Title	A Study on Design and Development Support for Cooperative Wireless Sensing Systems
Author(s)	Mori, Shunsuke
Citation	大阪大学, 2013, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/27486
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

情甲 16075

A Study on Design and Development Support for Cooperative Wireless Sensing Systems

January 2013

Shunsuke MORI

A Study on Design and Development Support for Cooperative Wireless Sensing Systems January 2013 Shunsu

A Study on Design and Development Support for Cooperative Wireless Sensing Systems

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2013

Shunsuke MORI

List of Publications

Journal Papers Corresponding to Thesis

1. Shunsuke Mori, Takaaki Umedu, Akihito Hiromori, Hirozumi Yamaguchi, and Teruo Higashino: "A design support environment for wireless sensor networks" *IPSJ Journal*, vol. 50, no. 10, pp. 2556–2567, (October 2009), (in Japanese).

Conference Papers Corresponding to Thesis

1. Kazushi Ikeda, Shunsuke Mori, Yuya Ota, Takaaki Umedu, Akihito Hiromori, Hirozumi Yamaguchi, and Teruo Higashino, "D-sense: An integrated environment for algorithm design and protocol implementation in wireless sensor networks", *Proceedings of the 11th IFIP/IEEE international conference on Management of Multimedia and Mobile Networks and Services (MMNS 2008)*, pp. 20–32. Samos Island, Greece, (September 2008).

2. Shunsuke Mori, Yu-Chih Wang, Takaaki Umedu, Akihito Hiromori, Hirozumi Yamaguchi, and Teruo Higashino, "Design and Architecture of Cloud-based Mobile Phone Sensing Middleware", *Proceedings of the Second Symposium on Network Cloud Computing and Applications (NCCA 2012)*, 102–109, London, UK (December 2012).

3. Shunsuke Mori, Takaaki Umedu, Akihito Hiromori, Hirozumi Yamaguchi, and Teruo Higashino, "Data-Centric Programming Environment for Cooperative Applications in WSN", *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, Ghent, Belgium, (in press)

Abstract

Recently, wireless sensor networks (WSNs) are expected as important techniques to monitor real world phenomena. On the other hand, mobile phone sensing techniques which use mobile phones such as smartphones as sensing devices are also prospective because mobile phones such as smartphones are in widespread used recently. For collecting data from the real world, there are several researches on WSN and mobile phone sensing applications such as collecting activities of people and environmental information. However, it is difficult to implement collaborative behavior of entire sensing systems by integration of low-level implementation for node programs. Therefore, in WSN system development, if developers design and give an abstract behavior of entire system, it is desired that programs of node behavior in low-level code for each platform and codes for simulators can be obtained automatically. In addition, because both of WSNs and mobile phone sensing become popular recently, both types of sensing should be supported. Thus, to support sensing system development comprehensively, we propose development and management support methods such as (i) a development support environment for network-level programming and performance evaluation, (ii) a system for node specification generation from application-level requirements for cooperative sensing of sensor node groups, (iii) a middleware to support cloud-based mobile phone sensing based on application-level queries.

At first, we design D-sense: a development support environment for network-level programming and performance evaluation. In order to support as many types of protocols as possible, D-sense offers algorithm-level APIs for network-layer level processes. The APIs are derived based on property analysis of existing typical protocols. D-sense also provides two types of translators. A translator for D-sense API expands the embedded APIs that are implemented as macros

into sensor node programming language (e.g. NesC) implementations automatically. A translator for sensor node programming language which translates into codes for simulator (e.g. QualNet). By these translators, a description with APIs are translated into both codes for sensor node and those for simulator. Therefore, developers can evaluate the performance of WSN protocols through both simulation and real environment only by specifying them as algorithm-level sensor node programming language descriptions with APIs provided by D-sense.

Secondly, we propose a methodology to support design and development of collaborative WSN applications by generation of node specifications from application-level requirements. The approach provides a language to specify the high-level behavior of applications which is given as a program specifying time, location and network-based constraints (conditions) on event occurrences and their processing which are carried out by collaborative nodes in WSN. This approach enables to program WSN without referring to the real deployment of sensor nodes. For example, if a developer gives a specification such as "obtain sensing information every 10 minutes from any 10 nodes in a certain area", the system derives a node program which contains modules such as a module to examine whether the node is in the area and a protocol module for sensing data uploading. This method automates design and implementation of complex co-operation protocols from this developer-friendly form of behavior specifications.

Thirdly, we propose a middleware to support cloud-based mobile phone sensing. Each mobile phone works as a sensor node and a cloud server manages sensing data and real-time location of mobile phones because they have mobility and, thus, the network topology always varies. Therefore, we design a middleware to support easy operation of mobile phone sensing. The middleware consists of applications on mobile phones and the server-side module. The middleware enables to give queries specified by real-world conditions such as time, location, and sensor data on event occurrences and their processing. It commands mobile phones to execute sensing according to given queries and manages real-time location of mobile phones to change assigning of sensing tasks when they leave or join the sensing.

These development and management support methods for sensor networks and mobile phone sensing help the sensing system development comprehensively.

Our methods abstract details of node program codes and supports performance evaluation. Especially, we can execute sensing by giving requirements which contain attributes of nodes such as time, location, topology, and sensor data for WSN and mobile phone sensing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, wireless sensor networks (WSNs) are expected as important techniques to monitor real world phenomena. For collecting data from the real world, there are several researches on WSN applications for such as weather observation, volcano monitoring [1], flooding-detection [2], intruder detection [3], and traffic monitoring [4]. Also, several systems are designed as sense and react applications such as traffic control [5] and air conditioning control [6]. In these applications, WSN is considered as a key platform for sensing and it will be more significant for next generation affluent and ubiquitous life and society.

For these several applications, there are various requirements due to heterogeneity of architecture, network scale and applications. New protocols for WSNs are often developed or existing protocols are tuned accordingly. Thus many protocols have been designed with different design goals [7, 8, 9, 10, 11, 12, 13, 14]. In addition, wireless sensor nodes are expected to be more energy-efficient and powerful in near future, and accordingly they can be more intelligent and cooperative to reduce traffic volume and delay, which are caused by gathering all the sensor readings onto cloud servers.

On the other hand, recently mobile phones such as smartphones are in widespread used, which have high functionality and multiple sensors. Therefore they can be used as sensing devices, using much richer storage space and processing power than networked sensor nodes in WSNs, which are cheaper and simpler for massive deployment. Since their features enable to sense many types of data at various location wherever human can visit, useful information such as crowd of walking people, air condition and pollution in human-living area and

real-time public transportation information can be obtained if a large number of mobile phones can participate in sensing activities.

However, there are problems in the WSN development such as effort for node program development such as low-level implementation and experiments in both real and simulated environments, and that for designing node behavior specification to achieve collaborative behavior. There are also problems in sensing by mobile phones such as effort for management real-time location of mobile phones.

In WSN protocol development as sensor node programs, protocol designers and developers face with typical problems which have been experienced in designing distributed systems. Even though the developers wish to concentrate on abstract behavior of protocols, they at last need to write target-dependent low-level codes. Then they carry out performance analysis and validation in simulated networks or real environments. However, additional efforts may be required to simulate the implementation of a target protocol using network simulators, since in most cases such simulation code is not compatible with the corresponding real code. Also experiments in real environments require configuration of many sensor nodes, to log their behavior, and to manipulate them to validate (debug) the implementation. Obviously all of these tasks are really hard and complex.

On the other hand, each node should act to accomplish collaborative behavior in fully decentralized, homogeneous environment for more efficient and powerful sensing. However, most of the current WSN systems and architecture assume simple, limited capability of sensor nodes. Therefore, the main task of those nodes is assumed to send the sensor readings to a gateway which is connected to the back-end cloud servers. This does not scale as cyber-physical computation becomes more essential and the amount of sensor readings accordingly becomes larger. Thus, each node should act to accomplish collaborative behavior in fully decentralized, homogeneous environments. In particular, WSNs should sometimes act as a part of computing modules as well as data collection and delivery, where sensor readings are processed and routed among sensor nodes to enable local and collaborative event processing. However, the implementation of such collaboration requires the designers to make enormous efforts since writing codes of collaborative sensor nodes in a node-centric way,

while keeping the global, data-centric behavior in mind, is extremely a hard task.

In mobile phone sensing, it is not an easy task to manage, organize and control a large number of mobile phones and a large volume of sensor readings to accomplish a given task. Although a cloud-based solution is a reasonable option to store data, we still need software-support to accomplish such complex tasks that involve particular mobile phones at particular time and locations and to have those mobile phones under control. An example sensing scenario is a real-time public transportation location system. Bus passengers at a bus stop may want to know real-time location of the bus, and it can be estimated by the collective GPS traces of some passengers on the bus. To implement this, we need to identify mobile phone users on the bus by finding a set of GPS traces moving together along the bus route and stopping at bus stops. A naive approach is to collect all the traces from all the users, which is too unrealistic due to privacy concern. Therefore, we need a mechanism to send a request including time and location conditions toward mobile users to ask the corresponding users (i.e. bus passengers) to participate in this collaborative task and provide their GPS traces. However, few approach has been considered to achieve this requirement.

In this thesis, we propose development and management support methods for sensor networks to support sensing system development comprehensively according to the following three themes: (i) a development support environment for network-level programming and performance evaluation, (ii) a system for node specification generation from application-level requirement for cooperative sensing of sensor node groups, (iii) a middleware to achieve mobile phone sensing with servers based on application-level queries. The development support environment abstracts details of node program codes and enables developers to implement network-level node programs such as routing protocols without considering difference of platforms. It also supports performance evaluation of node programs on real devices and simulators by the code sharing mechanism. The system for node specification generation derives a behavior specification of each node on WSNs from a given application-level requirement based on node groups which are defined by the attributes of nodes such as location, topology, and sensor data. The middleware for mobile phone sensing enables to achieve sensing using mobile phones by giving requirements based on sensor node group

in the same concept with (ii). The cooperative approach in WSN cannot be applied to mobile phone sensing because they are time- and location- dependent such as nodes which can sense a certain area changes as time advances. Thus, we design the middleware for mobile phone sensing by cooperation among the server and each mobile phone. The middleware achieves mobile phone sensing in hiding detail information of each mobile phone such as its location, mobility, and ID.

First, we discuss the support of node program development and node management in WSNs. At first, we design the core of D-sense: a method to support protocol development, such as node program implementation and evaluation in WSNs. D-sense mainly assumes NesC on TinyOS as the target language and experiments have been carried out on Mica Motes accordingly. For other languages such as C or Java, D-sense's design concept can be applied to support algorithm design and performance evaluation. we assume QualNet [15] simulator for simulation of wireless communication. The advantages of D-sense are three-fold. First, D-sense offers algorithm-level APIs for network-layer level processes which are derived by classifying and studying existing protocols. Since those APIs are written in NesC, the developers can design similar protocols directly using the NesC language. Secondly, it enables seamless integration of simulated and real sensor networks. To accomplish this, we provide a translator from NesC codes into QualNet application codes. Also the physically sensed events and sensor node status observed in real environment are made available in the simulator. These capabilities increase repeatability and fidelity of experiments. Thirdly, monitoring and run-time manipulation of sensor node behavior is possible. We will later show how this functionality can powerfully support developers in test and maintenance of WSN protocols.

Secondly, we discuss the support design and development of sensing systems with multiple node collaboration in WSNs. We propose a methodology to support design and development of collaborative WSN applications. The approach provides a language to specify the high-level behavior of applications without referring to the real deployment of sensor nodes, and an algorithm to automatically translate the given application specification into a platform-dependent program code of each sensor node. We provide a set of event sensing and communication primitives to achieve the given specification in WSN.

The application behavior may include time, location and network-based constraints (conditions) on event occurrences and their processing, and the description is independent of the physical placement of sensor nodes. We provide a concept that hides the details of wireless sensor network configuration, communication and processing inside the network but all the event occurrences are visible to the virtual node. In this architecture, the specification is given as a program on this node specifying pre- and post-conditions of events which are carried out by collaborative nodes in WSN. The translation algorithm automates design and implementation of complex cooperation protocols from this developer-friendly form of behavior specifications.

Thirdly, we discuss about the support for cooperative sensing in mobile phone networks. In this thesis, we propose a middleware to support mobile phone cooperative sensing with a cloud server. Since we have designed in the second theme to support design and development of collaborative WSN applications, we use the basic high-level specification language specification part to describe the behavior of whole sensing system. However, it is very different from the second theme in terms of the target architecture. Mobile phones have mobility and, thus, the network topology always varies and it is difficult to manage it. Thus, we need to tackle (i) cloud-server architecture and (ii) time and mobility into consideration, while the second theme assumes homogeneous, decentralized architecture without centralized servers. The middleware to achieve the mobile phone cooperative sensing consists of applications on mobile phones and the server-side module. Each mobile phone and the server communicate through WAN (e.g. 3G), and even two mobile phones through short-range communication such as Bluetooth or WiFi-direct.

Our middleware automatically translates the given sensing query into *server-side queries* which need to involve multiple mobile phones and *phone-side queries* which are executed by single mobile phones. Based on these queries, each mobile phone monitors and reports some conditions such as sensing data, time, and location, and the server manages mobile phones and assigns sensing task to them based on their reports. We provide a concept that hides the details of network configuration, communication and processing inside the network but all the event occurrences are visible. The sensing query contains time, location and network-based constraints (conditions) and their processing. The process

to achieve the given sensing query is very complex since it requires cooperation among mobile phones and servers. Thus, our method hides the physical placement of mobile phones and enables to execute cooperative sensing specified by abstract query descriptions. The proposed method reduces the effort to design and implementation of complex cooperation protocols by this developer-friendly form of behavior specifications format.

We provide a set of event sensing and communication primitives to achieve the given sensing query in the networks. Especially, since the proposed method is extended for mobile phone sensing, we have designed interface and mechanisms to handle mobility and human-mediate processes. Mobility predicates enables to handle mobility conditions about velocities, trajectories and so on. Opt-in predicates enables to human-mediate sensing to ask owners to work for sensing. For example, an owner of mobile phone is required to take a video from the opt-in interface and he takes the video if he agrees with it.

The following simple crowd sensing example helps to understand the concept; each mobile phone sends beacon to each other and thus can detect neighbors. When a crowded situation is detected from the number of neighbors, the system reacts and starts sampling the neighbor count of the surroundings. Based on the sample readings, the system predicts the crowded area and informs to users. This system requires mobile phones collaboration to obtain samples from appropriate location at required intervals.The proposed scheme allows us to write the system in a simple form that consists of three steps, (i) start sampling on detection with required density and intervals, (ii) crowd prediction on obtaining enough samples and (iii) notification, without being aware of physical configuration of mobile phones and the server. We have shown some examples of mobile phone sensing system by our proposed method to show its usefulness. We have also demonstrated the performance of our proposed method in terms of successful data collection and generated packet to validate the quality of processing the given sensing query.

This thesis is organized as follows. In Section 2, we address the related work and show the features of our methods. In Section 3, we describe the functions of D-sense that support node-centric design of WSN and show example implementation of the existing WSN protocols by using the D-sense design APIs. In Section 4, we propose a methodology to support design and development

of collaborative WSN applications. In Section 5, we propose a middleware to support mobile phone cooperative sensing with a cloud server. Section 6 concludes this thesis.

Chapter 2

Related Work

2.1 Sensor Network Development Support

WSN is an important technique to remote environment monitoring and target tracking and reduction of effort and difficulty of development is important. Therefore, there are several researches to support development, management, and operation of sensor networks [16].

