 Pitfalls

2.2.3.1 Application-specific data schema and implicit data processing

It seems quite natural to define application-specific data schema as a common data model when implementing the use cases. Fig.2.2 (a) shows the definition of schemas for power and switch monitoring applications. System operator wants to know the power usage and working statuses of the PCs and the projector. This system implicitly calculates the working time in the background, based on the status of the switches.

This system is totally customized for this type of application. We can expand the number of smart meters and switches, but we cannot include weather data, motion detector, HVAC statuses without adding new schemas. Of course, we can do this. However, this approach multiplies the engineering cost. The maintenance of the wide varieties of data schema and related software becomes a burden.

2.2.3.2 Concretely defined interfaces

It seems quite natural to define multiple interfaces and access methods concretely for identified use cases by system integrators as Fig.2.2 (b). It can surely implement the identified use cases if we could pay a number of attentions in the design, review, software development, system test and so on.

However, if we concretely define many interfaces and develop many subsystems, such engineering cost becomes multiplied. Besides, we cannot easily extend or change the implementation of the system on the concretely defined interfaces. Such burdens follow for several decades.

2.2.3.3 Data transfer on a remote procedure call

Many platforms support remote-procedure call (RPC) as a web service for accessing remote objects and it is getting easier to develop RPC-based communications over the Internet. Data-centric sensor actuator networks have taken this advantage with the development of the web service technologies.

RPC-based communication performs well at small dataset transmission. However, it causes timeout failure or out-of-memory error if the amount of dataset becomes large as Fig.2.2 (c) illustrates.

2.2.4 Challenges

2.2.4.1 Application-Independent Data Model

In order to use the system at many applications, the common data model must be designed independently from application-specific schema. The least requirement on the data model is (1) that it allows managing time-series data sequence and (2) that it allows identifying the sequences. This also omits implicit data processing

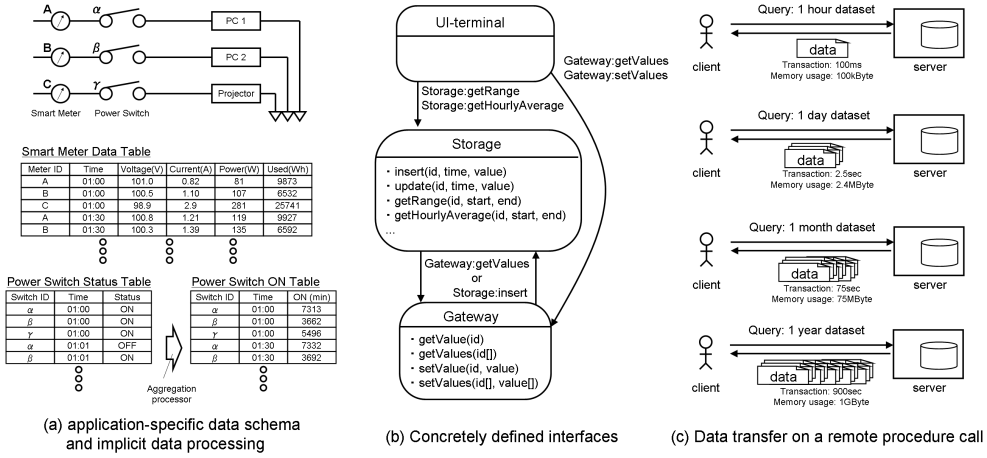

Figure 2.2: Pitfalls in the design of data-centric sensor actuator networks. (a) Application-specific data schema and (b) concretely defined interfaces multiply the engineering cost (i.e., installation and maintenance cost) of the system. (c) Data transfer on a remote procedure call causes failure at large dataset transmission.

inside the system (if we need to process data inside, we should explicitly specify the processing scheme).

2.2.4.2 Interface Generalization

In order to simplify the management of interfaces in data-centric sensor actuator networks, we must design a common interface that can inter-connect system components. This interface becomes a generalized interface that can implement gateways, storages, UI-terminals and any other functionalities.

2.2.4.3 Scalabile Data Transmission

A data-centric sensor actuator network must be able to transfer very large dataset from a component to another. Though data transmission on a single RPC is not scalable, we must make use of RPC-style communication to get scalability because RPC is well-supported by many platforms.

2.3 Facility Information Access Protocol

FIAP defines an application-independent data model for managing data sequences produced by sensors and actuators. It generalizes the interface for gateways, storages and UI-terminals. We also designed the data exchange procedure that allows scalable data transmission.

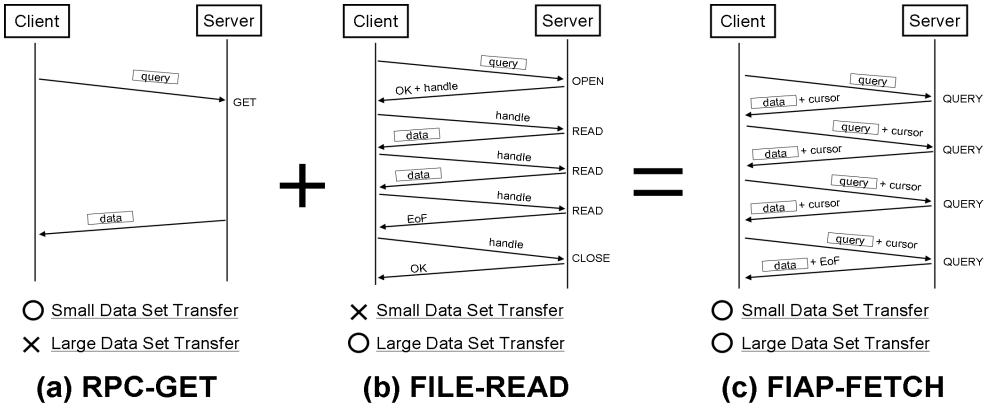

Figure 2.3: FIAP-FETCH procedure takes both advantages of RPC-GET and FILE-READ. (a) RPC-GET works efficiently at small data transfer but involves heavy load at large data transfer. (b) FILE-READ enables very large data transmission but has overhead at small data transfer. (c) FIAP-FETCH works as RPC-GET when the size of dataset is small but changes to FILE-READ style when it becomes large.

2.3.1 Management of Data Sequence by Point

A sensor produces a sequence of data. Signals for controlling an actuator also forms a sequence of data. Generally any entities work in the same manner in data-centric sensor actuator networks. For example, a data aggregator (e.g., hourly-average temperature calculator) generates sequences of data from their sources.

We define *point* to identify those sequences in the system. A point has only a sequence of data, which meaning (e.g., information of sensing target) must not be defined here in the common model. System components such as gateways, storages and UI-terminals manage them by points in their memory space or exchange them with others on flows.

More formally, let P be a set of points managed in a component or transferred on a flow. A point $p \in P$ has a set of time-value pairs, which we denote by $V(p)$. A pair $v \in V(p)$ has time and value. The format looks like as follows.

This data model itself is designed to be generic. It can be used at power monitoring applications, weather information gathering, and management of virtual machine working statuses.

2.3.2 Generalization of Gateways, Storages and UI-terminals

FIAP defines a common interface for gateways, storages, and UI-terminals. They can be inter-connected with other components with the common interface. The differences of components come from their implemented functionalities. A gateway

```
<point id="p1">
 <value time="2011-05-01T00:00:00">35.5</value>
 <value time="2011-05-01T00:01:00">35.4</value>
 <value time="2011-05-01T00:02:00">35.3</value>
</point>
<point id="p2">
 <value time="2011-05-01T00:00:00">true</value>
 <value time="2011-05-01T00:01:00">true</value>
 <value time="2011-05-01T00:02:00">false</value>
</point>
```

has a data bridge between a field-level bus and FIAP data model. A storage has a huge capacity of buffers to archive the history records of data. An UI-terminal has a data bridge between user interfaces and FIAP data model. Although these components are implemented differently, they can be inter-connected by the common interface.

2.3.3 Data Exchange Procedures

FIAP defines three procedures for data exchange among components. These procedures are (1) WRITE to send data to other components, (2) FETCH to retrieve data from other components, (3) TRAP to configure the other components to notify the change of status. FIAP defines them on RPC style communication because RPC is well-supported by many platforms.

2.3.3.1 WRITE

WRITE procedure initiates data transportation at the sender-side. The client (i.e., the sender) sends a message formatted as section 2.3.1. The server (i.e., the receiver) returns "OK" if it accepts. In WRITE procedure, the sender-side can avoid huge data transmission in one procedure call.

2.3.3.2 FETCH

FETCH procedure initiates data transportation at the data receiver-side. The receiver-side sends a query with specifying the range of dataset, but it does not know the amount of data for the returning data. If the amount of dataset is not large, the server returns all of them at the response. However, if the amount exceeds a certain limit, the server returns only the first subset of the specified dataset with a cursor, indicating that there is the next subset. If the client receives a cursor, it requests again with the cursor, and the server returns the next subset. They repeat this procedure until all the data transfer finishes.

Fig. 2.3 shows that FETCH procedure takes both advantages of RPC-GET and FILE-READ. RPC-GET returns all the dataset in one procedure call. It has smaller overhead in communication but it puts heavy load at the server if the amount is large. FILE-READ is scalable for reading very large dataset, however, it has overhead for small dataset transfer. FIAP-FETCH procedure performs well for both small dataset and large dataset.

2.3.3.3 TRAP

FIAP defines TRAP procedure to dynamically subscribe event-like data sequences from other components. A subscriber sends a stream (or trap) query to a notifier. Then, the notifier sends updated data to the subscriber. The stream query has a lifetime, and it must be updated by the subscriber.

2.4 FIAP-based System Integration

In FIAP-based system integration, we assemble individually developed components (i.e., gateways, storages and UI-terminals) into a data-centric sensor actuator network. We configure them to properly exchange data with each other so that these components can collaboratively work. The system becomes a diagram of component and flow as Fig. 2.4. Thus, we call this assemble process *component-flow programming*.

Formally, let $\Psi = (C, F)$ be a component-flow system. C is a set of components and F is a set of flows between components. A component $c(\in C)$ implements functionalities such as gateways, storages and UI-terminals. A flow $f(\in F)$ specifies how and what data shall be exchanged between the components.

For example, Fig. 2.4 shows a data-centric sensor actuator network. It has four components $C = \{c_1, \ldots, c_4\}$, and these components are connected by flows $F = \{f_1, \ldots, f_4\}$.

2.4.1 Programming of Components

Each component is programmed as follows.

c_1: p_1 gives the status of motion detector by *true/false*. p_2 gives the summarized time of motion detection in second. p_3 gives the status of the lamp by *true/false*. The lamp can be controlled at p_4 by *true/false*.

c_2: p_5 gives the summarized power usage in Wh.

c_3: c_3 archives the data of any points written to this component, and returns the specified range of dataset.

c_4: c_4 shows one day history of p_2, p_3 and p_5. It creates lamp control commands at p_4.

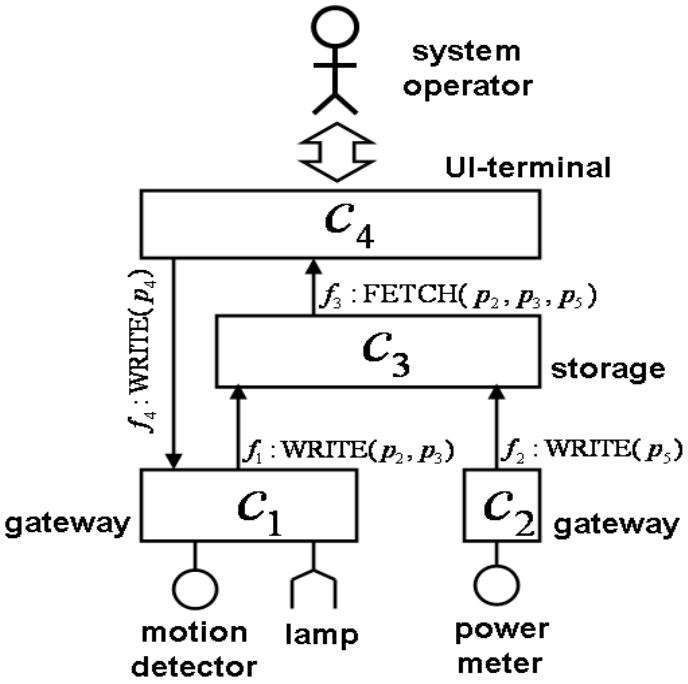

Figure 2.4: FIAP-based system integration. We assemble individually developed gateways, storages, and UI-terminals into a data-centric sensor actuator network, which can be drawn by component-flow diagram.

2.4.2 Programming of Flows

Each flow is programmed as follows.

f_1**:** c_1 sends the latest values of p_2 and p_3 periodically to c_3 by WRITE procedure.

f_2**:** c_2 sends the latest value of p_5 periodically to c_3 by WRITE procedure.

f_3**:** c_4 retrieves one day history of p_2, p_3 and p_5 from c_3 by FETCH procedure when requested by the system operator.

f_4**:** c_4 sends p_4 to c_1 by WRITE procedure when requested by the system operator.

In this way, the specification of $\Psi = (C, F)$ determines the total behavior of the system, which works as a data-centric sensor actuator network.

Practically, these programming can be made by script-based configuration. We can make use of text editors or command line interfaces (CLIs) for configuring the components and flows just as the conventional Ethernet switches and IP routers. We do not have to prepare integrated development environments (e.g., Eclipse, Microsoft Visual Studio) for this purpose.

2.5 Application

2.5.1 Implementation and Deployment

We have been operating a FIAP system for about one year from the beginning of 2010 at Engineering Bldg.2 in the University of Tokyo. It manages 1714 points[9], which in detail are:

- Electricity at distribution boards (908 points)
- Electricity at outlets (67 points)
- HVAC working modes (639 points)
- Motion detection and light status (40 points)
- Room environment (36 points)
- Gas and water supply (17 points)
- Weather information (7 points)

The frequency of data recording is configured for each point. Some points record at every minute, but others at every 30 minute. The total number of data elements in the data storage for year 2010 was about 430 million records.

2.5.1.1 Benefit from application-independent data schema

We first implemented storage and UI-terminals, and deployed for the building. We, then, implemented gateways for BACnet, oBIX, SNMP and other proprietary systems, and incrementally applied to the multiple applications as above: i.e., electricity, HVAC, motion detection, light status, weather. We could include various applications because FIAP has taken application-independent data schema for the design principle.

2.5.1.2 Benefit from interface generalization

We had only to assemble individually developed components into the data-centric sensor actuator network. Actually, we have implemented gateways, storages, and UI-terminals on many platforms, including Java (with Axis2[10]), PHP, Microsoft .NET Framework, Python, Ruby and Linux C. Those implementations came from different vendors and developers, and independently packaged as FIAP components. What we had done for integration is script-based configuration with a text editor.

[9] The number of points is the number of independent data sequences. E.g., The three sequences for voltage(V), current(A) and power(W) of an outlet are counted as three.

[10] Axis2: apache web services engine.

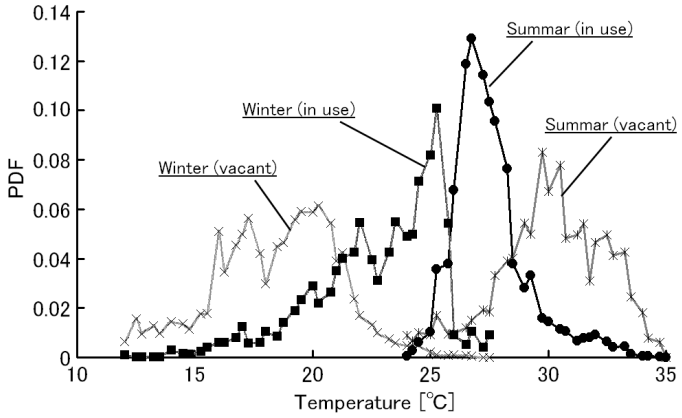

Figure 2.5: Temperature distributions of a meeting room

2.5.1.3 Lightweight implementation

Application-independent data schema and interface generalization has certainly reduced the engineering cost for software design, implementation, test and so on. The source code for Java platforms is made only by about 9000 lines with 52 class files (not including auto-generated codes). This includes the implementation of gateway (for BACnet), gateway (for oBIX), gateway (for SNMP), storage, and several UI-terminals. They use external libraries of Axis2 and JDBC[11]. If we implement more functionalities, the size of source code will increase, but our prototype indicates that FIAP is basically lightweight.

The most lightweight implementation is programmed in C for Linux platform. Though it has only WRITE client functionality, it is implemented with only 264 lines without using XML and SOAP libraries.

2.5.2 Data Analysis

Fig. 2.5 is the distributions of temperature of a meeting room in the building. The room has a motion detector, which can identify the time in use. Using such information, we categorized temperature data into four classes: i.e., (1) summer (in use), (2) summer when not used (vacant), (3) winter (in use), and (4) winter (vacant). We used dataset for [2010-07-01, 2010-09-30] as summer, and for [2010-01-01, 2010-02-28] $\cup$ [2010-12-01, 2010-12-31] as winter.

From this result, we can see that temperature of the meeting room has been well controlled. When the meeting room was in use, the average temperature was 27.5 degrees Celsius in summer, and 22.8 degrees in winter. They did not become too low in summer or too hot in winter. This means that people set appropriate

[11] JDBC: Java data base connectivity.

temperature for the room. When the meeting room was vacant, the temperature became higher in summer and colder in winter. This result indicates that if no body used the room, the air controller was switched off.

In order to enable this analysis, the analyzer had to read 3 months' history twice (for winter and summer) for 7 sensors: 6 sensors for motion detector, 1 sensor for temperature. Totally, there were about 1.5 million records. The FETCH procedure worked very well for retrieving such large amount of dataset.

2.6 Related Work

Researchers from database communities have tried to develop wireless sensor networks [6, 7, 8] with the application of database management systems (DBMS) or data stream management systems (DSMS) [9, 10]. GSN[11] and Daniel J. Abadi et al.[12] has also studied DBMS-based or DSMS-based sensor networking over the Internet. The main focus in their research has been the reduction of data traffic by in-network data aggregation[13], and the engineering cost was not mainly focused.

From the system development point of view, DBMS-based or DSMS-based platform is too general. The system integrator has to identify the use cases and design the data schema. They also develop the software for the designed data schema, and need to maintain the software. As we have identified, this system integration manner falls into the pitfall as Fig. 2.2(a), indicating that it multiplies the engineering cost for both installation and maintenance.

The researches of sensor web [14] have challenged to allow the management of global environmental data such as weather information over the Internet. The main goal of their researches seems to allow searching application-specific statuses from shared "pre-defined environmental information". Thus, the data model presented in IrisNet[15] is designed to tell whether parking slot is available or not. The main goal of our research is to propose a system development method which minimizes engineering cost at the system assembly phase. Thus, we considered application-independent data model and interface generalization for the design of individual system components.

2.7 Conclusion

We identified three major design pitfalls that multiply the engineering cost (i.e., installation and maintenance cost) of data-centric sensor actuator networks. System owners operate their intelligent buildings for several decades. Thus, such engineering cost must be minimized, otherwise the operation becomes a burden.

We presented technical challenges that enable lightweight system integration. The challenges include application independent data model, interface generalization, and scalable data transmission.

We presented the design principles of FIAP and FIAP-based system integration. We have also developed and deployed FIAP-based data-centric sensor actuator network in Engineering Bldg.2 in the University of Tokyo. This experiment has shown that FIAP allows incremental installation for wide-varieties of applications (i.e., electricity, HVAC, motion detection/lights, room environment, gas and water supply, weather) with CLI-based configuration.

Chapter 3

Central Controller-based Device Management(CCDM)

3.1 Introduction

In-network data processing or in-network data aggregation[13] is one of the most effective methods to reduce radio transmissions in wireless sensor and actuator networks. Instead of delivering all the sensor data to a remote actuator and making a decision there, it makes the decision at the very early stage of delivery and only the result reaches the final destination. The placement capability of data processors allows network-level optimization for traffic reduction, delivery delay, load-balancing and fault tolerance.

The implementation of data streams and data processors onto wireless sensor and actuator networks is challenging. First, the application-level instructions must be programmed by users, then the network must instanciate the program over the distributed environment. Here, it should optimize the placement pattern of the data streams and data processors, for example, to minimize radio transmittions. The algorithm that provides even nearly optimal becomes complex and they sometimes have $O(nc^3)$ for computation cost; n is the number of network nodes and c is the number of data processors. The implementation of such algorithms in distributed manner could be theoretically possible, but would have large execution time and seems infeasible.

We propose central controller-based device management (CCDM) which allows implementation of data streams and data prcessors onto wireless sensor and actuator networks even with moderately complex algorithms for placement optimization. In this work, we focus on traffic reduction case and propose two algorithms: (1) minimizing maximum traffic (MMT) and (2) minimizing maximum summary traffic (MMST). MMT and MMST takes $O(c^3 + c^2n)$ and $O(nc^3)$ respectively for the computation cost.

In this work, we propose ladder-logic for sensor actuator network programming.

We recognize that the database reseach community has provided sufficient work in terms of CQL-based sensor networking[6] for the last one decade. However, ladder-logic programming has been used in industrial automated systems or circuit programming[16][17][18] for more than two decades. The problem is that there is no clear and standard ladder-logic model for the current sensor actuator networks. Thus, in this work, we also provide component-and-flow model for the ladder-logic programming of sensor actuator networks, and make use of them in the CCDM architecture.

In our definition, a ladder-logic program (LLP) is an abstract description of applications provided by system users in component-and-flow diagram. In LLP, a component has input and output interfaces, and those interfaces are connected by flows. A component may be bound to physical sensors and actuators. It may also implement a processor or an aggregator. Some components provide, for example, an average value of several individual values gathered from different components. By associating the interfaces of components by flows, users compose a program (as Fig. 3.2).

Instanciation of an user LLP involves the placement of components to physical nodes. Though some components must be placed at pre-defined nodes, other components can be theoretically placed at anywhere in the network. Such components include data aggregator, data selector, data register and state machine. Intelligent placement of those components should be made in order to optimize, for example, total network traffic, delivery delay, load-balancing and fault-tolerance.

In general, optimization algorithms are not lightweight. The traffic optimization algorithms presented in this chaper take $O(c^3 + nc^2)$ or $O(nc^3)$. Other optimization algorithms (e.g., for delivery latency, load-balancing and fault-tolerance) also require large computation time. If they should be implemented in distributed manner, they would take larger time and sometimes may be infeasible. In this work, we focus on traffic reduction case and show that the optimization is feasible in CCDM architecture.

This chapter is organized as follows. In section 3.2, we address the related works. Section 3.3 defines CCDM and LLP model. Section 3.4 provides the evaluation. We discuss our experiment results in section 3.5 Section 3.6 gives the conclusion of this chapter.

3.2 Related Work

Centralized control and management for distributed data objects and data processors is actually a well-known architecture. Netconf[19] allows configuration of multiple switches and routers deployed in buildings or data centers at the same time with minimum configuration cost. Openflow[20] allows runtime configuration of communication flows over distributed openflow switches. Centralized architecture allows optimization of dataflows, and these systems take the advantage of such

capabilities.

ORINOCO[21] has proposed such centralized architecture for the management of distributed sensor-actuator networks, and studied some optimization algorithms. They assumed CQL-based[22] sensor network and Java distributed runtime environment for the data plane. Our work also inherits the advantage of the centralized architecture, but we applied it to a ladder-logic programming model.

CQL-based sensor networking is well-studied in database communities[6][7][23][8]. These works inherits the works on distributed data streaming managment systems (DSMS) such as Borealis[10], Aurora* and Medusa[24]. Not only ORINOCO[21], but also other optimization works[25][26] have based on the system model.

Ladder-logic programming, which we mainly focus on in this work, has not been sufficiently studied. In our survey, LonMark[18] applies this kind of programming model to building automation systems. In more past, ladder-logic programming was mostly discussed with programmable logic controllers[16][17]. However, their focus was the programming of circuits rather than the programming of dataflows and processes. We consider that there is no clear and standard model of ladder-logic programming for the currrent sensor actuator networks.

We have presented our first prototype implementation of CCDM in [27]. The previous work has assumed wired environment, and the main target was the reduction of operation cost. In this work, (1) we provide more formal definition of LLP in this architecture, (2) we apply them to wireless sensor and actuator networks, and (3) we propose traffic optimization algorithms.

3.3 CCDM System Model

3.3.1 Terminology

Let $G = (N, E)$ be the physical network topology organized by wireless nodes. The nodes are denoted by N, and the links are by E. We assume non-directional graph for G and that G is stable and does not change. A node has physical sensors and actuators. We denote them by $S(n)$ and $A(n)$; $S(n)$ is a set of physical sensors behind node $n \in N$, and $A(n)$ is actuators. We also assume that a node can exchange packets with any other nodes in the network by its shortest path. We denote the hop count between the two nodes by $h(n, k)$ $(n, k \in N)$.

We denote a LLP by $\Psi = (C, F)$. C is a set of components and F is a set of flows. Component $c \in C$ has input and output interfaces which we denote by c_{in_i} and c_{out_j} $(i, j \in J$: index number). Such interfaces are connected by a user-defined flow $f \in F$. We describe a sample definition of f as follows.

$$f : c_{in_i} \leftarrow d_{out_j} \tag{3.1}$$

Here, $c, d \in C$ and $i, j \in J$. In this example, flow f represents the link from d_{out_j} to c_{in_i}.

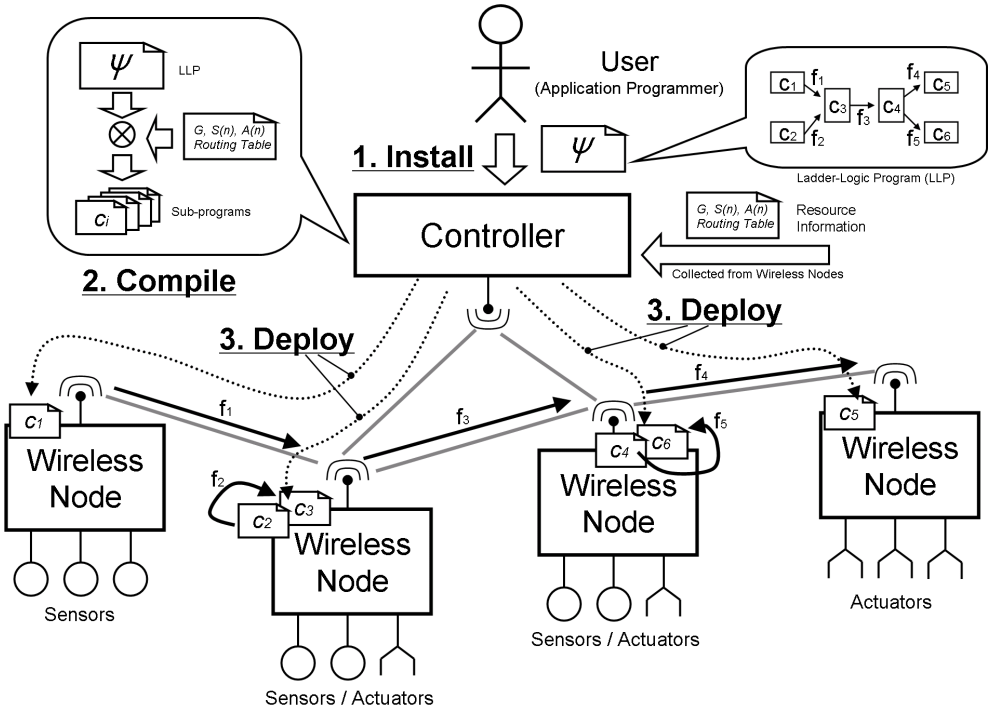

Figure 3.1: Central Controller-Based Device Management. User specifies the dataflow and process by LLP and installs to the controller. To instanciate the LLP, it compiles into sub programs, carries out placement optimization, and deploys onto the network nodes.

We denote the traffic weight on flow f (e.g., from c_{in_i} to d_{out_j}) by $w(f)$. It always gives non-negative values. We also allow describing by $w(c_{in_i}, d_{out_j})$ or $w(d_{out_j}, c_{in_i})$; both gives the same value.

In the following discussion, we sometimes use aggregated traffic between two components. We identify the difference by not specifing the interfaces in the parameters of w. The formal definition of aggregated traffic between component c and d $(c, d \in C)$ is as follows.

$$w(c, d) = \sum_{x \in I(c), y \in I(d)} w(c_x, d_y) \tag{3.2}$$

Here, $I(c)$ provides a set of interfaces of c.

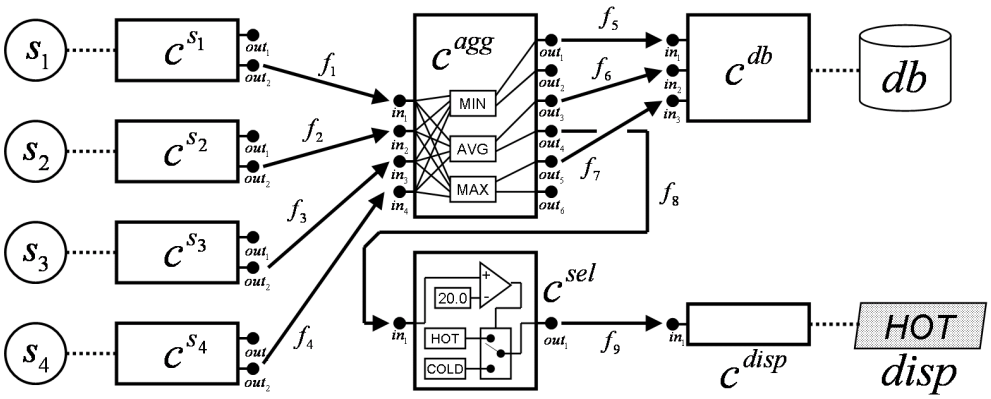

Figure 3.2: A sample user LLP on four temperature sensors $(s_1, \ldots, s_4)$, one database (*db*) and one display unit(*disp*). $C = \{c^{s_1}, c^{s_2}, c^{s_3}, c^{s_4}, c^{avg}, c^{sel}, c^{disp}, c^{db}\}$ is the set of programmed components, and $F = \{f_1, \ldots, f_9\}$ is the set of programmed flows. In this scenario, the database archives the maximum, minimum and average temperature and the display shows "HOT" or "COLD" based on the average.

3.3.2 Architecture

Figure 3.1 shows the architecture of CCDM. It is organized by an user (application programmer), a controller, wireless nodes, sensors and actuators. The user installs their program Ψ into the controller. Then, it compiles Ψ into component-based small program pieces, and deploys them to the wireless nodes. The instances of the small programs collaboratively carry out the application scenario provided by Ψ. During this process, the controller uses network resource information that includes topology G, routing tables, the list of sensors $S(n)$ and actuators $A(n)$ $(n \in N)$. It also makes optimization of total network traffic by assigning the small programs to appropriate nodes.

3.3.3 Ladder logic programming

Application programmers implement their instructions in the form of $\Psi = (C, F)$. Here, component $c \in C$ has input and output interfaces (i.e., c_{in_i} and c_{out_j}), and they are connected by flows $f \in F$. Programmers should not only draw such flows among components, but they should also know or program the internal behaviour (i.e., semantics) of the components.

We consider two phases in ladder-logic programming: (1) programming of components and (2) programming of flows. We explain these two steps in detail with a sample program provided at Fig.3.2. The sample program defines an application program for four temperature sensors in a room $(s_1, \ldots, s_4)$, a database (*db*) and

Table 3.1: Specifications of interfaces in the sample program (Fig. 3.2)

interface	type	mode	semantics
$c^{s_i}_{out_1}$	out	client	sends the latest value of sensor s_i every one second.
$c^{s_i}_{out_2}$	out	server	returns the latest value of sensor s_i when requested.
$c^{agg}_{out_1}$	out	client	sends the latest minimum collected by in_1, in_2, in_3, in_4 every one minute.
$c^{agg}_{out_2}$	out	server	returns the latest minimum collected by in_1, in_2, in_3, in_4 when requested.
$c^{agg}_{out_3}$	out	client	sends the latest average collected by in_1, in_2, in_3, in_4 every one minute.
$c^{agg}_{out_4}$	out	server	returns the latest average collected by in_1, in_2, in_3, in_4 when requested.
$c^{agg}_{out_5}$	out	client	sends the latest maximum collected by in_1, in_2, in_3, in_4 every one minute.
$c^{agg}_{out_6}$	out	server	returns the latest maximum collected by in_1, in_2, in_3, in_4 when requested.
$c^{agg}_{in_i}$	in	client	gets the value from the linked components.
$c^{sel}_{out_1}$	out	client	sends the text based on the collected value from in_1 (the text becomes "HOT" when it exceeds 20.0, otherwise the text becomes "COLD") every one minute.
$c^{sel}_{in_1}$	in	client	gets the value from the linked components.
$c^{db}_{in_i}$	in	server	receives values and archives them to the backend physical database as channel i.
$c^{disp}_{in_1}$	in	server	receives values and shows them by its physical display.

a display unit(*disp*). Here, $C = \{c^{s_1}, c^{s_2}, c^{s_3}, c^{s_4}, c^{avg}, c^{sel}, c^{disp}, c^{db}\}$ is the set of components, and $F = \{f_1, \ldots, f_9\}$ is the set of flows. This application archives the maximum, minimum and average of the four temperature sensors deployed in a room. It shows whether the room is hot or cold on the display, based on the average. Though the sensors can observe the readings at every second, this application sets the interval of the archive and the display update at 60 seconds.

3.3.3.1 Programming of Components

Programming of components is the first phase for ladder-logic programming. Before linking the interfaces of different components, the semantics of components and interfaces must be specified and implemented.

Table 3.1 shows the semantics of the sample program. It defines the type, mode and detailed behaviour of interfaces. For example, $c^{s_1}_{out_1}$ is an output interface

(type="output"), works as client (mode="client") and sends the latest value of sensor s_1 every second. To actually realize the application, these definitions must be described in computer language.