Since it is not easy to develop, install and manage program codes over a number of sensor nodes, several testbeds have been proposed to support those activities [17, 18, 19]. Large-scale testbeds such as MoteLab [20] and CitySense [21] usually provide management functions like online distribution of execution codes to mitigate maintenance costs. D-sense differs from them since it is aimed at comprehensive support of design, development and performance analysis.

WSNs have been used for a variety of applications such as indoor/outdoor environmental monitoring, health and wellness monitoring and object/human tracking [22, 23, 24]. Thus, WSN development supports should consider that network scales are very different, *e.g.* TWIST [25] focuses on indoor environment while Kansei testbed [26] has an unique feature supporting simulated nodes to deal with city-scale networks.

There are several approaches using mobile agent to assign and execute sensing tasks in WSNs. Agilla [27] is a mobile agent system for WSNs. An Agilla agent is similar to a virtual machine and migrates across nodes for sensing tasks. SensorWare [28] is also a mobile agent system. Unlike Agilla, the agents accomplish coordination by using direct communication instead of sharing memory

spaces.

In order to guarantee the success of the sensor network inthe real environment, several diagnostic and debugging systems are proposed. They enable measure and monitor the sensor networks and supports the development and operation. Sympathy [29] is a diagnosis tool for detecting and debugging failures in sensor networks by monitoring metrics such as connectiity, data flow, node's neighbor and next hops. In Sympathy, the sink/base station runs and identifies the type of failure and reports it to the user by analyzing received metrics. The work in [30] provides analyzing of data packet delivery in a sensor network. The system monitors physical and MAC layers on real devices in several different environment settings.

There are several methods to support WSN development by various approaches. Spatial Programming [31] system is based on a logical addressing scheme which enables to access nodes by spatial references. Its run-time system provides the mapping from spatial references to the physical nodes. Generic Role Assignment [32] is a feasible tool for the development of sensor network applications by providing a declarative role specification language and distributed algorithm for dynamic role assignment. TeenyLIME [33] is a middleware whose foundation is the notion of distributed tuple space [34] for data sharing in sense-and-react WSN application. SINA [35] is a sensor network information architecture which overcomes the limitation of easy integration of custom data operators by using SQL-like language SQTL which enables to the injection of arbitrary code in to the network. Market-based programming [36] is a framework to achive globally efficient behavior under dynamic conditions. To meet system wide goals of lifetime, accuracy, or latency based on the requirement. Sensor nodes act as self-interested agents that operate in virtual market and consider execution costs as *prices*.

In our approach, we propose support methods for sensor node programming and performance evaluation. Our methods support sensing system development comprehensively and it reduce complexity and effort for implementation. This approach enables developers to concentrate on designing algorithms and application-level specifications.

2.2 WSN Programming Support

There are several approaches to support sensor network programming as reported in [37]. TinyDB [38] and COUGAR [39] support designing query processing in sensor networks. They provide SQL-like APIs to implement event acquisition and search processes. MATE [40] also provides APIs for more generic purposes, but only low-level APIs like sensing events, pushing data to stack or sending data are designed. Meanwhile, we attempt to help high-level design of more generic protocols including geographic/random-based routing and data fusion/diffusion by extracting their typical behavior. This appropriately hides both distributed and low-level behavior so that developers concentrate on algorithm description. For example, geographic routing protocols like GPSR which employ greedy forwarding strategy need a series of the following atomic actions at each node; (i) obtaining positions of neighbors, (ii) computing distances between the node and the neighbors and between the neighbors and the destination, (iii) finding the neighbor which is closer to the destination than the node and is the closest to the destination among the other neighbors, and (iv) sending a packet. D-sense defines an API for each atomic action, and also provides a single API for a series of these actions by using those atomic APIs.

In summary, as far as we know, no environment has been provided that comprehensively supports algorithm design, low-level implementation, seamless use of simulator and real terminals, and online debugging/monitoring in real environment.

There are some researches supporting node programming of WSNs. Abstract Regions [41] provide abstract node group definition schemes based on connectivity and locations. Developers can design applications by hiding low-level communications, data sharing, and aggregations. Data Space [42] proposes a method of query multicast and data aggregation. By assigning an address to each area with some nodes, it can operate query procedure of a large number of nodes without accessing each individual node. Object-based distributed middleware EnviroTrack[43] provides many useful interfaces. They contain methods which enable operating a large number of nodes having similar sensing data as logical groups. In Ref. [44], ideas of data-centric design of WSNs without considering behaviors of each individual sensor node are presented. Especially, a data-centric design support framework PIECES [45] is proposed. However,

they do not establish languages and distributed behaviors. Ref. [46] proposes script functions of wireless networks. In the proposed script, an application can be specified by sensing, communication and procedure of data instead of actual node behavior. Therefore, by using the script, we can easily specify applications for such environment where positions and connectivity of many nodes are changed dynamically. Kairos [47] provides functions, such as management of node IDs, group construction of neighboring nodes, and getting data from a designated node, and hides them from programs. This enables specifying behaviors of overall WSNs. Pleiades [48] extends Kairos' programming model by allowing a program to be partitioned into independent execution unit which may run on different node and move among nodes.

In addition, there are various WSN programming models. Abstract Task Graph (ATaG) [49] is a programming framework providing a mixed declarative-imperative approach for sense-and-react applications. Flask [50] provides a hybrid approach with a programming model for network-level programs with a data-flow language and node-level program to compose dataflow graphs. FACTS [51] is a middleware which provides a rule-based programming model which works by exchange information called *facts* for data specification, execution trigger, and data exchanging.

RuleCaster [52] provides a programming model with *rules*. Each rule specifies state transitions in each region based on sensor data and consists of the condition for the rule to fire and the action to perform in datalog-like languages. SnBench [53] is a programming framework of multi-user sensor networks and enables to specify application processing with loops and assignments to local variables by a central entity managing their current status.

My approach is similar to them in a meaning that it is based on construction of sensor node groups. However, I provide application requirement description language which enables abstract description, such as conditions of node group construction, and sequential data procedure of the groups. This language has high description capability like a high-level programming language and a service description language. For example, we can define such a behavior; "if there is a node set to which more than 10 nodes belong and their average measurement of temperature is above a certain level, neighbor nodes within 2 hops from of the first group compute their average measurement of temperature".

Furthermore, there are some methods to assign operations to each sensor node in decentralized environment. Ref. [54] proposes Wireless Sensor and Actor Networks (WSANs), which are composed of a low-cost, low-power, multi-functional nodes (called sensors) and nodes which have better processing capabilities, higher transmission powers, and longer battery life (called actors). WSANs are based on event-driven clustering manner. When cluster formation is triggered by an event, sensors send their data to the actors. Based on the event features and the positions of the neighboring actors, the event information is collected to the optimal actor nodes. In this way, energy resources are better utilized, since clusters are formed only when necessary. In paper [55], a method to solve adaptive and decentralized operator placement problems for query processing in WSNs is presented. The position of the operator in query processing, such as aggregate, correlate or filter data streams, is decided based on the rate at which data is produced by the operator and sources, as well as on the path length between the sources, the operators and the sink. This method realizes reducing the data traffic.

Parts of our method of automated generation of sensor node program based on monitoring requirement description take similar approaches to these methods, such as event-driven node group formation and leader node election. However, it enables to describe repeating formation and a process where formation of a group becomes a trigger and invokes another group formation and realizes complex operation of monitoring.

2.3 Cooperative Sensing Development Support

Since it is not easy to develop, install and manage program codes over a number of sensor nodes, several testbeds have been proposed to support those activities [56, 57, 25, 26, 58]. By those support environments, the developers can test their protocols, algorithms and network applications by configuring the environment settings, running the applications and monitoring the results in the testbeds. Some of them have unique features, for example, Dunkels et. al [58] propose a run-time dynamic linking in WSNs so that the developers can change a part of programming codes dynamically.

Some toolsets can diagnose applications for detecting and debugging failures in WSNs [59, 60]. Sympathy [59] is designed for data collection to a central-

ized server, and can monitor and diagnose network status such as connectivity, neighbor nodes and forwarding node status as well as data traffic. Khan, et al. [60] proposes a framework to log run-time status of the system for offline analysis. Quanto [61] can measure and analyze energy consumption on wireless sensor nodes. Although these testbeds and toolsets are quite useful for developers, they still need the developers' effort to fill the gap between high-level specifications and low level codes.

On the other hand, several approaches have been presented so far that support entire process of design and development [62, 63, 64]. Woehrle et al. [62] have proposed a procedure for systematic and strategic testing of target applications to verify robustness and reliability of the applications. Liu et.al [63] have proposed a method to break a given single program down into several pieces that are executed by multiple nodes in ad-hoc networks. MacroLab [64] can also derive distributed codes from a given single program, and the developers can concentrate on designing policies to collect sensor readings and manipulate them. As for different approaches, Ref. [65] has designed a framework in which a task mapping problem can be abstracted to mathematical formulations and tasks generated from the formulations are mapped to sensor nodes. However, the above approaches do not provide the concept of design support for cooperative event processing with time-, location- or network-dependent conditions.

There several approaches to support macroprogramming for sensor networks. Regiment [66] provides declarative macroprogramming system which enables to perform aggregation over a region. DNS [67] is also a declarative sensor network platform for data management and network design. Semantic streams [68] also provides a platform for declarative query over semantic interpretations of sensor data.

In this context, the most relevant approaches with ours are Refs. [69, 70, 71]. Hood [69] privide a macroprogramming language to specify the behavior of the entire network or a group of nodes in physical policy and Logical Neighborhoods [70] provides that in logical policy. Virtual Node [72] is an extension of Logical Neighborhoods which abstracts subsets of nodes as a single logical node. In particular, [71] proposes a set based programming approach where requirement is given by a set of nodes, a set of sensor values and so on. However, the most significant difference is that [71] basically adopts a node-centric view of

programming, while we allow a node-independent approach where a specification can be fully-independent of nodes and networks including neighbors and sink nodes (our scheme allows higher abstraction in other words). Cooperation among nodes to implement such a specification is more complex and challenging, *e.g.* cluster heads should appropriately be chosen to collect and process sensor data if necessary, networks should dynamically be built and sensor data should be routed efficiently in a fully decentralized environment. We believe this is the first approach to consider such highly-abstracted specifications and provide cooperative, cost-effective solutions to achieve the given requirement in WSN.

The papers shown by the reviewer are categorized in domain-specific framework that is development framework for particular needs. [73] and [74] provide several APIs and components for applications in wireless body sensor networks and our approach is different in terms of target networks and applications.

2.4 Mobile Phone Sensing Support

There are some approaches of mobile phone sensing for several purposes. For example, Ear-phone [75] proposes design of a system for noise mapping and a method to recover the noise map from incomplete and random samples obtaind by smartphones. CSN [76] is a classification system for human activity recognition by providing a unique classifier tuned for each user. The system exploits crowd-sourcing to extract inter-person similarity. However, these researches are designed to support mobile phone sensing for particular purposes.

Some approaches are intended to support development of mobile phone sensing applications so that developers can develop such applications easier. In [77], the design, impementation, evaluation, and user experiences of the CenceMe application, which is a personal sensing system that enables members of social networks to share their sensing presense. SoundSense [78] is a scalable framework for modeling sound events on mobile phones (e.g., music, voice), PEIR [79] is a participatory sensing application that uses location data sam- pled from everyday mobile phones to calculate personalized estimates of environmental impact and exposure. These applications are works on each single node. Kobe [80] is a tool that aids mobile classifier development and provides a SQL-like interface for sensor data classification which can be used for mobile applications

development.

EEMSS [81] is an energy efficient mobile sensing system and provides a hierarchical sensor management scheme that specifies a particular sensing criteria and each user state in an XML-format description. Ref [82] proposes a cloud-based integrated framework for mobile phone sensing. This framework is designed as a part of cloud infrastructure, which coordinates a large number of mobile phone users and applications. Medusa [83] is a novel programming framework for crowd-sensing that manages not only computer resources but also human resources by auditing user acknowledgement and giving monetary incentives. CoMon [84] is a cooperative ambience monitoring platform which reduces energy consumption for sensing by sharing sensor data among nearby mobile users. This platform

PhoneGap [85] is is an open source solution for building cross-platform mobile apps with standards-based Web technologies like HTML, JavaScript, CSS.

PRISM [86] proposes a platform for collecting sensor data from a large number of mobile phones on specified location. *Code in the air* [87] is a platform for developing mobile crowdsourcing applications. It have explored tasking smartphones crowds and provide complex data processing primitives and profile-based compile time partitioning. The system enables developers to program mobile applications in a single source and works on multiple platforms. Our middleware enables to specify more complex sensing which considering relation amoung multiple node groups.

Several researches propose systems providing conseptual description for collaborative sensing. Movi [88] is a system for video documentation which collavoratively senses the ambience through multiple mobile phones and captures social moments worth recording and the sensing is triggered by specified events. Darwin Phones [89] is also a system for collaborative sensing and provides automated approach to updating sensor data models over time for the variability in sensing conditions. [90] video documentation systems which considers energy-delay tradeoffs.

In addition, some researches also consider about contribution for participants such as incentives and worker mediation. Nericell [91] proposes a system for road-bump monitoring and Micro-Blog [92] is a system which allows smartphone-equipped users to generate and share geotagged multimedia called

microblogs. Kitokito [93] is a participatory sensing system which allows participants to easily create small sensing tasks. This system is based on server-client architecture and assign tasks such that take a geo-tagged picture to appropriate participants based on their location. AnonySense [94] is an infrastructure for anonymous tasking and reporting and adopts a polling model for task distribution. This approach does not reveal the node's location to the infrasructure. Bubble-Sensing [95] allows to assign sensing task to specific physical locations to sense interest regions and the tasks are broadcasted by local and backend communications.

Compared with the above approaches, our contribution is to provide a middleware that supports higher level abstraction for mobile phone sensing. Query developers can specify time-, location- and topology-based conditions to choose a group of mobile node that should participate in the mobile phone sensing task. The developers are not necessary to be aware of physical locations of mobile phones and the middleware can find such a group that accomplishes the given query. In this context, we believe this has a novel concept of supporting distributed query executions.

Chapter 3

D-sense : A Design Support Environment for Wireless Sensor Networks

3.1 Introduction

In this chapter, we design and develop D-sense, an integrated development environment to support protocol development in WSNs efficiently.

D-sense mainly assumes NesC on TinyOS as the target language and experiments have been carried out on Mica Motes accordingly. For other languages such as C or Java, D-sense's design concept can be applied to support algorithm design and performance evaluation. We assume QualNet [15] simulator for simulation of wireless communication. The advantages of D-sense are threefold. First, D-sense offers algorithm-level APIs which are derived by classifying and studying existing protocols. Since those APIs are written in NesC, the developers can design similar protocols directly using the NesC language. Secondly, it enables seamless integration of simulated and real sensor networks. To accomplish this, We provide a translator from NesC codes into QualNet application codes. Also the physically sensed events and sensor node status observed in real environment are made available in the simulator. These capabilities increase repeatability and fidelity of experiments. Thirdly, monitoring and runtime manipulation of sensor node behavior is possible. We will later show how this functionality can powerfully support developers in test and maintenance of

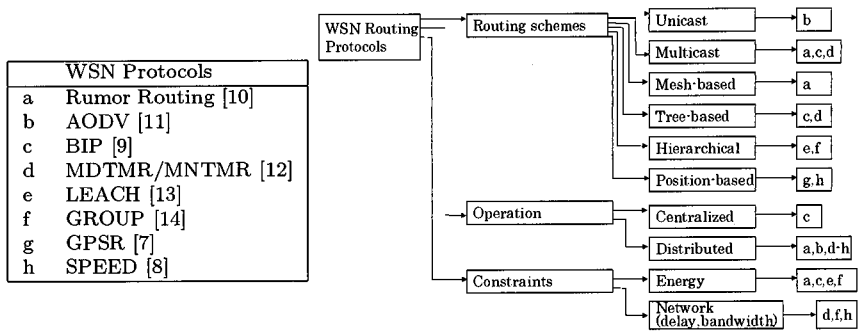


Figure 3.1: Classification of WSN Protocols

WSN protocols.

Using the D-sense APIs, We have implemented GPSR [7], SPEED [8], BIP [9] and Rumor Routing [10]. In particular, We have evaluated the performance of SPEED in both real and simulated networks and compared the results with Ref. [8] to validate the D-sense implementation. It is also confirmed how D-sense contributes to alleviate the development cost.

This chapter is organized as follows. In Section 3.2, We describe the functions of D-sense that support node-centric design of WSN and show example implementation of the existing WSN protocols by using the D-sense design APIs. Section 3.5 shows the evaluation of reducing implementation effort. Section 3.6 concludes this chapter.

3.2 Functions of D-sense

Developers can evaluate the performance of WSN protocols through both simulation and real environment only by specifying them as algorithm-level NesC descriptions with D-sense design APIs. In addition, automated generation function of sensor nodes' application (discussed in chapter 4) supports monitoring and managing of WSNs. The functions D-sense provides are described in the following.

Design Support

One of the most important features of D-sense is high-level design support. Using the *D-sense design APIs*, developers can give algorithm-level NesC de-

scriptions. Then the *D-sense design API translator* takes them as inputs, and expands the embedded APIs that are implemented as macros into pure NesC implementation automatically. In order to support as many types of protocols as possible, the D-sense design APIs are developed based on property analysis of existing typical protocols. These protocols are classified by the criteria which are inspired from Ref. [96]. A classification example by these criteria is given in Figure 3.1. For example, GPSR ("g" in Figure 3.1) is a position-based routing method and is used in GHT [97] or some other methods that employ position-based event accumulation and search mechanisms. In implementing this protocol, we may use the APIs for "position-based routing" and "store and search application". Similarly, some other known protocols like Rumor Routing [10], AODV [11], BIP [9], MDTMR [12], LEACH [13], GROUP [14] and SPEED [8] are classified in the figure.

For each type in the classification, we provide type dependent APIs, and also provide generic APIs which are commonly used in all the types. Furthermore, we design more functional APIs that are realized by using these APIs. For example, since obtaining the IDs of one-hop neighbors is carried out in many protocols, it is designed as a generic API. On the other hand, obtaining the position of a designated node is obviously used by location-aware protocols and applications only. Obtaining the ID of the node which has the maximum residual battery in a node cluster is required in some cluster-based protocols, and the corresponding API is designed as a functional API. The details of the APIs and their implementation are shown in our web site [98].

In Section 3.3, we will exemplify how typical protocols are implemented using these APIs.

Seamless Integration of Simulated and Real networks

D-sense provides functions of supporting seamless integration of simulated and real networks and supports performance evaluation. Integrated functions are (1) sharing the same code between both environments, (2) visualization of performance in real environment, and (3) setting of simulation parameters based on logs of real environment.

The NesC codes derived by the D-sense design API translator can be directly executed on Mica Mote, or can further be translated into codes for QualNet sim-

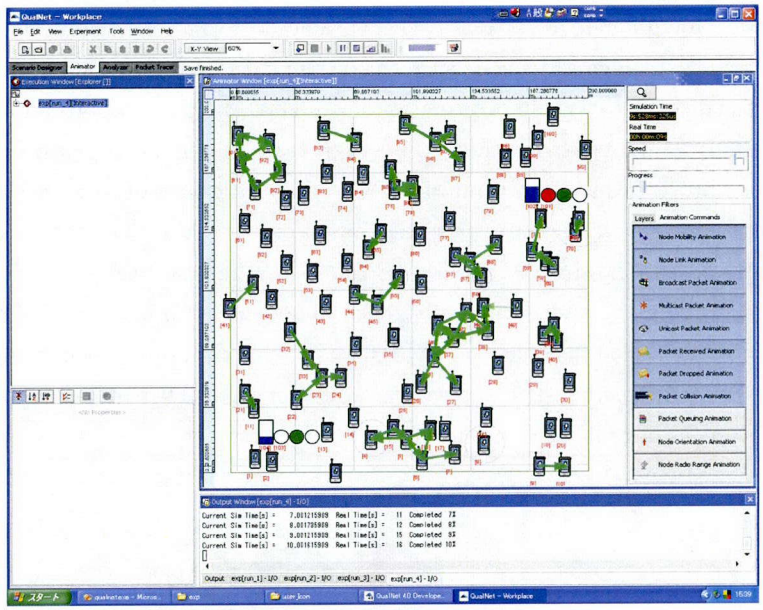


Figure 3.2: A Snapshot from QualNet Simulator (Sensor Node Status is Visualized)

ulator [15] written in C++ by the *D-sense NesC translator*. Also environmental events and sensor node status logged automatically by the D-sense debugging component in real environment and can be animated by the QualNet animator (Figure 3.2) in which we provide special graphics to visualize residual amount of battery and LED status (we assume Mica MOTE here) for more realistic animation.

The function of simulation parameter setting helps making similar environments as real environments based on logged information such as node positions and message receptions in real environment. This function enables simulation to inherit parameter settings and radio connectivity from real environment. This enables performance evaluation in both small-scale real WSNs and large-scale simulated WSNs.

Especially, there are large gaps between simulation results based on isotropic radio range model and anisotropic radio range. To solve this problem, [99] proposes Radio Irregularity Model (RIM), which assigns different rate of atten-

uation for different directions. The parameter degree of irregularity (DOI) is introduced, which is defined as the maximum path loss variation per unit degree of the direction of radio propagation, into RIM. By giving appropriate settings of DOI, RIM can bridge the gap between real and simulated environment. D-sense provides functions to derive DOI value from logs of some nodes in real environment and simulate with RIM based on this value. Derivation of DOI is executed as following: For a node i, the system makes neighbor node set $N(i)$ by listing up the nodes whose distances to node i are similar. Node i broadcast a message and each node in $N(i)$ records values of radio strength when it receives message. Based on the values, we can derive DOI parameter considering differences of radio strength in different directions.

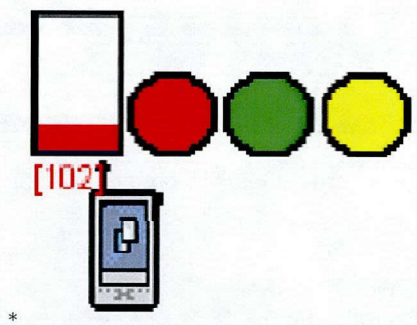


Figure 3.3: Visualization of LED and Battery for Mica MOTE

Monitoring Functions for sensor nodes' program

We explain our powerful support facility for monitoring of WSN programs. At each sensor node, we run a "monitoring agent" to monitor node status in distributed environment by cooperation among agents. Configuration of monitoring agents is based on a method which is discussed in chapter 4. Based on description of monitoring requirement for a target application, a monitoring sub-program is generated and executed with the target applications. The monitoring application can monitor states of sensor nodes and networks. For example, we can define some kind of invalid processes and states, such as overage traffic and deadlock by my monitoring functions, and sensor nodes can detect them as troubles, report about them, and operate to recover.

```
01: get_planar_graph(graph g){
02:    len = get_neighbors(neighbor_IDs, sizeof(neighbor_IDs));
03:    for (i = 0; i < len; i++){
04:       nodeID = neighbor_IDs[i];
05:       for (j = 0; j < len; j++){
06:          nodeID' = neighbor_IDs[j];
07:          if (get_distance(myID,nodeID)
               > max(get_distance(myID,nodeID'),
                     get_distance(nodeID,nodeID')))
08:             g.remove_edge(myID,nodeID);
09: }
```

(a) RNG Generation

```
01: len = get_neighbors(neighbor_IDs, sizeof(neighbor_IDs));
02: forwardID = get_my_ID();
03: for (i = 0; i < len; i++){
04:    nodeID = neighbor_IDs[i];
05:    if(get_distance(nodeID,targetID)
         < get_distance(forwardID,targetID))
06:       forwardID = nodeID;
07: if(forwardID != myID) // forward toward a nearer node
08:    send_unicast_packet(forwardID,packet);
09: else peremeter_mode == true; // perimeter mode (omitted)
```

(b) Greedy Forwarding

Figure 3.4: Example Implementation of GPSR

3.3 Protocol Implementation Examples

In this section, we show example implementation of four existing WSN protocols; GPSR, SPEED, BIP and Rumor Routing by using the D-sense design APIs.

GPSR[7] is a position-based protocol, where each sensor node forwards a packet to the neighbor node nearest to the destination by using a planar graph. Figure 3.4(a) denotes an example implementation of the algorithm to make a Relative Neighborhood Graph (RNG), which is a kind of well-known planar graph. We can see that the algorithm is implemented simply by using APIs, such as listing neighbor nodes ("get_neighbor" in line 02) and getting the distance between nodes ("get_distance" in line 07). Figure 3.4(b) shows an example implementation of routing process of GPSR. In this code, each node lists its neighbor nodes (line 01) and forwards a packet to the node nearest to the destination in the listed nodes (lines 05–09).

SPEED[8] is also a position-based routing protocol. In SPEED protocol,

```
01: SNGF(message){
02:     my_length = get_distance(message->target_ID, myID);
03:     len = get_neighbors(neighborIDs, sizeof(neighborIDs));
04:     num_FS_first = 0; num_FS_Second = 0;
05:     for (i = 0; i < len; i++){
06:         nodeID = neighbor_IDs[i];
07:         diff = my_length
              - get_distance(message->target_ID, nodeID);
08:         if(diff > 0){
09:             if(diff / get_delay(myID,nodeID) > set_point)
10:                 FS_first[num_FS_first++] = nodeID;
11:             else
12:                 FS_second[num_FS_Second++] = nodeID;
13:         }
14:     }
15:     if(num_FS_first > 0){
16:         forwarding_probability = 0;
17:         forwarding_nodeID = FS_first[0];
18:         for(i = 0;i < num_FS_first; i ++){
19:             nodeID = FS_first[i];
20:             fp = get_probability
                  (get_distance(myNodeID,nodeID)
                   ,get_queue_size(nodeID));
21:             if(fp > forwarding_probability){
22:                 forwarding_probability = fp;
23:                 forwarding_nodeID = nodeID;
24:             }
25:         send_unicast_packet(nodeID, message);
26:     }
27:     else // forward toward the nodes in FS_second
28: }
```

Figure 3.5: Example Implementation of SPEED

Stateless Non-deterministic Geographic Forwarding (SNGF) algorithm is used to select a node which is nearer to the destination and handles lighter traffic to forward packets. Figure 3.5 shows an example implementation of SNGF. A node receiving a packet finds nodes that are nearer to the destination than itself (lines 02–07) and classifies them into two groups. If the transmission efficiency of a node is larger than a threshold *set_point*, it is put into group *FS_first*. The other nodes are put into group *FS_second* (lines 08–13) Then a node is selected from the group *FS_first* of nodes having better transmission efficiency according to the length of the transmission path and the levels of congestion (lines 15–26). We can see that GPSR and SPEED, both of which are position-based routing protocols, can be implemented by using similar APIs.

BIP [9] is a centralized protocol managing sensor nodes in tree topology, and designed to minimize the total energy consumption of the network. In centralized protocols, information of any nodes and any node pairs may be used for managing network topology and routing packets. Design APIs gather and manage such information and provide them for developers. The right figure shows an example implementation of BIP. At first, the source node is set as the root node of the tree (line 01). Then the tree is constructed by adding the node that can be reached from a node in the tree with minimum radio transmission power one after another (lines 06-14). In this process, if a descendant node is in the radio transmission range of one of its ancestor nodes, relaying packets by the nodes between them is needless. If there is such a pair, the tree is modified such that the ancestor node sends packets to the descendant node directly and intermediate path is removed (lines 17-24).

In centralized protocols, information of any nodes and any node pairs may be used for managing network topology and routing packets. Design APIs gather and manage such information and provide them for developers. Figure 3.6 shows an example implementation of BIP. At first, the source node is set as the root node of the tree (line 01). Then the tree is constructed by adding the node that can be reached from a node in the tree with minimum radio transmission power one after another (lines 06–14). In this process, if a descendant node is in the radio transmission range of one of its ancestor nodes, relaying packets by the nodes between them is needless. If there are such a pair, the tree is modified such that the ancestor node sends packets to the descendant node directly and intermediate path is removed (lines 17–24).

Rumor Routing [10] is a routing protocol based on mesh topology, and is designed for accumulation and search of data. Figure 3.7 shows an example implementation. In Rumor Routing, an agent manages event tables kept in sensor nodes as follows. A node receiving an agent adds information written in the agent to its event table. The information consists of the number of hops to the event *num_hops*, and the direction and the hop count from the node that sends the agent *source_ID* to each event (lines 01–02). At the same time, the node puts information recorded in its event table to the agent and sends it to another node. In this process, a node where agents have not visited long time is selected (lines 03–06). When a node receives a query packet, the node searches

```
01: num = get_tree_nodes(root_node_ID,
                  treeNodes,sizeof(treeNodes));
02: while (num < N){
03:    parentNode = NULL;
04:    childNode = NULL;
05:    energy = MAX_VALUE;
06:    for (i = 0; i < num; i++){
07:       node = treeNodes[i];
08:       min_ID = get_minimum_energy_node(node->ID);
09:       if(energy > get_energy_consumption(
                        node->ID,min_ID)){
10:          parentNode = node;
11:          childNode = get_node(min_ID);
12:          energy = get_energy_consumption(
                        node->ID,min_ID);
13:       }
14:    }
15:    if(parentNode) add_child(parentNode,childNode);
16: }
17: num = get_tree_nodes(root_node_ID,
                  treeNodes,sizeof(treeNodes));
18: for(i = 1; i < N;i++)
19:   for(j = 1; j < N; j++)
20:    nodeI = get_node(i); nodeJ = get_node(j);
21:    if(has_children(nodeI,nodeJ)){
22:      len = get_children(nodeJ,childrenIDs, sizeof(childrenIDs));
23:      if(is_neighbours(nodeI,childrenIDs))
24:            set_parent(childrenIDs,nodeI);
```

Figure 3.6: Example implementation of BIP

the path to the event queried by the packet using its event table (line 10). If the node has the target event information itself, it processes the query, otherwise the node searches a direction to forward it to (lines 11–13). If the node has no information regarding the query at all, the query is forwarded to a node where the query packet has not visited recently (lines 14–15).