Table 3.2 shows a sample description of components for c^{s_1} and c^{sel} in a computer language. In this example, c^{sel} defines a client output interface ($out1$) and a client input interface($in1$). c^{sel} gets value from $in1$ and compares it with 20.0, it the value exceeds 20.0 c^{sel} sends "HOT" via $out1$, otherwise "COLD" via $out1$. Then it sleeps for 60 seconds and repeats the same process.

The semantics of the interface types change depending on the mode of the interface. In output mode case, client interface actively sends data to the linked target, and server interface returns data when requested from others. In input mode case, client interface actively gets data from the linked target, and server interface accepts data posted from others.

3.3.3.2 Programming of Flows

After defining the semantics of components and interfaces, programmers link them by flows. For example, the sample program (i.e., Fig. 3.2) has following flows to link component interfaces.

$f_1 : c^{agg}_{in_1} \leftarrow c^{s_1}_{out_2}$
$f_5 : c^{sel}_{in_1} \leftarrow c^{agg}_{out_4}$
$f_9 : c^{disp}_{in_1} \leftarrow c^{sel}_{out_1}$

With the semantics definition of components and interfaces, this means that c^{agg} gets the latest values of sensor s_1 from c^{s_1}, that c^{sel} gets the average values calculated by c^{agg}, and that the display unit shows the text selected by c^{sel}.

In this way, application programmers can implement their instructions on a LLP. However, LLP itself is just a program description, which needs to be instanciated and deployed over a wireless sensor actuator network.

3.3.4 LLP Instanciation

A LLP Ψ is just an abstracted description that defines processes and flows of an user application. Instanciation of Ψ needs to be made to actually carry out the program. In this process, CCDM controller divides Ψ into small program pieces, finds the better nodes (regarding to traffic optimization) for the divided programs to be carried out. In CCDM model, Ψ is divided by components; i.e., if Ψ has 100 components, Ψ shall be divided into 100 program pieces. Some components are deterministic and the others are non-deterministic. Deterministic components directly access sensors and actuators, thus, they must be deployed to the nodes of such sensors and actuators. Non-deterministic components do not belong to any physical devices, but implement a processing scheme or an aggregation scheme.

In Fig. 3.2 example, $\{c^s_1, c^s_2, c^s_3, c^s_4, c^{db}, c^{disp}\}$ are deterministic components, and $\{c^{agg}, c^{sel}\}$ are non-deterministic components.

3.3.4.1 Placement of deterministic components

Components that directly use physical sensors and actuators (e.g., c^{s_1}, c^{disp}) must be assigned and deployed to the nodes of such sensors and actuators.

More formally, let $N(c)$ be the set of nodes that component c can be deployed. If c directly uses a physical sensor s, there is only one node n_0 that satisfies $s \in S(n_0)$. Thus, $N(c)$ has only one element n_0 (i.e., $N(c) = \{n_0\}$). It also applies to actuator cases. If component d directly uses a physical actuator a, there is only one node n_1 that satisfies $a \in S(n_1)$. Thus, $N(d)$ has only one element n_1 (i.e., $N(d) = \{n_1\}$).

3.3.4.2 Optimized placement of non-deterministic components

The other components, which do not directly use any physical sensors or actuators, can be theoretically deployed anywhere in the network: i.e., $N(c) = N$. However, system resource usages drastically change depending on assignment patterns, and the controller should make some efforts to reduce the cost.

In this work, we focus on the reduction of network traffic load. We recognize that there are several targets to optimize: e.g., propagation delay, CPU load, fault tolerance. We provide algorithms for traffic load optimization as an example for CCDM-based optimization.

Here, we provide the definition of total traffic load under $G = (N, E)$ and $\Psi = (C, F)$,

$$T(G, \Psi) = \sum_{c,d \in C} w(c, d) h(n_c, n_d) \tag{3.3}$$

$$n_c \in N(c), n_d \in N(d) \tag{3.4}$$

The final goal is to develop algorithms that minimize $T(G, \Psi)$ with smaller computation cost. However, it seems a NP-hard problem because minimizing total traffic (MTT) algorithm takes $O(n^c)$; n is the number of nodes, c is the number of non-deterministic components. In this thesis, we only propose two algorithms (MMT and MMST) that nearly optimize the total traffic. We left the algorithm design for better optimization to be an open research item.

Minimizing maximum traffic (Fig.3.3): MMT algorithm calculates the traffic weight between unassigned components and the physical nodes of assigned components, and finds the pair of unassigned component $u(\in C)$ and physical node $l(\in N)$ that gives the maximum traffic weight. Then, it assigns u to the node l. MMT repeats this until all the components are assigned. The computation cost is $O(c^3 + nc^2)$; n is the number of nodes, c is the number of components.

Minimizing maximum summary traffic (Fig.3.3): MMST algorithm calculates the summary of traffic weights for an unassigned component at each node to all the assigned components, and finds the pair of unassigned component $u(\in C)$ and the physical node $l(\in N)$ that gives the maximum summary traffic. Then, it

MMT Algorithm

C: all the components
A: assigned components (A ⊂ C)
U: unassigned components (U ⊂ C)
N: all the nodes
L[c]: node location of (c ∈ C) (∈ N)
w(u,a): traffic between component u and a
m: maximum traffic
c: candidate component
l: candidate location

```
while(U.length>0){
  m=0; c=null; l=null;
  for each u in U {
    W.clear();
    for each a in A {
      n=L[a]; W[n]=W[n]+w(u,a);
    }
    for each n in N {
       if(m<W[n]){
          m=W[n]; l=n; c=u;
       }
    }
  }
  L[u]=l; A.add(u); U.remove(u);
}
```

MMST Algorithm

C: all the components
A: assigned components (A ⊂ C)
U: unassigned components (U ⊂ C)
N: all the nodes
L[c]: node location of (c ∈ C) (∈ N)
w(u,a): traffic between component u and a
h(l,n): hop count between node l and n
m: maximum summary traffic
c: candidate component
l: candidate location

```
while(U.length>0){
  m=0; c=null; l=null;
  for each u in U {
    // find local m and l for u
    l_m=+∞; l_l=null;
    for each n in N {
      s=0;
      for each a in A {
        s=s+w(u,a)*h(L[a],n)
      }
      if(s<l_m){
         l_m=s; l_l=n;
      }
    }
    if(l_m>m){
      m=l_m; l=l_l; c=u;
    }
  }
  L[u]=l; A.add(u); U.remove(u);
}
```

Figure 3.3: Minimizing maximum traffic algorithm

assigns u to the node l. MMST repeats this until all the components are assigned. The computation cost is $O(nc^3)$; n is the number of nodes and c is the number of components.

3.4 Evaluation

We carried out some simulation-based experiments and evaluated (1) the optimization effect by MMT and MMST regarding to the total traffic and (2) the CPU time for performing the optimization.

The result indicates that CCDM can carry out moderate optimizations even with moderately complex algorithms.

3.4.1 Total traffic

The goal of this evaluation was to study the optimization effects performed by the traffic optimization algorithms. We tested the performance at five use cases in this experiment (Fig. 3.4).

Ψ_1 **(one-tier aggregation)** determines actions according to several inputs; e.g., takes the minimum temperature of a room and switch on heaters so that it never decreases below the lowest limit.

Ψ_2 **(two-tier aggregation)** provides the statistics of larger area: e.g., maximum, minimum, average temperature of a building floor, aggregating the rooms of the floor.

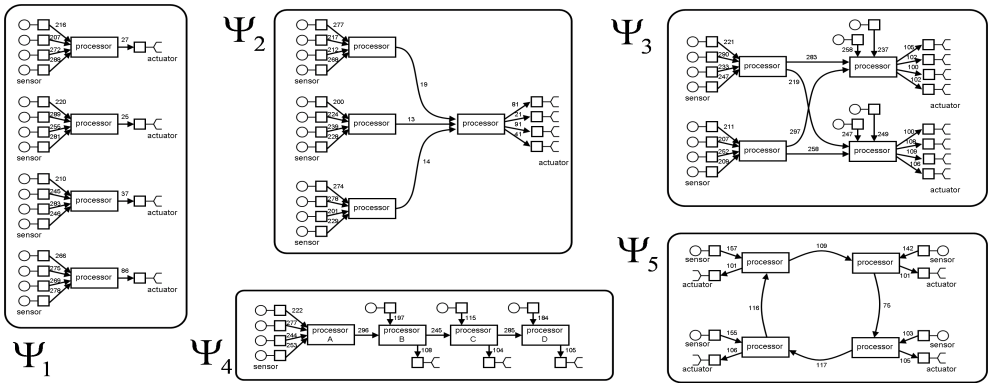

Figure 3.4: Use cases tested in the experiment

Ψ_3 **(two-type input and two-type output)** determines two different types of actions based on two different types of inputs; e.g., let air conditioner and room lights control based on infrared human detectors and switch boards.

Ψ_4 **(chain processing)** enables the detection of horizontal movements or synchronized actions; e.g., a walking person would be sensed by multiple floor sensors with time differences, which would be detected as a moving objects by a chain detection process.

Ψ_5 **(cyclic chain processing)** another form of Ψ_4.

Here, we assume that the traffic of each flow is the average traffic for statistically enough time periods. In our experiment, they are already given, however, in the practical deployment, it should be estimated by taking the statistics or by analysing the program logic.

We deployed these LLPs on a sensor actuator network installed in an office, and then we studied how the total traffic was reduced by MMT and MMST. Actually, we compared with minimumizing total traffic (MTT) algorithm, which actually solves the most optimal case. MTT is not practially useful because it takes too much CPU time especially if the number of components increases by $O(n^c)$: n is the number of network nodes and c is the number of components (see Section 3.4.2). We just present the result for a reference.

We assumed an office environment in the experiment. We deployed 40 nodes uniformly in an area of 10[m] $\times$ 25[m] square meters and connected the nodes if the distance between them are within 5[m]. Fig. 3.5 illustrates the network topology generated in this way. We let each node have one sensor and one actuator.

We admit that wireless links are usually unstable and the connectivity depends on the distances, obstacles, multipath interferences and so on. However, as our

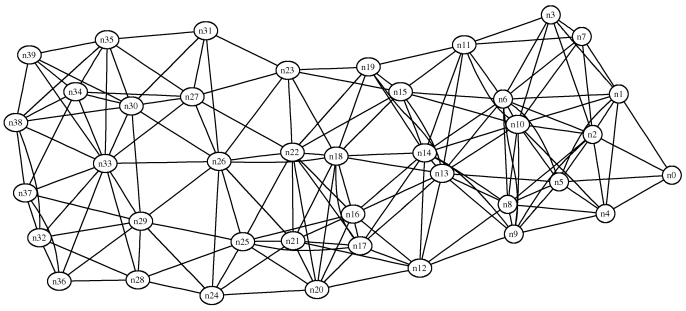

Figure 3.5: The network topology for total traffic evaluation

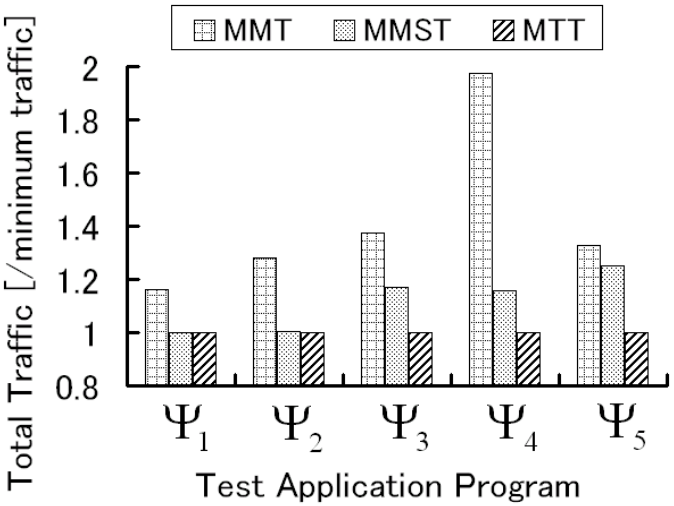

Figure 3.6: Total traffic optimization by MMT and MMST (compared with MTT)

focus in this evaluation study is the effects of traffic optimization algorithms, we modelled the network as above.

Fig. 3.6 is the result of total traffic on the deployment plan produced by MMT, MMST and MTT. It shows the magnitude of them compared to the ideal (i.e., most optimal) cases because the base traffic differs among the user LLPs.

In general, MMST performed better optimization than MMT. For Ψ_1 and Ψ_2, MMST achieved the ideal optimization. MMT sometimes gives 2 times larger traffic than the ideal optimization as Ψ_4.

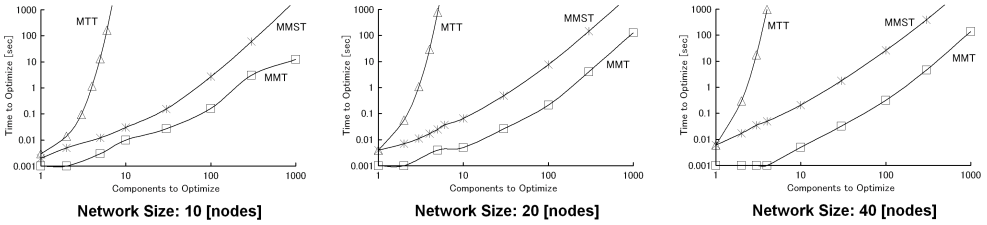

Figure 3.7: CPU time consumed in the optimization by MMT, MMST and MTT.

3.4.2 CPU time

The goal of the following experiment was to evaluate the scalability of algorithms regarding to CPU time. As the number of components in a LLP increases, the optimizer takes larger time to find the optimal deployment plan. We tested the CPU time on different scales of LLPs.

We did simulation-based experiments for the evaluation. We first prepared networks of 10 nodes, 20 nodes and 40 nodes. We then increased the number of components to be assigned by optimizer from 1 to 1000. We implemented the optimizer in Java 1.6 and carried out it on Intel(R) Celeron(R) CPU 430 at 1.80GHz.

Fig. 3.7 shows the CPU time for finding the optimal deployment plan by MMT, MMST and MTT. CPU time by MTT increased suddenly around 4 to 6 components, indicating that MTT cannot be used in most of practical cases. CPU time by MMST and MMT gradually increased in the same form, but MMST took 10 or 100 times larger than MMT. In some large scenarios, MMST might be stressful. However, MMST performs better optimization in reducing total traffic, thus, debugging of LLP can be made with MMT optimization, and permanent deployment can be made with MMST optimization.

3.5 Discussion

The experiment has shown that MMT and MMST nearly optimize the total network traffic, though those algorithms require $O(c^3 + nc^2)$ or $O(nc^3)$ for computation. This result indicates that other optimization algorithms (e.g., for delivery latency, load-balancing and fault-tolerance[28][29]) can be also applied to wireless sensor actuator networks in the CCDM architecture.

We here summarize the issues left untouched. First, we assumed stable links among wireless nodes. This was because we focused on the evaluation of MMT and MMST, and the study of fault-tolerance against unstable wireless links were out of focus. Second, we considered that user LLP does not change after the deployment. Some users may want to change LLP dynamically on instances by adding other components and flows to the existing LLP incrementally (e.g., [26]). Third, the

traffic on each flow was already given. They should be somehow estimated by statistical analysis or simulation before carrying out the optimization when we apply MMT and MMST to the practical systems.

3.6 Conclusion

LLP and CCDM allowed the implementation of user instructions onto wireless sensor and actuator networks. In this research area, CQL-based sensor networking[6] has been widely-studied. However, we proposed to make use of LLP for sensor actuator networking and demonstrated the possibility of this model.

In our experiment, CCDM has shown its capability of computing moderately complex algorithms with the traffic optimization case. We consider that this capability makes other optimizations feasible, such as delivery latency, load-balancing, and fault-tolerance.

Table 3.2: Sample program for c_1^s and c^{sel}

component	program

c^{s_1}	`<define interface="out1" type="output" mode="server" />`
	`<define interface="out2" type="output" mode="client" />`
	`<define device="s1" type="sensor" />`
	`<thread>`
	`  <while>`
	`  <true/>`
	`  <OUTPUT interface="out1"><READ device="s1" /></OUTPUT>`
	`  </while>`
	`</thread>`
	`<thread>`
	`  <while>`
	`  <true/>`
	`  <progn>`
	`  <OUTPUT interface="out2"><READ device="s1" /></OUTPUT>`
	`  <sleep><int>1</int></sleep>`
	`  </progn>`
	`  </while>`
	`</thread>`

c^{sel}	`<define interface="out1" type="output" mode="client" />`
	`<define interface="in1" type="input" mode="client" />`
	`<thread>`
	`  <while>`
	`  <true/>`
	`  <progn>`
	`  <OUTPUT interface="out1">`
	`  <if>`
	`  <gt>`
	`  <INPUT interface="in1" />`
	`  <double>20.0</double>`
	`  </gt>`
	`  <text>HOT</text>`
	`  <text>COLD</text>`
	`  </if>`
	`  </OUTPUT>`
	`  <sleep><int>60</int></sleep>`
	`  </progn>`
	`  </while>`
	`</thread>`

Part III

Data Transportation over Intermittently Connected Networks

Chapter 4

Potential-Based Entropy Adaptive Routing

4.1 Introduction

Delay or disruption tolerant networking (DTN)[1] has enabled application message delivery over intermittently-connected mobile networks by its hop-by-hop store-carry-and-forward scheme. Applications of DTN include pervasive communication [30, 31, 32], car-to-car communication[33, 34], village-to-village communication, natural disaster situations and interplanetary communication. It is widely acknowledged that we cannot apply the traditional Internet and MANET routing protocols[35, 36, 37, 38] to such networks because of the intermittent link connectivity and highly dynamic network topology.

Researchers have proposed several routing schemes for DTNs but often assuming particular mobile environments. DTLSR[39] has studied link-state routing in village-to-village scenarios. MaxProp[40] and RAPID[41] discussed message delivery in the context of city buses. PRoPHET[42] and SOLAR[43] proposed particular sociological mobile scenarios and the routing evaluation were based on them. Many other works(e.g.,[44, 45, 46, 47]) were proposed with random waypoint (RWP) mobility. Past literatures did not discuss *entropy-adaptiveness* in DTN routing.

Entropy, which we introduce in this work, seamlessly classifies networks or mobile environments. According to the original definition (provided in section 4.6), entropy shows how uniformly a node contacts with other nodes. For example, like village-to-village scenario, where a node is expected to contact with only a small set of pre-known nodes, provides small entropy. Levy walk[48] and RWP, where a node unexpectedly encounters a larger number of (sometimes previously unknown) nodes, provides large entropy. Fig. 4.1 shows how the contact entropy depicts mobile environments. It shows aggregated contacts of mobile nodes and the contact entropy associated to the mobile environment.

We propose potential-based entropy adaptive routing (PEAR), which adap-

tively changes message delivery formations for different entropy situations. In small entropy cases, it delivers messages to their destinations in a straightforward manner with almost shortest path. In large entropy cases, it creates redundant paths for message propagation and maintains delivery probability and latency. Interestingly, PEAR achieves this entropy-adaptiveness without being aware of contact entropy. Opportunistic contact patterns enables this change under PEAR routing algorithm.

In general, PEAR takes utility-based replication scheme. However, in this thesis, we use the term *potential* instead of *utility*. PEAR inherits the concept of potential-based routing (PBR)[49] that a router has scalar values called potentials, each associated to a particular destination. A router forwards a message to its neighbor that has the lowest potential. Instead of just forwarding messages, PEAR propagates them by replication. Replication, which makes redundant delivery paths, maintains delivery propability especially in large entropy cases.

We recognize several methods have been proposed for utility computation [50, 32, 42, 51]. However, these methods do not provide good performance in small entropy cases. Some methods only use direct hop statistics or other particular models (e.g., velocity of nodes, sociological structure). Although, PRoPHET seems to care multihop topology in the literature, it does not appropriately develop utility values.

Generally, hop-by-hop routing takes more important role in small entropy cases. Each node must know what nodes exist behind which neighbor node to forward messages appropriately. However, in large entropy cases, movement or contacts take more important role than hop-by-hop routing. If the contacts are not deterministic, to find the most optimal path is almost impossible. Replication would be the only method to increase the possibility of message encountering the destination.

We associate our potential computation method with the traditional distance-vector (DV) routing and the well-known diffusion equation. The basic idea behind DV routing is to forward a packet to the router that has lower *hop count* to the destination. Here, in the DTN routing terms, the hop count is *potential* or *utility*. Routing information protocol (RIP)[36], an implementation of DV routing, generates hop counts by diffusing subnet information with increasing them hop-by-hop. Potentials that PEAR develops in static (i.e., very small entropy) networks become the same as DV routing does. Our method is very similar to the diffusion theory at the equation level. We discuss how it is similar in section 4.4.3.

We have carried out (1) simulation-based and (2) implementation-based experiments for evaluation. The main purpose of the simulation-based studies is to compare PEAR with other routing methods. We create several mobile environments from small entropy to large entropy by using community-structured environment model as Fig.4.1. In the simulation, we assume an ideal environment (e.g., ideal data link and infinite capacity of storage), while recognizing the real performance would be affected by many environmental features, for example, by media-access control (MAC) protocol and radio interference as well as the structure of mobility.

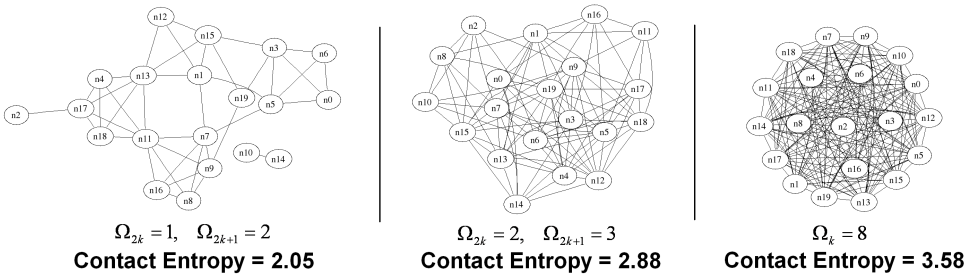

Figure 4.1: Aggregated contacts of mobile nodes and its contact entropy ($\overline{S}$). As the entropy becomes large, the network becomes closer to mesh topology. These mobile environments were generated by community-structured environment model (section 4.7.1).

The implementation-based evaluation is for proof-of-concept. With real software and hardware implementation, we verify and demostrate how PEAR works on several experiment cases.

This chapter is organized as follows. In section 4.2, we present related works. In section 4.3, 4.4 and 4.5, we propose potential-based entropy adaptive routing. Section 4.6 provides the definition of contact entropy and explains how the message delivery formations change depending on the entropy. In section 4.7, we describe our simulation results. Our implementation-based works are presented in section 4.8. We discuss the evaluation results in section 4.9. Section 4.10 provides the conclusion of this chapter.

4.2 Related Work

Several routing schemes for DTNs were proposed in the past literatures. Demmer et al.[39] applied link-state routing (LSR) to village-to-village communication in developing regions. Depending on the methods of computing link costs, maximum delivery probability, minimum expected delay and minimum expected dependent delay has been studied in[52]. However, in our study, LSR is effective only in small entropy cases. In large entropy cases, it does not work well because a node does not forward messages to potential carrier nodes encountered unless it is designated as the most optimal next hop.

Epidemic routing [44] ensures message delivery even in partitioned networks of highly dynamic topology. Basically, epidemic routing is flooding-based routing scheme, which copies a message to all the nodes encountered, and the copy-received nodes start to copy the message in the same manner. It ideally achieves minimum delivery latency, but, it surely consumes lots of transmissions and buffer space, which results in traffic congestion and poor performance in some situations.

Spray and Wait (SW)[45] improves the overhead of message replication by controlling the number of message copies. Message routing in SW is composed of two phases. First, the network sprays the limited number of message copies in epidemic way. Then, it waits until one of the nodes encounters the destination. In our study, controlled-replication like SW is useful in large entropy cases. It cannot appropriately deliver messages in small entropy situations.

Utility-based routing was proposed by Chen et al.[50] in mobile ad hoc networks (MANET) to achieve disconnected transitive communication. Utility is a scalar value that shows a heuristic proximity to the destination. In that, utility-based routing is the same as potential-based routing in nature. A node relays a message to the higher utility nodes that are expected to deliver the message with higher probability. Utility-based routing for DTNs is often combined with controlled replication. There are a number of utility computation methods in literature. Chen et al. [50] proposed most-recently-noticed, most-frequently-noticed. History-based protocol was proposed with ZebraNet[32]. PRoPHET[42] proposed delivery predictability with aging constant and transitive property. Spyropoulos et al.[51] proposed last-seen-first, most-mobile-first and most-social-first. CAR[53] provides attribute-based general framework for computing utility that can be aware of remaining battery and other properties as well as contact statistics. In these works, though they take multihop message forwarding scheme, they use only direct hop statistics or other particular models (e.g., velocity of nodes, sociological structure) for utilitity computation. PRoPHET, which seems to care multihop network topology by *transitive property*, in fact, does not appropriately develop utility – in our study, very far nodes from the destination sometimes have great predictability to the destination.

Potential-based routing (PBR) was proposed by Basu et al. [49] in the context of traffic engineering in stable networks. In PBR, a node forward messages toward the neighbor that has the lowest potential. Followed by this work, PWave[54] has applied PBR to wireless sensor networks for routing of sensor data to sink nodes. Volcano routing scheme (VRS)[55] is also an extension of PBR that creates potential-field to diffuse messages from densely message buffered areas.

We propose potential-based entropy adaptive routing (PEAR). It can be applied to both small and large entropy cases. In small entropy cases, PEAR behaves as distance-vector routing on multihop network topology and provides cost-effective routing. In large entropy cases, it replicates messages and provides high delivery probability. In our survey, past works did not mention such entropy-adaptiveness.

We proposed the basic idea of PEAR in the past[56]. However, the evaluation was made only by simulation and it did not use contact entropy to describe mobile environments. DTIPN[57] carried out testbed-based experiments making use of PEAR. However, the main focus in the DTIPN work was the proposal of another architecture for disruption tolerant networking. It has just used PEAR as one of the routing components to evaluate the architecture. Inheriting these works, this

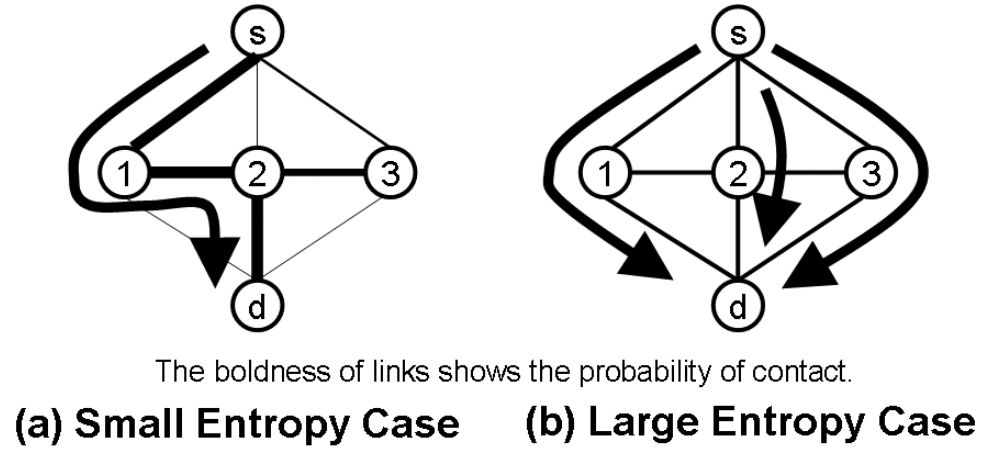

Figure 4.2: Entropy-adaptive message delivery on small entropy network and large entropy network. The boldness of links shows the probability of contacts. (a) chooses the best path on multihop network topology, (b) makes redundant paths and maintains delivery probability and latency

chapter provides the summary of our works on PEAR with some additional theoretical descriptions for distance-vector aspect, similarity to the diffusion equation and entropy-adaptiveness.

4.3 Potential-Based Entropy Adaptive Routing

This section describes the design principles of PEAR and the system architecture with defining some terminologies. Details of the proposal are provided in the following sections. Section 4.4 describes its potential-field construction method, and section 4.5 describes its message propagation method. Section 4.6 provides the definition of entropy and how the delivery formations change depending on the entropy of mobile environments.

4.3.1 Overview and design principles

PEAR autonomously performs effective routing over wide-range of entropy cases. Every node learns what nodes are nearby and what nodes exist over intermittent connectivity in ad hoc manner. It adaptively changes the delivery formations sometimes replicating messages in the network to improve delivery probability and latency depending on the contact or mobility model.

Fig. 4.2 shows how the formations of message delivery should change depending on the pattern of node contacts. A bolder link indicates higher probability of node contacts, and thinner link indicates less contacts. This figure presents two network

environments; the left one is characterized as small entropy case, and the right one is as large entropy case. In the small entropy case, a node mostly meets with some particular nodes. As entropy increases, contact probability between nodes becomes almost uniform. We provide the formal definition of *entropy* in section 4.6.

In our design principles, in small entropy cases, PEAR should perform message delivery in the best form with shortest path, by choosing the closer (and it is the best) hop at every transmission. In large entropy cases, where estimation of the best path becomes meaningless, the main delivery force should be the replication of messages in the network. It increases the probability to encounter the message for the destination.

We design PEAR to achieve this adaptiveness autonomously without being aware of entropy itself. PEAR takes potential-based replication scheme for message routing. A node copies a message to the node that has lower potential among the neighbors, and finally the message will reach the lowest point, which should be the destination. In large entropy cases, potentials of PEAR become very dynamic and it produces lots of branched delivery paths.

The potential-field construction scheme of PEAR has tight relationship with the well-known diffusion equation. Each node advertises itself (i.e., node ID) with potential 0, and PEAR diffuses the potential information associated to the node ID in the network with periodically increasing it. If the network is stable and connected with each other, the potential-field converges into the same pattern that distance-vector protocols develop. Even if the network becomes less stable and more sparse, it still has tolerance for network disruption. When the node movement becomes complicated, the potential-field becomes very dynamic, and PEAR creates redundant paths with maintaining delivery probability and latency.

4.3.2 Notations

Let N be a set of nodes and $nbr(n)$ be a set of neighbors of node $n(\in N)$. Here, in our definition, $nbr(n)$ includes n itself: i.e., $\{n\} \subseteq nbr(n)$.

A node has potential values, each of which is associated to a particular destination. We denote a potential value by $V^d(n,t)$: the potential of node $n(\in N)$ associated to destination $d(\in N)$ at timeslot t. We define the dynamics of potentials (i.e., how autonomously PEAR develops potential values) in section 4.4.

Potential-field is a subset of potential values which are associated to a particular destination. For example, potential-field for destination $d(\in N)$ is provided by $\{V^d(n,t)|n \in N\}$.

A node should have message next hop information for each destination. We denote it by $nexthop^d(n,t)$, which shows the next hop node for destination d at node n at timeslot t. A node n generates $nexthop^d(n,t)$ from available and valid potential values advertised by neighbors. Section 4.5.1 gives the formal definition to generate it.

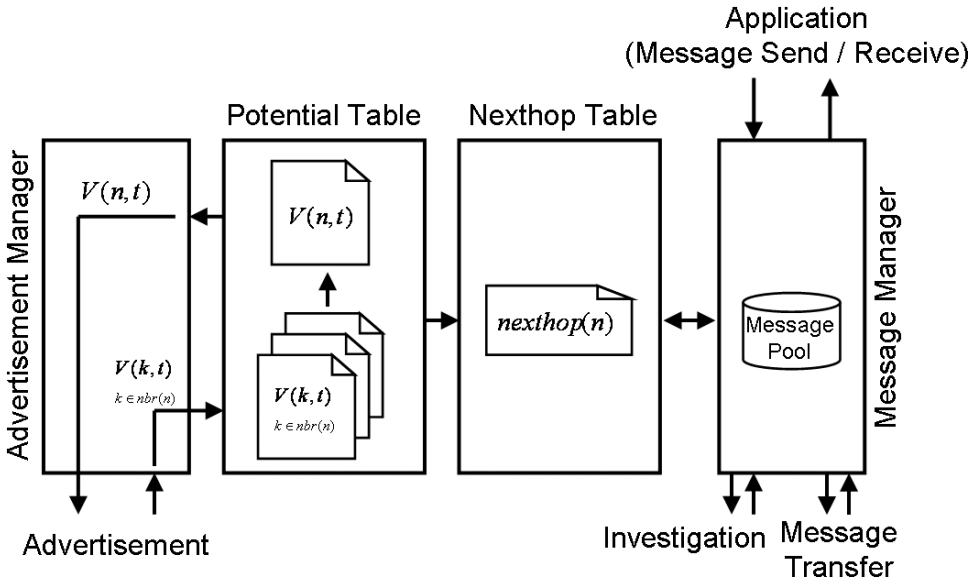

Figure 4.3: PEAR node design. Advertisement manager(AM) and potential table(PT) construct potential-fields (section 4.4). Nexthop table(NT) and message manager(MM) propagate application messages (section 4.5).

$M(n,t)$ is a set of messages stored in node n at timeslot t. $m \in M(n,t)$ has several attributes that denote, for example, destination or source of it. We describe the detail in section 4.5.2 replica management.

4.3.3 PEAR node design

Fig. 4.3 provides the component diagram of node n. The role of each functional component is as follows.

- **Advertisement Manager(AM):** AM periodically advertises $\{V^d(n,t)|d \in N\}$ to and receives $\{V^d(k,t)|d \in N\}$ from the neighbors ($k \in nbr(n)$) by, for example, radio-range multicast. AM submits the received $\{V^d(k,t)|d \in N, k \in nbr(n)\}$ to potential table (PT), and publishes the current $\{V^d(n,t)|d \in N\}$ obtained from PT to the neighbors.
- **Potential Table(PT):** PT manages all the potentials of neighbors $\{V^d(k,t)|d \in N, k \in nbr(n)\}$ and computes the potential of the next time step $\{V^d(n,t+1)|d \in N\}$ from $\{V^d(k,t)|d \in N, k \in nbr(n)\}$. Section 4.4 provides the formal definition of this computation.
- **Nexthop Table(NT):** NT generates and maintains nexthop table for each destination d, using potentials $V^d(n,t)$ and $\{V^d(k,t)|k \in nbr(n)\}$ managed

at the PT. Section 4.5.1 provides the formal definition.

- **Message Manager(MM):** MM implements a replica management scheme discussed in section 4.5.2. It provides application programming interface (API) for sending and receiving application messages.

4.4 PEAR Potential-Field Construction

4.4.1 Algorithm

We take diffusion approach for potential-field construction. It has twofold: (1) the advertisement of node existence and (2) the diffusion of the existence information. Each node advertises itself as potential 0, and other nodes diffuse the information to neighbors with increasing the potential value.

Nodes periodically exchange their potential vector (i.e., $\{V^d(n,t)|d \in N\}$) among their neighbors by means of, for example, radio-range multicast. And, each node computes potential values in the following rule.

$$\begin{aligned} V^d(n,t+1) &= V^d(n,t) \\ &+ D \min_{k \in nbr(n)} \{V^d(k,t) - V^d(n,t)\} \\ &+ \rho \qquad (4.1) \\ V^d(d,t) &= 0 \qquad (4.2) \\ 0 < \rho < D \quad &, \quad 0 < D < 1 \qquad (4.3) \end{aligned}$$

The potential of destination is always tied to 0 (Eqn. 4.2). Other potential values dynamically change depending on node-contact patterns. A potential normally grows by ρ at every timeslot, but decreases when the node has encountered a node of smaller potential value (Eqn. 4.1). D is a diffusion parameter. If it becomes larger, potential values decrease faster when it has encountered lower nodes, and dissemination of low-potential information becomes faster.

At the very early stage of potential-field construction, a node does not know all the nodes in the network over intermittent connectivity. In this phase, the node does not make any potential computation for such unknown destinations. However, the potential values associated to such destinations disseminate over the intermittent connectivity. First, a node opportunistically encounters another node and exchanges potential values each other. Then, it finds that some destinations are not in the local list of potentials. So, it adds to the list and starts computation for the new destinations. The initial potential could be the received potential. In this way, every node autonomously learns what nodes exists in the network even they are intermittently connected.

4.4.2 Distance-vector routing in stable scenarios

If the network is stable and connected, a potential-field converges into the same pattern that DV protocols develop; it increases linearly hop-by-hop from the destination. More formally, if the network is stable ($\Leftrightarrow \{nbr(n)|n \in N\}$ are static) and connected, PEAR gets,

$$\forall n \in N, \lim_{t \to +\infty} V^d(n,t) = h(d,n)\frac{\rho}{D} \tag{4.4}$$

Here, $h(d,n)$ is the minimum hop count from node d to n. Thus, it has an aspect of distance-vector routing. PEAR reveals this in small entropy cases especially in the stable networks.

4.4.3 Similarity to the diffusion equation

The potential-field construction method diffuses potential values with increasing originally advertised 0 potential. We associate this PEAR's diffusion scheme with the widely-known diffusion equation.

First, we remove parameter $+\rho$ from Eqn. 1, then we get,

$$\begin{aligned} V^d(n,t+1) &= V^d(n,t) \\ &+ D \min_{k \in nbr(n)} \{V^d(k,t) - V^d(n,t)\} \end{aligned} \tag{4.5}$$

Next, we swap min operator with $\sum$ operator.

$$\begin{aligned} V^d(n,t+1) &= V^d(n,t) \\ &+ D \sum_{k \in nbr(n)} \{V^d(k,t) - V^d(n,t)\} \end{aligned} \tag{4.6}$$

This is the diffusion equation for discrete systems. This corresponds to the following continuous forms.

$$\frac{\partial V^d(n,t)}{\partial t} = D \text{ div grad } V^d(n,t) \tag{4.7}$$

$$\text{thus, } \frac{\partial V^d(n,t)}{\partial t} = D\nabla^2 V^d(n,t) \tag{4.8}$$

This means that we can associate the potential-field construction scheme with the analogy of temperature diffusion among contacted metallic balls. However, though the $\sum$ operator gets both lower and higher temperature from its neighbors, the min operator gets only the lower temperature from them.

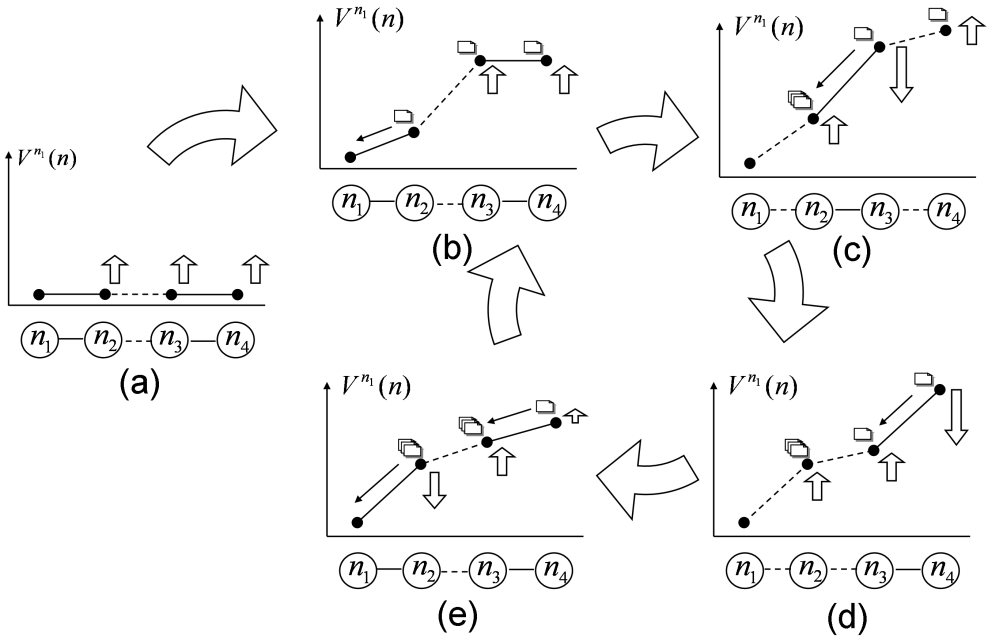

Figure 4.4: Dynamics of a potential-field for n_1 and message delivery flow in a small entropy case

When two different potential nodes encounter, this means that the larger potential node has a link to a more possible (i.e., smaller potential) node than itself. According to the potential-field construction method, the potential at the larger side decreases with showing that it has got more delivery possibility to the destination.

This concept was not proposed in the past utility-based routings. In ZebraNet[32], Chen et al.[50] and Spyropoulous et al.[51], though message propagation takes multihop forwarding or replication, they take direct hop statistics or other particular models (e.g., velocity of nodes and sociological structure) for utility computation, and multihop cases were not sufficiently discussed. PRoPHET[42] tried to include multihop cases into its utility computation scheme by *transitive property*. However, in our study, very far nodes from the destination could have great predictability to the destination. This phenomenon drives messages unpreferable way and works against delivery probability and the cost of transmissions.

4.4.4 Example

Fig. 4.4 demonstrates how PEAR constructs a potential-field and delivers messages. We use a simple example, which entropy is small, to make the discussion clearer. In this example, n_1 contacts to n_2, n_2 contacts to both n_1 and n_3, n_3 contacts to both

n_2 and n_4, and n_4 contacts to n_3. The links between these nodes are intermittent. In the figure, contacts are denoted by lines, and disconnection by dashed lines. The vertical axis shows the potential for destination n_1: i.e., $V^{n_1}(n)$. In the following discussion, we omit n_1 as $V(n)$ for simplicity.

We must note that we have simplified the figure to keep it clear. First, although the initialization process should be taken as we have described in section 4.4.1, every node has potential from the beginning in the figure. Second, this figure only shows the message flow, and the replica management scheme which we define in the next section is not illustrated.

In the following discussion, $m.dst$ identifies the destination of message m.

(a) Assuming an ideal environment, initially, let all the nodes have 0 as the potential value. At this moment, $V(n_1)$, $V(n_2)$ and $V(n_3)$ increases by ρ.

(b) As time elapses, $V(n_2)$ stays at $\frac{\rho}{D}$, while $V(n_3)$ and $V(n_4)$ continues to increase at speed ρ. Here, n_2 copies $\{m|m.dst = n_1 \wedge m \in M(n_2)\}$ to n_1, while n_3 and n_4 store messages in their local memory.

(c) In this situation, only n_2 and n_3 are in contact. $V(n_2)$ begins to increase at speed ρ, while $V(n_3)$ decreases toward $V(n_2)$. n_3 copies $\{m|m.dst = n_1 \wedge m \in M(n_3)\}$ to n_2.

(d) Now, n_2 and n_3 are disconnected. Instead, the link between n_3 and n_4 has been setup. $V(n_2)$ and $V(n_3)$ increases at speed ρ, while $V(n_4)$ decreases toward $V(n_3)$. n_4 copies $\{m|m.dst = n_1 \wedge m \in M(n_4)\}$ to n_3.

(e) n_2 is now in contact with n_1. $V(n_2)$ decreases to $\frac{\rho}{D}$. n_2 transfers $\{m|m.dst = n_1 \wedge m \in M(n_2)\}$ to n_1. The topology of (e) is the same as (b), and the pattern of potential-field at (e) will become (b).

In this way, PEAR dynamically develops potential-field and delivers messages to the destination.

4.5 PEAR Message Propagation

PEAR propagates a message hop-by-hop by replication. After the delivery has been confirmed, PEAR removes the replicated messages stayed in the network. We combined these two tasks (i.e., message propagation and deletion) as a replica management algorithm in PEAR.

This section describes (1) nexthop selection method and (2) the replica management algorithm. They are implemented in the nexthop table (NT) and the message manager (MM) respectively (Fig. 4.3).

4.5.1 Selection of the next hop

NT creates next hop information for each destination in the following rule.

if $\max_{k \in nbr(n)}\{F_k^d(n,t)\} > \alpha$,

$$nexthop^d(n,t) = \mathrm{argmax}_{k \in nbr(n)}\{F_k^d(n,t)\} \tag{4.9}$$

else,

$$nexthop^d(n,t) = \{\} \tag{4.10}$$

Here, α, a positive constant parameter, is the threshold of message transfer. If $F_k^d(n)$ exceeds α, the nexthop is the neighbor node k that gives maximum $F_k^d(n,t)$. If not, no nexthop is provided for destination d, which means that the messages should be stored in the local buffer. $F_k^d(n)$ is the force that affects on the messages $\{m|m.dst = d \wedge m \in M(n)\}$ from node n toward neighbor k, which we define as

$$F_k^d(n,t) = V^d(n,t) - V^d(k,t) \tag{4.11}$$

4.5.2 Replica management

MM manages propagation and deletion of messages in the network. The basic idea of the algorithm is as follows. After a node decides a next hop, it asks the next node what kind of information it has for a particular message. Depending on the response, it (1) makes a copy of the message into the next hop, or (2) delete the message body from the local buffer.

Fig. 4.5 defines the replica management algorithm. m represents a message, and the *nexthop* is the next hop candidate for m. Here, m has attributes as follows.

- $m.dst$: destination of the message.
- $m.src$: source of the message.
- $m.id$: unique message ID.
- $m.ttl$: the left validity time of the message.
- $m.delivered$: whether the delivery has been certified or not. $m.delivered$ is a local attribute. Every node initializes it to *false* when got a new message copy. This turns *true* when the node has fetched a delivery certification from the neighbor. The first delivery certification should be provided by the destination node after it has received the message.
- $m.disseminationTTL$: left time to actively make its copy when it has chance. $m.disseminationTTL$ is a local attribute. If this has expired, the node does not replicate the message any longer. Every node initializes it to $m.ttl$ when it

```
if nexthop != null and m.delivered = false then

   stat := m.investigation(nexthop)
   if stat = NEIGHBOR_NOT_FOUND then
      return
   endif

   if stat = MESSAGE_DELIVERED then
      m.deleteContent()
      m.delivered := true
      return
   endif

   if stat = MESSAGE_NOT_HAVE and
     (m.disseminationTTL>0 or nexthop = m.dest)
     then
      m.copy(nexthop)
   endif

   if m.disseminationTTL>DISSEMINATION_TIME then
      m.disseminationTTL := DISSEMINATION_TIME
   endif
endif
```

Figure 4.5: Replica management algorithm for message propagation and deletion

got a new message copy. After making another copy, it sets $m.disseminationTTL$ to DISSEMINATION_TIME, a constant parameter that defines how long it can make copies.

- $m.body$: message body. $m.body$ can be *null*, if the node no longer needs the body part.

PEAR defines the following three local methods for m. The replica management algorithm makes use of these methods.

- $m.investigation(n)$ tries to find node n, asks the status of message m at neighbor n and returns the result.
- $m.copy(n)$ tries to copy m to node n.
- $m.deleteContent()$ clears the body of the message m and sets $m.delivered$ to *true*. This sets $m.body = null$ and frees the allocated memory space.

Here, the response of $m.investigation(n)$ is one of,

- NEIGHBOR_NOT_FOUND: n is not in the transmission range.
- MESSAGE_NOT_HAVE: n does not have the message.
- MESSAGE_ALREADY_HAVE: n already has the message.
- MESSAGE_DELIVERED: n saids that it heard the message has been delivered (delivery certification).

If NEIGHBOR_NOT_FOUND, it does nothing and tries again at the next timeslot. If MESSAGE_DELIVERED, it removes the body of the message from the local buffer and marks that it has been delivered. If MESSAGE_NOT_HAVE and during the $m.DisseminationTTL > 0$, it tries to copy the message to the nexthop node.

PEAR applies this algorithm to messages in two ways. When a node has received a message from another node, it applies to the received message. It applies to all the stored messages when a periodic timer has been triggered.

In the above description, in order to simply the discussion, investigations are made one-by-one, but practically, at the implementation phase, they should be done by summary vector as Epidemic routing[44] does to reduce the overhead.

4.6 PEAR Entropy Adaptiveness

This section describes how the delivery formations change depending on mobile environments from small entropy cases to large entropy cases. First, we introduce the formal definition of *contact entropy*, and then we describe how PEAR delivers a message in both small and large entropy cases. We must remind that PEAR itself does not use (or even calculate) entropy for message routing. Physical movements of nodes or physical contact patterns provide the change to message delivery formations under the PEAR routing scheme.

4.6.1 Contact entropy

We define contact entropy for each node and denote it by $S(n)$. It shows the uniformness of meeting with other nodes in the network. The formal definition of $S(n)$ in this thesis is,

$$S(n) = -\sum_{k \in (N-\{n\})} q_{(n,k)} log_2(q_{(n,k)}) \tag{4.12}$$

$$q_{(n,k)} = \frac{p_{(n,k)}}{\sum_{k \in (N-\{n\})} p_{(n,k)}} \tag{4.13}$$

Here, $p_{(n,k)}$ is the ratio of contact-time between node n and k taken over statistically enough time period.

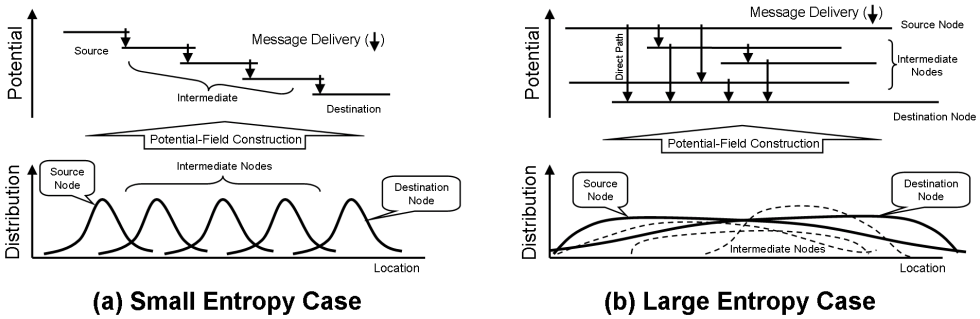

Figure 4.6: Potential-field construction and message delivery. (a) Small entropy case; nodes are locally distributed with small overlaps. A message reaches destination with few redundant paths (no branches in the figure). (b) Large entropy case; nodes are widely distributed with large overlapped areas. PEAR creates redundant paths for message propagation.

If node n encounters only a small set of nodes or particular nodes longer than others, $S(n)$ is small. If it encounters larger number of nodes uniformly, $S(n)$ becomes large.

4.6.2 Entropy adaptive delivery formations

Fig. 4.6 illustrates potential-fields and the formations of message delivery in small entropy and large entropy cases.

The X-axis is location, the Y-axes are the distribution of nodes, and the constructed potential-field associated to a destination. Potential-fields are usually dynamic, but in the figure only the snapshot values are shown.

4.6.2.1 Small entropy cases

Nodes are locally distributed with small overlaps. This mobility pattern would correspond to village-to-village or static-node scenarios, where a node is expected to meet with only a small number of pre-known nodes. PEAR develops the potential-field like distance-vector routing. A message goes down the potential-field at the overlapped areas in a straightforward manner, with few redundant paths, and finally it reaches the destination.

4.6.2.2 Large entropy cases

Nodes are widely distributed with large and shared overlapped areas. Pedestrians and car-to-car contacts, where a node is possible to meet with large number of unexpected nodes, would be classified into large entropy cases. PEAR develops the

potential-field such that several candidates exist for the next hop (not at the same time). Contacts with those candidate nodes trigger the replication of messages, taking a cost of finding faster delivery path. This improves delivery probability and latency as we have described in Fig. 4.2.

4.7 Simulation Results

In order to study entropy-adaptiveness of PEAR routing scheme, we have carried out simulation-based evaluation. We have analyzed message delivery rate and transmission costs for different entropy cases. We compared with link state routing (MED and MDP), Spray and Wait (Tree, LSF), PRoPHET and Epidemic routing. This study was not intended to evaluate them in some particular application contexts. The main purpose of this study was the analysis of the features (e.g., entropy-adaptiveness) of routing schemes. Thus, we have used community-structured environment (CSE) that can generate wide range of mobility patterns from small entropy to large entropy.

Though we have assumed an ideal environment (e.g., infinite bandwidth, infinite capacity of storage), simulations were useful for this purpose. Another reason for taking simulations was its reproductivity; i.e., we could setup the same environment for different routing schemes. We recognize that real performance would be affected by many environmental factors such as radio interference, media-access control (MAC) protocol, buffer capacity as well as mobility pattern. Thus, we also carried out implementation-based evaluation to demonstrate it really worked with the prototype software and hardware (section 4.8).

4.7.1 Community-structured environment

Traditionally, in order to evaluate the performance of routing in this research area, random-based mobility models have been used. Such mobility models include random waypoint (RWP), random walk (RW) and random direction (RD)[58]. Recently, researchers have realized such mobility modes are unrealistic and proposed application-oriented mobility models such as bus[4], taxi[59], sociological orbit[43], pedestrians[60] and levy walk[48]. Research communities like CRAWDAD[1] provide real traces of cars and pedestrians. Application-oriented mobility models or traces might be useful to study the applicability to such scenarios. However, we could not use them to study the feature (especially entropy-adaptiveness) of PEAR. Thus, we developed community-structured environment (CSE), which enables seamless change of entropy simply by setting parameter Ω (described below).

Let N be a set of nodes in the network, and C a set of communities. A community is a holder of nodes like a room which is bound to a physical location. A node $n \in N$ belongs to several communities, which is a subset of C denoted by C_n.

[1] http://www.crawdad.org/

Table 4.1: Environmental Settings

Class	Parameter	Value
Time	no messages	[-50000,0)
	message publish	0
	message routing	[0,20000]
CSE	community	50
general	node	100
setting	p	0.02
	v	50
	(Width, Height)	(1000, 1000)
CSE#1	$(\Omega_{2k}, \Omega_{2k}; \overline{S})$	(1, 2; 1.9)
CSE#2	$(\Omega_{3k}, \Omega_{3k+1}, \Omega_{3k+2}; \overline{S})$	(1, 2, 2; 2.1)
CSE#3	$(\Omega_k; \overline{S})$	(2; 2.6)
CSE#4	$(\Omega_k; \overline{S})$	(3; 3.4)
CSE#5	$(\Omega_k; \overline{S})$	(4; 3.9)
CSE#6	$(\Omega_k; \overline{S})$	(6; 4.4)
CSE#7	$(\Omega_k; \overline{S})$	(32; 4.9)

The node n moves among communities in C_n and it does not visit at any other communities. In CSE, we define two mobility statuses: *stay* and *transition*. In the stay mode, node n stays at one of C_n, which is given by $community(n)$. In the transition mode, n moves from community c_i to community c_j where $c_i, c_j \in C_n$ and $c_i \neq c_j$. A node is in contact with the nodes that stay in the same community. That is,

Node n and k are within direct transmission range

$\Leftrightarrow$

Node n and k are in contact with each other

$\Leftrightarrow$

$$\exists c \in C, community(n) = c \wedge community(k) = c \tag{4.14}$$

When node n is in transition state, $community(n)$ gives *unallocated*.

By changing the number of belonged communities, we can easily change average *contact entropy*. Let Ω_n provide a number of communities node n belongs to: i.e., $\Omega_n = ||C_n||$. For example, if we allocate more communities to n, which increases Ω_n, then node n encounters larger number of nodes and $S(n)$ also increases.

One of the implementations of CSE, which defines movement of nodes, could be as follows, and this is what we have used in the simulation.

1. Node n stays at community $c_i \in C_n$.

2. Choose a random value r uniformly in $[0, 1)$.

3. If $p < r$ or $\Omega_n = 1$, goto 1. Parameter p is probability of transition from *stay* mode to *transit* mode (we used 0.02 in the simulation).

4. Else, choose a destination community c_d from $C_n - \{c_i\}$ at random.

5. Let n move to c_d with transitive time $T(c_i, c_d) = distance(c_i, c_d)/v$. Here, $distance(c_i, c_d)$ is the physical distance between c_i and c_d, and v is the velocity of a node, which is 50 in the simulation setting.

6. After n reached the destination, $c_i := c_d$ and goto 1.

Fig. 4.1 was generated by the CSE model. These CSEs have 20 nodes in 10 communities. By changing Ω_n from 1 to 8, these mobility patterns have got $\overline{S} = 2.05$, $\overline{S} = 2.88$ and $\overline{S} = 3.56$. Here, $\overline{S}$ provides average contact entropy.

Though the CSE implementation is random-based, there is this kind of structure in the contacts of nodes. In that, CSE is different from RWP, RW, RD and Levy walk.

4.7.2 Experiment settings

We summarize environmental settings in table 4.1 and routing parameters in table 4.2.

We prepared 7 CSEs from $\overline{S} = 1.9$ to $\overline{S} = 4.9$. Each CSE has 100 nodes in 50 communities. The experiment was carried out from time $t = -50000$ to $t = 20000$. No messages were sent during $-50000 \leq t < 0$, just learning contact patterns in some routing schemes. At $t = 0$, every node sent messages to all the destinations. Totally, 10000 messages were sent in the network from 100 nodes to 100 nodes. This includes the messages that source and destination are the same. Message routing were simulated during the validity time of the messages: $0 \leq t \leq 20000$.

As for routing schemes, we studied PEAR, Epidemic routing[44], Spray and Wait (Tree-based Greedy and last-seen-first(LSF))[51], PRoPHET[42], and link-state routing (minimum expected delay (MED) and maximum delivery probability (MDP))[52]. Table 4.2 lists the detailed routing parameters.

4.7.3 Delivery rate

Delivery rate is the ratio of delivered messages to all the sent messages.

Fig. 4.7 shows the delivery rate for different entropy cases. Epidemic routing provided the ideal delivery rate on each CSE. The reason even the Epidemic routing did not provide 100% on CSE#1, #2 and #3 is that some nodes were isolated from the major network as Fig. 4.1's left case ($n10$ and $n14$ are isolated).

PEAR provided very good feature in delivery rate – almost the same rate as Epidemic routing did. Link-state routing (MED and MDP) performed well in small

Table 4.2: Routing Parameters

Class	Parameter	Value
PEAR	D	0.01
	ρ	0.001
	α	0.04
	TTL	20000
	DisseminationTTL	500
	AdvertiseTTL	60
Spray and Wait (Greedy)	scheme	tree
	max copy	10
	TTL	20000
Spray and Wait (LSF)	scheme	tree
	max copy	10
	increasing rate($\frac{\partial \tau}{\partial t}$)	0.1
	TTL	20000
PRoPHET	P_{init}	0.75
	γ	0.98
	β	0.25
	TTL	20000
Link State Routing (MED, MDP)	study time period	[-50000,0)
	TTL	20000

entropy cases, but not in large entropy cases. Spray and Wait (Greedy and LSF) and PRoPHET performed well in large entropy cases, but not in small entropy cases.

Fig. 4.8 shows the change of the delivery rate during $0 \leq t \leq 20000$. In the smallest entropy case (CSE#1, $\overline{S} = 1.9$), link-state routing(LSR) and PEAR provided almost the same rate as Epidemic routing did. However, message delivery in SW and PRoPHET were made only at the very early stage and were not enough. In the largest entropy case (CSE#7, $\overline{S} = 4.9$), SW, PRoPHET and PEAR delivered in $1 - e^{-\omega t}$ form (which is fast), but the delivery rate of link-state routing increased almost linearly and could not achieve 100% delivery during the message validity time.

4.7.4 Transmissions

Total message transmission is the total count of copy events that all the nodes did; i.e., how many times they have carried out $m.copy(nexthop)$. Each node must copy a message to forward it to the nexthop, which involves, for example, radio

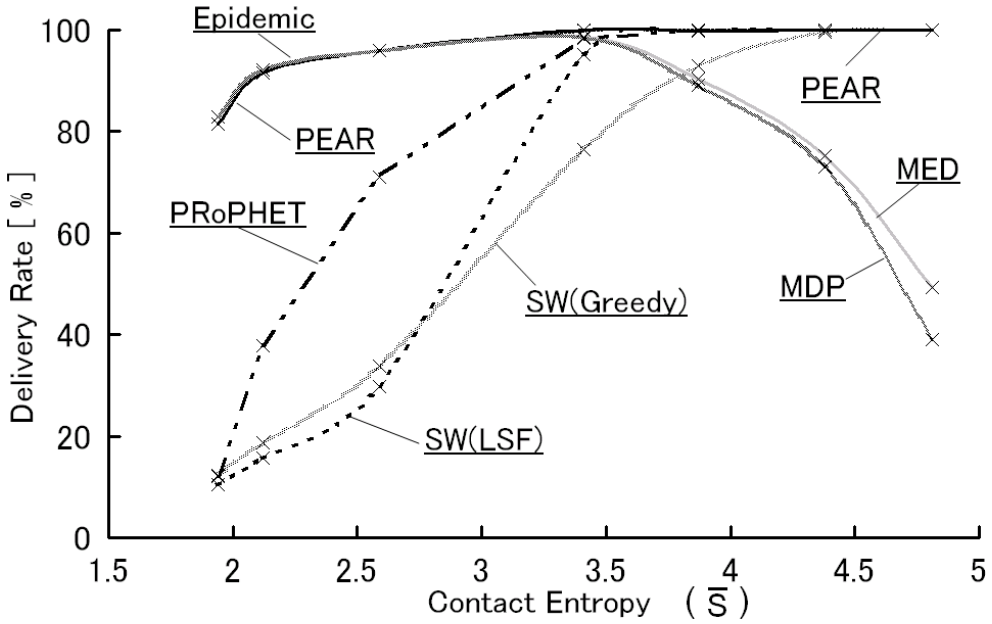

Figure 4.7: Message delivery rate for different entropy cases

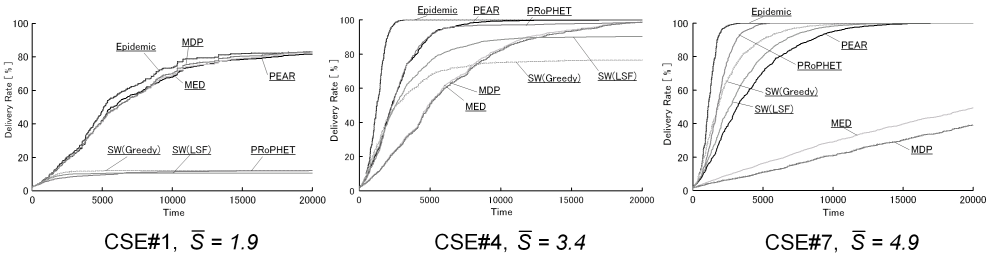

Figure 4.8: Message delivery rate: delivered messages during $0 \leq t \leq 20000$.

transmission. We counted all the events that the whole network did.

Fig. 4.9 shows the transmission results from CSE#1($\overline{S} = 1.9$) to CSE#7($\overline{S} = 4.9$). Epidemic routing provides the upper bound (i.e., maximum limits) of transmissions. PEAR has achieved in 10% transmissions of the upper bound. PRoPHET consumed almost 90% in large entropy cases where it achieved high delivery rate.

SW(Greedy and LSF) provided high delivery rate with about 10% transmissions in large entropy cases. Link-state routing (MED and MDP) also provided high delivery rate with less than 10% transmissions in small entropy cases.

Transmissions of SW(LSF) in small entropy cases and Link-state routing in large entropy cases were about 1% of the upper bound. However, the delivery rates

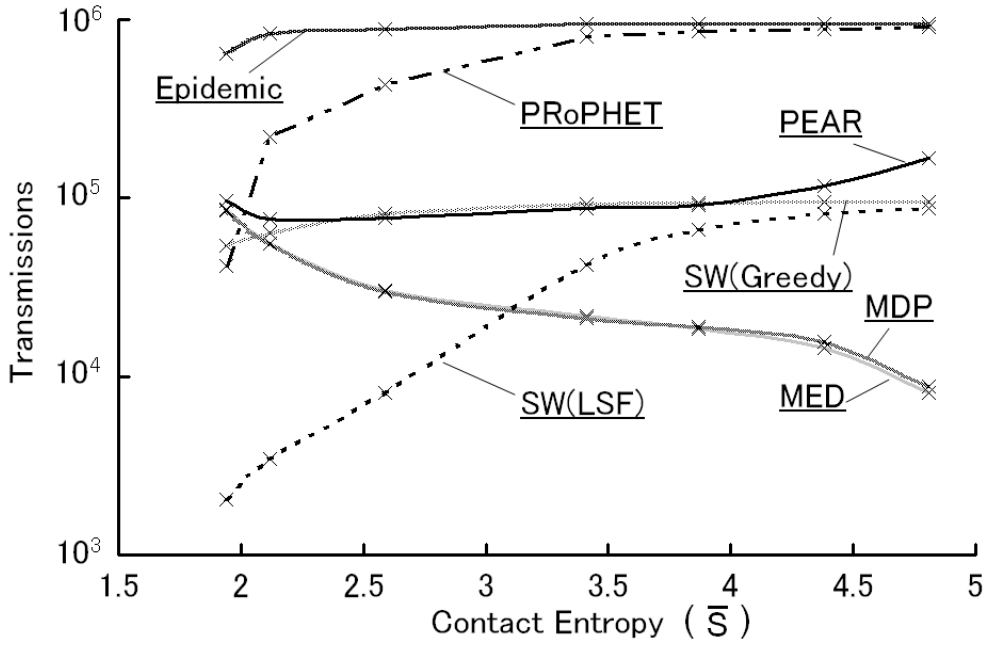

Figure 4.9: Total message transmissions for different entropy cases

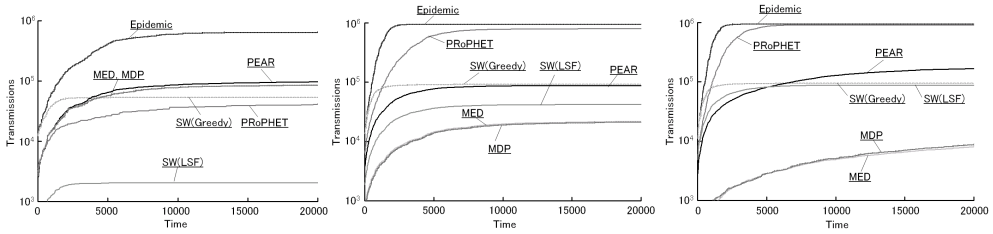

Figure 4.10: Message transmissions from $t = 0$: the number of messages exchanged in the network from $t = 0$.

were not good in those cases (Fig. 4.7).

Fig. 4.10 shows the change of transmissions during $0 \leq t \leq 20000$. In CSE#1, transmissions of PEAR were almost the same as that of link-state routing. This indicates that PEAR forwarded messages in a straightforward manner in the small entropy case. In CSE#7, transmissions of PEAR were about 10 times of link-state routing. This indicates that PEAR replicated messages and increased delivery path redundancy in the large entropy case.

Table 4.3: Testbed Experiment Settings

Class	Parameter	Value
Message	source	1
Transmission	destination	99
	traffic	83[message/min]
	message size	1024[byte]
PEAR	D	0.2
	ρ	0.02
	α	0.04
	TTL	2400
	DisseminationTTL	1800
	AdvertiseTTL	30
	buffer entry	4096
	estimated maximum buffer occupancy	3320

4.8 Implementation and Testbed Experiments

We implemented PEAR and carried out experiments on our testbed system. We have assembled battery powered 11 wifi ad-hoc nodes, and carried out the experiments for four scenarios. We have implemented PEAR as an independent message delivery platform with about 3000 lines in C programming. It has worked appropriately and achieved 100% delivery in the experiments.