3.4 Case Study: A Demonstration of Routing Protocol Evaluation

In order to validate the D-sense implementation and show its usefulness, we show a demonstration to evaluate the performance of the SPEED protocol in simulation and real environment by using D-sense, and compared the performance in the simulation to the performance reported in Ref. [8].

```
01:  on_agent_received(source_ID,agent_packet){
02:     event_table = set_event_distance(
                agent_packet->num_hops,source_ID);
03:     agent_packet = set_event_info(
                agent_packet, event_table);
04:     if(agent_packet->ttl-- > 0)
05:        send_unicast_packet(
             get_not_visited_neighbor(agent_packet),
             agent_packet);
06:  }
07:
08:  on_query_received(source_ID,query_packet){
09:     query_packet->ttl--;
10:     if(get_num_hops(event_table,query_packet->data)==0)
11:        doQuery(query_packet)
12:     else if (get_hops(query_packet->data) > 0)
13:        send_unicast_packet(query_packet,
          get_forwarding_direction(event_table,query_packet))
14:     else
15:        send_unicast_packet(
             get_not_visited_neighbor(query_packet),
             queryPacket)
16:  }
```

Figure 3.7: Example Implementation of Rumor Routing

We used the same scenario as Ref. [8]. This scenario is aimed at testing the congestion avoidance capability of the SPEED protocol. A few nodes are randomly selected from the left side of the terrain and send periodic data to the base station at the right side of the terrain. Each sender generates one CBR flow with 1 packet/second. To create congestion, two randomly chosen nodes in the middle of the terrain create a flow between them at half time of the 150 second experiment. In order to evaluate the congestion avoidance capability under different congestion levels, the rate of this flow is increased by 10 from 0 up to 100 packets/second over several simulations. We have evaluated the delay and loss ratio of the packets to the base station.

We show the experimental environment in Table 3.2 and its topologies in Figure 3.8 and 3.9. Because of the limitation on the number of MOTEs, we evaluated the SPEED protocol with 25 nodes in real environment. To compare the reported performance with the real environmental performance, simulation experiments were also conducted in the same configurations. We adjusted the wireless ranges of MOTEs and simulator according to the network scale. Figure

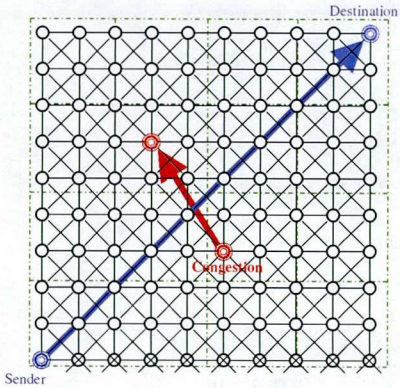


Figure 3.8: Topology of the Experiment with 100 nodes

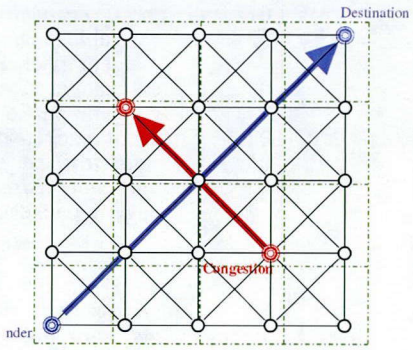


Figure 3.9: Topology of the Experiment with 25 nodes

3.10 shows a snapshot from the experiments in real environment where MOTE terminals were uniformly arranged.

,

Figure 3.11(a) shows the end to end delay. In the experiments with 100 nodes, the performance observed in the simulation well follows the reported performance although small difference is seen around 40 packet/sec congestion. We observed the same level delays in the experiments with 25 nodes as observed in those with 100 nodes. In each congestion level, delays in real environment were smaller than those in simulation.

Figure 3.11(b) shows packet loss ratio (the ratio of packets that failed to reach the base station). In the experiments with 100 nodes, the simulation performance is nearly equal to the reported performance. In the experiments with 25 nodes, the packet loss ratio is greatly higher than that in the experiments with 100 nodes. This is mainly because each node had too few nodes in its neighbor table to avoid the congestion area at the center of the network in the experiments with 25 nodes. In particular the packet loss ratio is much higher in real environment than that in the simulation. In each congestion level, we observed the same level packet loss ratio in real environment as observed in simulation although small difference is seen around 50 packet/sec congestion

As shown in Figure 3.11, compared to the simulation results, we can see small delays and large packet loss ratio in real environmental results. We attribute

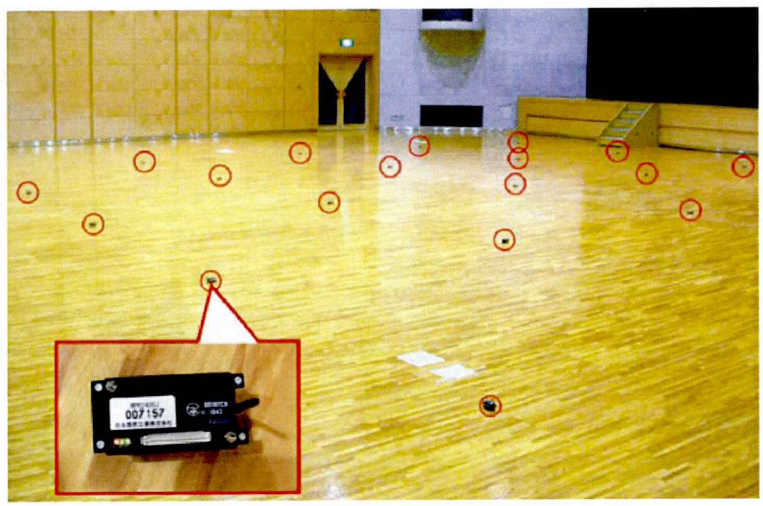


Figure 3.10: Arrangement of MOTEs in Real Environment

these differences to large fluctuation of radio ranges in real environment. In real environment, nodes can receive beacons from further nodes and store the node IDs in its neighbor table. Then, nodes send packets to those further nodes, which have both lower delays and higher probability of packet loss.

From these performance evaluations, we could validate the D-sense implementation. In addition, we could find some real environmental problems and their causes, discuss their solutions, and improve reality of the simulation by considering them. This shows the importance of implementing and evaluating WSN protocols in real environment, and also shows that D-sense well supports these activities.

3.5 Performance Evaluation

3.5.1 Reduction of Implementation Effort

In order to evaluate the effectiveness of implement effort reduction by D-sense Design APIs, we show analysis of reduction in lines of codes and complexity.

We counted the LOC (lines of code) of example codes of SPEED protocol implemented (1) by using the design APIs, (2) in C++ for QualNet simulator and (3) in NesC for MOTE terminals. The lines of code are shown in table 3.3.

Without using APIs, the implementation required more than 1000 lines. On the other hand, by using the APIs, the LOC is decreased to about 200 lines. Therefore, the APIs seem to reduce the effort of protocol implementation.

In detail, the APIs are classified into Functional APIs, which are provided as functions, and Procedural APIs, which are expanded to codes of multiple processes by the API translator.

The Functional APIs provide processes such as storing data to neighbor table, getting data from the tables, and calculating distance between two nodes. These APIs are designed to provide processes which are used frequently for implementation. These APIs reduce efforts for first implementation of functions. Thus, lines of code which are reduced are constant even if they are used in multiple times.

The Procedural APIs provide implementation of procedures for such as sending and receiving packets and hide codes for their procedures such as processes to control MAC and lower layers of sensor devices for each types of communication (e.g. data request and data reply). However, from the APIs with parameters specifying details of the processes such as payload data and destination node of the communication, codes of the communication procedures are generated (the image of translation is shown in Fig.3.12). The APIs absorb differences of code for each platform and reduce complexities which are not essential for algorithms. These APIs reduce lines of code which are in direct proportion to number of communication types at constant rate.

3.5.2 Availability and Correspondence for Various Protocols

On the other hand, APIs for development support should satisfy many factors such as role expressiveness and domain correspondence as commonly described (e.g. [100]). Therefore, we have implemented example codes of position-based routing SPEED [8] and GPSR [7] by using the APIs and evaluated the number of each API to analyze the expressiveness and domain correspondence in some cases. We have also implemented and evaluated that of LAR [101]. LAR is a routing protocol which uses location information to limit the search for a new route to a smaller "request zone". We show the result in Table 3.4.

Generic APIs such as *send_unicast_packet* and *send_broadcast_packet* are used frequently and commonly in each protocol and the API reduces effort for implementation of communication procedures. About other functional APIs, *get_distance* is used commonly in SPEED and GPSR, which select neighbors to send packet based on distances to them, and *is_within_area* is used commonly in GPSR and LAR, which have processes of checking whether a certain position is within a certain area. The result shows that APIs provide role expressiveness to support implementation of general processes such as communication and also have domain correspondence to implement domain-specific processes such as position-based processes. Our future works are evaluating the usefulness of the proposed APIs for various protocols and improving their designs.

3.6 Conclusion

In this chapter, We have designed and developed an integrated environment called D-sense for supporting development of WSNs. D-sense supports protocol design by high-level design APIs. Also it provides seamless collaboration of simulated and real networks for performance evaluation, and a powerful distributed debugging scheme. We have conducted performance evaluation of the SPEED protocol in simulation and real environment to show the effectiveness of D-sense. Our ongoing work includes developing a complete set of design/debug APIs and related tools, and opening them to public domain.

Table 3.1: Example of D-sense Design APIs

Generic APIs	
get_neighbors(IDs[],len_IDs)	Get the IDs of the one hop neighbor nodes
send_unicast_packet(ID,pkt)	Send a packet to the designated node
send_broadcast_packet(pkt)	Broadcast a packet
get_num_hops(ID)	Get the number of hops to a node
Position based protocol APIs	
store_position(ID, position)	Store the position of the designated node to neighbor table
get_position(ID)	Get the position of the designated node
get_distance(ID,ID)	Compute the distance between the designated two nodes
get_nearest_neighbor(ID)	Get ID of the nearest node to the designated node
get_smallest_azimuth_neighbor (ID)	Get ID of the smallest azimuth node to the designated node
is_within_area(c, r)	Return true if the node is within the circle centered at c with radius r
Tree Protocol APIs	
get_ancestor(IDs[],len_IDs)	Get IDs of ancestor nodes
get_downstream_tree (IDs[],len_IDs)	Get downstream tree
evaluate_tree_cost(Tree)	Compute the cost of the tree
Hierarchical protocol APIs	
get_cluster_nodes(IDs[],len_IDs)	Get the IDs of the cluster members
get_neighbor_clusterhead()	Get the cluster head of the neighbor cluster
Collect to BS Application APIs	
get_bs_neighbors(IDs[],len_IDs)	Get neighbours of BS node
get_bs_queue_size()	Get packet queue size of BS node
Store and Search Application APIs	
get_num_hops(Data)	Get the number of hops to the specific data
Multicast APIs	
set_multicast_group(IDs)	set multicast group
send_multicast_packet(Group)	send multicast packet
Energy Constraint Protocol APIs	
get_residual_energy(ID)	Get the residual battery of the designated node
get_transmission_energy()	Get the energy consumption to send a packet
Network Constraint Protocol APIs	
get_delay(ID,ID)	Get the delay between the two nodes
get_packet_loss_rate(ID)	Get the packet loss ratio at the designated node
Functional (Combined) API	
Get the ID of the maximum residual battery node in a cluster	
Get the minimum delay node which has the specific data	
Get the packet loss ratio in the tree	

Table 3.2: Experimental Environment

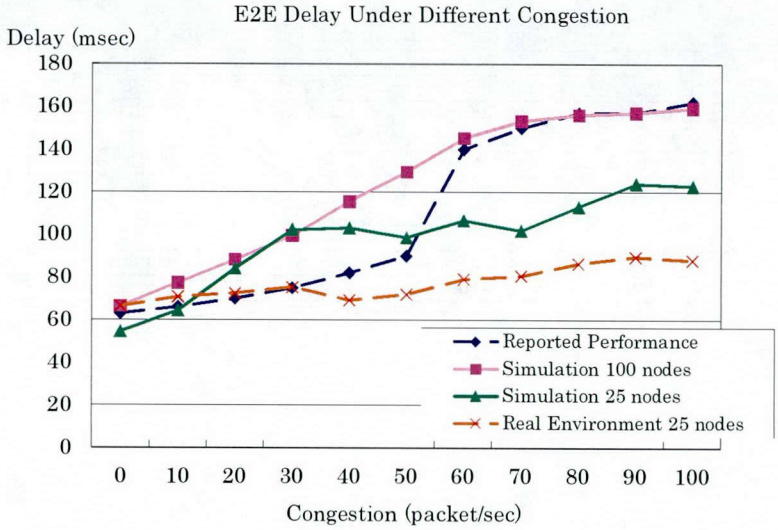
	Reported	Simulation	Real Env.
PHY & MAC	802.11	802.11	802.15.4
Bandwidth	200 Kb/s	200Kb/s, 250Kb/s	250Kb/s
Payload Size	32 Bytes	32 Bytes	32 Bytes
Terrain	(200m,200m)	(200m,200m), (20m,20m)	(20m,20m)
# of Nodes	100	100, 25	25
Node Placement	Uniform	Uniform	Uniform
Radio Range	40m	40m, 8m	8m

Table 3.3: Lines of Code of SPEED

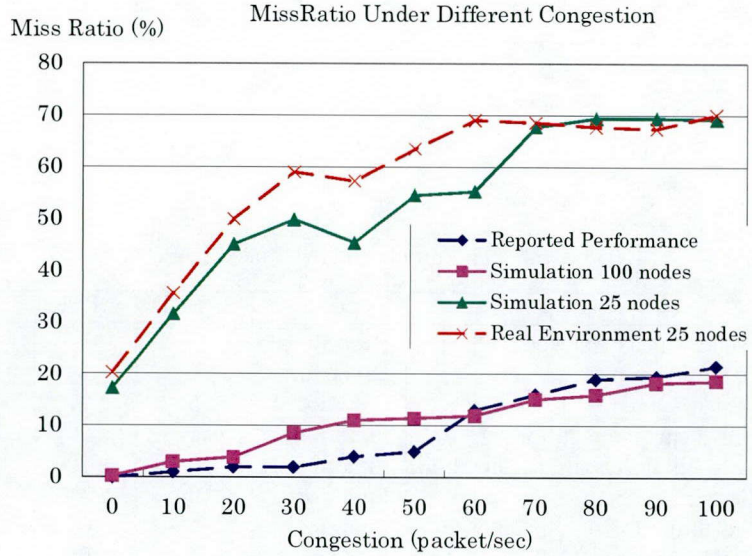
	(1) NesC with Design APIs	(2) C++ (with D-sense modules)	(3) C++	(4) NesC
LOC	221	1058	860	1147

Table 3.4: The number of API used in SPEED

API	Type	SPEED	GPSR	LAR
send_unicast_packet	Procedural	5	3	2
send_broadcast_packet	Procedural	1	1	1
get_distance	Functional	5	2	0
get_nearest_neighbor	Functional	1	2	0
get_smallest_azimuth_neighbor	Functional	0	1	0
is_within_area	Functional	0	1	1
get_queue_size	Functional	1	0	0
get_neighbor	Functional	1	0	0
get_delay	Functional	1	0	0
get_position	Functional	1	1	2



(a) E2E Delay



(b) Packet Loss Ratio

Figure 3.11: Performance of SPEED Protocol

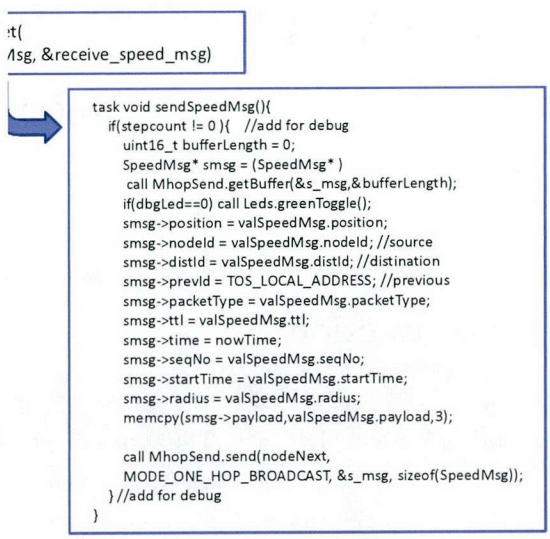


Figure 3.12: The Image of API translation

Chapter 4

Data-Centric Programming Environment for Cooperative Applications in WSN

4.1 Introduction

In this chapter, we propose a methodology to support design and development of collaborative WSN applications. The approach provides a language to specify the high-level behavior of applications without referring to the real deployment of sensor nodes, and an algorithm to automatically translate the given application specification into a platform-dependent program code of each sensor node. We provide a set of event sensing and communication primitives to achieve the given specification in WSN.