Extentional works of PEAR for intermittently-connected mesh networks are provided in chapter 5.

4.8.1 Experiment settings

We carried out the experiments inside and outside a building in a campus, the University of Tokyo. The experiments include static-node cases and dynamic-node cases. We did on static cases, because even if nodes are statically setup, wireless links frequently disrupts in the real environment, causing intermittent connectivity between nodes. This work presents four types of experiment scenarios, which we call (1) StaticA, (2) StaticB, (3) DynamicA and (4) DynamicB.

Fig. 4.11 shows the physical deployment for each experiment, and Table 4.3 provides the summary of configuration. Here, we configured node 1 to send messages to node 99.

StaticA: Nodes were statically deployed, one node for one floor in our building – Eng. Bldg.2, in the university of Tokyo.

StaticB: Nodes were statically deployed, two nodes for one floor in our building.

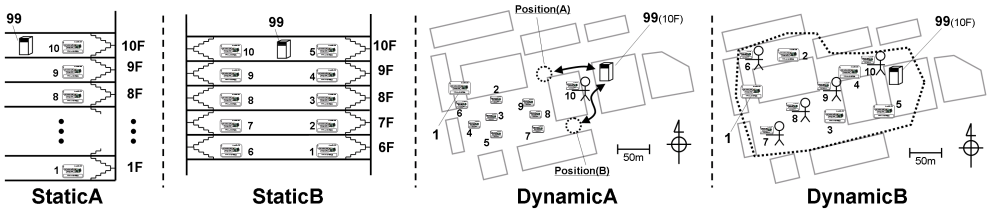

Figure 4.11: Experiment settings in the testbed-based evaluation. We deployed the DTN nodes in Eng. Bldg. 2 (StaticA and StaticB) and in Hongo-campus (DynamicA and DynamicB) of the University of Tokyo.

DynamicA: Nine nodes were statically deployed in the campus, and one node (node 10) has moved between Position(A) and the home position (around node 99), and Position(B) and the home position alternately in 20 minutes per cycle.

DynamicB: Five nodes were statically deployed in the campus, and other five nodes have moved inside the dashed-area freely. They returned to the home position about three or four times during the experiment.

For StaticA and StaticB, we conducted the experiment for 12 hours, and for DynamicA and DynamicB, the experiment was made for three hours and one hour respectively.

4.8.2 Testbed

We programmed PEAR in C and deployed into Armadillo 220[2]. The source code is available from SourceForge.net[3].

Armadillo-220 is an embedded computer with 8Mbyte program memory and 32MByte working memory. It works with ARM9 200MHz CPU and Linux operating system. We added an USB wifi (IEEE802.11g) [4] for ad-hoc communication with radio-range neighbors, and an USB storage to archive the working logs. We used linux-2.6.12.3-a9-15 for its kernel image.

The footprint of PEAR is not large. The C program source code has 3225 lines, and built into about 34k-byte object code.

Our prototype PEAR has 4096 entries for message buffer on its working memory. We consider that it should make use of storage-based buffer for message store, though our current prototype implementation does not support it. However, we have experienced that even with this small buffer, it is likely to be useful for some applications, like pervasive communication.

We packed all of them into a handy box (Fig. 4.12) and assembled 11 packages totally. The system is powered by both battery and external power source.

[2]http://www.atmark-techno.com/en/

[3]http://sourceforge.net/projects/pear/files/

[4]GW-US54Mini2, Planex Communications Inc.

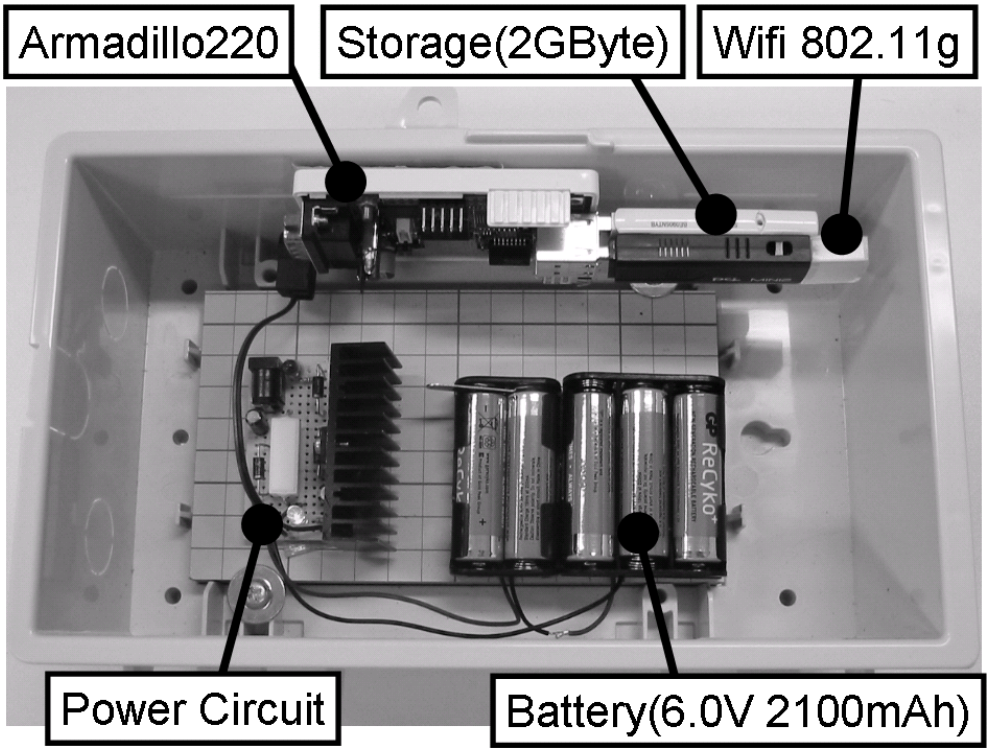

Figure 4.12: A prototype DTN node for PEAR evaluation. We assembled 11 DTN nodes.

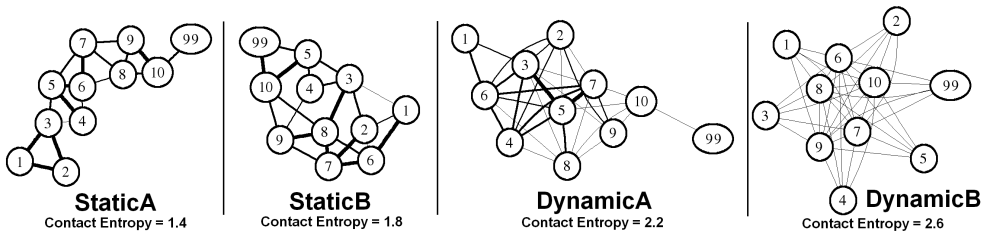

Figure 4.13: Aggregated contacts and average contact entropy. The bolder link indicates higher contact probability.

4.8.3 Contact features

Fig. 4.13 shows the aggregated contact patterns of each experiment. These contact graphs were made by the analysis of the received advertisement published from the neighbor nodes. The links between nodes shows the contacts of them. The bolder link indicates larger average contacted time. The contact entropy (Eqn. 4.12) for

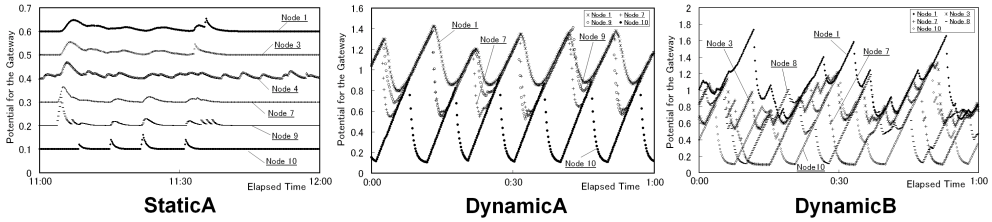

Figure 4.14: The patterns of potential-fields. As entropy increased, the potential-fields became complex.

each case is also presented.

As the topology became complex, the contact entropy increased. The ratio of contacted time has varied, which indicates that the links even between static nodes were intermittent.

In fact, we have observed asymmetric contacts (e.g., node 2 received advertisement from node 3 though node 3 did not receive from node 2). However, it is not presented here because it makes the graph complicated.

4.8.4 Potential-field construction

Fig. 4.14 shows the pattern of potential-field for StaticA, DynamicA and DynamicB. This shows the transition of node's potential-value associated to node 99. In drawing these sequences, we have chosen only a few nodes, for example, the DynamicA only presents the potential of node 1, 7, 9 and 10, because presenting all the potentials made it too much complicated.

In StaticA, potential-field is almost stable but with some ripples caused by intermittent connectivity between nodes. In DynamicA, we can read that node 10 always stayed at lower potential and node 7 and 9 followed in turns. In DynamicB, the potential-field became quite complicated, but we can see that potential of node 1 kept higher than the potential of other nodes in most of the time.

4.8.5 Message delivery pattern

Fig. 4.15 shows the pattern of message delivery from the source (1) to the destination (99).

Each delivery pattern is associated to the propagation of a certain message. The number along with an arrow shows the time when the message has been copied; the time is adjusted so that the arrival time at the destination (99) to be zero. The delivery pattern was always tree. A node did not receive the same message from different nodes because of the replica management algorithm (section 4.5.2).

As we can see, PEAR has created branch or redundant paths for message delivery. Although it has increased the overhead of transmissions and buffer occupancy

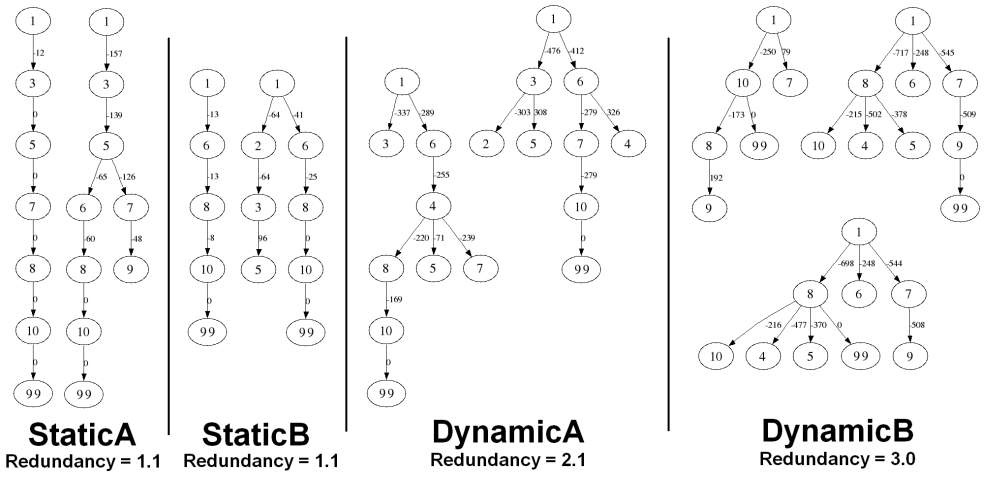

Figure 4.15: Examples of message delivery patterns. Average redundancy became large in larger entropy cases.

but certainly improved delivery latency. For example, in the right case of Static A, the message was first delivered as $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$, then it made a branch from 5 as $5 \rightarrow 6 \rightarrow 8 \rightarrow 10$. During that 7 copied the message to 9. Finally the destination received it from 10. Though it made some redundant transmissions, it has acted effectively for reducing delivery latency.

We also studied the redundancy of delivery. The definition of the delivery redundancy for message m, which we denote by $R(m)$, is,

$$R(m) = \frac{copyCount(m)}{hopCount(m)} \tag{4.15}$$

Here, $copyCount(m)$ is the number of copies made in the network, and $hopCount(m)$ is the number of actual hops directly counted from the source to the destination. If no branches are made for message delivery, $R(m)$ provides 1.0. As the number of copies increases compared to the hop count, $R(m)$ gets larger than 1.0.

In small entropy cases (i.e., StaticA and StaticB), the redundancy were 1.1, indicating that messages were propagated without path branches in most of the cases. In medium and large entropy cases (i.e., DynamicA and DynamicB), delivery paths have made lots of branches and the average redundancy has increased to 2.1 and 3.0.

4.8.6 Latency and delivery rate

Fig. 4.16 shows the distribution of delivery latency for each experiment. 90% of the messages were delivered within 150[sec] in StaticA, 60[sec] in StaticB, 1200[sec] in DynamicA and 800[sec] in DynamicB.

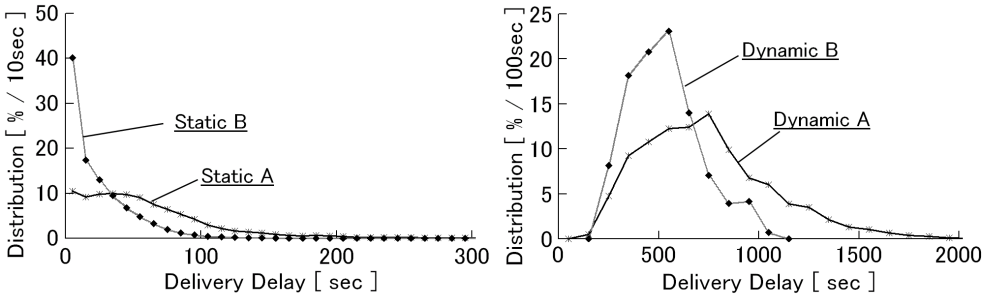

Figure 4.16: Distribution of message delivery latency

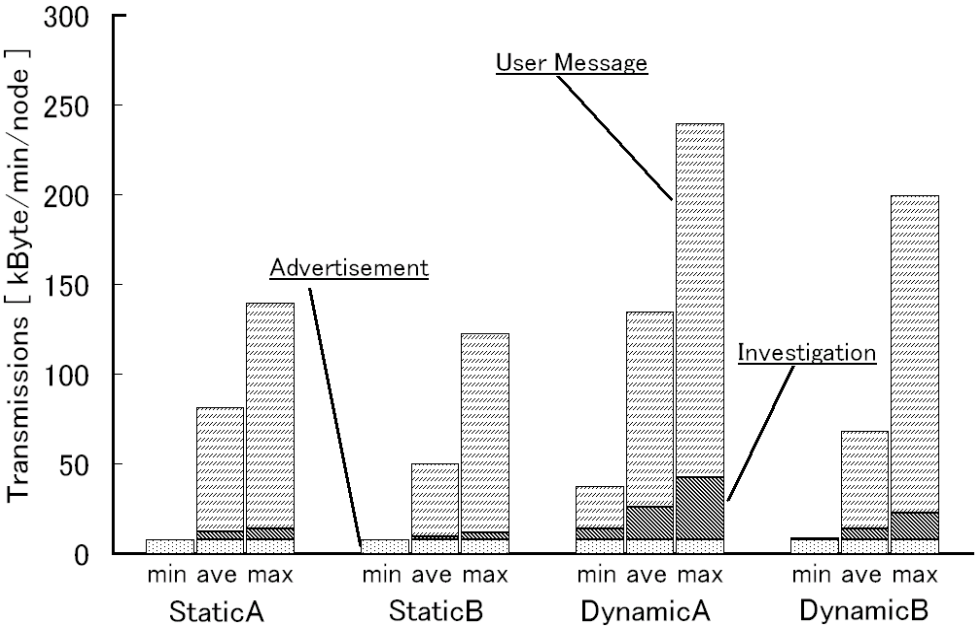

Figure 4.17: Transmission data size a node made per minute. Most (80% – 90%) of the transmissions were consumed by data plane (user message) at average nodes.

The message delivery ratio from node 1 to node 99 were 100% for all the scenarios. Sent and received messages (sent/received) were 59444/59444 in StaticA, 59535/59535 in StaticB, 13811/13811 in DynamicA and 3876/3876 in DynamicB (we did not count the messages that were still propagating in the network at the end period of the experiment).

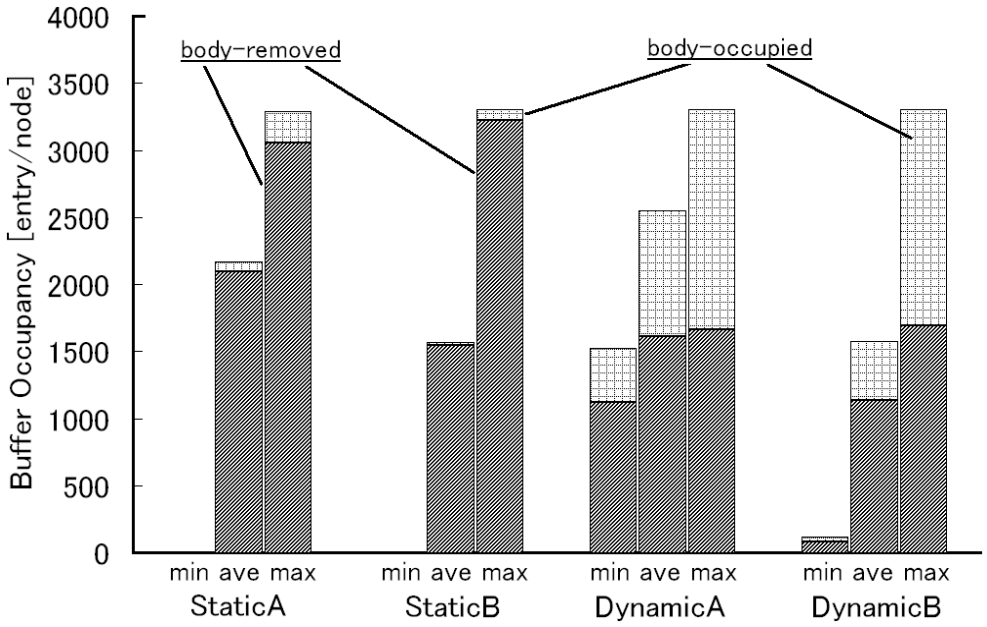

Figure 4.18: Buffer usage patterns. In StaticA and StaticB, more than 90% of the entries had removed message bodies. In DynamicA and DynamicB, about 60% to 75% of the entries of average nodes had removed the bodies.

4.8.7 Transmissions

Fig. 4.17 shows transmissions a node made per minute. We classified them into advertisement, investigation and user message.

The main purpose of this study was to analyze the ratio of control plane (advertisement and investigation) and data plane (user message) in transmissions. As we can see, the ratio of control-plane was approximately 10% – 20%, which indicates that transmission power was consumed mostly by data transfer rather than control signals.

4.8.8 Buffer usage

Fig. 4.18 shows average buffer occupancy of a node. We classified them into body-occupied and body-removed entries. Body-removed entries are such entries that have received delivery-certificate associated to the message (section 4.5.2).

The main purpose of this analysis was to analyze the usage pattern of entries. In StaticA and StaticB cases, more than 90% of the entries had removed message bodies. This is because most of the messages were delivered in a short time (less than 100[sec] in average) and delivery-certificate with body-removed entries stayed there until the TTL had expired.

In DynamicA and DynamicB cases, average nodes had removed message bodies from about 65%–75% of the entries. There were two reasons for this entry usage pattern. First, average delivery time was about 500[sec] and 700[sec], much more than StaticA and StaticB cases. It had to keep the message body for a longer time. Second, since the network had redundantly replicated messages in the network, it took time to remove those messages.

4.9 Discussion

We have introduced the concept of *contact entropy* to express the feature of mobile environments. In small entropy cases, a node encounters a small set of pre-known nodes. A message must take several hops to reach its destination. This requires appropriate routing over multihop network topology. In large entropy cases, a node encounters a larger set of nodes. Since we have assumed that the movement or contacts of nodes are nondeterministic, to find the best delivery path becomes meaningless in these cases. The only way to improve the delivery probability in a limited time is to replicate messages in the network to find faster delivery path in parallel.

In small entropy cases, PEAR has developed such potential-fields that distance-vector protocols do. It propagated messages hop-by-hop directly to the destination by almost shortest path with few branched paths. Link-state routing (LSR) schemes have also taken almost the shortest path. PEAR and LSR have achieved high delivery rate with small transmission cost. Controlled-replication (Greedy Spray and Wait) and previously proposed utility-based routing (PRoPHET and LSF Spray and Wait) could not adapt to the multihop network topology, and could not achieve sufficient delivery rate.

In large entropy cases, the potential-fields of PEAR were very dynamic and the nexthop candidate had changed frequently. PEAR replicated messages in the network and maintained high delivery probability but with small transmissions. Spray and Wait (Greedy and LSF) have also achieved high delivery rate with small transmissions. Moderate replication of messages increased the probability of delivery. In LSR, a node sticked to forward a message to its particular next hop node, while encountering other nodes which could sometimes deliver the message faster.

Interestingly, PEAR has achieved this entropy-adaptiveness without being aware of contact entropy. In small entropy cases, the next hop candidate did not change so frequently. Thus, PEAR created few branched paths in the experiment. However, in large entropy cases, the next hop candidate did change more frequently. PEAR replicated messages to those candidates. This means that the large entropy contacts initiated the message replication.

4.10 Conclusion

We have proposed potential-based entropy adaptive routing (PEAR). PEAR adaptively changes message delivery formations depending on the entropy of the node contacts. In small entropy cases, PEAR works as distance-vector routing, delivering messages with almost shortest path directly, which is the most cost-effective propagation. In large entropy cases, where optimal paths are no longer static and it is very hard to find them, PEAR replicates messages in the network, taking a risk of finding faster delivery paths, and maintains delivery probability.

Prototype-based experiments have shown this feature. In small entropy cases, PEAR developed more stable potential-field and delivered most of the messages in a straightforward manner with few redundant paths. In large entropy cases, it developed dynamic potential-field, and the next hop candidate frequently changed. Redundant paths were generated moderately during the propagation of the messages.

With our simulation-based experiments, we have also confirmed the entropy-adaptiveness of PEAR to the different entropy cases. PEAR has achieved almost the ideal (i.e., Epidemic routing's) delivery rate in any entropy cases with about 10% transmission cost of the wrost (i.e., Epidemic) routing. This feature was not provided by other utility-based and/or controlled replication routing schemes and link-state routing.

Chapter 5

Intermittently-Connected Mesh Networks

5.1 Introduction

Wireless mesh networks frequently suffer from intermittent connectivity even if the network nodes do not move. Intermittent connectivity has been pointed out mostly in the context of mobile cases[42][41][34], and considered as an application area of delay tolerant networking (DTN)[61][1] for such challenged network environments. However, according to the study with real equipments, even static neighbors frequently disappear and become disconnected.

This raises intermittently-connected mesh networks (ICMeN), indicating that this is also an application area of DTN. For this reason, the traditional communication schemes, such as (1) best effort forwarding with end-to-end reliability and (2) message propagation on a single path, sometimes do not work well[62]. Instead, we should take the approaches that DTN researches have explored; i.e., (1) store-and-forward with hop-by-hop reliability and (2) parallel message propagation. We must note that ICMeN is different from intermittently-connected mobile networks (ICMN)[63][58][53][64] because nodes are physically fixed.

In this work, we implemented such DTN communication schemes into UTMesh (Fig. 5.1) and studied how they enabled scalable message propagation. UTMesh is our real world ICMeN: 50-node scale wireless mesh network. We demonstrate that links among static nodes are highly dynamic and that we have to take those schemes in order to increase the scalability in hop counts and the propagation speed.

The traditional approach [38][37] for wireless mesh networking was mostly based on the Internet design principles: (1) best effort forwarding with end-to-end reliability, and (2) single message path. However, the target environment became quite different, which is characterized by intermittent-connectivity and unintentional packet loss. Packet loss sharply kills the traffic and sometimes causes delivery failure under best-effort communication. Intermittent-connectivity frequently

Figure 5.1: UTMesh overview: 51 wireless nodes gathered at the laboratory, in Eng. Bldg. 2 at the University of Tokyo.

changes the physical network topology and the best path soon becomes obsolete or unavailable even during one message delivery. For these reasons, they do not have scalability in hop counts[62].

We analyzed the features of wireless links using UTMesh. The result shows that the mesh network topology is certainly highly dynamic – almost always changing (see Section 5.3). This supports the hypothesis that traditional approaches do not work well in such wireless networks.

The researches of DTNs have recently shown that hop-by-hop reliability and parallel delivery (i.e., redundant-delivery over multiple paths) greatly improve the message propagation speed as well as the scalability with regard to the hop count. However, those studies were made mostly in mobile cases[65][56][66]. Hop-by-hop reliable transfer is made by assuring that the message has been certainly transferred to the nexthop node. In this scheme, it repeatedly retries this forwarding process if the receiver could not get the message in the previous transmission. This scheme certainly propagates messages to the nexthop and achieves the scalability in hop counts. By making branch paths at the intermediate nodes during the delivery, DTN performs message propagation in parallel.

We applied those communication schemes to UTMesh and studied the features of message propagation. We have used potential-based entropy adaptive routing (PEAR)[56] as an implementation of DTNs.

We consider that wireless mesh networking enable rapid and costless deployment of smart meters into green buildings[67]. In such applications, the network must autonomously route messages under the given deployment configurations. This work itself does not focus on the method of node placement[68][69], or high bit rate application[70], or reduction of power consumption[71]. Sensors at power distribution boards, lights and HVAC systems actually do not suffer from power constraints.

The rest of this chapter is organized as follows. Section 5.2 addresses the related works. In section 5.3, we provide our analysis of wireless links on UTMesh. Section 5.4 describes PEAR focusing on the behavior in static cases. Section 5.5 provides our evaluation work. In section 5.6, we provide the discussions on the results. Finally, we conclude this chapter in section 5.7.

5.2 Related Work

Link-level measurement on mesh networks was well studied by Daniel et. al.[72] with Roofnet: wireless mesh networking testbed (38 nodes). They concluded "there is no clear distinction between working and non-working links". As for routing on mesh networks, Tschudin et. al.[73][62] discussed the existence of "Ad Hoc Horizon" – "at 2-3 hops and 10-20 nodes where the benefit from multihop ad hoc networking virtually vanishes" from the experiences on APE testbed (37 nodes)[74]. The researches on our UTMesh (51 nodes) have also verified their conclusions.

The benefits of hop-by-hop transfer are well summarized by Heimlicher et. al. [75] (also discussed in many literatures). Related to this, delay or disruption tolerant networking (DTN)[61][1], originally proposed for deep space communication, has been identified as an applicable scheme for intermittently-connected networks. Message routing on DTNs (or intermittently-connected networks) were studied mostly for mobile cases[65][56][42][41] in the last 5 years, and now it is widely acknowledged that multipath delivery improves message propagation speed and delivery probability. We applied the communication scheme even to static cases in this work.

There has been several studies on multipath or redundant-path packet propagation for mobile ad hoc networks (MANET) for a decade. Stephen et. al.[76] discussed the fault tolerance of multipath routing in MANETs. Tsirigos et. al.[77] provided an analytical work on the benefits obtained from multipath schemes. However, these works seem to assume best-effort forwarding and end-to-end reliability. In our survey, most of the studies are not tested with real implementations.

The system model of PEAR, which we describe in this thesis, was defined by our previous literature[56]. However, we described in the context of mobile cases. In

this chapter, we describe PEAR in static cases and provide our testbed experiments on UTMesh.

5.3 Intermittently-Connected Mesh Networks

The definition of intermittently-connected mesh networks (ICMeN) is as follows. An ICMeN is composed of stable wireless nodes, however, the links among them are disruptive and unreliable; i.e., they sometimes become connected but also frequently become disconnected. According to our testbed experiments, the typical wireless mesh networks (i.e., composed of 802.11b ad hoc mode) falls into this category.

In this section, we first describe our experiment settings, then show the results. We deployed 50 wireless nodes in our university campus and studied the features of the links. The results strongly indicate that the network topology frequently changes even if nodes are static.

5.3.1 Experiment setting

We deployed 50 nodes in Hongo campus at the University of Tokyo as Fig. 5.2 (a). Each node was working with Armadillo-220, a Linux[1] embedded computer with an USB IEEE802.11b/g/n module[2]. The IEEE802.11b/g/n module was working in ad hoc mode (of 11b) at channel 1; all the nodes had the same frequency.

The embedded computer was powered by enough batteries (actually, we do not focus on the power usage – it just equipped enough power for the experiment). We packed all of them into a plastic box. Fig. 5.1 is the overview of the testbed (before deploying into the campus).

In order to analyze the features of wireless links (e.g., availability, contact time, inter contact time), we installed the software that made radio range advertisement in every 10 second. By recording the advertisements received from neighbors at each node, we performed this investigation. The experiment was carried out for 6 hours.

5.3.2 Link availability and network topology

Fig. 5.2 (b) is the summarized network topology. The boldness of links indicates the availability between the nodes, which is specified by,

$$A(a,b) = \frac{1}{2}\left(\frac{R_{a\leftarrow b}}{S_b} + \frac{R_{b\leftarrow a}}{S_a}\right) \tag{5.1}$$

[1]Kernel: linux-2.6.12.3-a9-15

[2]GW-USMicroN, Planex Communications Inc.

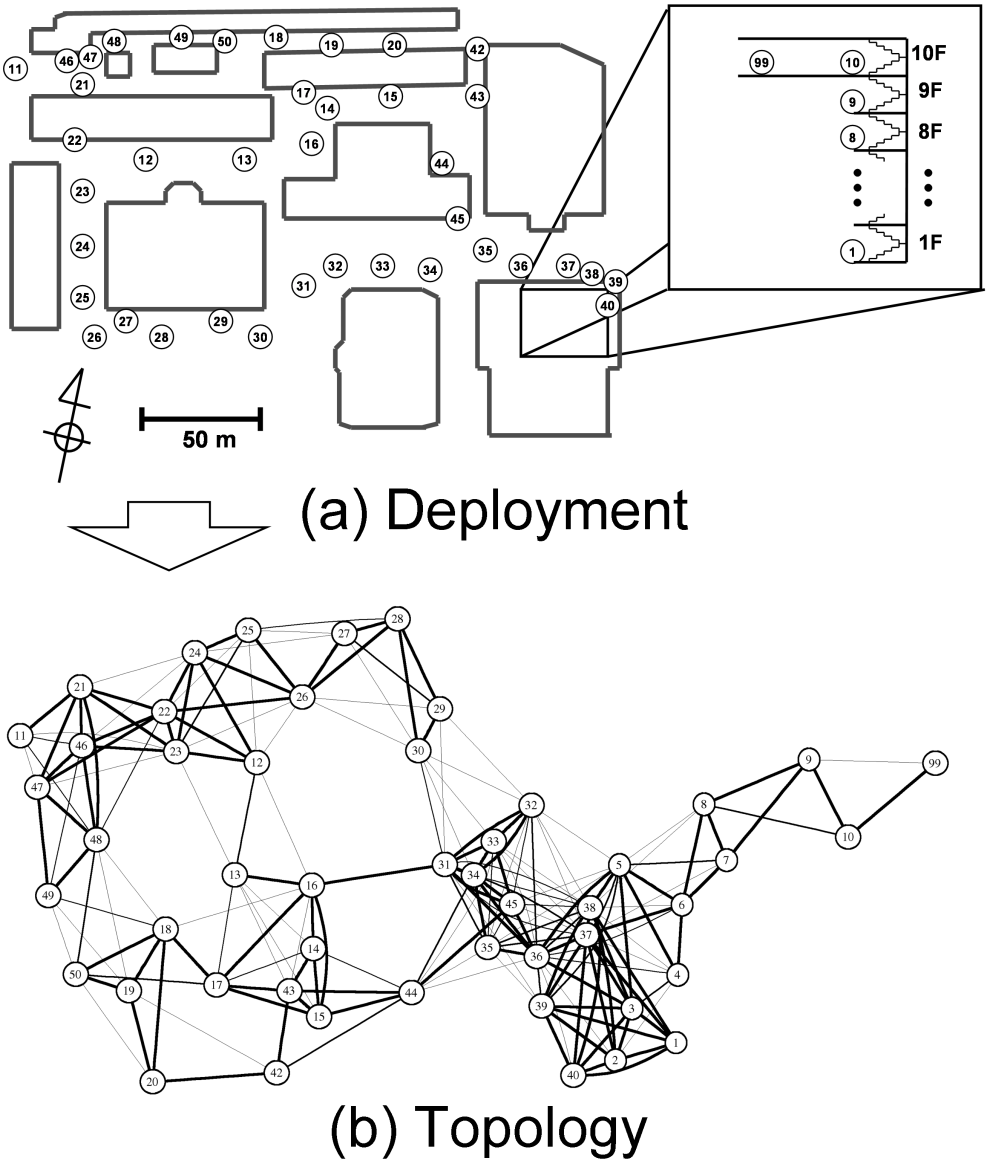

Figure 5.2: (a) The experiment setting to analyze intermittent links between wireless nodes deployed in ad hoc manner. (b) Summarized network topology; bolder link indicates higher link availability.

Here, $A(a, b)$ denotes the availability of links between a and b. $R_{a \leftarrow b}$ is the received advertisement from b at a, S_b is the total advertisement sent by b during the experiment.