The application behavior may include time, location and network-based constraints (conditions) on event occurrences and their processing, and the description is independent of the physical placement of sensor nodes. We provide a concept that hides the details of wireless sensor network configuration, communication and processing inside the network but all the event occurrences are visible to the virtual node. In this architecture, the specification is given as a program on this node specifying pre- and post-conditions of events which are carried out by collaborative nodes in WSN. The translation algorithm automates design and implementation of complex cooperation protocols from this

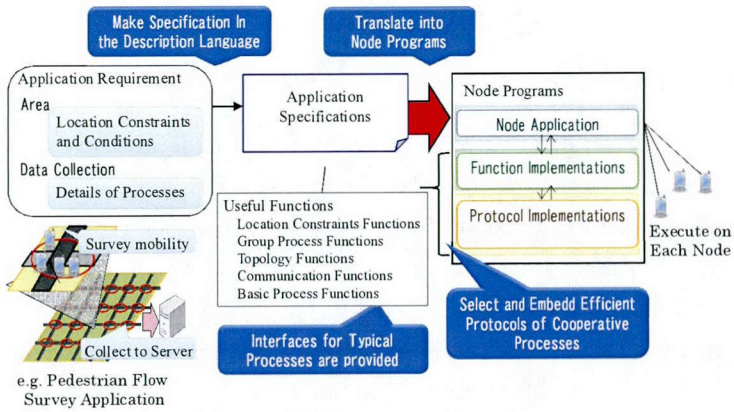


Figure 4.1: Approach Overview

developer-friendly form of behavior specifications.

The following simple fire detector example helps to understand the concept; Each wireless sensor node is equipped with CO (smoke) and thermal sensor, and forms a wireless sensor network with neighboring nodes. When a smoke or fire is detected, the application reacts and starts sampling the air temperature and smoke of the surroundings. Based on the sample readings, the application predicts the potential spread of the smoke and fire, and alerts nearby people appropriately. This application requires sensor node collaboration to obtain samples from appropriate location at required intervals, and needs to determine nodes that take over processing like fire prediction and notification. The proposed scheme allows us to write the application in a simple form that consists of three steps, (i) start sampling on detection with required density and intervals, (ii) smoke and fire prediction on obtaining enough samples and (iii) notification, without being aware of physical configuration of WSN.

We have compared derived codes with given application specifications to quantitatively understand the gap between the two different specification levels. We have also demonstrated the performance of program codes generated by our proposed method in terms of successful data collection ratio, data collection delay and traffic volume to validate the quality of automatically-generated program codes.

4.2 Approach Overview

4.2.1 Outline

Our method can derive a program code of each node for a given specification that describes actions to be taken by a group of nodes and conditions to be examined before the actions. Developers can easily describe applications by specifying such conditions that should be checked by cooperation of nodes. We show a simple but essential example in Fig. 4.2 where a group of nodes that satisfy the following conditions is defined as the first group to detect a fire; (1) all the nodes in the group have detected temperatures higher than $40°C$, (2) they are located in a circle with 100m diameter and (3) the average temperature by 30 or more nodes in the group is higher than 50 $°C$ (we explain these predicates in Section 4.3, but readers may refer to Table 4.1). In addition, in order to alert the approach of fire, the second, larger group of nodes that contains the previous group having the similar center with the previous group (*DetectedFireSpot*) but with larger radius, which is five times of the maximum distance (diameter) between nodes in *DetectedFireSpot* group, is defined. The nodes in the second group (*EstimatedFireSpot*) warn people to escape from the fire. In this way, conditions on geometry, sensing data values and their manipulations can be written in our specification.

However, such a specification is not easy to implement since checking conditions and executing actions need cooperative operations among nodes. For example, in order to check a condition on sensing data, (i) a node group with a leader node needs to be organized, (ii) the sensing data needs to be collected onto the leader node, and (iii) it needs to be checked if the condition is met or not. The action should be executed if the condition is satisfied, or the group is dismissed.

Thus, our method can automatically derive the program of cooperative nodes. This hides the details of node behavior, which are often complex, from the developers. Therefore they can concentrate on application logic.

4.2.2 Code Derivation for WSN nodes

For a given specification described by a set of nodes with pre-conditions and post-actions, we classify the predicates that constitute the condition into two

```
nodegroup DetectedFireSpot
  condition:
      TestEach(temperature, ">40")
      && InFloatCircle(100)
      && AverageSelect(temperature, 30)>50
  action:
      centroid = GetCentroid()
      diameter = GetDiameter()

nodegroup EstimatedFireSpot
  condition:
      InGeoCircle(DetectedFireSpot.centroid,
                  5·DetectedFireSpot.diameter)
  action:
      ExecuteEach("ActivateAlert()");
```

Figure 4.2: Specification of Fire Detection and Alert System

categories, single-node predicates and multi-node predicates. An example of single-node predicate is *TestEach* that checks if variable on each node satisfies a given condition (see *TestEach*(temperature, "> 40") in Fig. 4.2). Meanwhile, both *InFloatCircle(100)* and *AverageSelect*(temperature, 30)>50 are multi-node predicates since they cannot be examined by single nodes. For example, *InFloat-Circle(100)* needs distance calculation for every pair of nodes, meaning that it can be checked only when a group of nodes is given. Considering this fact, we take the following strategy; Firstly, we let each node periodically check single-node predicates, and let the node be a potential constitute of the group if it satisfies the conditions. If a node becomes a potential constitute of the group, it establishes a link with neighboring potential constitutes of the same group if any. This is done by periodic neighbor discovery messages by each potential constitute. These nodes finally form a tree with a *leader node*. The choice of a leader node is simply done by determining link direction (parent-child relation), and the root node can be the leader node. Then the data values to check the multi-node predicates are collected to the leader node, and the node checks if all the multi-node predicates are satisfied or not. If true, those nodes take actions as specified. Moreover, we allow to describe conditions of groups that depend on some other groups. For example, the second group (*EstimatedFireSpot*) in Fig. 4.2) is such a group that refers to the "center" and "diameter" of *Detect-*

edFireSpot as a part of its conditions. In this case, the centroid of coordinates of nodes in *DetectedFireSpot* and the diameter between nodes in it has been calculated by the leader node of *DetectedFireSpot* to prepare for creation of *EstimatedFireSpot*, and the information is broadcast to potential constitutes of *EstimatedFireSpot* (in this case, all the nodes).

In summary, each node needs periodically to check the single-node conditions of each group, and then forms a tree with neighboring nodes which also satisfy the same conditions. During the tree construction process, a leader node has been selected and the leader node collects all the data necessary to check multi-node conditions through the tree. Then it actually checks the conditions and executes the post-actions in cooperation with the nodes on the tree if necessary. During the process, it prepares and calculates the data for the other groups' conditions if any.

4.3 Language and Algorithm Details

4.3.1 Specification Language

A specification consists of two types of profiles, *node profiles* and *nodegroup profiles*.

The *node profiles* define the attributes of sensor nodes. For example, if a WSN consists of wireless sensor nodes and base stations, then we prepare two profiles that correspond to them. In their profiles, local variables (storing sensor data and so on) and methods they hold are defined. We omit example descriptions here because they just consist of definitions of variables and functions.

In *nodegroup profile*, each block of description starts with a keyword **nodegroup** (words highlighted by bold fonts are reserved words hereafter). Conceptually, this corresponds to a group of nodes that cooperatively execute tasks. Developers can define pre-conditions with **condition** keyword and post-actions with **action** keyword. The condition part must be a logical formula using predefined *single-node* and/or *multi-node* predicates, and the action part must be a list of functions (or procedures).

The example specification of a fire detection and alert system shown in Sec.4.2.1 (Fig. 4.2) is an example of formal description. *DetectedFireSpot* and *EstimatedFireSpot* are nodegroup definitions blocks. In *DetectedFireSpot*, three

predicates are speficied in the condition part. *TestEach* is a single-node predicate, and *InFloatCircle* and *AverageSelect* are multi-node predicates. As we explained in the previous section, groups may refer to other groups by directly specifying their group names. For example, *EstimatedFireSpot* group refers to the *DetectedFireSpot* group. Since the node groups defined by *DetectedFireSpot* may not be unique (*i.e.* there may appear multiple groups), *DetectedFireSpot* is assumed to be the reference to the first-generated group in our language definition. The condition of *EstimatedFireSpot* contains a single-node predicate *InGeoCircle* with 2 parameters. In the specification, these values of paramters have been determined in *DetectedFireSpot* group by *GetCentroid* and *GetDiameter* functions.

Tables 4.1 and 4.2 show the list of predicates and functions, respectively. As for the predicate table, we add how the predicates are examined in distributed environment in the last column. *Single* means it can be tested by each node independently (*i.e.* single-node predicates), while *Multi* means cooperation among nodes is necessary (*i.e.* multi-node predicates). For example, *InGeoCircle* can be examined by each node independently based on its own coordinates and the given center and radius information. On the other hand, *InFloatCircle* needs to know the coordinates of all the nodes in the group since it does not relate to the specfic geographical area but to relative locations among nodes. These attributes will be used in the derivation algorithm in the following section. Due to space limitation, we omit some of predicates and functions, and the complete list can be found in [98].

4.3.2 Distributed Program Generation

In this subsection, we explain how to generate code executable on each node from a given specification. As shown in Section 4.2, each node repeats the following step sequence; (i) periodic sensing from sensors, (ii) periodic evaluation of single-node predicates, (iii) tree construction (potential group generation) and leader election, (iv) data collection on the tree, (v) evaluation of multi-node predicates and (vi) execution of actions, to check if conditions are satisfied or not, and to execute actions if satisfied. Therefore, we generate a programming code that corresponds to each step of the sequence according to the given specification. The outline of a generated code for TinyOS is shown in Fig. 4.3 for

Table 4.1: Predicates for Condition Part (Excerpt)

Type	Predinate	Description	Examined by
General	TestEach(v, exp)	true *iff* variable v satisfies exp at every node	Single
Location	InGeoCircle(c,r)	true *iff* all the nodes in the group are within the circle centered at c with radius r	Single
Topology	InFloatCircle(d)	true *iff* all the nodes in the group are within a circle with diameter d	Multi
Location	InGeoRectangle($c1$, $c2$)	true *iff* all the nodes in the group are within the rectangle determined by two coordinates $c1$ and $c2$	Single
Topology	InFloatRectangle(w,h)	true *iff* all the nodes in the group are within the rectangle with width w and height h	Multi
Topology	Size(min, max)	true *iff* the number of nodes in the group is in $[min, max]$	Multi

`DetectedFireSpot` group of the specification in Fig. 4.2.

(i) Periodic sensing from sensors This phase describes routine tasks like periodic reading from sensors, which are used in the given specification. Correspondingly, we prepare a code block that periodically measures data shown in the given specification as variables so that the code can refer to these data at steps (ii), (v) and (vi). The code also sends a beacon to know neighbor nodes. This information is used for step (iii). Each node has a value *Root* to select a leader node among nodes in the potential constitute, and periodically sends a beacon packet that includes following information: (1) evaluation results of single-node predicates, (2) value of *Root*, and (3) value of *Root* of a root node in a potential constitute if the node has already joined any potential constitute. We explain how a node handles this packet in step (iii).

(ii) Periodic evaluation of single-node predicates We generate a code that checks if single-node predicates in each group are satisfied or not. Since

Table 4.2: Functions for Values and Actions (Excerpt)

Function	Description
Average(v)	Calculate the average of variables p among all the nodes in the group
AverageSelect(v, n)	Calculate the average of variables v among randomly-chosen n or more nodes in the group
GetCentroid()	Calculate the centroid of the coordinates of nodes in the group
GetDiameter()	Calculate the maximum distance between nodes in the group
Sleep(t)	sleep in t
UploadData(d, c)	Let exactly one node in the group upload d through network interface c
ExecuteEach(f)	Let each node execute function f

each predicate can be examined by single node or multiple nodes as indicated in Table 4.1, the code checks if only single-node predicates are satisfied or not. If necessary, the code manipulates local variables according to equations such as addition, subtraction and multiplication. If the predicates are met, the code continues executing the following steps since the node may be able to meet all the conditions specified in the group (this is checked later in step (v)). Otherwise, the code goes to step (i) again. In Fig. 4.3, we can see that a temperature is checked in the function SinglePredicateCheck. If all the predicates in the group condition are single-node predicates, our method skips code generation for steps from (iii) to (v).

(iii) Tree construction and leader election After the evaluation of single-node predicates, we need to check multi-node predicates. However, it requires some preparations. In step (iii), our method generates a code that constructs a potential constitute to collect data required for evaluating the rest of the predicates. We apply a tree-based protocol to organize a potential constitute for managing nodes and collecting data from them. In constructing a potential group in which nodes meet the single-node predicates, these nodes connect with each other by direct wireless link (one-hop link). This is done by the code generated for step (i). In step (iii), at first, a node assumes to be a leader node of a potential constitute which contains only the node itself. If the node

receives a beacon packet from one of the neighbor nodes that have also satisfied the single-node predicates of the same group, the node reacts as follows. (1) If *Root* value of the leader node is smaller than *Root* of the node, the node lets the leader node become a leaf node or an intermediate node by sending a special packet, and sends the other nodes a packet to tell that the node becomes a new leader node. (2) If *Root* value of the neighbor node is bigger than *Root* value of the node, the node becomes a leaf node. (3) If *Root* value of the neighbor node is equal or smaller than *Root* value of the node, the node becomes an intermediate node. After a certain period of time, the leader node sends a packet to stop constructing a tree and all the nodes in the tree move to step (iv). The functions *TreeConstruction()*, *StartLeaderNode()*, *ExpireTimer.fired* and *ReceiveTreeConstruction.receive* in Fig. 4.3 assume this part.

(iv) Collection of data After step (iii), the leader node collects the sensing data from the other nodes so that the multi-node predicates can be checked in a certain place. This data collection step will be done through the constructed tree by data replies sent from all nodes except the leader node. An intermediate node waits for the replies from all the children of the node, and merges them into a single packet before replying to it. Thus, our method generates the following three kinds of codes; (1) a leaf node sends its data immediately at step (iv), (2) an intermediate node waits for replies from children and sends its data reply, and (3) a leader node waits for replies from children and moves to step (v).

(v) Evaluation of multi-node predicates Once the leader node collects all data required for checking the rest of the predicates, the node can know all the nodes which meet the condition and become true members of the group. In this step, multiple groups may be created according to a definition of a group because there are many combinations of nodes that can satisfy the given multi-node predicates. For example, if one of the predicates is *Size*(8, 10), at least three different groups with 8, 9 and 10 nodes can be considered. In Fig. 4.3, the function *MultiNodePredicates* generates groups with the calculated average of *temperature* variables by using a pre-defined function *GenerateSets* which derives all possible sets of nodes satisfying a given condition. Thus, the proposed method generates a code that eliminates nodes which do not satisfy the condition. The code also generates several sets of nodes that can meet the

multi-node predicates from the rest of nodes. These derived sets become groups specified in the given specification.