From this result, we can see that the availability of links is apparently hetero-

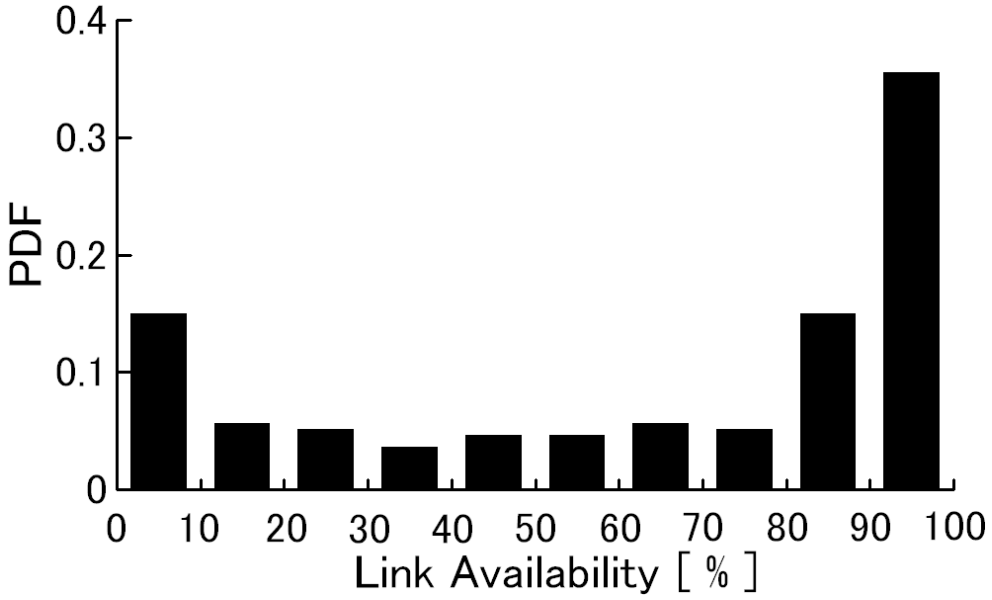

Figure 5.3: The distribution of link availability – only 35% was tightly connected.

geneous. Some links seem tightly connected but others are lightly connected. Fig. 5.3 is the distribution of the availability per link. Only 35% of the links were tightly (more than 90%) connected, and others were disruptive. Actually, 90% availability is not enough if the two nodes need to make session-based communication (e.g., TCP). Furthermore, if the hop count between two nodes increases, the packet loss ratio increases suddenly. This fact indicates that the principle of best-effort and end-to-end reliability cannot be applied to such networks.

5.3.3 Contact time and inter contact time

We also looked into the detail of each link, and analyzed the distributions of contact time and inter contact time. Contact time, in this work, is the duration that a node received advertisements from the other node without losses. If it received five succeeding advertisements but not the sixth advertisement, the contact time is 50 seconds. Inter contact time is the interval between the received advertisements. For example, if it could not receive three succeeding advertisements before receiving the fourth advertisement, the inter contact time is 40 seconds.

Fig. 5.4 and Fig. 5.5 shows the distribution of contact time and inter contact time for the links at $1 \leftrightarrow 2$, $30 \leftrightarrow 31$, and $30 \leftrightarrow 32$. They respectively had 91%, 55% and 3% availability.

Links become disconnected at 23%(1↔2), 76%(30↔31) and 100%(30↔32) in 60 seconds. They reconnected again at 100%(1↔2), 98.9%(30↔31) and 59.6%(30↔32)

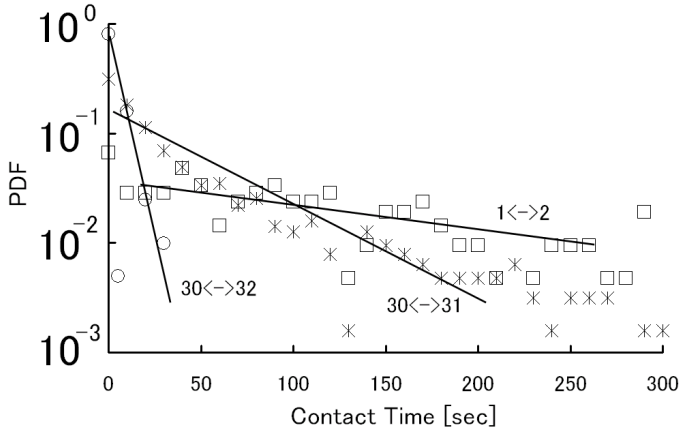

Figure 5.4: Distributions of contact time at $1 \leftrightarrow 2$ (91% availability), $30 \leftrightarrow 31$(55%) and $30 \leftrightarrow 32$(3%).

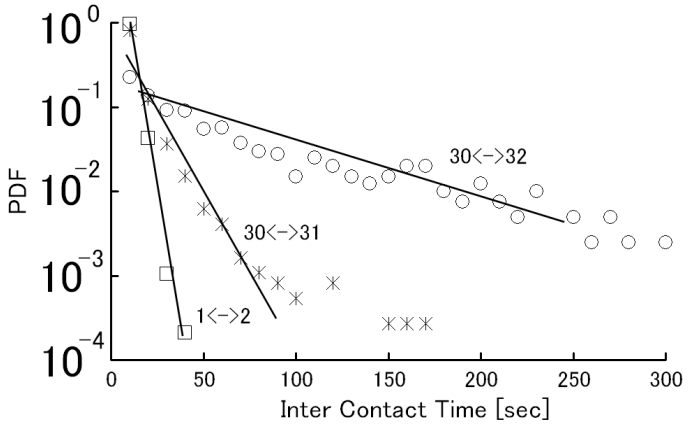

Figure 5.5: Distributions of inter contact time at $1 \leftrightarrow 2$ (91% availability), $30 \leftrightarrow 31$(55%) and $30 \leftrightarrow 32$(3%).

in the next 60 seconds after disconnected.

5.3.4 Summary

From these results, we concluded that (1) the connectivity of links changed very frequently even nodes was static, that (2) more than half of them were such links and that (3) such links often have longer distance.

The shortest path, which gives the smallest hop count to the destination, would not be the best path regarding to delivery latency. In order to get the smallest hop

count, each hop must reach long distance. However the links of longer distance frequently become disconnected. Instead, shorter links are available. Propagating messages on shorter links sometimes work faster. The results of our evaluation (section 5.5) clearly have shown this.

This experiment was made with small advertisement traffic at 10[sec] sampling rate. Burst traffic may cause larger packet losses.

5.4 Potential-Based Entropy Adaptive Routing

In order to implement hop-by-hop reliable communication and parallel message propagation, we use potential-based entropy adaptive routing (PEAR) in this work. PEAR inherits the concept of DTNs such as store-carry-and-forward and multipath message propagation. Actually, we have already presented the definition of PEAR model in chapter 4, but in the context of mobile nodes. Thus, in this chaper, we focus on the description of behavior at static cases.

5.4.1 Forwarding scheme

We define two forwarding schemes for PEAR: best candidate selection (BCS) and multiple candidate selection (MCS). BCS chooses the most possible nodes and MCS chooses the better nodes among the contacted neighbors for the nexthop. More formally, we define them as,

Best Candidate Selection (BCS):

$$F^d_{max}(n,t) = \max_{nbr(n)} \{V^d(n,t) - V^d(k,t)\} \tag{5.2}$$

$$\begin{aligned} nexthop^d_{BCS}(n,t) = \{k | k \in nbr(n) \wedge \\ F^d_{max}(n,t) = V^d(n,t) - V^d(k,t) \\ > \alpha\} \end{aligned} \tag{5.3}$$

Multiple Candidate Selection (MCS):

$$\begin{aligned} nexthop^d_{MCS}(n,t) = \\ \{k | k \in nbr(n) \quad \wedge \quad V^d(n,t) - V^d(k,t) > \beta\} \end{aligned} \tag{5.4}$$

Here, α and β are positive constants that give threashold of forwarding.

PEAR implements hop-by-hop reliability scheme, and takes copy-based approach in transferring messages. This makes parallel delivery and achieves faster message propagation.

After the nexthop candidates are determined, the node asks whether the nexthop already has the sending-message or not. If the nexthop has no knowledge

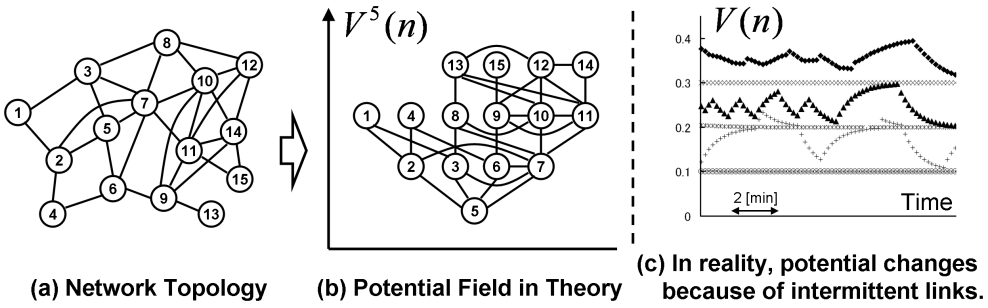

Figure 5.6: Potential field construction (in theory and in reality); potential-field converges into (b) if the given network topology (a) does not change. However, links are intermittent and the topology dynamically changes. Potential-field follows this change as (c). (c) is a real trace on UTMesh.

about the message, the node forwards the message to it. If the nexthop already has the message, the node does not forward. Actually, the node makes copies of the message, instead of just forwarding. Traditional ad hoc network removed the message after the forwarding process. However, PEAR copies it to the nexthop and does not remove at the forwarding source. This forwarding strategy, which is widely acknowledged in DTN research community[44][65][42], certainly improves delivery performance. This scheme allows to make branch paths from the intermediate nodes and to propagate them in parallel. Even if a certain path became wrong, it finds another path and delivers the messages (see section 5.4.3).

The forwarding scheme of PEAR also implements message deletion mechanism for those remained in the network. For more detail, see our previous paper[56].

5.4.2 Potential field in stable scenarios

If the network is stable and connected, a potential-field converges into the same pattern that distance-vector protocols develop; it increases linearly hop-by-hop from the destination. More formally, if the network is stable ($\Leftrightarrow \{nbr(n)|n \in N\}$ are static) and connected, PEAR gets,

$$\forall n \in N, \lim_{t \to +\infty} V^d(n,t) = \frac{\rho}{D} h(d,n) \tag{5.5}$$

Here, $h(d,n)$ is the minimum hop count from node d to n. Thus, it has an aspect of distance-vector routing.

Fig. 5.6 (b) shows the developed potential field for destination node n_5 by PEAR. In this example, the potential value increments as the hop count increases for n_5. Theoretically, it converges as Eqn. 5.5 presents, however in reality, because

links are intermittently-connected as we have noted, the potential values change all the time as Fig.5.6 (c). Here, (c) is the real trace obtained at UTMesh.

5.4.3 Parallel delivery in ICMeN

The same messages propagate in parallel in ICMeN whether it is BCS or MCS. In BCS case, each node chooses only one nexthop at one time. However, as we mentioned, because potential-field changes according to the status of links, the best nexthop candidate also changes in the next time slot. Thus, the node also copies messages to the new nexthop candidate. In this way, it makes multiple paths in message propagation. In MCS case, each node chooses several nexthops at one time, which itself makes multiple path without waiting the change of potential-field.

The major difference of BCS and MCS is the redundancy level in message propagation. MCS apparently creates larger number of delivery paths, indicating that small number of link failure does not cause fatal delay.

Choosing the best single path from the source to the destination is very difficult or impossible in ICMeN, because (1) the best path soon becomes obsolete or unavailable during the delivery of messages and (2) each message has its own best path. PEAR enables to choose the best path for each message by propagating itself in parallel.

5.5 Evaluation

We carried out experiments for two deployment cases (campus case and building case) and evaluated the features such as delivery latency and buffer usages. We compared the performance of BCS and MCS in both deployment configurations. In this section, we describe the experiment settings, the profile of the experimental networks and the results of delivery latency, message delivery patterns and buffer usages.

5.5.1 Experiment setting

Fig. 5.7 shows the deployment configuration in Hongo campus and Eng. Bldg. 2 in the University of Tokyo(UT). We used 51 nodes (1, . . . , 50, 99) in the experiments. First, all the nodes were gathered at Esaki laboratory at the 10th floor in Eng. Bldg. 2. We powered on between 5 and 10 nodes at the same time and shipped to the specified location. We repeated this until all the nodes were deployed. The wireless interfaces were deployed about 10cm high on the ground (outside) or on the floor (inside). We here configured node 1 to send one 100-byte message to node 99 at every 5 second. In this setting, the ICMeN delivers the messages from 1 to 99 by PEAR routing algorithm.

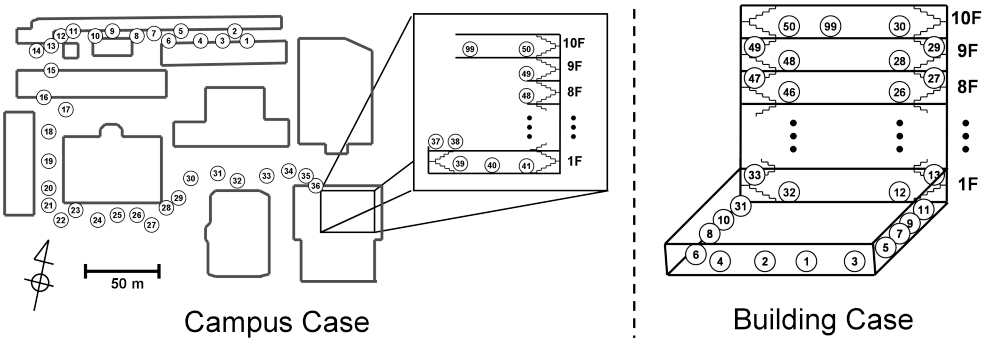

Figure 5.7: Deployment configuration; we deployed 51 nodes in Hongo campus (campus case) and Eng. Bldg. 2 (building case) in the University of Tokyo.

Before the deployment, we installed two programs into these nodes – the one is for BCS operation and the other is for MCS, and the running mode was automatically changed at specified times. In the campus case, nodes were operated by BCS for the first 3.5 hours, then swapped by MCS and operated for the last 3.5 hours. In the building case, the first 6 hours were operated by BCS and the last 6 hours were by MCS. Here, we setup 5 minute break between the two modes.

Finally, we collected the deployed nodes into the laboratory, and retrieved the working logs for analysis.

All the wireless interfaces were operated in ad hoc mode of 802.11b at the same frequency: 2.412GHz (channel 1). The parameters of PEAR were as follows.

- POTENTIAL_TTL: 30[sec]
- MESSAGE_TTL: 3600[sec]
- DISSEMINATION_TTL: 300[sec]
- D: 0.2
- ρ: 0.02
- α, β: 0

We implemented PEAR to achieve small tasks at 10-second step. For example, it exchanged potential values in every 10 second; it generated the next potential-field every 10 second. It also tried re-transmissions for stored messages at every 10 second. Thus, when it failed forwarding messages, the messages must wait there for the next re-transmission, which caused 10 second delay.

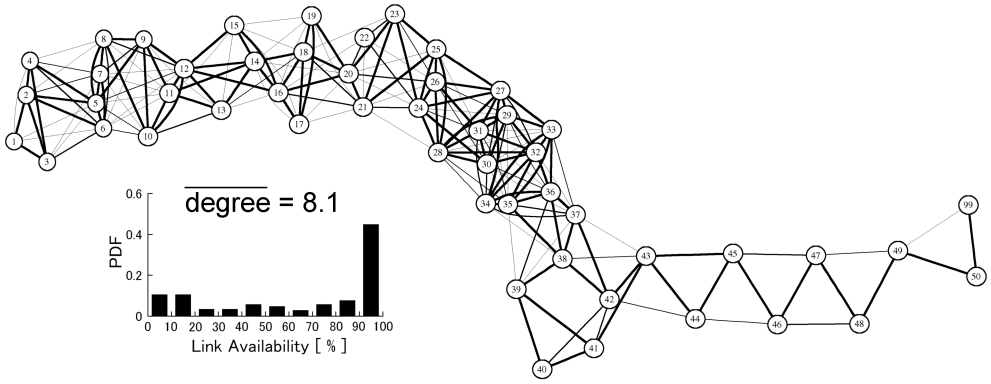

Figure 5.8: Topology, link availability and average degree of the deployed network (campus case). Bolder link has larger availability.

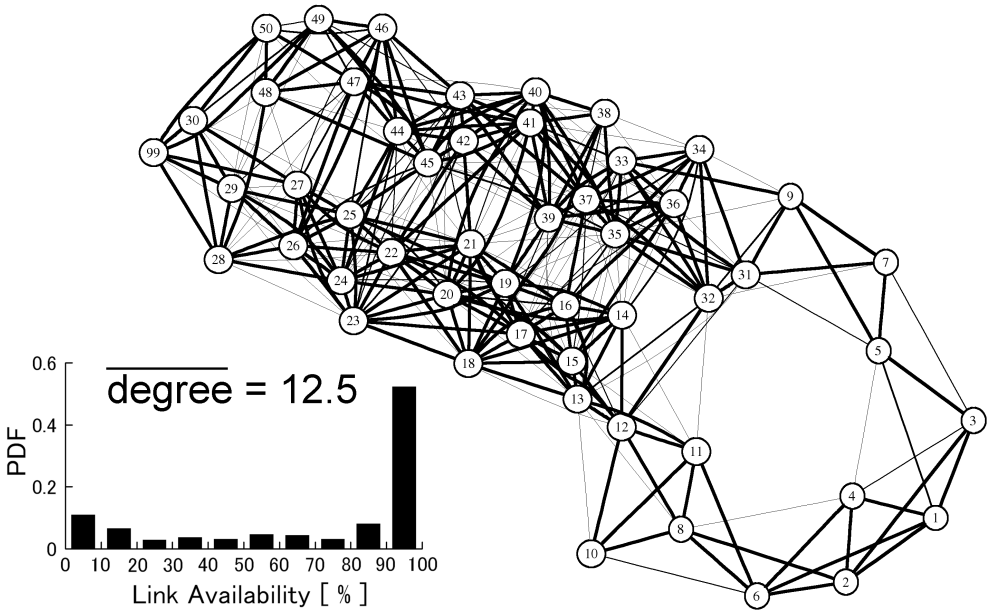

Figure 5.9: Topology, link availability and average degree of the deployed network (building case). Bolder link has larger availability.

5.5.2 Features of the deployed networks

Fig. 5.9 and Fig. 5.8 are the summarized topology and the distribution of link availability of the deployed networks. The boldness of links are provided by Eqn. 5.1. As it becomes bolder, the availability increases. The distribution of the availability was almost the same between the two configurations. About one half of the

links were tightly (more than 90%) connected. The average degree (the number of links at an average node) was 8.1 (campus case) and 12.5 (building case).

5.5.3 Experiment results

We here summarize the evaluation results. The results indicate that hop-by-hop reliability scheme achieves scalable message propagation (e.g., 23 hops), that message propagation speed increases as the redundancy-level increases.

- **Campus Case:** PEAR achieved 26% (BCS) and 29% (MCS) delivery in the given message life time: i.e., 3600[sec]. The average hop count from 1 to 99 was 21.2 (BCS) and 22.3 (MCS). Messages were copied 30.4 times (BCS) and 36.9 times (MCS) in average. 96 entries (BCS) and 194 entries (MCS) were used at average nodes for replica management and message deletion control. 36.4 buffers (BCS) and 51.5 buffers (MCS) were occupied by messages at average nodes.
- **Building Case:** PEAR achieved 100% delivery probability for both BCS and MCS. The average delivery latency was 238[sec] (BCS) and 46.6[sec] (MCS). The average hop count from 1 to 99 was 7.08 (BCS) and 7.72 (MCS). Messages were copied 14.5 times (BCS) and 27.7 times (MCS) in average. 197 entries (BCS) and 408 entries (MCS) were used at average nodes. 17.3 buffers (BCS) and 15.4 buffers (MCS) were occupied by messages at average nodes.

Delivery probability is the ratio of the arrived messages during their life time to the sent messages. Average delivery latency is the average time of message travel from the source to the destination. Because the delivery latency is given for each message, we cannot calculate the average if any messages disappear before reaching the destination. Thus, we could present the average delivery latency only for the building case, which has achieved 100% delivery.

Average hop count is the average of message hop counts from the source to the destination. Average copy count is the average of copies for each message made in the network during delivery.

Entry is used to maintain the state of the message. Thus, when a new message arrives at node n, the message consumes one entry at n. The entry remains there until the message expires. Average entry usage shows how many entries are used in the average node, and average buffer usage shows how many message bodies are stored in the average node. Buffer usage is usually smaller than entry usage because PEAR deletion algorithm removes the message body but entry remains there until TTL expires to delete messages.

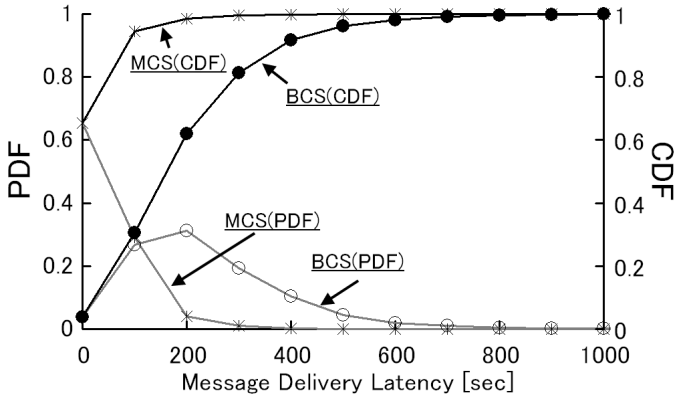

Figure 5.10: Delivery Latency in Building Case

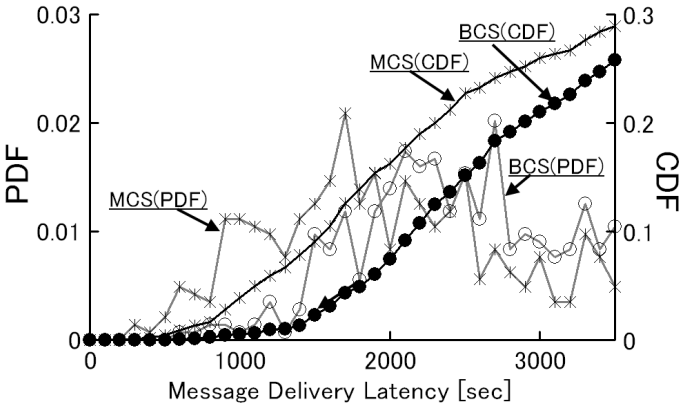

Figure 5.11: Delivery Latency in Campus Case

5.5.3.1 Delivery latency

Fig. 5.10 and Fig. 5.11 show the distribution of message delivery latency.

In the campus case, the fastest message was delivered in 600[sec] (BCS) and in 300[sec] (MCS). 10% of the messages were delivered in 2200[sec] (BCS) and in 1600[sec] (MCS). 20% of the messages in 3000[sec] (BCS) and 2400[sec](MCS). They delivered 26% (BCS) and 29% (MCS) of the messages during the given TTL.

In the building case, BCS delivered 4% of the messages in between 0[sec] – 100[sec], whereas MCS achieved 65% for the same latency. 99% of the messages were delivered in 700[sec] (BCS) and 300[sec] (MCS).

Apparently, MCS achieved faster message propagation than BCS did. Especially, in building case, the average performance of MCS was 5.1 times faster (5.1=238[sec]/46.6[sec]).

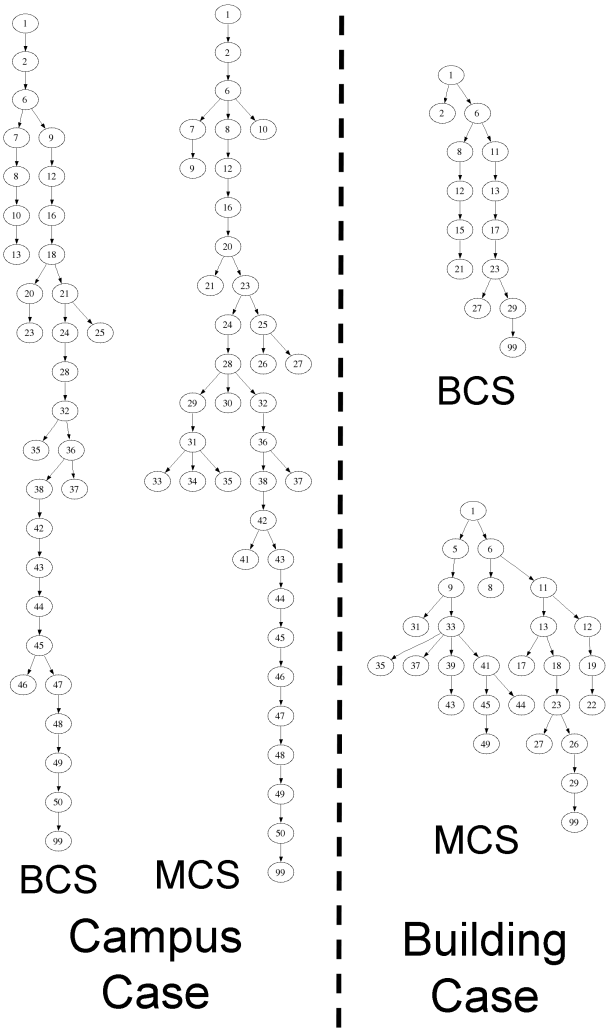

Figure 5.12: Message delivery pattern examples from source 1 to destination 99. PEAR achieved scalable message propagation in hop count. MCS made larger number of replicas, found faster delivery path, and took bigger hop count than BCS.

5.5.3.2 Message delivery pattern (hop and copy count)

Fig. 5.12 shows the typical message delivery patterns in each scenario (we picked them up from thousands of delivery pattern samples). In campus case, 21.2 hops and 30 copies were average for BCS, and 22.3 hops and 36 copies were for MCS. In building case, 7.08 hops and 14.5 copies were for BCS, and 7.72 hops and 27.7 copies were for MCS. Fig. 5.13 and Fig. 5.14 shows the distributions of hop count

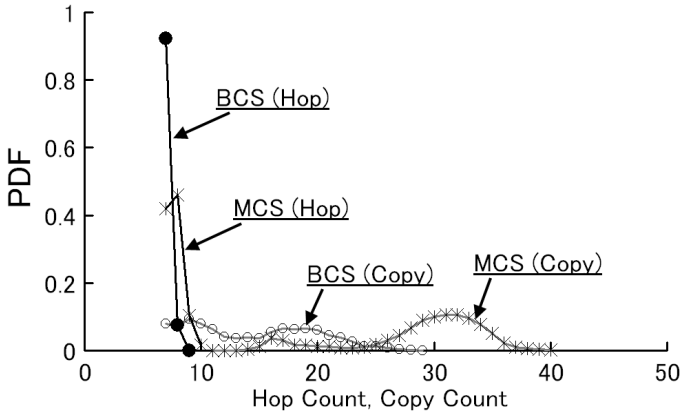

Figure 5.13: Hop and Copy Count in Building Case

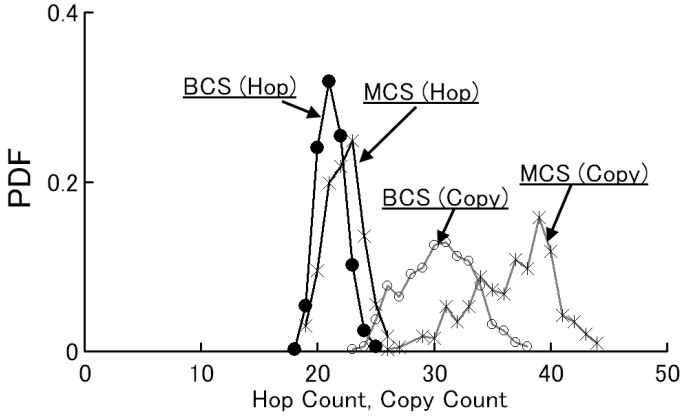

Figure 5.14: Hop and Copy Count in Campus Case

and copy count. Because messages took different delivery paths, each message gave different hop count and copy count.

Hop-by-hop reliability certainly improved the scalability in hop count. (According to [62], MANET could make only several hops for packet propagation).

MCS made larger number of copies especially in building case: i.e., about 1.9 times. Interestingly, MCS also took slightly larger hop counts than BCS did. This indicates that faster delivery path is not always the shortest path. Longer path (i.e., larger hop count) sometimes increases message propagation speed.

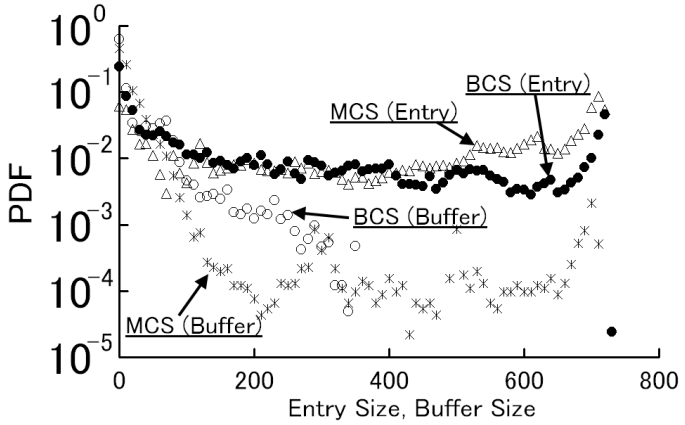

Figure 5.15: Entry and Buffer Usage in Building Case

5.5.3.3 Entry and buffer usage

Fig. 5.15 and Fig. 5.16 show the distributions of entry and buffer usages. The maximum limit of the usages is 720[count] – given by the message generation traffic (0.2[count/sec]) and its life time (3600[sec]).

In both campus and building cases, most of the nodes had small entry and buffer usages at most of the time. In building case, buffer sizes (BCS and MCS) were much smaller than entry sizes. This indicates most of the message body was removed from the network after the delivery by PEAR deletion mechanism. However, in campus case, although buffer sizes were smaller than entry sizes in total, they were almost the same at large spectrum. This indicates that farer nodes from the destination removed fewer message bodies because they had to remain in the network until they reached the destination.

5.6 Discussion

The reason why message delivery took large time (e.g., 100[sec], 1000[sec]) originates in PEAR implementation. PEAR was originally developed for delay tolerant networking, and thus it tried re-transmission in every 10 second. If the message could not be transferred in one transmission (this frequently happens probably because of [72]), it had to wait for the next trial. Because the hop count is large, the total delivery took plenty of time. We would be able to improve the re-transmission scheme for faster propagation, which is our future work.

The experiments have clearly shown that MCS, which gives larger redundancy, delivered messages to the destination especially when the network was densely configured (building case). Here, MCS took larger hop count than BCS did. This indicates that shorter distant links propagate messages faster though it must take

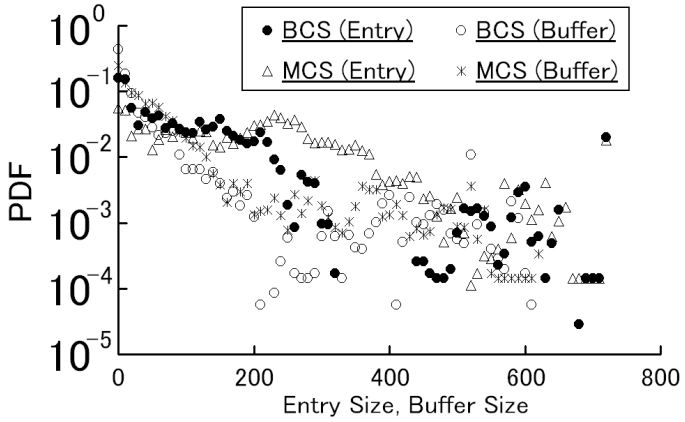

Figure 5.16: Entry and Buffer Usage in Campus Case

larger number of hops.

The experiments were made on IEEE802.11b links (in ad hoc mode). Though the wireless interface itself supports IEEE802.11g and 11n, they do not work in ad hoc mode. The main contribution of this research, we consider, is the implementation report of hop-by-hop reliability and parallel propagation schemes for such unstable networks. Study on other wireless links (11g, 11n and 11s[78][79]) is open research items.

5.7 Conclusion

In this thesis, we have raised intermittently-connected mesh networks(ICMeN) on the basis that links of typical wireless mesh networks (i.e., composed of 802.11b ad hoc mode) are disruptive and unreliable. The connectivity of links, and the whole network topology, frequently changes, and this makes scalable message propagation in traditional communication schemes difficult.

We proposed to apply hop-by-hop reliable and parallel message propagation, which DTN researches have explored, to such ICMeN. We implemented them to UTMesh – 50 node scale wireless mesh testbed, and studied the delivery patterns.

On the evaluation result with UTMesh, we have confirmed (1) that hop-by-hop reliability achieves scalable message propagation (e.g., 23 hops), and (2) that message propagation speed increases as the redundancy-level increases. We have also observed that the smallest hop count path does not always achieve the fastest message delivery. This was probably because longer distant links were unstable and message paths over short distant links provided faster propagation.

Chapter 6

Delay Tolerant IP Networking

6.1 Introduction

In the challenged network environment, any communication nodes are always virtually networked over intermittent physical connection. This idea opens up the new communication paradigm that extends the communication infrastructure everywhere from tiny embedded devices to even out of the Earth.

The impact of the emergence of delay tolerant networks (DTN)[1, 61, 80] is remarkable. Sensor networking community is now applying the concept to its framework[30, 81]. Mobile ad-hoc networks (MANET) reseach community is also tring to use it[82, 83]. Researches of vehicular ad-hoc networks (VANET) cannot be discussed without DTN[33, 34].

However, the widely-discussed DTN, which was originally designed for interplanetary communication, is still one of the approaches for the challenged network environment. IP-based sensor networking or MANETs with 100-node scale can take another approach with proposing alternative architecture for delay tolerant networking. The architecture of widely-discussed DTN is not well-adaptive to the Internet, and the management and operational issues have not likely been deeply discussed, which is mandatorily required in the deployment phase.

This work proposes delay tolerant IP networking (DTIPN) architecture, which we have developed as a different style of delay tolerant networking, assuming about 100 nodes in a challenged network segment. The architecture does not rely on *Bundling*, which has characterized the well-known DTN. It uses IP address for node locators, still providing delay tolerant support for application message delivery. DTIPN is much more adaptive to the existing IP network than the widely-discussed DTN, regarding to practical management and network operation.