(vi) Execution of actions After organizing groups, each leader node of a group executes actions in the given specification. Besides, if the group is accessed by another group, the leader node has to notify the values of variables to those nodes which need them, Therefore, our method generates a code that not only executes the actions but also notifies those data. This notification is performed in two ways. If the group of nodes which accesses those values are explicitly identified at that moment and if the locations of those nodes are known, the notification can be delivered to the location via geocasting to reduce redundant messages. Otherwise, the notification is distributed by message broadcast. In Fig. 4.3, the code sends a packet via geocasting to a circle since *DetectedFireSpot* group has to notify its centroid and diameter, and *EstimatedFireSpot* group refers to the centroid and the diameter to make a circle.

4.4 Performance Evaluation

We first demonstrate the benefit from our method in terms of developers design effort-saving. This is done by comparison of a given specification and the derived code in terms of simplicity and readability. Then we measure the communication performance in order to show that automatic derivation algorithm can derive a reasonable code. For this purpose, the performance is compared with a naive approach where all the data is collected to a single sink.

4.4.1 Application Examples

We consider two applications. The first one is simple and similar with the fire detection and alert system in Figs.4.2, but it can present applicability of our method to various applications. It is a noise detection system (Fig.4.4) where environmental noise is monitored by sensor nodes. If a sensor node detects a certain noise, then the sensor nodes in the surroundings are organized to calculate the average noise level and upload it. Each node has a facility to upload data to base station, but we would like to limit the number of nodes to upload data to only one node in the group since duplication of reports means waste of computation and communication resources. There are two groups called

Initiator and *SensorGroup* which represent the first-detector of noise over 80db , and the group of sensor nodes in its surrounding area, respectively. `AREA_RADIUS` is a system parameter (constant).

Another example is a crowd estimation system. In a theme park or huge exhibition space, we assume each visitor is given a battery-operated dedicated information terminal with positioning system (like GPS or WiFi-based location system), 3G device, and ad-hoc communication facility (like Zigbee or Bluetooth). This is similar with audio guide offered at museums and historical places, and we simply call it *node*. The objective of using such a device is to exploit location-based guidance or navigation and to obtain crowd information (*e.g.* how each attraction is crowded in a theme park in real-time). Each node broadcasts its position to neighboring nodes, and some nodes collect neighbor positions, generate people crowd information (*i.e.* perform crowd estimation) and report them via 3G networks. However, we would like to prevent all nodes from performing crowd estimation due to the limited number of 3G subscriptions for this purpose or battery efficiency. The specification is given in Fig. 4.5.

CellCrowdEstimator is a group of nodes that estimate the density of a crowded cell. We assume a region is divided into square cells, and (i, j)-th cell is specified by *InGeoRectangle* predicate with a pair of left–bottom coordinate $c(i, j)$ and right-up coordinate $c(i+1, j+1)$. In the condition of *CellCrowdEstimator*, a group of nodes where (i) each node detected more than 5 neighbors, (ii) all the nodes exist in a same cell, and (iii) the number of nodes in the group is at least 10, is organized, and one node in the group is selected to calculate the estimated density of the cell from the information sent by the group member, and report it via 3G network.

4.4.2 Lines of Code Comparison

The node programs are generated from the two applications in the previous section, and are assessed in comparison with the original specifications in terms of the lines of codes (LoCs) and abstraction levels.

Fig.4.3 shows LoCs of the specifications and derived codes. As we mentioned earlier, the translation is done by extracting parameters and conditions from specifications, choosing and composing primitives for collaboration, and

Table 4.3: Lines of Codes (LoCs) Comparison

Applications	Specifications	Derived Codes
Noise Detection	18	547
Crowd Estimation	17	443

embedding them into a skeleton code. Since the derived codes need a lot of implementation level descriptions, they essentially need much more lines than the specifications. Due to space limitations, we did not show the derived codes, but as we exemplified in Fig. 4.2 and Fig. 4.3, we can see the difference of abstract levels between the specification and the derived code.

4.4.3 Performance Analysis of Derived Systems

We have conducted simulation experiments to observe that the automated program derivation performs well. We have used the Scenargie network simulator [102] version 1.4 where IEEE802.11g was used in the MAC and PHY layers of the ad-hoc communications. By assuming small Tx power in wireless sensor networks, the ad-hoc communication range r was about 40 m. We have targeted the noise detection application and the simulation was performed for 60 seconds. The size of the area was $250m \times 250m$ and nodes are deployed uniformly (grid-based deployment),

To present that the derived program can achieve reasonable performance levels, we have evaluated the following metrics.

- *Node coverage ratio*, which is the ratio of the number of actually-found nodes in the simulation to the number of nodes to be found according to the specification and node deployment. In other words, it shows the "completeness" of data collection.
- *Data collection delay*, which is the time duration from the first detection of over 80db noise occurrence to the completion of data collection process.
- *Number of packets in network*, which is the total number of data and control packets in the network layer.

For reference purpose, we have also measured those metrics by a sink-initiated data collection and computation called *sink-based collection* where a sink node broadcasts a data request packet to all the nodes in the field by a simple flooding mechanism, and each receiving node replies to this packet by sending data back

to the sink. In order to verify the performance in various environments, we have prepared the following four scenarios where the number of nodes and/or the number of events is different. 100 nodes (10 $\times$ 10 nodes) with single big noise at 20 seconds) and two big noises at both 20 and 23 seconds, and 144 nodes (12 $\times$ 12 nodes) with single and two big noises at the same timing.

Fig.4.6(a) and Fig.4.6(b) respectively show the node coverage ratios with 100 nodes and 144 nodes. In these graphs, the number of nodes that are expected to be in the group is also shown as bars.

We can see that the ratios are very close to 1.0 in all the cases. We note that sink-based collection achieves very low ratio (0.2 in average). This is due to unreliable message delivery back to sink nodes where these messages concentrated on a few nodes around sink nodes and some of them have been lost.

Fig.4.7 shows the data collection delay. They are not affected by the circle radius that determines the group size. As seen, the sink-based protocol could achieve the shortest delay, but this is mainly due to (very) low node coverage ratios (most packets were not delivered to sink nodes). On the other hand, we can observe that our algorithm could achieve reasonable trade-off between the node coverage ratio and delay.

Finally, Fig. 4.8(a) shows the number of packets observed in the network layer. The number of packets grows as the radius of circles becomes larger, but the growing tread is linear in any case. From this fact, we can say that our group-based local data collection and processing works preventing the growth of traffics to data collection toward a single point, which is often located far from the event occurrence place. We have further analyzed the types of messages in case of $R = 150$ in Fig. 4.8(b). The most of the packets are beacon messages that are used for neighbor discovery purpose. Since the number of beacon packets per nodes is constant, we can confirm that our method will scale to network size and node density growth.

4.4.4 Crowd Sensing System Design and Evaluation

In order to show that our proposed method have practical usefulness in realistic situations such as large-scale fields with non-uniform node distribution, we have executed the experiments in realistic fields for the sensing application. We have supposed a crowd-sensing application to survey crowded regions which

enables to collect data from arbitrary regions. We have evaluated the number of discovered visitors (nodes) and that of missed nodes to assess the applicability of our method. The field for the experiment is shown in Fig.4.9(a), which is a $1000m \times 1000m$ region of a theme park in Osaka. We have arranged 1000 visitors (nodes) with a realistic distribution and crowd-sensing events have occured at the following 4 spots: spots A (830, 680) and B(400, 340) as high node density regions (popular attractions) and C(610, 510) and D(680, 380) as low node density regions. Other settings are the same as those in Sec.4.4.3.

Fig.4.9(b) shows the number of discovered (*i.e.* counted) and missing nodes, which should be discovered but have not been done. In most cases more than 80 % of node were found in average. We note that at Spot D with $R = 100$ (R is the cell size in the specification) the ratio is low, but this is due to the difficulty of node connectivity maintenance due to geography (mainly buildings there) and larger number of nodes to be discovered (the sum of discover and missing nodes), which is larger than that at Spot C with same region radius. Thus, the cell size should be small in low-connectivity area.

In this experimentation, we have shown the practical usefulness of the proposed system at most cases with appropriate determination of the target region size corresponding to the environment.

We are beneficial from our approach since we can conduct this kind of assessment of the system very easily. We may modify the specification (not the implementation code) if expected performance is not achieved or more improvement can be applied. Since real systems need a lot of system configurations, we strongly believe we need this type of support systems.

4.5 Conclusion

In this chapter, we have proposed a support methodology for cooperative wireless sensor network application development. We have designed a language to describe high-level specification of such applications where we can specify the whole system's behavior from developer-friendly viewpoint based on group of node concept, and have provided an algorithm to translate a given high-level specification into program codes for wireless sensor nodes. Our contribution compared with the existing work is that we focus on cooperative applications in WSNs and design a methodology to implement given applications in a fully-

distributed way, assuming computing and communication capabilities of intelligent sensor nodes. In this viewpoint, we believe this is the first approach to tackle with such a problem. We have shown some example descriptions of practical applications and have evaluated the quality of generated programs in the experiments.

Our future work is deriving optimal selection of communication and cooperative protocol and algorithms for various situations. As shown in performance evaluation, there is overhead of the delay and number of generated packet for control and it may cause to shorten life time of each sensor. Therefore, we believe the extension of our method for deriving programs with protocols and parameters which are selected to satisfy application requirements and QoS restrictions, such as delay, life time and reliability, which are given as specifications. For example, in the performance evaluation in Sec.4.4.3, the number of beacon can be configured by parameters for data collection protocols such as a value of interval for beacon sending. Thus, node programs which satisfy the required performances can be derived by giving requirement considering trade-off among delay, completeness of data collection, and energy consumption.

```
// (i) Periodic sensing from sensors
void PeriodicSensing {
  post SendBeacon();
}
// (ii) Periodic evaluation of single-node predicates
void SinglePredicateCheck() {
  if ( temperature > 40 ) { TreeConstruction(); }
}
// (iii) Tree construction and leader election
void TreeConstruction() {
  StartLeaderNode();
}
void StartLeaderNode() {
  myRole = ROLE_LEADER_NODE;
  call ExpireTimer.start( TIMER_ONE_SHOT, 30S );
}
event ExpireTimer.fired() {
  if ( myRole == ROLE_LEADER_NODE ) {
    post StopTreeConstruction();
  }
}
event ReceiveTreeConstruction.receive( void *payload ) {
  TreeConstruction *msg = (TreeConstruction *)payload;
  if ( msg.type = TYPE_BEACON ) {
    if ( msg.rootroot < root ) { post StopLeaderNode(); }
    else if ( msg.selfroot > root ) {
      myRole = ROLE_LEAF_NODE;
    } else if ( msg.selfroot <= root ) {
      myRole = ROLE_BRANCH_NODE;
    }
  } else if ( msg.type == STOP_LEADER_NODE ) {
    myRole = ROLE_LEAF_NODE or ROLE_BRANCH_NODE;
  } else if ( msg.type == STOP_TREE_CONSTRUCTION ) {
    DataCollection();
  }
}
// (iv) Collection of data
void DataCollection() {
  if ( myRole == ROLE_LEAF_NODE ) {
      data = new Data(temperature, myPosition);
      post SendData();
      }
}
event ReceiveData.receive( void *payload ) {
  DataPkt *msg = (DataPkt *)payload;
  if ( myRole == ROLE_BRANCH_NODE ) {
    CombineData(data, msg );
  if ( HasAlreadyReceivedFromAllChildlen()) {
      DataType myData = new Data(temperature, myPosition);
      CombineData(data, myData );
      post SendData();
  }
  } else if (myRole == ROLE_LEADER_NODE) {
    CombineData(data, msg );
  if ( HasAlreadyReceivedFromAllChildlen()) {
      DataType myData = new Data(temperature, myPosition);
      CombineData(data, myData );
      MultiNodePredicates();
    }
  }
}
// (v) Evaluation of multi-node predicates
void MultiNodePredicates() {
  newSets
    = GenerateSets( InFloatCircle( 100 ) &&
                    AverageSelect( temperature, 30 ) > 50 );
  ExecuteEach( newSets, ExecuteAction() );
}
// (vi) Execution of actions
void ExecuteAction(){
  NotificationPkt pkt;
  pkt.center = centroid;
  pkt.diameter = diameter;
  post SendNotificationPkt();
}
```


Figure 4.3: A Generated Code for DetectedFireSpot Group.

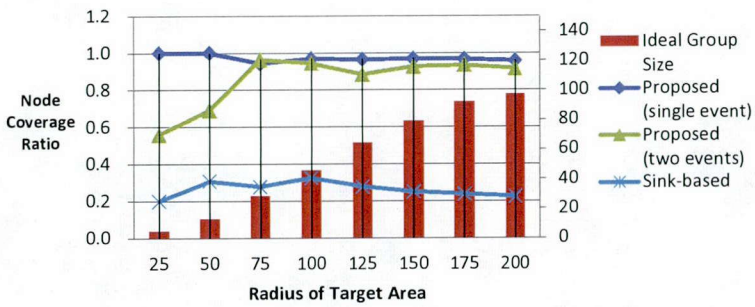
```
nodegroup Initiator
  condition:
      TestEach(noiseLevel, ">80db")
      && Size(1,1)
  action:
      centroid = GetCentroid()

nodegroup SensorGroup
  condition:
      InGeoCircle(Initiator.centroid, AREA_RADIUS)
  action:
      UploadData(Average(noiseLevel),BS);
```

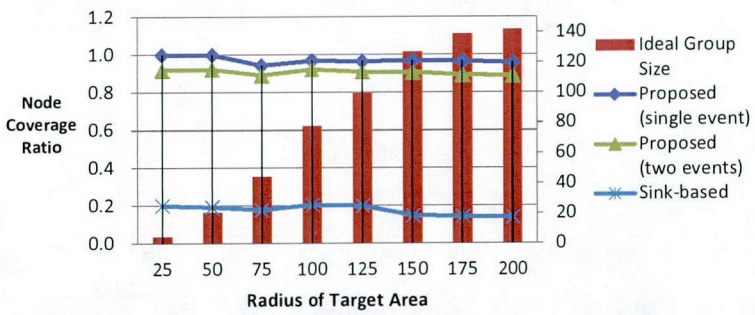
Figure 4.4: An Example Specification of Noise Detection Application

```
nodegroup CellCrowdEstimator
  condition:
      TestEach(neighborCounter, ">5")
      && ( InGeoRectangle(c(0,0), c(1,1))
              || InGeoRectangle(c(1,0), c(1,2))
              ...
              || InGeoRectangle(c(m-1,n-1),c(m,n)) )
      && Size(10, INFINITY);
  action:
      UploadData(EstimateDensity(neighborCounter),
                          3G_INTERFACE)
```

Figure 4.5: Crowd Estimation System



(a) 100 nodes



(b) 144 nodes

Figure 4.6: Node Coverage Ratio (vs. AREA_RADIUS)

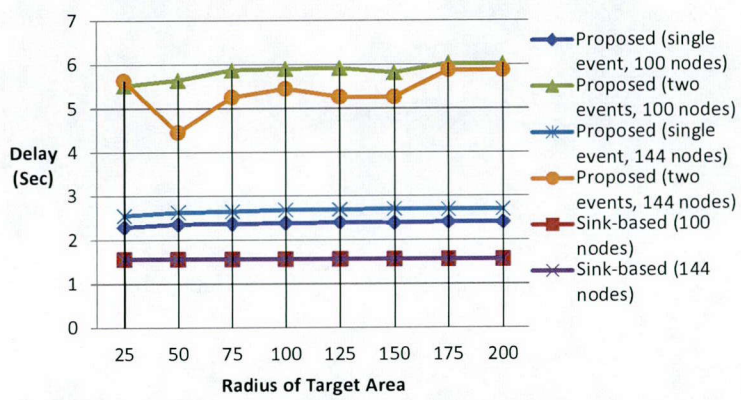
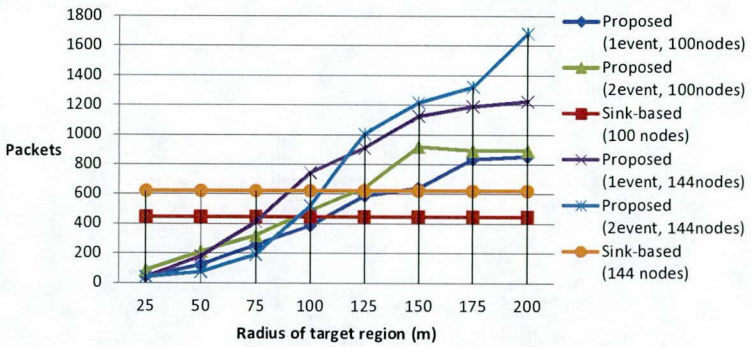
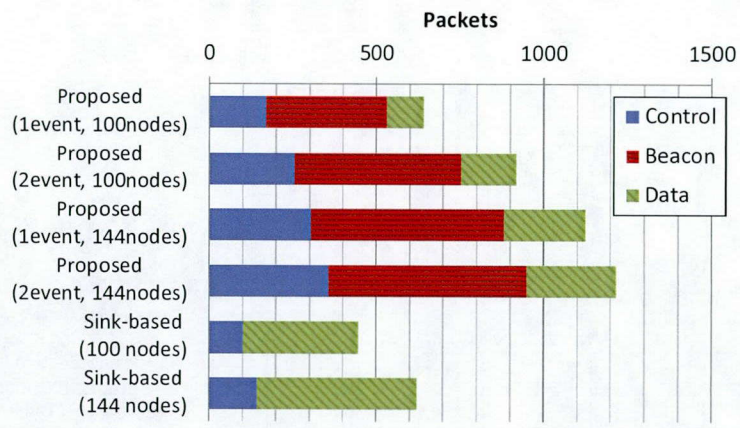


Figure 4.7: Data Collection Delay

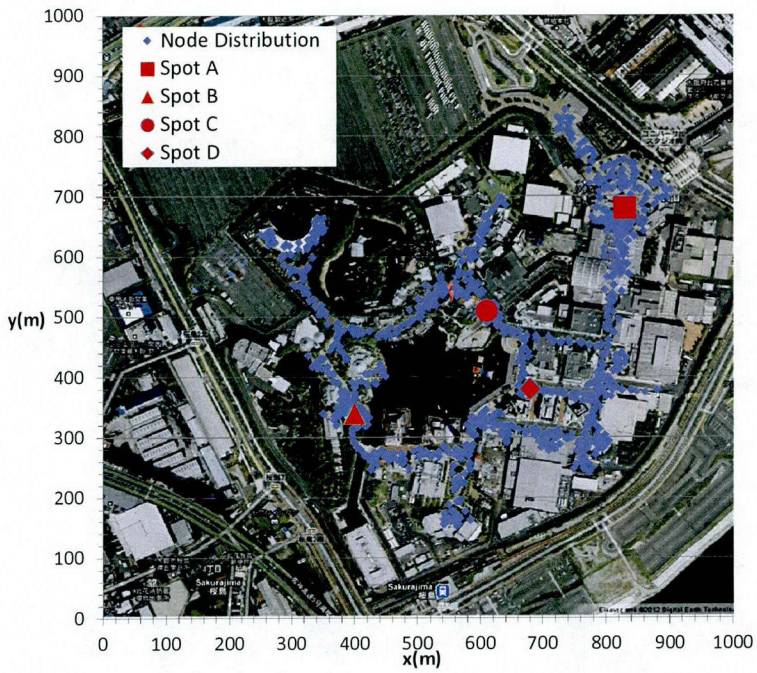


(a) the number of L2 packets

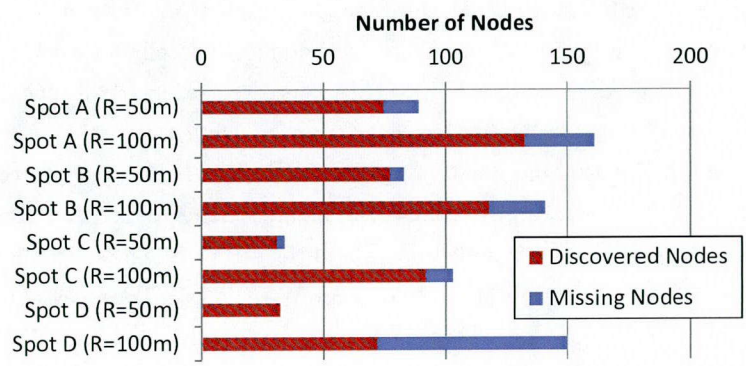


(b) Packet Types Breakdown (R=150)

Figure 4.8: Packets for Data Collection



(a) Visitor Distribution in Theme Park



(b) Number of Coveraged and Missing Nodes

Figure 4.9: Experimental Results

Chapter 5

Design and Architecture of Cloud-based Mobile Phone Sensing Middleware

5.1 Introduction

In this chapter, we propose a middleware to support mobile phone cooperative sensing with a cloud server. Since we have designed in our previous work a methodology to support design and development of collaborative WSN applications [103] (shown in Chapter 4), we use the basic high-level specification language specification part to describe the behavior of whole sensing system. However, it is very different from the methodology for collaborative WSN application in terms of the target architecture where we need to tackle (i) cloud-server architecture and (ii) mobility into consideration, while the method for WSN application assumes homogeneous, decentralized architecture without centralized servers. The middleware to achieve the mobile phone cooperative sensing consists of applications on mobile phones and the server-side module. Each mobile phone and the server communicate through WAN (e.g. 3G), and even two mobile phones through short-range communication such as Bluetooth or WiFi-direct.

Our method automatically translates the given sensing query into *server-side queries* which need to involve multiple mobile phones and *phone-side queries* which are executed by single mobile phones. We provide a concept that hides the details of network configuration, communication and processing inside the

network but all the event occurrences are visible. The sensing query contains time, location and network-based constraints (conditions) and their processing. The process to achieve the given sensing query is very complex since it requires cooperation among mobile phones and servers. Thus, our method hides the physical placement of mobile phones and enables to execute cooperative sensing specified by abstract query descriptions. The proposed method reduces the effort to design and implementation of complex cooperation protocols by this developer-friendly form of behavior specifications format.

We provide a set of event sensing and communication primitives to achieve the given sensing query in the networks. Especially, since the proposed method is extended for mobile phone sensing, we have designed interface and mechanisms to handle mobility and human-mediate processes. Mobility predicates enables to handle mobility conditions about velocities, trajectories and so on. Opt-in predicates enables to human-mediate sensing to ask owners to work for sensing. For example, an owner of mobile phone is required to take a video from the opt-in interface and he takes the video if he agrees with it.

The following simple crowd sensing example helps to understand the concept; each mobile phone sends beacon to each other and thus can detect neighbors. When a crowded situation is detected from the number of neighbors, the system reacts and starts sampling the neighbor count of the surroundings. Based on the sample readings, the system predicts the crowded area and informs to users. This system requires mobile phones collaboration to obtain samples from appropriate location at required intervals.The proposed scheme allows us to write the system in a simple form that consists of three steps, (i) start sampling on detection with required density and intervals, (ii) crowd prediction on obtaining enough samples and (iii) notification, without being aware of physical configuration of mobile phones and the server. We have shown some examples of mobile phone sensing system by our proposed method to show its usefulness. We have also demonstrated the performance of our proposed method in terms of successful data collection and generated packet to validate the quality of processing the given sensing query.

5.2 Approach Overview

5.2.1 Middleware Architecture

In the proposed method, a mobile phone sensing system is defined to collect sensor data matched with given queries from mobile phones. Thus, this system should satisfy the following requirements.

- To reduce the complexity of sensor data and node management, the system enables to select nodes abstractly by conditions of time, location and sensor data attributes.
- To avoid large consumption of device energy and wireless bandwidth and concentration of traffic and process load, the system should communicate with fewer nodes while obtaining enough data. (Thus, processes which are executed by all nodes should be avoided.)
- To detect conditions concerning multiple nodes, the system executes collaborative processing by nodes and the server.

Fig.5.1 shows an image of the mobile phone sensing system by our proposed middleware for these requirements. This system is organized by the server on the cloud and multiple mobile phones and it enables cooperative sensing by their collaboration. A requestor gives a query, which contains a abstracted requirement for mobile phone sensing such as getting density information of a certain area, to the server and the server distribute the query to mobile phones. Each mobile phone determines whether it participates the sensing or not by itself to reduce the probability to upload waste data. When it participates, it executes sensing and uploading data to the server. The server analyzes the data uploaded by mobile phones to extract information. In addition, sometimes it selects mobile phones based on the analyzed data to satisfy the more complex queries concerning multiple nodes. These processes satisfy the above requirement.

Fig.5.2 is the architecture of our proposed middleware to realize such systems. The middleware is composed of the program on the server and the application on each mobile phone. The collaboration between the mobile phones and the server is achieved by communication through WAN (e.g. 3G) and the collaboration among mobile phones is achieved by short-range communication such as Bluetooth or WiFi-direct. The server program provides functions for

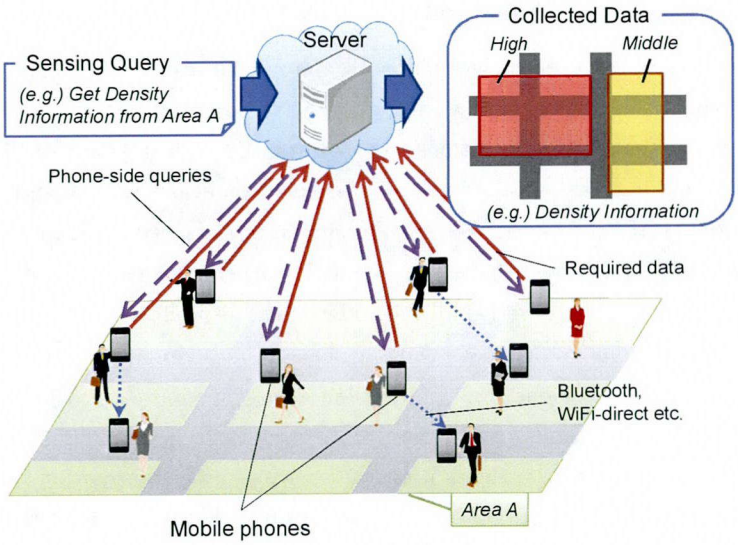


Figure 5.1: Mobile Sensing System Architecture

distributing queries, analyzing uploaded data, managing each mobile phone, and selecting some mobile phone to participate the more complex sensing based on analyzed data. The applications on each mobile phone has functions such as determinating to participate the sensing, sensing according to the query information, uploading the sensing data to the server. The functions provided by this middleware enable such mobile phone sensing.

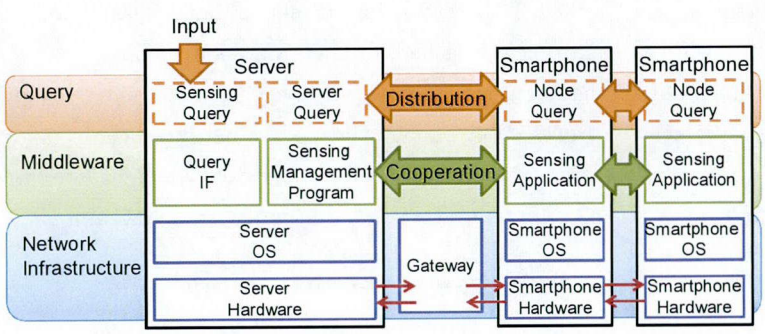


Figure 5.2: Proposal Middleware architecture

5.2.2 Query Description Outline

Our method can execute sensing on each mobile phone (hereinafter called node) and the server for a given sensing query that describes actions to be taken by a group of nodes and conditions to be examined before the actions. Requestors can easily describe systems by specifying such conditions that should be checked by cooperation of nodes. We show a simple but essential example in Fig. 5.3 where a group of nodes that satisfy the following conditions is defined as the first group to detect a crowd; (1) all the nodes in the group have detected numbers of neighbors higher than 10, (2) they are located in a circle whose radius is 100m and whose center is one of given positions of intersections, and (3) the size of the group is more 30 nodes (we explain these predicates in Section 5.3, but readers may refer to Table 5.1). In addition, in order to sample information of the crowd, the second, larger group of nodes that contains the previous group having the similar center with the previous group (*CrowdDetector*) but with larger radius 200m is defined. The nodes in the second group (*SamplingSpot*) sample and upload numbers of neighbor as crowd information. In this way, conditions on geometry, sensing data values and their manipulations can be written in our query description.

However, such a sensing query is not easy to satisfy since checking conditions and executing actions need cooperative operations among nodes. For example, in order to check a condition on sensing data, (i) a node group needs to be organized, (ii) the sensing data needs to be collected onto the server, and (iii) it needs to be checked if the condition is met or not. The action should be executed if the condition is satisfied, or the group is dismissed.

Thus, our method can automatically derive the single-node and multi-node queries which are processed by cooperative nodes and the server from given sensing queries. This hides the details of node behavior, which are often complex, from the developers. Therefore they can concentrate on system logic.

5.2.3 Distributed Execution on mobile phones and the Server

For a given sensing query described by a set of nodes with pre-conditions and post-actions, we classify the predicates that constitute the condition into two categories, single-node predicates and multi-node predicates. An example of

```
nodegroup CrowdDetector
  condition:
      TestEach(neighborCount, ">10")
      && InFloatCircle(100)
      && Size(30, INFINITY)
  action:
      centroid = GetCentroid()

nodegroup SamplingSpot
  condition:
      InGeoCircle(CrowdDetector.centroid, 200)
  action:
      OutputData(
          GetSamplingDataSelect(
              neighborCount, 1min, 10min, 3))
```

Figure 5.3: A Query Description of Crowded Sensing

single-node predicate is *TestEach* that checks if variable on each node satisfies a given condition (see *TestEach*(neighborCount, "> 10") in Fig. 5.3). Meanwhile, both *InFloatCircle*(100) and *Size*(30, INFINITY) are multi-node predicates since they cannot be examined by single nodes. For example, *InFloatCircle(100)* needs distance calculation for every pair of nodes, meaning that it can be checked only when a group of nodes is given. Considering this fact, we take the following strategy; Firstly, from the given sensing query, the server generate single-node queries which contain information of single-node predicates in it and multi-node queries which contain information of multi-node predicates in it. Each node contacts to the server and get single-node queries periodically. We let each node periodically check single-node predicates, and let the node be a potential constitute of the group if it satisfies the conditions. If a node becomes a potential constitute of the group, it report to the server. The server constructs a node group based on these reports. Then the data values to check the multi-node predicates are collected to the server, and it checks if all the multi-node predicates are satisfied or not. If true, those nodes take actions as specified. Moreover, we allow describing conditions of groups that depend on some other groups. For example, the second group (*SamplingSpot* in Fig. 5.3) is such a group that refers to the "center" of *CrowdDetector* as a part of its conditions. In this case, the centroid of coordinates of nodes in *CrowdDetector*

has been calculated by the server to prepare for creation of *SamplingSpot*, and the information is distributed to potential constitutes of *SamplingSpot* (in this case, all the nodes). by the server in two ways. (1) polling by each node or (2) broadcasting by some nodes which are nearby the target location.