Generally, all the communication protocols targeted at the challenged network environment must avoid synchronous communication style. Making state synchronization over physically disrupted connectivity is basically impossible or it would get only a weak synchronization. The DTN has avoided synchronous communica-

tion between communication ends and enabled asynchronous message delivery, by dividing an end-to-end TCP session into multiple sub-sessions.

DTIPN, our approach, takes IP packets as asynchronous data delivery units. IP networks originally provide asynchronous packets delivery, and it is basically allowed for large delay: even one hour or one day delay. In DTIPN, the data link layer for the challenged environment enables IP packet delivery over physical intermittent connectivity, and the transport (or, the application) protocol layer supports the long-delay delivery of application protocol data units (APDUs).

DTIPN is much more adaptive to the Internet architecture. Anyone can easily deploy independently around their Internet edges with the conventional management operation. The widely-discussed DTN requires new management and operational schemes, especially when we come to deploy it as a globally managed network, because it basically establishes a new network as an overlay.

One of the contributions of this work is to demonstrate how the architecture can be implemented and how it is practically useful. For this purpose, we have implemented DTIPN and carried out some experiments, and we present the performance of our prototype system in this thesis. At the data link layer, we adopted potential-based entropy adaptive routing (PEAR), which was proposed in our previous work[56], as an IP packet delivery framework for wide-range of mobility models. At the application protocol layer, we implemented a forward error correction (FEC) scheme, which would be practically useful for moderate packet loss in the Internet space. Thus, the evaluation work is totally based on the real implementation. We have carried out four types of experiments with 10 nodes using the campus of the University of Tokyo.

This chapter is organized as follows. Section 6.2 provides the related work. We propose DTIPN architecture in section 6.3. Section 6.4 and 6.5 provides our evaluation work with presenting the prototype implementation. Section 6.6 gives the discussion, and finally this paper provides the summary in section 6.7.

6.2 Related Work

The widely-discussed DTN was proposed by Burleigh et. al. [61] and Kevin Fall[1], and the internet engineering task force(IETF) has published it as RFC4838[80]. The architecture is characterized by the Bundle layer, which deploys DTN framework as an overlay network and enables asynchronous message delivery. The background of the architecture came from the study of TCP performance reduction and failure over large delay and high packet loss networks[84]. In order to avoid TCP-based synchronous communication between communication ends, they have proposed to take hop-by-hop delivery at the Bundle layer.

DTIPN takes different architecture to enable asynchronous message delivery over the challenged environment. It uses IP packets as asynchronous data delivery units, and enables application message delivery by the transport or the application

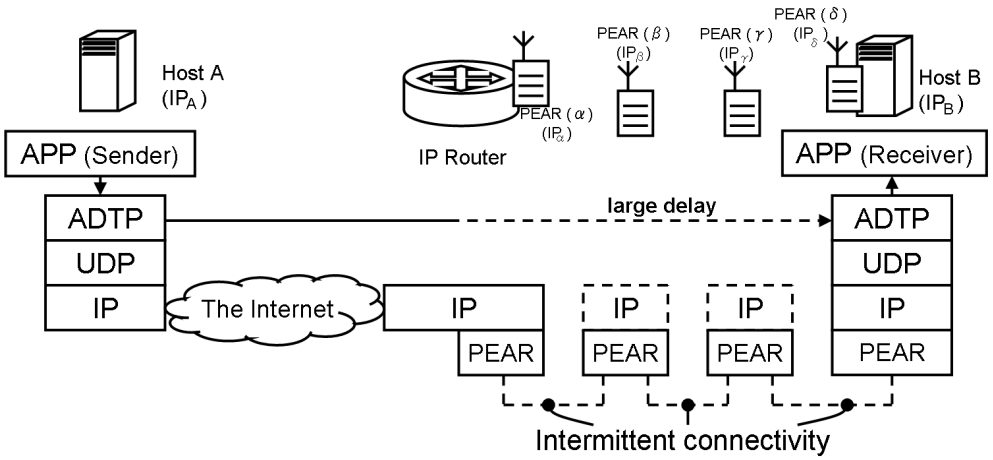

Figure 6.1: Delay tolerant IP networking architecture

layers. This approach directly fits into the Internet protocol architecture, indicating that DTIPN has great adaptivity to the current Internet.

This work inherits our previous works. IP over DTN[85] has proposed the basic concept for DTIPN. Potential-based entropy adaptive routing (PEAR)[56] is an autonomous message routing algorithm over the intermittently connected networks. One of the goals of this work is to demonstrate the feasibility of deployment and the usability for practical applications.

IP multicast practically takes non-reliable communication styles because of its difficulty in making state synchronization (i.e., data acknowledgement and retransmissions)[86], just as delay tolerant networking. In this regard, DTIPN can also take non-reliable communication styles. However, it would get reliability to a certain extent by forward error correction schemes just as Parity-based loss recovery[87] and Uni-DTN[88] has presented. We also take this approach at the transport (or, the application protocol) layer.

6.3 Delay Tolerant IP Networking

This section describes the architecture for delay tolerant IP networking (DTIPN). It takes IP address as a communication endpoint identifier, uses IP packets as asynchronous data delivery units.

6.3.1 Requirements

We, here, summarize the basic requirements for DTIPN.

(1) A communication endpoint must be identified by an IP address. Even when the communication end is physically isolated, we consider that it is virtually connected to the network. Taking IP address for endpoint locator reduces the operational cost compared to the additionally required costs in the well-known DTN.

(2) The network must use an IP packet as an asynchronous data delivery unit. The delivery of an IP packet is one of the asynchronous communications. An IP network provides *send* and *recv* method, and it carries IP packets in the best effort manner. Basically, it is allowed to have large delay for packets delivery. Whether it delays one minute, one hour, or even one day, it does not matter if the delay is expected at the application level.

(3) The data link layer for the challenged network environment must assume possible link disruption and must save IP packets against possible losses as much as possible. We must design a framework that provides delay and disruption tolerant support for IP packet propagation over isolated network boundaries to deliver them even with large delay.

(4) The application layer protocol must not have shared states across a network. Interactive communication in a short time is no longer available in our target environment. Thus, all the application programs must assume asynchronous message delivery. The protocols used by those applications must not rely on synchronous communication style, too.

6.3.2 Architecture

We present the architecture of DTIPN in Fig. 6.1. The architecture itself fits into the Internet protocol suite. The major features of this architecture are potential-based entropy adaptive routing (PEAR) at the data link layer and asynchronous data transfer protocol (ADTP) at the application protocol layer.

In fact, PEAR is one of the implementations of the link layer. Another message delivery scheme can be also applied here (e.g., [44, 42, 40, 39, 65]) as long as it provides delay and disruption tolerant support for IP packet delivery.

PEAR, in this architecture, autonomously enables the delivery of IP packets over intermittently connected networks in ad-hoc manner. Even if it takes a long time because of physical network disruption, it manages to deliver packets by its store-and-forward scheme. It adapts to wide-range of mobility models. Section 3.3 describes the internal design of PEAR.

ADTP enables asynchronous delivery of APDUs between remote application instances, making use of UDP for the under layer transport protocol. It provides *send* and *recv* methods as the application programming interface(API). ADTP must be tolerant for the delay of IP packet delivery – the receiver side should wait until it gets the whole APDU. ADTP does not have to provide reliable communication. Thus, some data loss should be assumed at the application programs. However,

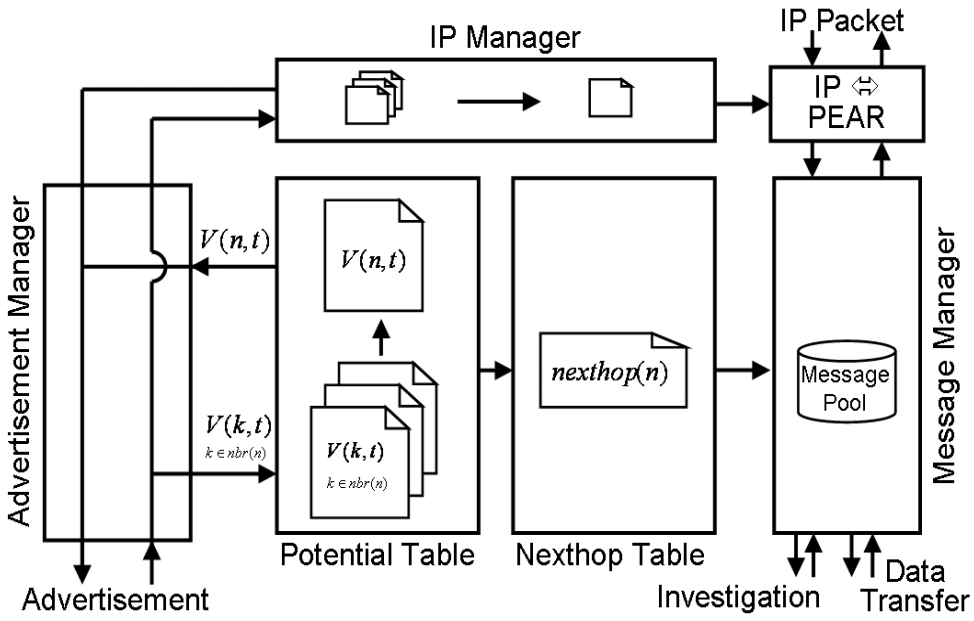

Figure 6.2: PEAR software design for DTIPN

some efforts can be made in ADTP implementation to get reliability to a certain extent. Section 6.3.4 discusses in more detail.

The major difference with the widely-discussed DTN is that all the hosts have IP address for communication endpoint identifier. The widely-discussed DTN has its own identification schemes, and thus it should have its own network management and operational schemes. DTIPN directly fits into the existing Internet, which would help global delay tolerant networking with smaller operational costs.

6.3.3 PEAR for link layer implementation

Potential-based entropy adaptive routing (PEAR) fits into the link layer in the DTIPN architecture. This section provides the overview and the software design of PEAR. For theoretical details (e.g., potentials and replica management scheme), please refer to chaper 4.

6.3.3.1 Overview

PEAR, which was originally designed as a DTN framework, is practically useful for various scenarios. It performs effective routing over wide-range of mobility patterns in ad-hoc manner. A node learns what nodes are nearby and what nodes exist over intermittent connectivity. It performs routing and propagation of messages, adaptively changing the form of delivery pattern depending on the contact or mobility model.

6.3.3.2 Software design

We provide a software design of PEAR in Fig. 6.2. It has six functional blocks. We, here, describe the details.

As for transport, PEAR in DTIPN has only to deliver typical-sized IP packets. This enables system implementation quite simple (see, section 6.5.2).

- **IP Manager:** IP Manager implements an epidemic-based IP-to-PEAR name map construction algorithm. This enables resolution of IP address to PEAR node name.
- **IP Encapsulator:** IP Encapsulator encapsulates an outgoing IP packet by a PEAR protocol frame, and decapsulates an incoming PEAR protocol frame.
- **Advertisement Manager(AM):** AM periodically advertises potentials to its neighbors by radio-range multicast. AM submits the received potentials to potential table (PT), and publishes the current potentials obtained from PT to the neighbors. AM also manages the dissemination of the map of IP address and PEAR node name.
- **Potential Table(PT):** PT manages all the potentials of neighbors and computes the potentials of the next time step by the potential-field construction algorithm[56].
- **Nexthop Table(NT):** NT generates and maintains nexthop table for each destination by the potentials managed at the PT.
- **Message Manager(MM):** MM implements the replica management scheme that PEAR defines[56]. It provides an interface of sending and receiving packets for IP Encapsulator.

6.3.4 Asynchronous data transfer protocol

The role of ADTP is to provide the API for application instances to send and receive messages by application protocol data unit (APDU). It decomposes an APDU into UDP datagrams when sending it, and composes the original APDU from the received UDP datagrams. It must be tolerant to (1) moderate loss, (2) large latency, (3) out-of-order arrival and (4) duplicate arrival of IP packets. It should also be aware of traffic congestion; i.e., huge number of packets must not be sent in a burst manner.

PEAR carries IP packets at the link level, saving them from fatal losses as much as possible. However, even when it provides 100% packet delivery, the Internet sometimes drops them from the network. FEC, as widely discussed, must be the most practical method for the moderate data loss or error in asynchronous communication, and we also take this method for ADTP. However, we must also

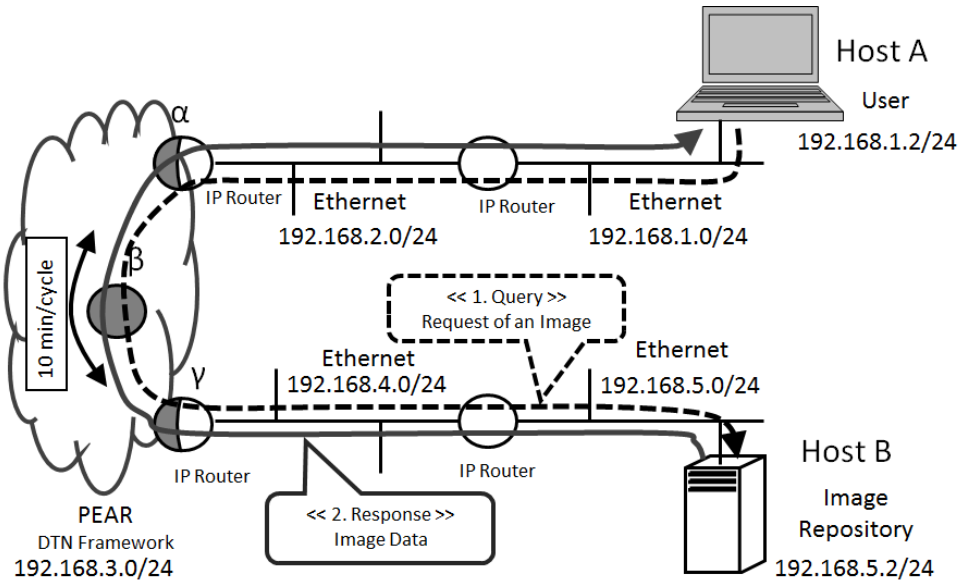

Figure 6.3: The detailed configuration of the preliminary experiment

be aware that the packet loss might occur randomly or sometimes in burst. We can read this kind of discussion in the past literatures: e.g., parity-based reliable IP multicast transmission[87], and Uni-DTN[88].

We basically take reed-solomon algorithm as a FEC scheme. In order to implement it, assuming some burst losses and congestion, ADTP takes interleave and slow transmission.

6.4 Preliminary experiment

We carried out a preliminary experiment in order to verify and demonstrate the behavior of DTIPN. Fig. 6.3 shows the detailed configuration of the experiment.

We prepared five network segments; four were deployed on Ethernet and one was deployed on PEAR. Each network was connected by IP routers as Figure 6.3 shows. We used three DTN nodes named α, β and γ. These three nodes have 802.11g radio interface with ad-hoc mode through which they exchange messages in DTN manner. Node α and γ were also networked by Ethernet and worked as IP routers. They were deployed in separate so that they cannot directly communicate with each other. In the experiment, they stored and forwarded IP packets, between Ethernet and the DTN framework. We physically carried node β, an embedded node with battery powered, between the two stations around α and γ in 10 minutes per cycle.

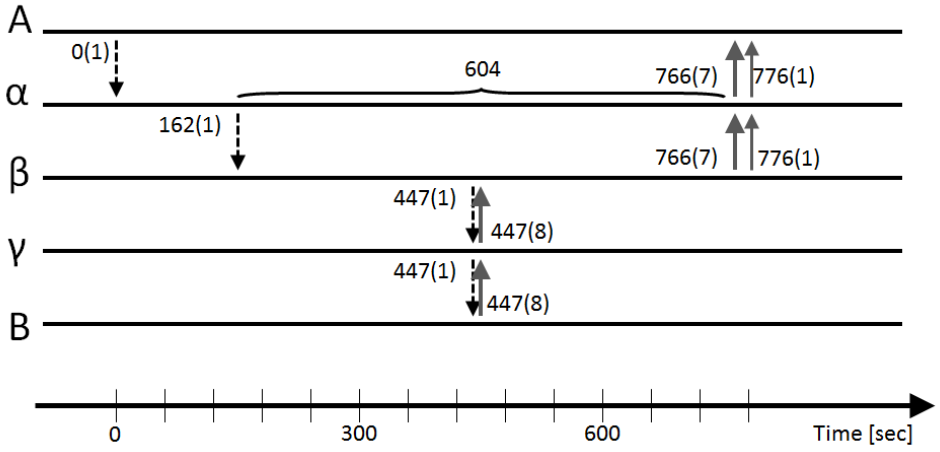

Figure 6.4: IP packet delivery pattern

We setup two hosts A and B in the network as an Internet application. Host B provides an image repository service, and host A tries to retrieve an image from host B. The communication protocol is implemented over UDP, and the application is designed assuming large-delay message delivery. In the experiment, host A sent an image request query to host B, and host B returned the image between five and nine UDP segments depending on the image size. Though the overall communication style is synchronous, the delivery of each APDU (i.e., query and image data) was done in asynchronous manner.

Figure 6.4 illustrates the IP packet delivery pattern observed in the experiment for one request-to-response transaction.

The dashed arrows show request delivery pattern, and the gray arrows show image data delivery pattern. The number along with the arrows shows the time in second when the transmission event had happened. The number inside the bracket shows the number of packets transmitted at the same time. Host A submitted a query to host B at time 0[sec], which was stored at α. At time 162[sec], β has contacted to α and got the query from it. β has arrived at γ and handed the query to it, then the network droved the packet to host B at time 447[sec]. The requested image was sent by eight UDP segments at 447[sec] and stored at β via γ. β has re-contacted to α and send seven packets at 766[sec] and the last one packet at 776[sec]. Here, the last one packet was also transmitted at 766[sec] but failed to receive at α thus re-transmitted 10 seconds later. The total time to retrieve an image over the network was 776[sec].

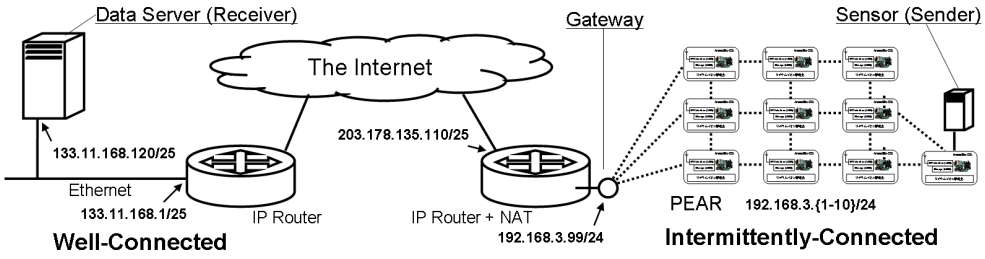

Figure 6.5: Network configuration for the experiment

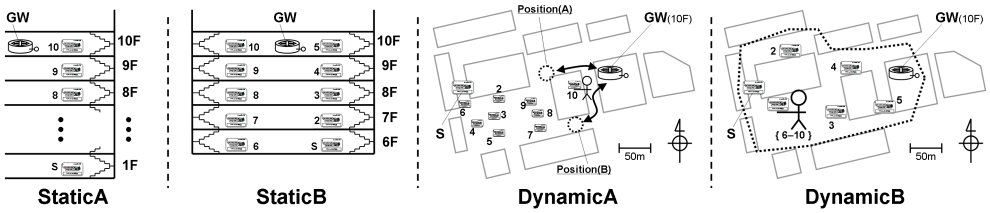

Figure 6.6: Experiment scenarios

6.5 Evaluation

One of the contributions of this work is to confirm the feasibility of deployment and to study the performance of the system with real prototype system in order to demonstrate the applicability to practical scenarios.

The first half of this section describes the settings of the experiment and the prototype implementation which we have developed. The latter part provides the analyses of performance and environmental features.

6.5.1 Experiment settings

We have carried out experiments for both static-cases and dynamic-cases. We did on static cases, because even if nodes are statically setup, wireless links frequently disrupts in the real environment, causing intermittent connectivity in communication. This work presents four types of experiment scenarios, which we call (1) StaticA, (2) StaticB, (3) DynamicA and (4) DynamicB.

Figure 6.5 is the common network configuration applied to all the experiments. The data server in the well-connected network (133.11.168.0/25) receives messages from the sensor over the intermittently-connected network. The sensor periodically sends its messages to the server (133.11.168.120) by ADTP. PEAR network manages to deliver the IP packets to the gateway.

Figure 6.6 shows the physical deployment for each experiment. S is the message

sender (i.e., sensor) and *GW* is the gateway for the upper Internet link.

StaticA: Nodes were statically deployed, one node for one floor in our building (i.e., Eng. Bldg.2, in the University of Tokyo).

StaticB: Nodes were statically deployed, two nodes for one floor in our building.

DynamicA: Nine nodes were statically deployed in the campus, and one node (No.10) has moved between Position(A) and GW, and Position(B) and GW alternately in 20 minutes per cycle.

DynamicB: Five nodes were statically deployed in the campus, and other five nodes have moved inside the dashed-area freely. They sometimes returned to the gateway (about three or four times) during the experiment.

For StaticA and StaticB, we conducted the experiment for 12 hours, and for DynamicA and DynamicB, the experiment was made for three hours and one hour respectively.

During the experiment, S has sent 5k, 15k and 45k-byte dummy messages every 90 second each to the data server. According to the ADTP's FEC configuration, this generates 120 packets in 90 second: i.e., 80 [packet/min].

The parameter settings of PEAR were as follows: $D = 0.2$, $\rho = 0.02$, $\alpha = 0.04$, message TTL = 2400[sec]. The TTL of advertisement was 30 [sec], the time for dissemination was 1800 [sec], and the maximum buffer entry size was 4096. With this setting, maximum buffer occupancy would be around 3200 [entry]. Thus, it never exceeds the available buffer entry.

6.5.2 Prototype implementation

We programmed in C and deployed into Armadillo-220[89]. Armadillo-220 is an embedded computer with 8Mbyte program memory and 32MByte working memory. It works with ARM9 200MHz CPU and Linux operating system. We here added an USB-stick IEEE802.11g module (GW-US54Mini2, Planex Communications Inc.) for ad-hoc communication with neighbors. We used linux-2.6.12.3-a9-15 for its kernel image. As for IPv6 support, please refer to section 6.6.

The footprint of PEAR is not large. The source code has only 3225 lines, and it works around 34k byte in object code size. We developed ADTP as a library for applications. The source code has 640 lines, including reed-solomon algorithm. The library has 32k byte in object code size.

We packed all of them into a handy box. We assembled 11 boxes; one for the gateway and others for static or mobile nodes (including the sensor).

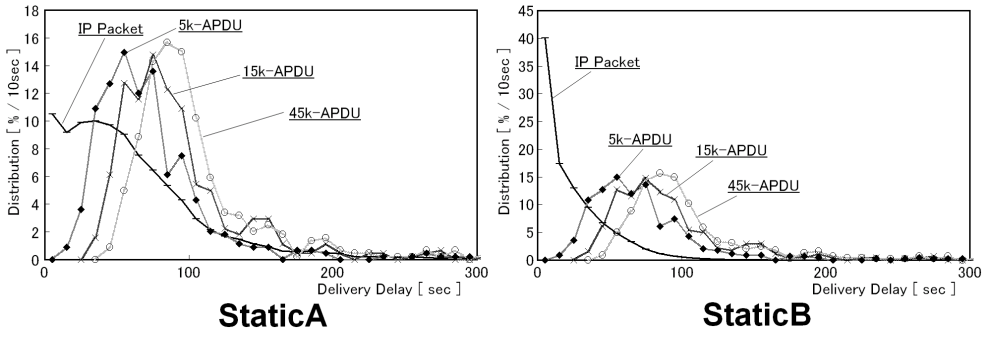

Figure 6.7: IP Packet and application message delivery latency (Static Cases)

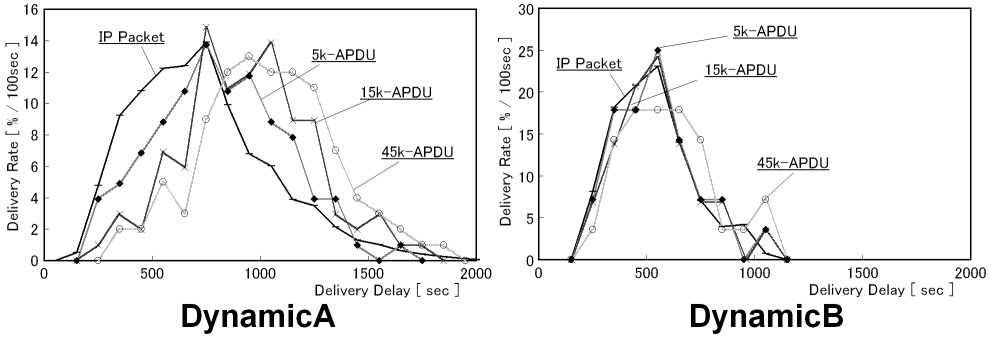

Figure 6.8: IP Packet and application message delivery latency (Dynamic Cases)

6.5.3 Delivery rate and latency of IP packets and APDUs

Figure 6.7 shows the distribution of IP packet and APDU delivery latency for each experiment. The distribution was certainly dependent on the physical settings.

As we have expected, ADTP delivery has taken more time than IP packet delivery, though in DynamicB case, distribution of IP packet and APDU was almost the same. However, APDU was recovered by ADTP before receiving all the packets associated to the original message.

The packet delivery rate from sensor to GW was 100% for all the scenarios. The packet drop rate from GW to the data server was 0.50%; there were 10 hops in IP network from the GW to the server. ADTP has achieved 100% delivery.daniel05borealis

6.6 Discussion

With our prototype-based experiment, we experienced that DTIPN would be practically useful for some opportunistic network applications. In our experiment settings, it has achieved 100% delivery of messages with acceptable overheads. This indicates that DTIPN is quite ready to be deployed in the real environment at the Internet edges.

Our prototype software also supports IPv6 already, though the experiment was carried out on IPv4 networks because the kernel version of the embedded computer was old, and the software could not use IPv6 properly. We confirmed that it worked on Linux kernel 2.6.28-15-generic (ubuntu 9.04), which means that the software itself is also ready for IPv6.

The architecture, especially the link layer, can be applied to any network configurations even to the core networks. However, we consider that it should be applied to edge networks in practice, and that it would be the major use case in the real situations.

DNS-based host identification and IP address resolution are not in the scope of this work. IP address should be resolved beforehand or other additional methods should be explored to allow DNS-based networking.

6.7 Conclusion

We have proposed DTIPN, an alternative architecture for delay tolerant networking. DTIPN takes IP packets as asynchronous delivery units, identifying the location of hosts by IP addresses. In DTIPN, the data link layer provides disruption tolerant support in delivering IP packets over the challenged network environment, and the application protocol layer enables message transfer by APDU.

We have implemented the architecture with PEAR for the link layer and ADTP for the application protocol layer. We have carried out several experiments in the campus of the University of Tokyo, and analyzed the result regarding to delivery rate and latency, transmissions, and buffer occupancy. The result indicates that it is practically useful in the real environment.

The evaluation result was made on the real implementation of PEAR and ADTP. We confirmed, with these implementations, that DTIPN could adapt to the Internet very easily with practically useful performance.

Part IV

Conclusion

We made two major contributions in this thesis. The one is related to component-and-flow programming for sensor actuator networks, and the other is about data transportation over intermittently-connected networks.

The first part was composed of two chapters, and presented facility information access protocol (FIAP) and central controller-based device management (CCDM). The second part was composed of three chapters; the first chapter presented potential-based entropy adaptive routing(PEAR), the second chapter presented intermittently-connected mesh networks (ICMeN), and the last chapter presented delay tolerant IP network (DTIPN).

In the chapter of FIAP, we identified (1) design pitfalls in data-centric sensor actuator networks, (2) challenges in the design of common protocol stack for implementing gateways, storages and UI-terminals, and (3) policies and algorithms for FIAP. We have developed FIAP protocol stack, implemented FIAP components (gateways, storages and UI-terminals), and integrated into a data-centric building automation system in Engineering Bldg.2 in the University of Tokyo. This experiment has shown that FIAP allows incremental integration for wide-varieties of applications (i.e., electricity, HVAC, motion detection/lights, room environment, gas and water supply, weather) only with script-based configuration, which is more lightweight for system integrators.

In the chapter of CCDM, we demonstrated the centralized architecture for managing sensor and actuator devices, and their dataflows. CCDM allowed the integration and configuration of large number of components in a centralized terminal. This also helps the integration of FIAP components into a facility management system. In our experiment, CCDM has shown its capability of computing moderately complex placement algorithms in the traffic optimization case. We consider that this capability makes other optimizations feasible, such as delivery latency, load-balancing, and fault-tolerance.

In the chapter of PEAR, we proposed a reliable communication framework for wide varieties of mobility models from stable networks to totally random networks. We carried out both simulation-based and prototype-based experiments to validate the behavior of PEAR. We evaluated delivery probability, latency, total transmissions and buffer usage. Though the delay increased according to the hop count, PEAR could achieve 100% message delivery in sensor data transportation. This result indicates that PEAR is quite adaptive to various mobility models and provides reliable communication framework compared to other proposed routing schemes.

In the chapter of ICMeN, we presented our experiment report on wireless mesh networks and the application of PEAR. We carried out experiments with 50 wireless nodes in Engineering Bldg. 2 and Hongo campus, in the University of Tokyo. The result demonstrates that wireless links are totally intermittent even if nodes are physically stable. It has also shown that hop-by-hop transfer and parallel message propagation, which PEAR takes, enables scalable message propagation in hop count and increases message propagation speed.

In the chapter of DTIPN, we proposed an alternative delay tolerant networking architecture that seamlessly connects intermittently-connected networks to the existing Internet. We have implemented the architecture with PEAR for the link layer and ADTP for the application protocol layer. We have carried out several experiments in the campus of the University of Tokyo, and analyzed the result regarding to delivery rate and latency of IP packets and application messages. The result indicates that it is practically useful at least in the application of sensor and actuator networking.

Acknowledgement

Professor Hiroshi Esaki has advised the contents and directions of my research for about six years. He has let me do anything I want, and I could take part in many activities. The activities include not only academic researches, actually, but also development and operation of Live E! system, promotions, industrial meetings, educational meetings, poster presentations, demonstrations and standardization. He has allowed me to visit foreign countries 36 times for those activities until the beginning of year 2011. These wide varieties of activities have led me to enthusiastically develop many systems, carry out many experiments, write and present papers, hold workshops and meetings, which has somehow affected to the contents of this thesis. Such research activities have been also supported by the secretariat: Ms. Fumi Takatashi and Ms. Kanae Tasaka.

In the development of FIAP, I have discussed with Dr. Masahiro Ishiyama (Toshiba Corporation), Mr. Noriaki Fujiwara (Panasonic Electric Works), Mr. Tsuyoshi Momose (Cisco Systems), Mr. Kosuke Ito (Ubiteq) and a number of researchers and engineers. We have worked together to develop FIAP and to make FIAP a standard protocol. We have proposed it to IEEE 1888 and ASHRAE BACnet Committee. Mr. Kyohei Otsu in our laboratory has made a demonstration tool using FIAP.

FIAP has inherited the four-year experiences of Live E! system design and operation. I have designed, implemented, deployed and operated Live E! weather sensor network with Dr. Satoshi Matsuura (Nara Institute of Science and Technology), Mr. Hiroki Ishizuka (University of Tokyo), Mr. Masato Yamanouchi(Keio University), Mr. Akihiro Sugiyama (University of Tokyo), Mr. Yusuke Doi(University of Tokyo and Toshiba Corporation) and many researchers. Live E! system works as an overlay network in autonomous and distributed manner. Dr. Ting-Yun Chi (National Taiwan University) has operated Live E! sensor network in Taiwan, and Professor Sinchai Kamolphiwong (Prince of Songkla University, Thailand) has helped the operation in Thailand. We have extended Live E! network globally with Mr. Toshifumi Matsumoto in asia pacific networking group (APNG). Professor Hideki Sunahara have given great comments on the development and direction of Live E! system.

I have developed CCDM with Mr. Akihiro Sugiyama. With XML-based script

engines I have developed, he has implemented and demonstrated CCDM with real sensors and actuators. This work has certainly shown an important architecture for sensor actuator networks. Mr. Hironori Kawaguchi has inherited the original CCDM to fault tolerant sensor actuator networks as one of the applications. Mr. Yoshihiko Kanaumi introduced the operational issues of openflow-switch networks, which architecture is similar to CCDM.

As for DTN, many people in our laboratory have helped my research. Ms. Sathita Kaveevivitchai, Mr. Leela-Amornsin Lertluck, Mr. Sergio Carrilho, and Mr. Kenichi Shimotada have been involved in DTN research and they have brought many ideas from academic papers. When I proposed DTIPN, Mr. Kenichi Shimotada has implemented the concept using the source code of PEAR. We have carried out many experiments related to DTN with UTMesh: the 50-node scale wireless mesh networking testbed. Ms. Sathita Kaveevivitchai, Mr. Leela-Amornsin Lertluck, Mr. Kenichi Shimotada, Mr. Yuya Kawakami, Dr. Sho Fujita, Mr. Luciano Aparicio, Mr. Takehiro Sugita, Mr. Takuya Motodate, Dr. Thomas Silverstone, Mr. Minshin, Mr. Wataru Ishida, Mr. Hisatake Ishibashi, Mr. Daigo Sonoda and Mr. Kyohei Otsu greatly helped the experiment.

Ph.D students have advised and directed me from many aspects. When I was a bachelor student, Dr. Seiichi Yamamoto taught me CLI-based network configuration for constructing a network, which has now become the basic concept for FIAP. Dr. Kaoru Yoshida advised my works from his great experiences. Dr. Sho Fujita introduced many ideas, including research topics, projects, tools and methods, from his rich survey. Mr. Hirochika Asai has also provided many ideas and information from his profound studies.

It has passed nine years from when I entered the University of Tokyo. For such long years, my parents have supported my research work from my home town. It feels like not so long now, but during such a long period, I did participate in many activities and did great researches with my parents' support.

Appendix A

Facility Information Access Protocol (FIAP) Specification

Abstract

This document specifies facility information access protocol (FIAP), targeting at building-scale and city-wide energy management. The scope and the purpose of this document is to establish facility networking infrastructure over the Internet by specifying an interoperable communication protocol among the common (building-scale) facility networking components such as access gateways, data storages and application units. Thus, this document itself just provides a tool for energy-aware facility networking, and energy-management or energy-saving algorithms should be provided by other documents.

A.1 Introduction

Facility networking in buildings, houses and factories is now considered to be a promising tool for energy-management or energy-saving, and networking of facilities with TCP/IP protocols has certainly enabled building-scale or city-wide energy management. However, most of the systems are proprietary and independently developed, deployed and operated, which made the installation and running cost quite expensive.

Traditionally, in order to extend access reachability to sensors and actuators at field-bus level via the Internet, gateway design has been introduced. However, recent applications of facility networking for such scales are required beyond just a simple access to devices. In most of the practical implementation, they have (1) a large storage to archive the history of sensor readings, (2) user interface for interactive operation, (3) reporting systems and (4) data analyser.

Collaboration of these system components is mandatory, especially in energy-aware facility networking. However, they cannot simply collaborate or interoperate

with each other without unpreferred analysis, integration and operation of systems, because these system components have been independently developed and integrated proprietary.

Interoperability of them by a common communication protocol certainly increases the efficiency of facility networking deployment. It reduces the cost of system integration and interoperability management, allowing small and medium-sized buildings and even houses to install them. For vendors, their developed components can be sold worldwide without any customized implementation, sometimes resulting in mass production with reasonable cost.

This document specifies facility information access protocol (FIAP) in order to allow interoperability and open development of those system components. First, we generalize all the facility networking components (e.g., device access gateways, storages, user interfaces, reporting systems and data analyser) by a simple component model. Then, we define communication protocol among them. We also introduce registry system to support autonomous collaboration of these system components.

It is a communication infrastructure to construct a new network for the renewal of the facilities, next generation's facility management and the energy conservation including a small and medium-sized scale facilities.The aspect is expanded from a past facility management to the operation management that aims at energy conservation and the integration of the management data, monitoring and controlling by using an open and a common protocol.This infrastructure will be used for some system-level collaborations in addition to the energy conservation.

(*) This specification allows co-existence of legacy TCP/IP field-bus gateways. The main targets of this specification are to generalize all the facility networking component (e.g., not only access gateways, but also storages, user interfaces and data analyzer) by a simple component model, and to enable interoperability among them.

A.1.1 Requirement for designing this specification

FIAP is designed under following requirement. The details of each requirement item are described another document. This edition covers the basic requirement only, but the extended requirements especially security requirement will be specified soon.

*** The requirement scope of this specification**

- Support for database and its applications
- On-demand data retrieval from databases or sometimes directly from sensors
- On-demand and scheduled actuator work mode setting
- Event data delivery scheme
- Encapsulation of existing field-bus protocols by FIAP with gateways:

  - A gateway must work as a translator between the existing field-bus protocols and FIAP.
  - FIAP gateway implementation should practically support multiple field-bus protocols to apply to multi-building environment.
- Support for IPv4 and IPv6.
- XML-based data format.
- Autonomous collaboration of access gateways, databases and user interface systems.
- Identify devices(points) globally.

*** Untouched items**

- Semantics management
  - Schema for Point description (e.g., location, measurementType, targetObject and so on)
  - Schema for Value description (e.g., true,false, ...)
  - Schema for relation description.
  - Guidelines for Point properties for specific applications (e.g., for home appliances, office buildings)
- Security issues (See, Section 7. security considerations)

A.2 Architecture

This protocol specification applies to a TCP/IP-based facility networking architecture as Fig. A.1. One of the main goals of this specification is to enable interoperability among facility networking components. Thus, GW, Storage and APP, what we call "Component" in this document, have the same generalized communication interface. A Component works as a part of data-plane, and a Registry works as a part of control-plane.

Registry works as a broker of Components. It manages meta information: e.g., the role of each component and the semantics of point ID, in order to bind components appropriately and autonomously. We here describe them in more detail and show how they collaborate with each other.

Gateway (GW) Gateway component has physical sensors and actuators. It generalizes the data model and the access method for those devices, encapsulating each physical (field-bus) data model and access method. It acts on its actuator according to the written value from a component (e.g., APP), and it provides physical sensor readings for other components (e.g., Storage and APP).

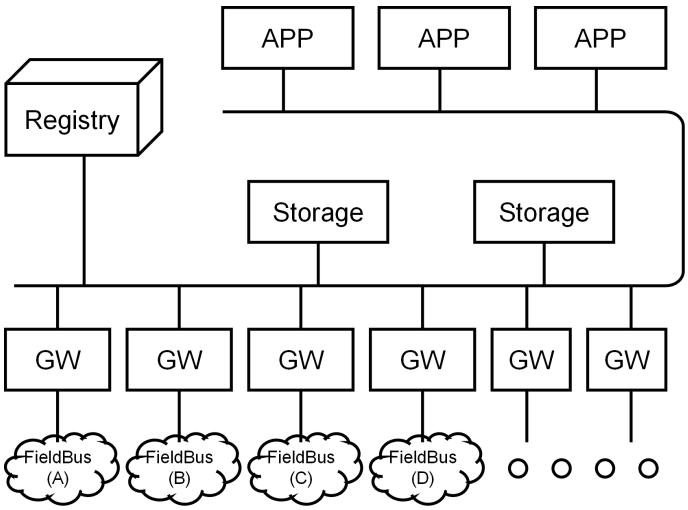

Figure A.1: Facility Networking Architecture

Storage Storage component archives the history of data sequences. The written values from other components should be permanently stored in the backend disks. It provides the archived values to the components that have requested.

Application (APP) APP component provides some particular works on sensor readings and actuator commands. It may have user interface to display the latest environmental state. It may allow an user to put some schedules of actuator settings. It may analyze some sensor data in realtime and provide the result as a virtual device.

Registry Registry works as a broker of GW, Storage and APPs. The main role of registry is to binding those components appropriately and autonomously. It is separated from the data-plane. It does not work on sensor readings or actuator settings directly.

A.2.1 Typical sequence

In this section, several typical sequences that would be taken among the components are shown. Fig. A.2 illustrates actual sequences at every case. The bold arrows indicate component-to-component communication, and the dashed arrows component-to-registry communication. There is one assumption that all the components are already configured as all end point references (EPRs) they manage and a registry EPR should be asked a point ID and its relations.

Registry manages the relationships between point ID and components. When all the components, except a GW that manages a point ID, try to connect to the

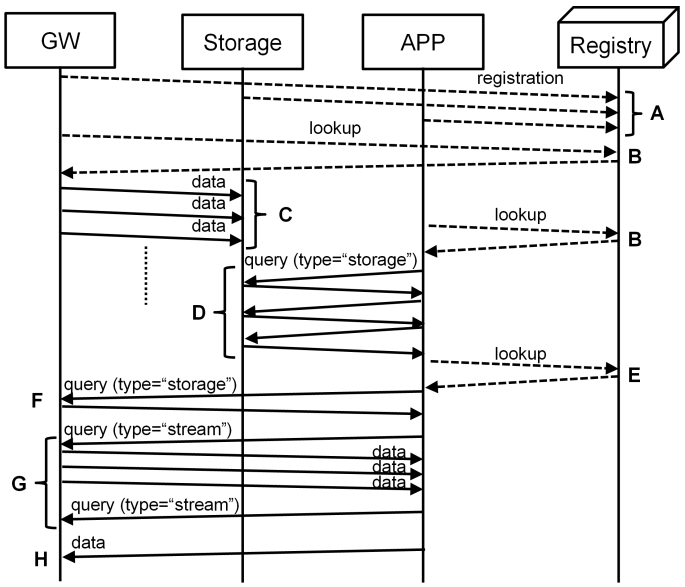

Figure A.2: Typical Sequence

point ID, that connection should be redirect to the registry. Then, the Registry replies the EPR of the actual GW that manages the point ID. Once the GW EPR was gotten at the component, it can access to the GW directly for the point ID.

Here, we describe the detail of communication from case A to case H.

Case A: Registration of Components. The registration includes the information of (i) what point IDs this GW has, (ii) what point IDs this storage manages, (iii) what point IDs this APP reads and provides.

Case B: Searching a storage component for a specified point. It returns the access URI for the resolved component.

Case C: Transmission of data; GW is sending observed sensor readings to Storage. See, section A.3.1.2 – WRITE protocol for detail.

Case D: Sequential retrieval of data; APP is retrieving data from Storage. If the retrieving dataset is large, data should be read sequentially. See, section A.3.1.1 – FETCH protocol for detail.

Case E: Searching a GW component for a specified point. It returns the access URI for the resolved component.

Case F: Retrieval of Data; APP is retrieving data from GW; If the retrieving dataset is small, data can be read in one RPC. See section A.3.1.1 – FETCH protocol for detail.

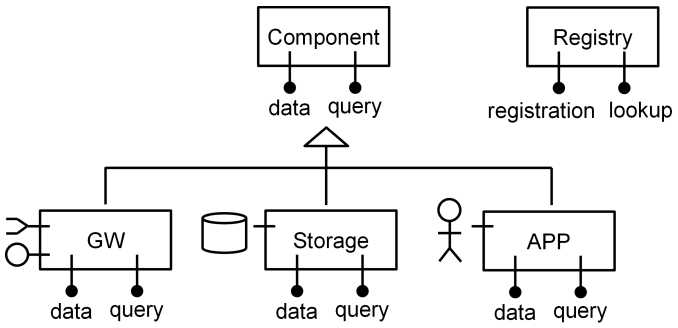

Figure A.3: FIAP System Model

Case G: Notification of Data; APP is putting event query to GW periodically, and the GW submits updates to the APP. See section A.3.1.3 – TRAP protocol for detail.

Case H: Writing of Data; APP is setting a command to a GW actuator. See section A.3.1.2 – WRITE protocol for detail.

A.2.2 Concerns of FIAP network design

As we see the above sequences, all components can behave both TCP initiator and receiver. It implies the components should be put on a flat network. A flat network means there are no middle boxes which could disturb bi-directional communications such like NAT routers and firewalls.

To avoid the issue, we strongly recommend building IPv6 network. There could be other solutions than IPv6 such like http proxy based or lots of NAT traversal solutions, however, they would depend on the requirement of the network configuration. This specification doesn't reject such solutions, but they are required to make interoperable with other FIAP based systems and not to disturb the specification. Proposing of specific solution other than IPv6 is out of scope of this document.

A.2.3 System model and deployment

Component is the basic unit for all the GWs, Storages and APPs. The interface of Component provides data and query method. GW, Storage and APP are the inherited classes of Component. Thus, they have the same interface (i.e., data and query method), and they communicate with each other using the same protocol. Here,

- query is a method for retrieving data (including event-based data transfer) from Component.

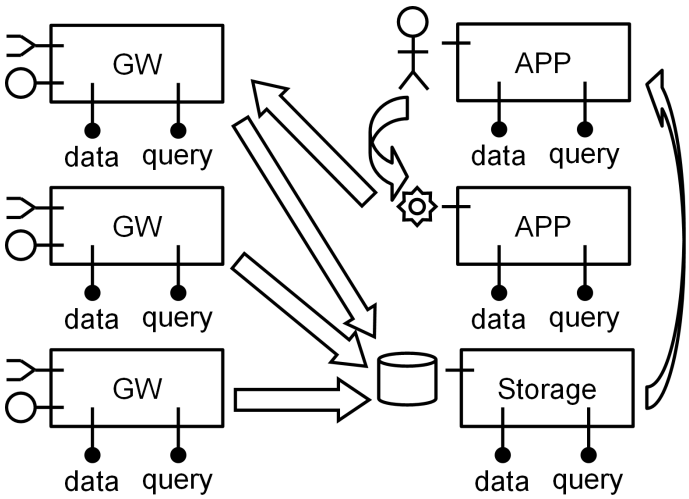

Figure A.4: Implementation of a Facility Networking System

- data is a method for pushing data into Component

Registry has another type of interface: i.e., registration and lookup method. Registration method is for registering the role of components and description of points, and lookup method is for finding a proper component for component-to-component collaboration.

Typical implementations for GWs, Storages and APPs would be:

- GW implementations encapsulate field-buses and provide INPUT/OUTPUT access for physical devices (by query and data method).
- Storage implementations archive the history of data posted by data method, and provide the historical data by query method.
- APP implementations provide other functionalities. For example, they might have user interface. Data processing component can be also categorized as an APP implementation.

(*) We also allow calling APP the system that accesses other components but does not have query and data method.

This generalization enables open development of facility networking components (i.e., GWs, Storages and APPs) by any vendors. And, we would deploy facility networking systems for customer buildings without customized programming, by binding these developed components (Fig. A.4).

The role of Registry is to increase the autonomousness of component-to-component collaboration. It allows autonomous collaboration of components, by sharing the

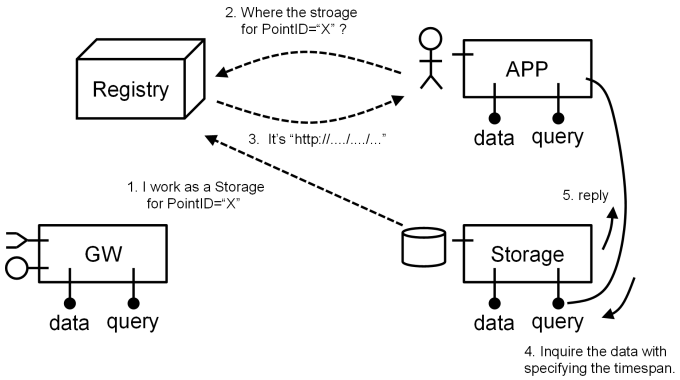

Figure A.5: Autonomous Collaboration of Components

information of component roles in an operational domain (in fact, not only in an operational domain but also with other external domains) (Fig. A.5).

A.2.4 Point

This section introduces the concept of Point. A point should have an URI-based globally unique identifier. It identifies a dataflow that exchanges data (i.e., sensor readings, actuator commands and meta-control signals) among components.

A.2.4.1 Definition

A Point is an elemental message channel for a specific data sequence among Components. A sequence of sensor readings, actuator commands and others (e.g., virtualized sensor readings, meta-control signals) should be bound to a Point. We denote a message in a Point (whether it is coming from a sensor or it is outgoing to an actuator) by value. Any object type is allowed for values in a Point.

Delivery of values among components should be made by invoking other components' interface. The provided methods are:

- query: to read objects from specified Points
- data: to write objects into specified Points

By using these methods, a component can get data of the specified Points from another component, and it also can transfer data to another component with specification of the Points.

(*) This specification extends the traditional concept of Point in facility networking. Traditionally, Point was originally given for a specific device to enable direct access (by read and write method). This definition is still true when the

target component is a gateway. However, in this specification, Storage and APPs have the same interface as GWs, so we extended the definition not only to access them but also to manage the data sequences related to them. In writing a value to a point, if the component has a GW implementation, the associated physical actuator would work according to the written value. If the component has a Storage implementation, the value would be archived in its disk.

A.2.4.2 URI-based identification

A Point is associated to a globally unique data sequence. The data must have been generated from a specific sensor or to a specific actuator in the world. Thus, in order to identify the data sequence globally, each Point should have a globally unique identifier.

In FIAP, every Point must have an URI for its identifier. Practically, we would first assign IDs for physical sensors and actuators, then we would take the IDs for the associated Point IDs. This operation goes well with the traditional facility networking operation.

Taking URI for identifiers enables global access (if the Point is public) to the Point. Let X(=http://gw.hogehoge/sensor1) be a PointID. To read the data of Point ID="X", an APP in the Internet should refer to X, and would find the Registry that manages X. Then, the APP finds related components such as Storage for X. Finally, it retrieves the historical data from the storage.

In case of the components but GW don't know the actual registry server, they should try to access the PointID directly. Then, the GW who is managing the PointID would reply with the Registry EPR that the pointID was registered. If all components already know the pointID's registry, pointID might not need to be reachable. However, from the point of the view of protocol and operation consistency, we recommend the host of the URI to be its GW host name (because physical sensors and actuators are attached to the GW). Thus, typical URI format would be:

point ID = "http://< GW host name >/< any format to identify the Point in the GW >"

A.2.4.3 PointSet

This specification also defines PointSet to enable hierarchical management of Points. A PointSet aggregates multiple Points and multiple PointSets. This definition allows the conventional operation of grouping of Points hierarchically. See, section A.5.6 PointSet Class for formal definition.

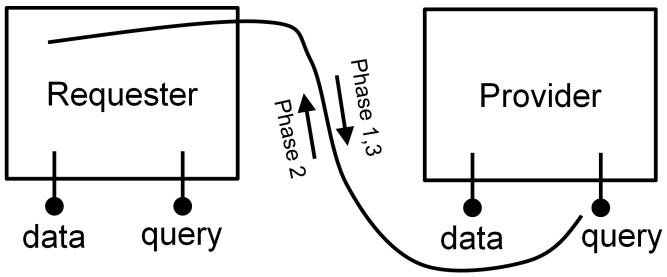

Figure A.6: FETCH Protocol

A.3 Protocol

A.3.1 Component-to-Component Communication Protocol

This section specifies the following three types of sub protocols for component-to-component communication.

(*) Instances of components are GWs, Storages and APPs. As for accessing method to a registry, see section A.3.2. component-to-registry communication protocol.

- FETCH protocol: for data retrieval from a remote component.
- WRITE protocol: for data transfer to a remote component.
- TRAP protocol: for event query registration and event data transfer

The latter part of this section describes these protocols in detail.

A.3.1.1 FETCH protocol

FETCH is a protocol for data retrieval from a remote component. We here denote the component that inquires data from the remote component by *Requester*, and the component that replies the data by *Provider*. Fig. A.6 provides a diagram of the interaction.

Phase 1: Requester invokes the query method of its Provider. The Requester must send the following information at the same time.

- query (i.e., the range of interested dataset)
- the size of acceptable dataset at the RPC-response by the number of value element

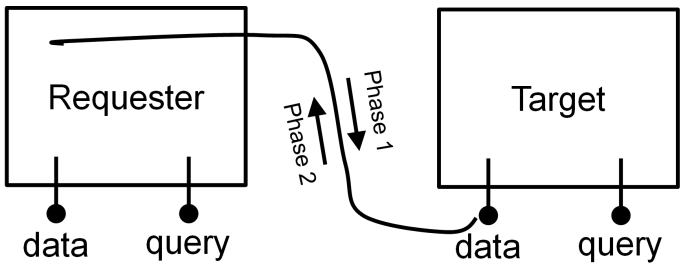

Figure A.7: WRITE Protocol

Phase 2: The Provider returns a subset of the whole dataset by the RPC-response. If the size of the returning dataset is exceeding the acceptable data size of requester, or if it is consuming too much computing resources at the provider, the provider returns only a subset of the whole dataset. For succeeding data retrieval, the provider also returns a cursor associated to it.

Phase 3: The Requester invokes the query method at the Provider again (with the given cursor), if there was a cursor in the previous response. – Go to Phase 2.

Phase 4: If not, all the dataset has been retrieved and the FETCH procedure should be completed.

(*) The cursor should have moderate validity time, and the provider must return Invalid Cursor Error if it received a query with an expired cursor.

(*) The Provider returns the information of error if it encountered: e.g., access control policy, malformed-XML or other system error.

A.3.1.2 WRITE protocol

WRITE is a protocol for data transfer to a remote component. We denote the component that submits data to the remote component by *Requester*, and the component that receives the data by *Target*. Fig. A.7 provides a diagram of the interaction.

Phase 1: Requester invokes the data method of the Target with data contents.

Phase 2: Target replies whether it was successful or ended with failure to the Requester.

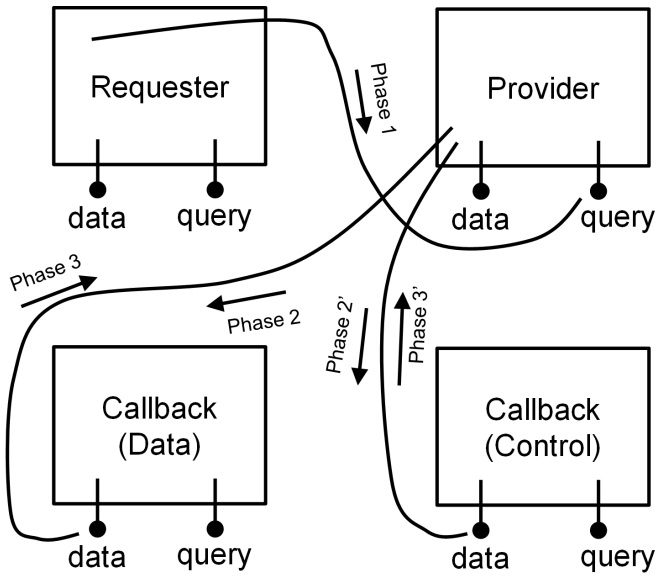

Figure A.8: TRAP Protocol (General Implementation)

A.3.1.3 TRAP protocol

TRAP is a protocol for event query registration and event data transfer. We here give names for components in the following manner.

- Requester: the component that sets event-based query to Provider.
- Provider: the component that transmits data when it has received query-matching updates.
- Callback (Data): the components that receives data from the Provider.
- Callback (Control): the components that receives control-signals from the Provider.

This subsection provides the definition of collaboration among these components(Fig. A.8). Though the roles are explicitly categorized in general, in most of the practical systems, Callback (Data), Callback(Control) and Requester will be the same component (Fig. A.9).

Phase 1: Requester invokes the query method at the Provider. Here, the arguments that should be transmitted at the same time with query-expression must include query validity time (TTL) in second, data callback URI and control-signal callback URI.

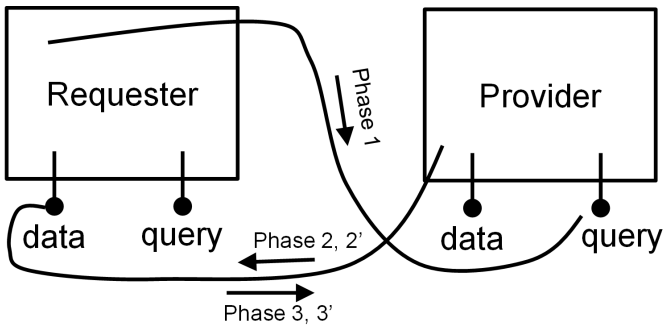

Figure A.9: TRAP Protocol (Practical Implementation)

(*) This action can be (should be) carried out periodically (e.g., with interval of one-half or one-third TTL) in order to update TTL at the Provider. Here, the same id must be set in the query.

(*) Provider should return error if it could not accept the query.

Phase 2: Provider posts the update that matches the query-expression to the specified data callback URI by its data method. The Provider sends the query at the same time.

Phase 3: The data callback replies whether it was successful or not. If it was failure, it also replies the error message.

Phase 2': Provider posts the communication errors (e.g., when it encountered an error at Phase 3) to the specified control callback URI by its data method.

Phase 3': The control callback replies whether it was successful or not. If it was failure, it also replies the error message.

(*) The TTL of the query-expression must be decreased in Provider by elapsed second. If the TTL reaches 0, Provider expires the query by removing from the query table entry.

(*) To remove a query explicitly, Requester should invoke query method with specifying 0 for TTL.

A.3.2 Component-to-Registry Communication Protocol

This section specifies the following two types of sub protocols for component-to-registry communication.

- LOOKUP Protocol: for searching appropriate components and points.

- REGISTRATION Protocol: for registration of the role of components and semantics of points.

(*) Registry has two methods (lookup and registration) for its access interface. LOOKUP protocol uses lookup method and REGISTRATION protocol uses registration method. See section A.4.3. Registry Access Interface for detail.

A.3.2.1 LOOKUP protocol

LOOKUP is a protocol for a component to search appropriate access components (for component-to-component communication), and to search points by semantic-query. The detail will be given in the future version of FIAP specification.

A.3.2.2 REGISTRATION protocol

REGISTRATION is a protocol for a component to register the role of components and semantics of points. The detail will be given in the future version of FIAP specification.

A.4 API

This section defines two types of application programming interfaces (APIs).

- Component access interface: for component-to-component communication
- Registry access interface: for component-to-registry communication

A.4.1 Transport data structure

Whether it is component-to-component communication or component-to-registry communication, access methods must be implemented by remote procedure call (RPC). In Fig. A.10, Caller is invoking a method of Callee with request-data, and response-data is returning to Caller from Callee. Here, the data structure of request and response is the same; it has a header part and a body part. The header part contains control information such as query-expressions, acknowledgement and failure information. The body part contains Point or PointSet objects with data values such as observed sensor readings and actuator commands. The control information associated to each Point or specific value should be included in the body part.

A.4.2 Component access interface

A component (i.e., GW, Storage and APP) implements this interface. This interface defines query and data method for other components to access.

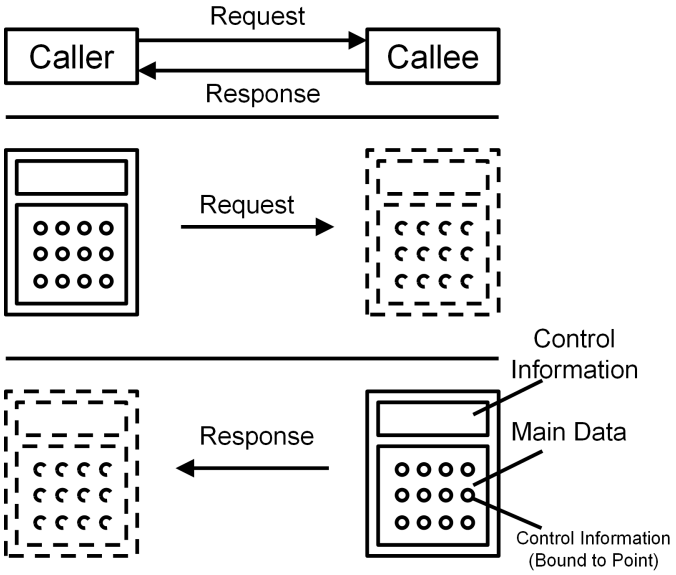

Figure A.10: Transport Data Structure

- query: to retrieve or to subscribe data from the component
- data: to post data to the component

The following three sub protocols defined in section A.3.1 are expected to be implemented with the query and data method.

- FETCH protocol
- WRITE protocol
- TRAP protocol

(*) All the components should basically implement these methods, however, some components (especially APPs) do not always need to implement them (i.e., in the case that they only access other components).

A.4.2.1 Query method

*** Format**

Transport query(Transport t)

*** Outline**

query is a method for a Requester to get data from a Provider by specifying query-expression. The query-expression must be contained in the header part of the Transport argument t.

This document specifies two types of queries: type="storage" and type="stream".

Depending on the type of the query the Provider must take the following behavior.

1. type="storage" → It works as FETCH protocol
2. type="stream" → It works as TRAP protocol

The latter part specifies the detail of query method for both cases respectively.

A.4.2.1.1 Query (type="storage")

*** Request**

The query expression must be put in the header part of t. The body part of t should be ignored.

**** Attributes of query expression**

id: For identifying the query (UUID).

type: "type" must be always set as "storage".

acceptableSize: Requester's acceptable number of value elements at a RPC-response.

cursor: For retrieval of the succeeding dataset. Cursor must not be set at the first query. This cursor should be given by the provider for the next dataset retrieval.

*** Response**

**** Header**

In the header part of the response, query expression and OK or Error object should be contained. If there is cursor attribute in the query expression, it means that there are succeeding dataset at the Provider side. It not, it means that all the data has been retrieved from the Provider.

OK object tells that the procedure at the Provider has successfully completed. Error object tells that there were some error occurred while carring out the procedure. The error information should be included in the Error object.

**** Body**

If it was successful (= if an OK object was contained in the header), objects of PointSet or Point should be contained in the body part. If not, body object should be ignored.

A.4.2.1.2 Query (type="stream")

*** Request**

The query expression must be put in the header part of t. The body part of t should be ignored.

**** Attributes of query expression**

id: For identifying the query (UUID).

type: "type" must be always set as "stream".

ttl: Valid time of the query at the Provider (i.e., time-to-live) in second.

callbackData: The callback URI of the query-match data.

callbackControl: The callback URI of the control information.

acceptableSize: callbackData's acceptable number of value elements in one transfer (optional).

*** Response**

**** Header**

In the header part of the response, query expression and OK or Error object should be contained.

OK object tells that the procedure at the provider has successfully completed. Error object tells that there were some error occurred while carrying out the procedure. The error information should be included in the Error object.

**** Body**

Body object should be ignored.

*** Other behavior**

The Provider invokes the data method of the callbackData to transfer the query-match data. Here, the provider adds the query-expression in the header part of the argument t in order that the callbackData can identify the context of the data transfer.

(*) Requester must use the same query id when it attempts to update the query TTL at the Provider. (It is TRAP protocol specification.)

A.4.2.2 Data method

*** Format**

Transport data (Transport t)

*** Outline**

data is a method for a Requester to transfer PointSet or Point objects to a Target. The data (whether they are pointset or point objects) must be contained in the body part of the Transport argument t.

When this method is used by TRAP protocol for transferring data from a Provider to a callback, the header part of t should contain the matched query-expression to the sending dataset.

*** Request**

Data objects (i.e., PointSet or Point) should be put in the body part of the argument t.

In the header part, query-expression and control-signal may be contained in the context of TRAP protocol.

*** Response**

**** Header**

In the header part of the response, OK or Error object should be contained.

OK object tells that the procedure at the provider has successfully completed. Error object tells that there were some error occurred while carrying out the procedure. The error information should be included in the Error object.

**** Body**

The body part should be ignored.

A.4.3 Registry access interface

A Registry implements this interface. This interface defines lookup and registration method for components (i.e., GW, Storage, APPs) to access.

- lookup: to find an appropriate component to access.
- registration: to register the role of component.

This interface definition corresponds to the component-to-registry communication protocol defined in section A.3.2.

A.4.3.1 Lookup method

*** Format**

Transport lookup(Transport t)

*** Outline**

lookup is a method for a component to find an appropriate component to access. LOOKUP protocol (defined in section A.3.2.1) uses this method. The detail will be given in the future version of FIAP specification.

A.4.3.2 Registration method

*** Format**

Transport registration(Transport t)

*** Outline**

registration is a method for a component to register the role of itself to a Registry. REGISTRATION protocol (defined in section A.3.2.2) uses this method. The detail will be given in the future version of FIAP specification.

A.5 Data and Query Model

This section defines data and query model for component-to-component communication. This data model basically takes the conventional tree data structure. As for component-to-registry communication, the future version of FIAP specification will provide its data and query model.

A.5.1 Point management with PointSet tree

Fig. A.11 is a typical structure of PointSet data tree. A PointSet aggregates Points and PointSets. This structure allows hierarchical management of Points by PointSet. A Point has a sequence of Value elements. A sensor reading or an actuator command is contained inside a Value element. Every PointSet and Point has a globally unique identifier by id attribute. A Value element itself usually is not

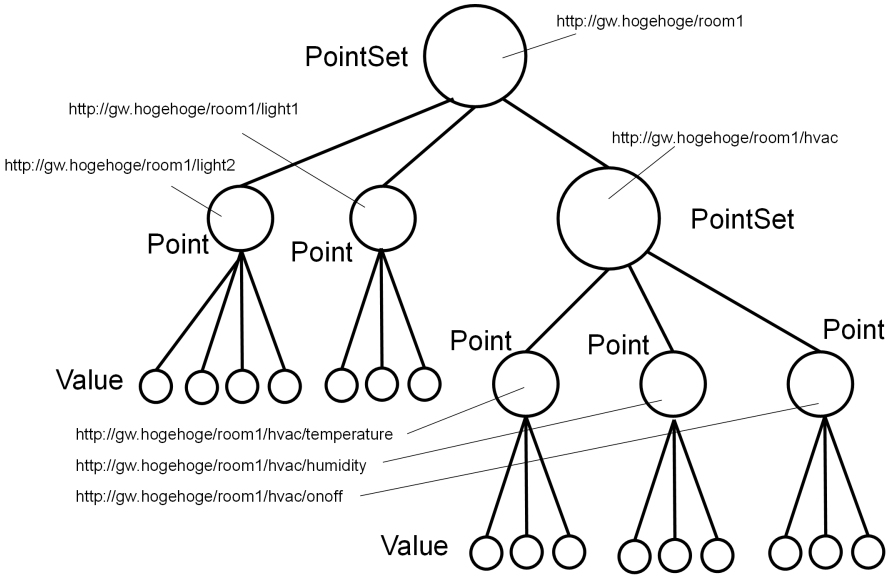

Figure A.11: PointSet Tree Data Model

bound to any globally unique identifiers, but has time attribute to identify when the value was taken from a sensor or when the value should be set to an actuator.

In this document version, we do not specify any attributes for PointSet, Point and Value except the identifier and time attribute. The attribute should be defined in the context of applications of facility networking. Thus, we remain other attributes open in order to make this protocol adaptive to any application scenarios.

A.5.2 Query model for PointSet tree

Query A get values of "http://gw.hogehoge/room1/hvac/light2" or "http://gw.hogehoge/room1/hvac/temperature", which time should be between 2009-10-01T00:00:00+09:00 and 2009-10-02T00:00:00+09:00.

```
<query>
  <key id="http://gw.hogehoge/room1/light2"
     attrName="time"
     gteq="2009-10-01T00:00:00+09:00"
     lteq="2009-10-02T00:00:00+09:00" />
  <key id="http://gw.hogehoge/room1/hvac/temperature"
     attrName="time"
     gteq="2009-10-01T00:00:00+09:00"
```

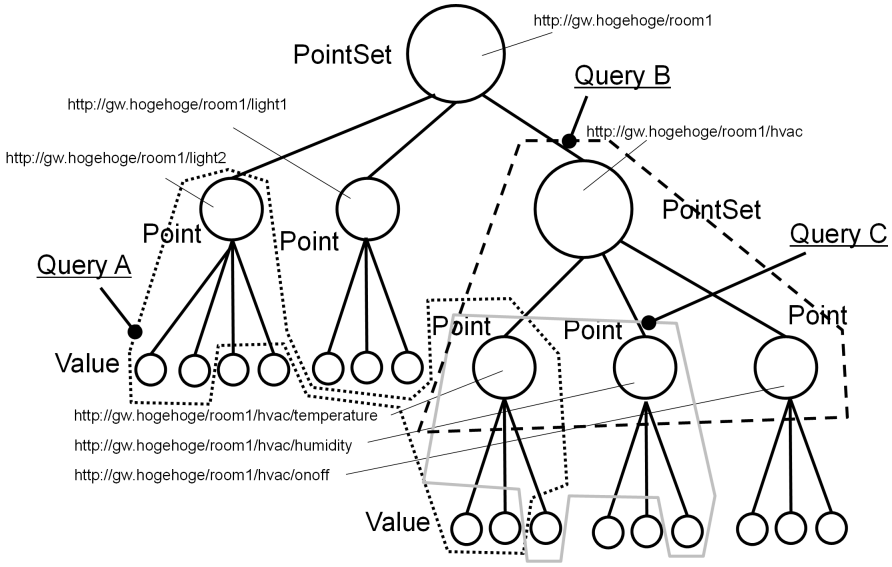

Figure A.12: Query for specifying the range of data set

```
    lteq="2009-10-02T00:00:00+09:00" />
</query>
```

Query B get point information at "http://gw.hogehoge/room1/hvac"

```
<query>
  <key id="http://gw.hogehoge/room1/hvac" />
</query>
```

Query C get the latest values of "http://gw.hogehoge/room1/hvac/temperature" and "http://gw.hogehoge/room1/hvac/humidity".

```
<query>
  <key id="http://gw.hogehoge/room1/hvac/temperature"
     attrName="time" select="maximum" />
  <key id="http://gw.hogehoge/room1/hvac/humidity"
     attrName="time" select="maximum" />
</query>
```

A.6 Data Structure

This section defines data structure for component-to-component communication protocol. All the messages must be described with XML-format. A class in this document corresponds to an XML element. The following set of class definitions provides the scheme of XML. The XML namespace for this data format is xmlns="http://gutp.jp/fiap/2009/11/".

(*) Data structure for component-to-registry communication protocol should be provided in the future version of FIAP specification.

A.6.1 Naming rules between object-class names and XML-element names

Names are case-sensitive. However, class name must start with a upper case, while XML-element name must start with a lower case. From the second letter, class name and XML-name must be identical. However, if the first two letters of class name are upper cases, the first letter of the XML must be an upper case, too.

A.6.2 List of classes

This version of FIAP protocol defines the following object classes.

- Transport
- Header
- Body
- PointSet
- Point
- Value
- Query
- Key
- OK
- Error

A.6.3 Transport class

*** Outline**

Transport class enables delivery of both data and control-plane information in one message.

*** Member**

Transport has a header object and a body object. They can be omitted when they do not have any data.

*** Attribute**

None

*** Example**

```
<transport>
  <header> ... </header>
  <body> ... </body>
</transport>
```

A.6.4 Header class

*** Outline**

Header class provides a container for control-plane information. Control-plane information includes query-expression, acknowledgement or failure of the access and redirection.

*** Member**

Query, OK and Error object

*** Attribute**

None

*** Example 1**

```
<header>
  <query ... > ... </query>
</header>
```

*** Example 2**

```
<header>
  <OK />
</header>
```

A.6.5 Body class

*** Outline**

Body object provides a container for the data of Points or PointSets.

(*) The control signals that are tightly attached to Points or Values, should be dealt with as data of them.

*** Member**

Body has PointSet objects and Point objects.

*** Attribute**

None

*** Example 1**

```
<body>
  <pointSet id="http://gw.hogehoge/tv1">...</pointSet>
  <pointSet id="http://gw.hogehoge/refrigerator1">...</pointSet>
  <pointSet id="http://gw.hogehoge/pot1">...</pointSet>
</body>
```

*** Example 2**

```
<body>
  <point id="http://gw.hogehoge/tv1/power">...</point>
  <point id="http://gw.hogehoge/tv1/switch">...</point>
  <point id="http://gw.hogehoge/pot1/power">...</point>
</body>
```

*** Example 3**

```
<body>
  <pointSet id="http://gw.hogehoge/room1/light">...</pointSet>
  <pointSet id="http://gw.hogehoge/room1/hvac">...</pointSet>
  <point id="http://gw.hogehoge/tv1/power">...</point>
  <point id="http://gw.hogehoge/pot1/power">...</point>
</body>
```

A.6.6 PointSet class

*** Outline**

PointSet aggregates related Point objects and PointSet objects. In the conventional operation, we develop hierarchical data structure to manage related Points by one identifier. PointSet class enables this type of aggregation.

*** Member**

PointSet has PointSet objects and Point objects.

*** Attribute**

id: URI-based identifier for this PointSet.

*** Example 1**

```
<pointSet id="http://gw.hogehoge/room1">
  <pointSet id="http://gw.hogehoge/room1/light/">...</pointSet>
  <pointSet id="http://gw.hogehoge/room1/hvac">...</pointSet>
</pointSet>
```

*** Example 2**

```
<pointSet id="http://gw.hogehoge/tv1/">
  <point id="http://gw.hogehoge/tv1/power">...</point>
  <point id="http://gw.hogehoge/tv1/switch">...</point>
  <point id="http://gw.hogehoge/tv1/channel">...</point>
</pointSet>
```

*** Example 3**

```
<pointSet id="http://gw.hogehoge/room1">
  <pointSet id="http://gw.hogehoge/room1/light">...</pointSet>
  <pointSet id="http://gw.hogehoge/room1/hvac">...</pointSet>
  <point id="http://gw.hogehoge/room1/temperature">...</point>
  <point id="http://gw.hogehoge/room1/humidity">...</point>
</pointSet>
```

A.6.7 Point class

*** Outline**

Point object is a representation of "Point".

*** Member**

A Point object has value objects.

*** Attribute**

id: identifier of the point (i.e., PointID)

*** Example**

```
<point id="http://gw.hogehoge/room1/temperature">
  <value time="2009-09-01T00:00:00.0000000+09:00">25.5</value>
  <value time="2009-09-01T00:01:00.0000000+09:00">25.6</value>
  <value time="2009-09-01T00:02:00.0000000+09:00">25.6</value>
  <value time="2009-09-01T00:03:00.0000000+09:00">25.7</value>
  ...
</point>
```

A.6.8 Value class

*** Outline**

A Value object contains one specific data value. An input data from a sensor, and an out going data to an actuator should be encapsulated by Value object.

*** Member**

None

*** Attribute**

time: generated time (INPUT case) or scheduled time (OUTPUT case) in W3CTimestamp format.

*** Example**

```
<value time="2009-10-19T00:00:00.0000000+09:00">true</value>
<value time="2009-10-19T00:00:00.0000000+09:00">false</value>
<value time="2009-10-19T00:00:00.0000000+09:00">10</value>
<value time="2009-10-19T00:00:00.0000000+09:00">0</value>
<value time="2009-10-19T00:00:00.0000000+09:00">3.4</value>
<value time="2009-10-19T00:00:00.0000000+09:00">0.5323</value>
<value time="2009-10-19T00:00:00.0000000+09:00">HIGH</value>
```

```
<value time="2009-10-19T00:00:00.0000000+09:00">MID</value>
<value time="2009-10-19T00:00:00.0000000+09:00">LOW</value>
```

A.6.9 Query class

*** Outline**

Query object manages query-expressions for storage-based and stream-based queries.

*** Member**

Query class has Key objects.

*** Attribute**

id: the identifier of this query (UUID)

type: the type of this query := storage, stream

cursor: the cursor for sequential dataset retrieval (valid when type=”storage”)

acceptableSize: Receiver's maximum acceptable size of value objects at one RPC.

ttl: validity time of query at the Provider (valid when type=”stream”)

callbackData: The URI of data callback in TRAP protocol (valid when type=”stream”)

callbackControl: The URI of control-signal callback in TRAP protocol (valid when type=”stream”)

*** Example**

```
<query id="6229c37f-970d-9292-83e4-7c0e54733f8a"
     type="storage"
     acceptableSize="20"
     cursor="dab751ed-0133-4ce4-8b7d-ba5c54ce4fb5">
   <key> ... </key>
   <key> ... </key>
</query>
<query id="9eed9de4-1c48-4b08-a41d-dac067fc1c0d"
     type="stream"
     ttl="60"
     callbackData="http://hogehoge/axis/services/GUTAPI"
     callbackControl="http://hogehoge/axis/services/GUTAPI">
   <key> ... </key>
   <key> ... </key>
</query>
```

A.6.10 Key class

*** Outline**

Key class allows description of query-expression. A Key object corresponds to a key predicate in Section 5.2. Query Model.

*** Member**

Key class has Key objects.

*** Attribute**

id: the target id of point or pointSet.

attrName: the attribute name for the following

eq: this predicate becomes true if the key attribute value is equal to the specified value, otherwise becomes false.

neq: this predicate becomes true if the key attribute value is not equal to the specified value, otherwise becomes false.

lt: this predicate becomes true if the key attribute value is less than the specified value.

gt: this predicate becomes true if the key attribute value is greater than the specified value.

lteq: this predicate becomes true if the key attribute value is less than or equal to the specified value.

gteq: this predicate becomes true if the key attribute value is greater than or equal to the specified value.

select: for selection of maximum, minimum attribute value

trap: for event detection := changed (valid if the query type="stream")

*** Example**

```
<key id="http://gw.foo.org/room1/temperature"
     attrName="time" select="maximum" />
<key id="http://gw.foo.org/room1/temperature"
     attrName="time"
     lteq="2009-10-01T00:00:00.0000000+08:00"
     gteq="2009-09-01T00:00:00.0000000+08:00" />
<key id="http://gw.foo.org/room1/temperature"
     attrName="time" trap="changed" />
<key id="http://gw.foo.org/room1/temperature"
     attrName="value" trap="changed" />
```

A.6.11 OK class

*** Outline**

OK object tells that a request has been successfully accepted.

*** Member**

None

*** Attribute**

None

*** Example**

```
<OK />
```

A.6.12 Error class

*** Outline**
Error class contains error messages.
*** Member**
None
*** Attribute**
type: category of error
*** Example**

```
<error type="syntax">Malformed XML Error at line X.</error>
<error type="authorization">
  You are not permitted to access the requested resource.
</error>
```

A.7 Security Considerations

In this edition, we don't provide any security specification at this time. But the security is the most important concerns especially for such open system. This section provides some requirements and considerations for designing security in later versions.

FIAP protocol is basically open; it assumes multi-domain operation and public access from other domain's system components. In this context, security requirements to the system would be listed as follows:

- To avoid unintended data disclosure to the public
- To avoid unauthorized access to writable resources
- Availability and confidentiality of remote communication host
- Integrity and confidentiality of data
- To avoid unintended access or operational conflicts

To get confidentiality of remote communication host, we would be able to take VPN, SSL, SSH and other related technologies. SOAP and its security extension would help in getting integrity and confidentiality of data. Access control and access confliction management must be other important but different types of security issues that should be discussed independently. Generally, access control is used to allow only specific users to access both readable and writable resources, which would certainly help to avoid unauthorized access from or unintended data disclosure to the public (sometimes anonymous) users. In order to manage this, the system would need to introduce the concept of users to identify who is accessing the resources. We assume URI-based identification for user authentication just as Point ID takes

URI for its identifier. Authentication of these users and components (probably by taking advantage of the existing authentication platforms) would be certainly need to be considered.

Bibliography

[1] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *ACM SIGCOMM*, 2003.

[2] Christian Bettstetter, Giovanni Resta, and Paolo Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, Vol. 2, No. 3, pp. 257–269, jul 2003.

[3] Injong Ree, Minsu Shin, Seongik Hong, Kyunghan Lee, and Song Chong. On the levy-walk nature of human mobility. In *IEEE INFOCOM*, 2008.

[4] Xiaolan Zhang, Jim Kurose, Brian Neil Levine, Don Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network: Mobility modeling and impact on routing. In *ACM MobiCom*, 2007.

[5] Sede M., Xu Li, Da Li, Min-You Wu, Minglu Li, and Wei Shu. Routing in large-scale buses ad hoc networks. In *IEEE WCNC*, 2008.

[6] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acuisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, Vol. 30, No. 1, pp. 122–173, mar 2005.

[7] Yong Yao and Johannes Ghrke. Query processing for sensor networks. In *CIDR*, 2003.

[8] Johannes Gehrke and Samuel Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, Vol. 3, pp. 46–55, 2004.

[9] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Don Carney, Ugur Cetintemel, Ying Xing, and Stan Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[10] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *CIDR*, 2005.

[11] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *IEEE MDM 2007*, may 2007.

[12] Daniel J. Abadi, Wolfgang Lindner, Samuel Madden, and Jorg Schuler. An integration framework for sensor networks and data stream management systems. In *30th VLDB Conference*, pp. 1361–1364, 2004.

[13] Bhaskar Krishnamachari, Deborah Estrin, and Stephen Wicker. The impact of data aggregation in wireless sensor networks. In *IEEE Distributed Computing Systems*, 2002.

[14] Magdalena Balazinska, Amol Deshpande, Michael J. Franklin, Phillip B. Gibbons, Jim Gray, Suman Nath, Mark Hansen, Michael Liebhold, Alexander Szalay, and Vincent Tao. Data management in the worldwide sensor web. *IEEE Pervasive Computing*, Vol. 6, No. 2, pp. 30–40, apr 2007.

[15] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. Irisnet: an architecture for a world wide sensor web. *IEEE Pervasive Computing*, Vol. 2, No. 4, pp. 22–33, dec 2003.

[16] L. A. Bryan and E. A. Bryan. Programmable controllers – theory and implementation, 1988. Industrial Text.

[17] M.C. Zhou and E. Twiss. Design of industrial automated systems via relay ladder logic programming and Petri nets. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 28, , feb 1998.

[18] Mario Neugebauer, Jorn Plonnings, Klaus Kabitzsh, and Peter Buchholz. Automated modeling of lonworks building automation networks. In *IEEE Factory Communication Systems*, 2004.

[19] Ed. R. Enns. RFC4741: NETCONF Configuration Protocol, dec 2006.

[20] Nick McKweown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, Vol. 38, , apr 2008.

[21] Takayuki Inamori, Yousuke Watanabe, Hiroyuki Kitagawa, and Toshiyuki Amagasa. A proposal of management system for distributed stream processing environments. Technical report, IEICE DEWS, feb 2007.

[22] Arvind Arasu, Shivnath Babu, and Jennifer Widom. CQL: a language for continuous queries over streams and relations. *LNCS*, Vol. 2921, pp. 1–19, 2004.

[23] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *IEEE MDM*, 2007.

[24] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Don Carney, Ugur Centitemel, Yuing Xing, and Stan Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[25] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. Operator placement for in-network stream query processing. In *ACM SIGMOD*, 2005.

[26] Sangeetha Seshadri, Vibhore Kumar, and Brian F. Cooper. Optimizing multiple queries in distributed data stream systems. In *IEEE Data Engineering*, 2006.

[27] Akihiro Sugiyama, Hideya Ochiai, and Hiroshi Esaki. CCDM: central controller-based device management architecture and method to split management scripts. In *IEEE SAINT*, 2009.

[28] Mehul A. Shah, Joseph M. Hellerstein, and Eric Brewer. Highly available, fault-tolerant, parallel dataflows. In *ACM SIGMOD*, 2004.

[29] Jeong-Hyon Hwang, Magdakena Balazinska, Alexander Rasin, Ugur Cetintemel, Michael Stonebraker, and Stan Zdonik. High-availability algorithms for distributed stream processing. In *ACM ICDE*, 2005.

[30] Bence Pasztor, Mirco Musolesi, and Cecilia Mascolo. Opportunistic mobile sensor data collection with scar. In *IEEE MASS*, 2007.

[31] Xu Li, Wei Shu, Minglu Li, Hongyu Huang, and Min-You Wu. DTN routing in vehicular sensor networks. In *IEEE Globecom*, 2008.

[32] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In *ACM SIGOPS*, pp. 96–107, 2002.

[33] Pei-Chun Cheng, Kevin C. Lee, Mario Gerla, and Jerome Harri. GeoDTN+Nav: geographic dtn routing with navigator prediction for urban vehicular environments, jun 2009.

[34] Franck Laurent and Gil-Castineira Felipe. Using delay tolerant networks for car2car communications. In *IEEE Inductrial Electronics*, 2007.

[35] J. Moy. RFC2328: OSPF Version 2, apr 1998.

[36] G. Malkin. RFC2453: RIP Version 2, nov 1998.

[37] C. Perkins, E. Belding-Royer, and S. Das. RFC3561: ad hoc on-demand distance vector (AODV) routing, jul 2003.

[38] T. Clausen and P. Jacquet. RFC3626: optimized link state routing protocol (OLSR), oct 2003.

[39] Dichael Demmer and Kevin Fall. DTLSR: Delay tolerant routing for developing regions. In *ACM NSDR*, 2007.

[40] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. MaxProp: Routing for vehicle-based disruption-tolerant networks. In *IEEE INFOCOM*, 2006.

[41] Aruna Balasubramanian, Brian Neil Levine, and Arun Venkataramani. DTN routing as a resource allocation problem. In *ACM SIGCOMM*, 2007.

[42] Andres Lindgren, Avri Doria, and Olov Schelen. Probabilistic routing in intermittently connected networks. *LNCS*, Vol. 3126, pp. 239–254, sep 2004.

[43] Joy Ghosh, Sumesh J. Philip, and Chunming Qiao. Sociological orbit aware location approximation and routing (SOLAR) in MANET. *ACM Ad Hoc Networks*, Vol. 5, No. 2, pp. 189–209, mar 2007.

[44] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, Duke University, 2000.

[45] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and Wait: An efficient routing scheme for intermittently connected mobile networks. In *ACM SIGCOMM*, 2005.

[46] Jorg Hahner, Christian Becker, and Kurt Rothermel. A protocol for data dissemination in frequently partitioned mobile ad hoc networks. In *IEEE ISCC*, 2003.

[47] Jay Boice, J. J. Garcia-Luna-Aceves, and Katia Obraczka. On-demand routing in disrupted environments. *LNCS Networking*, Vol. 4479, pp. 155–166, nov 2007.

[48] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, and Song Chong. On the levy-walk nature of human mobility. In *IEEE INFOCOM*, 2008.

[49] Anindya Basu, Alvin Lin, and Sharad Ramanathan. Routing using potentials: A dynamic traffic-aware routing algorithm. In *ACM SIGCOMM 2003*, 2003.

[50] Xiangchuan Chen and Amy L. Murphy. Enabling disconnected transitive communication in mobile ad hoc networks. In *ACM POMC*, 2001.

[51] Thrasyvoulos Spyropoulos, Thierry Turletti, and Katia Obraczka. Routing in delay tolerant networks comprising heterogeneous node populations. *IEEE Transactions on Mobile Computing*, Vol. 8, No. 8, pp. 1132–1147, aug 2009.

[52] Chao Chen and Zesheng Chen. Evaluating contacts for routing in highly partitioned mobile networks. In *ACM MobiOpp*, 2007.

[53] Mirco Musolesi, Stephen Hailes, and Cecilia Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. In *IEEE WoWMoM*, 2005.

[54] Haiyang Liu, Zhi-Li Zhang, Jaideep Srivastava, and Victor Firoiu. PWave: A multi-source multi-sink anycast routing framework for wireless sensor networks. *LNCS Networking*, Vol. 4479, pp. 179–190, nov 2007.

[55] Yashar Ganjali and Nick McKeown. Routing in a highly dynamic topology. In *IEEE SECON*, 2005.

[56] Hideya Ochiai and Hiroshi Esaki. Mobility entropy and message routing in community-structured delay tolerant networks. In *ACM AINTEC*, pp. 93–102, 2008.

[57] Hideya Ochiai, Kenichi Shimotada, and Hiroshi Esaki. DTIPN: delay tolerant IP networking for opportunistic network applications. In *ACM MobiOpp*, 2010.

[58] Apoorva Jindai and Konstantinos Psounis. Fundamental mobility properties for realistic performance analysis of intermittently connected mobile networks. In *IEEE Percom*, 2007.

[59] Natasa Sarafijanovic-Djukic, Michal Piorkowski, and Matthias Grossglauser. Island Hopping: Efficient mobility-assised forwarding in partitioned networks. In *IEEE SECON*, 2006.

[60] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on the design of opportunistic forwarding algorithms. In *IEEE INFOCOM*, 2006.

[61] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine*, Vol. 41, No. 6, pp. 128–136, jun 2003.

[62] Christian Tschudin and Evgeny Osipov. Estimating the ad hoc horizon for TCP over IEEE 802.11 networks. In *Med-Hoc-Net*, 2004.

[63] Joy Ghosh, Hung Q. Ngo, and Chunming Qiao. Mobility profile based routing within intermittently connected mobile ad hoc networks (ICMAN). In *ACM IWCMC*, 2006.

[64] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Efficient routing in intermittently connected mobile networks: the single-copy case. *IEEE/ACM Transactions on Networking*, Vol. 16, No. 1, pp. 63–76, feb 2008.

[65] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Efficient routing in intermittently connected mobile networks: the multiple-copy case. *IEEE/ACM Transactions on Networking*, Vol. 16, No. 1, pp. 77–90, feb 2008.

[66] Athanasios Kinalis and Sotiris Nikoletseas. Adaptive redundancy for data propagation exploiting dynamic sensory mobility. In *ACM MSWiM*, 2008.

[67] Kaoru Yoshida and Hiroshi Esaki. Energy saving with ICT: Green university of tokyo project. In *EcoDesign*, 2009.

[68] A.A. Franklin and C.S.R. Murthy. Node placement algorithm for deployment of two-tier wireless mesh networks. In *IEEE Globecom*, 2007.

[69] Joshua Robinson, Mohit Singh, Ram Swaminathan, and Edward Knightly. Deploying mesh nodes under non-uniform propagation. In *IEEE INFOCOM*, 2010.

[70] Shiwen Mao, Shunan Lin, Shivendra S. Panwar, Yao Wang, and Emre Celebi. Video transport over ad hoc networks: multistream coding with multipath transport. *IEEE Journal on Selected Areas in Communications*, Vol. 21, , dec 2003.

[71] Katayoun Sohrabi, Jay Gao, Vishal Ailawadhi, and Gregory J Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, Vol. 7, , oct 2000.

[72] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurement from an 802.11b mesh network. In *ACM SIGCOMM*, 2004.

[73] Christian Tschudin, Per Gunningberg, Henrik Lundgren, and Erik Nordstrom. Lessons from experimental manet research. *Elsevier Ad Hoc Networks*, pp. 221–233, 2005.

[74] Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordstrom, and Christian Tschudim. A large-scale testbed for reproducible ad hoc protocol evaluations. In *IEEE WCNC*, 2002.

[75] Simon Heimlicher and Bernhard Plattner. Reliable transport in multihop wireless mesh networks. In *Guide to wireless mesh networks*, pp. 231–254. Springer, 2009.

[76] Stephen Mueller, Rose P. Tsang, and Dipak Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. *Performance Tools and Applications to Networked Systems*, pp. 209–234, 2004.

[77] Aristotelis Tsirigos and Zygmunt J. Haas. Multipath routing in the presence of frequent topology changes. *IEEE Communications Magazine*, Vol. 39, pp. 132–138, nov 2001.

[78] Rosario G. Garroppo, Stefano Giordano, Davide Iacono, and Luca Tavanti. On the development of a IEEE 802.11s mesh point prototype. In *ACM TridentCom*, 2008.

[79] Timo Vanhatupa, Marko Hannikainen, and Timo D. Hamalainen. Performance model for IEEE 802.11s wireless mesh network deployment design. *Journal of Parallel and Distributed Computing*, Vol. 68, pp. 291–305, mar 2008.

[80] V. Cerf, S Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. RFC4838: delay tolerant networking architecture, apr 2007.

[81] Yu Wang and Hongyi Wu. Delay/fault-tolerant mobile sensor network (DFT-MSN): a new paradigm for parvasive information gathering. *IEEE Transactions on mobile computing*, Vol. 6, No. 9, sep 2007.

[82] Jorg Ott, Dirk Kutscher, and Christoph Dwertmann. Integrating DTN and MANET routing. In *ACM SIGCOMM workshop*, pp. 221–228, 2006.

[83] Gabriele Ferrari Aggradi, Flavio Esposito, and Ibrahim Matta. Supporting predicate routing in DTN over MANET. In *ACM CHANTS*, 2008.

[84] T. V. Lakshman and Upamanyu Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, Vol. 5, No. 3, pp. 336–350, jun 1997.

[85] Hideya Ochiai, Kenichi Shimotada, and Hiroshi Esaki. IP over DTN: large-delay asynchronous packet delivery in the Internet. In *IEEE ICUMT workshop*, 2009.

[86] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, pp. 784–803, dec 1997.

[87] Jorg Nonnenmacher, Ernst W. Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4, pp. 349–361, aug 1998.

[88] Drik Kutscher, Janico Greifenberg, and Kevin Loos. Scalable dtn distribution over uni-directional links. In *ACM SIGCOMM Workshop*, 2007.

[89] Armadillo220. http://www.atmark-techno.com/.

Publication

Journal / Transaction

[P1] Hideya Ochiai and Hiroshi Esaki, "Topology Change Tolerant Routing for Delay Tolerant Networks", IPSJ Journal, Vol. 50, No.9, pp.2312–2326, September, 2009.

[P2] Hideya Ochiai, Satoshi Matsuura and Masato Yamanouchi, "センサネットワーキングの新たな展開を目指して Live E! Workshop in APNG Camp 活動報告", IPSJ Journal, 情報処理, Vol.50, No.1, pp.55–63, January, 2009.

[P3] Hideya Ochiai, Satoshi Matsuura, Hideki Sunahara, Masaya Nakayama and Hiroshi Esaki, "Operating Architecture and Multi-Attribute Search for Wide Area Sensor Networks", IEICE Transaction on Communications, Vol.J91-B, No.10, pp.1160–1170, October, 2008.

[P4] Hideya Ochiai and Hiroshi Esaki, "Networked GPIO Control by High-Level Languages and Protocol Translator", IPSJ Journal, Vol.49, No.10, pp.3451–3461, October, 2008.

Conferences

[P5] Satoshi Matsuura, Hideya Ochiai, Shingo Kimura, Kazutoshi Fujikawa, Hideki Sunahara, "A Large Scale Content-Based Network Considering Publish / Process / Subscribe", GlobeCom 2010(ASIT'10), December, 2010.

[P6] Hideya Ochiai, Hiroki Ishizuka, Yuya Kawakami and Hiroshi Esaki, "A Field Experience on DTN-Based Sensor Data Gathering in Agricultural Scenarios", IEEE Sensors, November, 2010

[P7] Kaveevivitchai Sathita, Hideya Ochiai, Hiroshi Esaki, "Message Deletion and Mobility Patterns for Efficient Message Delivery in DTNs", IEEE PerCom, Mannheim, Germany, March, 2010.

[P8] H.Ochiai, K.Shimotada and H.Esaki, "DTIPN: Delay Tolerant IP Networking for Opportunistic Network Applications", ACM MobiOpp, Italy, February, 2010.

[P9] H.Ochiai, N.Fujiwara and H.Esaki, "Green UT Energy-Aware Facility Networking: a Challenge to Standardization of Architecture and its Protocol", ECODESIGN, Japan, December, 2009.

[P10] H.Shimonishi, H.Ochiai, N.Enomoto and A.Iwata, "Building Hierachical Switch Network Using Openflow", International Workshop on Information Network Design (WIND), Spain, November, 2009.

[P11] H.Ochiai, K.Shimotada and H.Esaki, "IP over DTN: Large-Delay Asynchronous Packet Delivery in the Internet", IEEE ICUMT, Russia, October, 2009.

[P12] H.Ochiai and H.Esaki, "Toward Open Facility Networking: Semantics Management for Higher-Level Interoperability", Asia Future Internet, Korea, August, 2009.

[P13] Elyes Ben Hamida, Hideya Ochiai, Hiroshi Esaki, Pierre Borgnat, Patrice Abry and Eric Fluery, "Measurement Analysis of the Live E! Sensor Network: Spatial-Temporal Correlations and Data Aggregation", IEEE/IPSJ SAINT workshop, USA, July, 2009.

[P14] Kaveevivitchai Sathita, Hideya Ochiai and Hiroshi Esaki, "RainWatch Project: Location-Aware Realtime Detection and Notification of Rain on Internet-Based Sensor Network", IEEE/IPSJ SAINT workshop, USA, July, 2009.

[P15] A.Sugiyama, H.Ochiai and H.Esaki, "CCDM: Central Controller-Based Device Management Architecture and Method to Split Management Scripts", IEEE/IPSJ SAINT workshop, USA, July, 2009.

[P16] H.Ochiai and H.Esaki, "Message Routing on Potential-Fields in Forwarding-Based DTNs", ACM ICUIMC, Korea, January, 2009.

[P17] H.Ochiai and H.Esaki, "Mobility Entropy and Message Routing in Community-Structured Delay Tolerant Networks", ACM AINTEC, Thailand, November, 2008.

[P18] H.Ochiai and H.Esaki, "Accuracy-Based Cache Consistency Management for Numerical Object Replication", IEEE/IPSJ SAINT workshop, Finland, July, 2008.

[P19] H.Ochiai and H.Esaki, "Topology Change Tolerant Routing for Delay Tolerant Networks", IPSJ DICOMO(in Japanese), pp. 932–943, Hokkaido, Japan, July, 2008.

[P20] S.Matsuura, S.Doi, H.Ochiai, H.Ishizuka, H.Esaki and H.Sunahara, "Live E!: control system for sensor stream and geographical location based overlay network managing Ubiquitous sensors", IPSJ DICOMO (in Japanese), pp. 1161–1167, Mie, Japan, July, 2007.

[P21] H.Ochiai, S.Matsuura, H.Sunahara and H.Esaki, "Architecture for Centralized Management of Sensor Data Upload Stream", (in Japanese), JSSST Workshop on Internet Technology (WIT), Tottori, Japan, June, 2007.

[P22] H.Ochiai and H.Esaki, "Architecture of Scalable Embedded Device Management System with Configurable Plug-In Translator", IEEE/IPSJ SAINT2007 Workshop on Practical Applications of Sensor Networking, Hiroshima, Japan, January 2007.

[P23] H.Ochiai, Z.Wang, R.Oguchi, A.Sugiyama, Y.Sakamoto, S.Ishida and H.Esaki, "Application of Content-Based Network for Sensor Data Distribution System", IEEE/IPSJ SAINT2007 Workshop on Practical Applications of Sensor Networking, Hiroshima, Japan, January 2007.

[P24] S.Matsuura, H.Ishizuka, H.Ochiai, S.Doi, S.Ishida, M.Nakayama, H.Esaki and H.Sunahara, "Live E! Project: Establishment of Infrastructure Sharing Environmental Information", IEEE/IPSJ SAINT2007 Workshop on Practical Applications of Sensor Networking, Hiroshima, Japan, January 2007.

[P25] S.Fujita, H.Ochiai and H.Esaki, "Cooperative Log-Taking Support System Exploiting RFID and Web2.0 Technologies", JSST 14th Workshop on Interactive Systems and Software (WISS 2006), Iwate, Japan, December 2006.

[P26] H.Ochiai, Z.Wang, R.Oguchi, A.Sugiyama, Y.Sakamoto, S.Ishida and H.Esaki, "Application of Content-Based Network for Sensor Data Distribution System", (in Japanese), In Proceedings of the 2006 IEICE Society Conference, Kanazawa, Japan, September 2006.

[P27] H.Ochiai, "The Application of XML Translator for a Large-Scale Sensor Node Network", (in Japanese), In Proceedings of the 2006 IEICE General Conference, Tokyo, Japan, March 2006.

Technical Report

[P28] Hironori Kawaguchi, Hideya Ochiai, Hiroshi Esaki, "Decision of Dataflow Based on Structure of LAN in Data Stream Processing", IEICE IA2010-70, December, 2010.

[P29] Hideya Ochiai, Masaya Nakayama, Hiroshi Esaki, "UTMesh: An Evaluation Testbed for Wireless Networking - Development and Experiment Report", IEICE IA2010-67, December, 2010.

[P30] H.Ochiai, S.Kimura, S.Matsuura, H.Sunahara, H.Fukuhara and H.Esaki, "Content-Based Network におけるデータ処理コンポーネント最適配置問題", IPSJ DPS139-12, June, 2009.

[P31] H.Ochiai, A.Sugiyama, Y.Sakamoto, M.Yamanouch, H.Ishizuka, S.Matsuura, H.Sunahara, M.Nakayama and H.Esaki, "Analysis of the Operating Status of Live E! Wide Area Sensor Network", IEICE IA2008-3, Japan, May, 2008.

[P32] A.Sugiyama, H.Ochiai and H.Esaki, "Merged Data Upload in Sensor Network for Lower Server Load", IEICE IA2007-36, Japan, October, 2007.

[P33] H.Ochiai and H.Esaki, "Evaluation of an Embedded OLSR Node for IP sensor System", IEICE IA2007-26, Japan, July, 2007.

[P34] H.Ochiai and H.Esaki, "Sensor Profile Consistency and Interoperability in Global Sensor Networks", IEICE IA2007-23, Japan, July, 2007.