In summary, each node needs periodically to check the single-node conditions of each group, and then report the result and some data to the server. The server forms the nodes whose report is received into a potential group with nodes which also satisfy the same conditions. During the server collects all the data necessary to check multi-node conditions. Then it actually checks the conditions and executes the post-actions in cooperation with the nodes. During the process, it prepares and calculates the data for the other groups' conditions if any.

5.3 Language and Algorithm Details

5.3.1 Query Description Language

A sensing query consists of two types of profiles, *node profiles* and *nodegroup profiles*.

The *node profiles* define the attributes of sensor nodes. For example, if a WSN consists of wireless sensor nodes and base stations, then we prepare two profiles that correspond to them. In their profiles, local variables (storing sensor data and so on) and methods they hold are defined. We omit example descriptions here because they just consist of definitions of variables and functions.

In *nodegroup profile*, each block of description starts with a keyword **nodegroup** (words highlighted by bold fonts are reserved words hereafter). Conceptually, this corresponds to a group of nodes that cooperatively execute tasks. Developers can define pre-conditions with **condition** keyword and post-actions with **action** keyword. The condition part must be a logical formula using predefined *single-node* and/or *multi-node* predicates, and the action part must be a list of functions (or procedures).

The example query description of a crowd sensing system shown in Sec.5.2.2 (Fig. 5.3) is an example of formal description. *CrowdDetector* and *SamplingSpot* are nodegroup definitions blocks. In *CrowdDetector*, three predicates are speficied in the condition part. *TestEach* is a single-node predicate, and *InFloatCir-*

cle and *Size* are multi-node predicates. As we explained in the previous section, groups may refer to other groups by directly specifying their group names. For example, *SamplingSpot* group refers to the *CrowdDetector* group. Since the node groups defined by *CrowdDetector* may not be unique (*i.e.* there may appear multiple groups), *CrowdDetector* is assumed to be the reference to the first-generated group in our language definition. The condition of *SamplingSpot* contains a single-node predicate *InGeoCircle* with 2 parameters. In the sensing description, the value of 1st paramter has been determined in *CrowdDetector* group by *GetCentroid* functions.

Tables 5.1 and 5.2 show the list of predicates and functions, respectively. As for the predicate table, we add how the predicates are examined in distributed environment in the last column. *Single* means it can be tested by each node independently (*i.e.* single-node predicates), while *Multi* means cooperation among nodes is necessary (*i.e.* multi-node predicates). For example, *InGeoCircle* can be examined by each node independently based on its own coordinates and the given center and radius information. On the other hand, *InFloatCircle* needs to know the coordinates of all the nodes in the group since it does not relate to the specific geographical area but to relative locations among nodes. These attributes will be used in the execution algorithm in the following section. Due to space limitation, we omit some of predicates and functions, and the complete list can be found in [98].

Additionally, the proposed method is designed for mobile phone sensing. Thus, we have designed some mobile-device-specific predicates. For example, *KeepUpWithCircle* is a mobility predicates which is evaluated based on the device's trajectory, and *AllowsToProvideVideo* is a opt-in predicates which requires the owner of the device to determine to take video for sensing through the GUI of the device. These type predicates are suitable for mobile phone sensing systems.

5.3.2 Mobile Phone Sensing Execution

In this subsection, we explain how to execute a given sensing query on each node and server. As shown in Section 5.2, each node and server repeats the following step sequence; (i) query generation from given sensing queries, (ii) periodic polling to get single node query and notification, (iii) periodic sensing from

sensors, (iv) periodic evaluation of single-node predicates, (v) data collection and potential group generation (vi) evaluation of multi-node predicates and (vii) execution of actions, to check if conditions are satisfied or not, and to execute actions if satisfied.

(i) Query generation from given sensing queries In this step, the developer give a sensing query of sensing to the server and the server generates a phone-side query and a server-side query. Each sensing query contains single-node predicates and multi-node predicates, respectively. Single-node predicates should be processed by each node and multi-node predicates should be processed by the server since it can collect and manage data of multi nodes. Thus, each sensing query is divided into a part of single-node predicates and that of multi-node predicates. The former becomes a phone-side query and the latter becomes a server-side query.

(ii) Periodic polling to get phone-side query and notification Queries and notifications (discussed later) should be distributed to all nodes in the field. But broadcasting to all nodes causes concentrations of a large amount of data traffics on the server and frequency distribution causes a large amount of energy consumption on each node. Thus, the server distributes them by a polling strategy. Each node asks the server if there are phone-side queries and notifications for every interval T. The server sends them to the node if they are updated.

(iii) Periodic sensing from sensors In this step, each node executes routine tasks like periodic reading from its sensors, which are used in the given single-node and server-side queries. Each node periodically measures data shown in the given sensing query as variables so that the code can refer to these data at steps (iv), (vi) and (vii). Especially, each node stores the history of its position since its trajectory may be required for single-node predicates.

(iv) Periodic evaluation of single-node predicates In this step, each node checks if single-node predicates in each group are satisfied or not. Since each predicate can be examined by single node or multiple nodes as indicated in Table 5.1, the code checks if only single-node predicates are satisfied or not. If

necessary, the code manipulates local variables according to equations such as addition, subtraction and multiplication. If all other single-node predicates are satisfied but opt-in predicates are not, each node asks its owner to determine to work for sensing and they becomes true if the owner agree. If the predicates are met, the node continues executing the following steps since the node may be able to meet all the conditions specified in the group (this is checked later in step (vi)). Otherwise, the code goes to step (i) again. If all the predicates in the group condition are single-node predicates, our method skips the steps from (v) and (vi).

(v) Data collection and potential group generation In this step, each node reports the result of checking the single-node predicates and its data for checking multi-node predicates. The server receives reports from nodes and organized the sender nodes as a potential node group after a certain interval from receiving the first report.

In addition, our proposed method also provides tree-based data collection protocol by using ad-hoc communication facility (like Zigbee or Bluetooth) for reduction of the communication cost of data uploading. Nodes which satisfy single-node predicates construct a tree and they collect data to root node through the tree. The root node uploads the collected data to the server. We can select direct uploading or tree-based uploading.

(vi) Evaluation of multi-node predicates Once the server collects all data required for checking the rest of the predicates, it can know all the nodes which meet the condition and become true members of the group. In this step, multiple groups may be created according to a definition of a group because there are many combinations of nodes that can satisfy the given multi-node predicates. For example, if one of the predicates is *Size*(8, 10), at least three different groups with 8, 9 and 10 nodes can be considered. The server generates groups with the calculated average of *temperature* variables by using a pre-defined function *GenerateSets* which derives all possible sets of nodes satisfying a given condition. Thus, the server sends a special packet to nodes, which do not satisfy the condition, to eliminates and also generates several sets of nodes that can meet the multi-node predicates from the rest of nodes. These derived sets become groups specified in the given sensing query.

(vii) Execution of actions After organizing groups, the server executes actions in the given specification. Besides, if the group is accessed by another group, the server has to notify the values of variables to those nodes which need them. Therefore, it not only executes the actions but also notifies those data. This notification is performed in two ways. If the group of nodes which accesses those values are explicitly identified at that moment and if the locations of those nodes are known, the notification can be delivered to the location via geocasting from a certain node in the group to reduce redundant messages. Otherwise, the notification is distributed by polling of each node in step (ii).

5.4 Performance Evaluation

We first demonstrates the benefit from our method in terms of developers design effort-saving. This is done by introducing example systems of our proposed method. Then we measure the communication performance in order to show proposed middleware works well by simulations. In addition, to demonstrate our method is available in real environment and provides useful interface for developers, we performed experimentations in real environment.

5.4.1 System Examples

To show that our proposed method have a system as a development environment for mobile phone sensing, we introduce two systems.

The first one is simple and similar with the crowd detection and alert system in Figs.5.3, but it can present applicability of our method to various systems. It is a traffic jam monitoring system (Fig.5.4). If a traffic jam is occurred, some mobile phone user takes videos of surrounding situation of the jam. This system is useful to understand details of the traffic jam and to support driver's determination. If mobile phones are car-mounted mode and detect a traffic jam by detecting that their velocity are continuously low, then the mobile phones in the surroundings are organized to assign tasks to taking videos and upload them. Each node has a facility to do it, but we would like to limit the number of mobile phones to upload data to only 10 node in the group since duplication of video uploading means waste of computation and communication resources. There are two groups called *TrafficJamSpot* and *MonitoringStreet* which represent the

```
nodegroup TrafficJamSpot
  condition:
      InFloatStreet(500m)
      && TestEach(mode, "==CAR_MOUNTED")
      && DurationTestEach(velocity, "<20kmph", 3min)
  action:
      centroid = GetCentroid()

nodegroup MonitoringStreet
  condition:
      InGeoCircle(Initiator.centroid, 2km)
      && AllowsToProvideVideo
  action:
      OutputData(
          GetSamplingDataSelect(movie, 1min, 1min, 10))
```

Figure 5.4: An Query Description of Traffic Jam Monitoring

```
nodegroup TransportationPassengers
  condition:
      IsFollowingPath(TRANSPOTATION_PATH, 10min)
      && KeepUpWithCircle(10m, 10min);
  action:
      OutputData(
          GetSamplingDataSelect(position, 1min, 10min, 2))
```

Figure 5.5: An Query Description of Public Transportation Monitoring

first-detectors of traffic jam, and the group of mobile phones in its surrounding area.

Another example is a public transportation monitoring system. For a public transportation (*e.g.* bus), each of passengers who ride on it has its mobile phone. If the information of the transportation is required, the system on the server finds passengers on it and collects real-time position data from them. The sensing query is given in Fig. 5.5.

TransportationPassengers is a group of nodes which are corresponding to passengers of transportation. The group provides real-time position data. We assume nodes, which follow the same path as the transportation and keep within a certain distance between each node, as passengers of the transportation. Thus, each node monitors its trajectory and, if it follows the path, report it to the

server. After the server receives reports from some nodes, node groups whose nodes keep their distance are organized and 10 nodes in the group is selected to upload the real-time position to the server for 10 minutes.

These systems supported by our middleware are useful example of mobile phones sensing. These demonstration show broad utility of the middleware.

5.4.2 Performance Analysis of the Middleware

Performance Evaluation in Simulated Environments

We have conducted simulation experiments to observe that our middleware performs well. We have used the Scenargie network simulator [102] version 1.4 where IEEE802.11g have been used in the MAC and PHY layers as wireless wide area network communication and all nodes can connect with the server by this network. We have targeted the crowd detection system and the simulation was performed for 50 seconds. The size of the area was $200m \times 200m$ and there are 4 cross point (as shown in Fig.5.6). Nodes are moving at a constant speed 1 m/sec and a crowd detection event occurs in the intersection at (150m, 150m) after 20 seconds. Nodes nearby the event report it to the server and it sends request packet to nodes in the target area.

To present that the middleware can achieve reasonable performance levels, we have evaluated the following metrics.

- *Node coverage ratio*, which is the ratio of the number of actually-found nodes in the simulation to the number of nodes to be found according to the query and node deployment. In other words, it shows the "completeness" of data collection.
- *Number of packets*, which is the total number of data and control packets in the network layer.

In order to verify the performance in various environments, we have prepared the scenario with 160 nodes which move along the streets on the field.

Fig.5.7 show the node coverage ratios. In these graphs, the number of nodes that are expected to be in the group is also shown as bars. We can see that the ratios are very close to 1.0 in all the cases. This shows a certain level of scalability to sense fields.

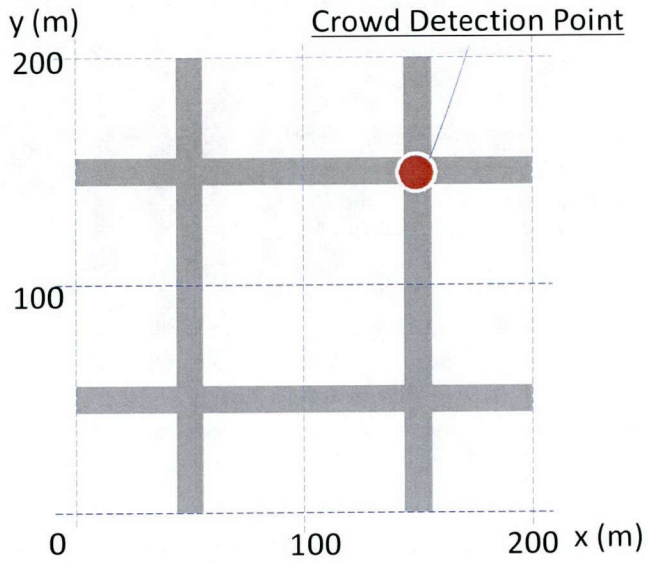


Figure 5.6: Experiment Field

Finally, Fig. 5.8 shows the number of packets observed in the network layer and ideal nodes to send data packets. The number of packets grows as the radius of circles becomes larger and shows slightly larger than the number of ideal nodes in cases (less than "150 m") . However, the growing tread is similar to the number of ideal target nodes. From this fact, we can say that our mobile phone sensing middleware can prevent excessive traffic growth during the data collection phase.

The simulated evaluation shows the middleware works well in the ideal environment. Our ongoing works includes more realistic evaluation in various environments to show the utility.

Performance Evaluation in Real Environments

To show practical utility of our middleware for mobile phone sensing, we have implemented a prototype of mobile phone sensing system and have evaluated its data collection performance in real environment.

The prototype system is the crowd sensing system as shown in Fig.5.3 to detect human locations in a certain region. This system asks mobile phones in the region to report their locations by sending queries to them. We have

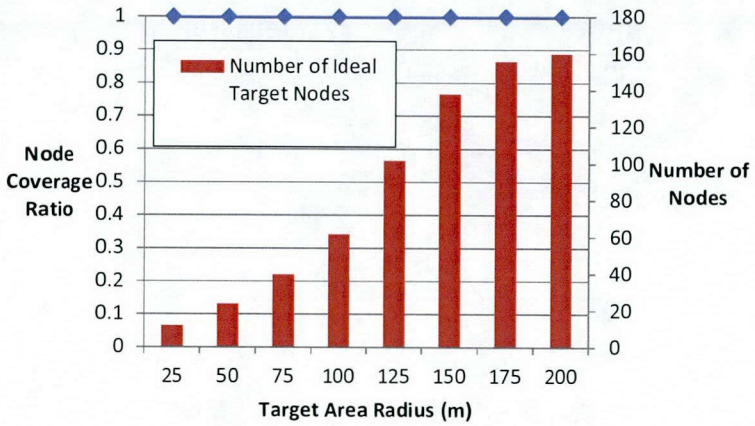


Figure 5.7: Node Coverage Ratio

conducted the experiments at a intersection in a campus of Osaka University (Fig.5.10). 12 mobiles phones are placed in the intersection and report their data by multi-hop forwarding if they are in a target circle. We have evaluated the performance of the system with several target circles, whose radii are 20m, 30m, 40m, and 50m. In our framework, it is easy to change such conditions since mobiles phones are controlled by queries and the queries are distributed to the mobile phones for each sensing.

The simulation results are shown in a GUI, which enables to monitor mobile phones and their locations easily and helps to manage and operate them (Fig.5.9). We have evaluated node coverage ratios and delays of location collection in the same way as Sec.5.4.2. Table 5.3 shows the results. We can see that node coverage ratios are higher than 70% in all scenarios. We can see that the delay becomes large as R becomes large. This is because it takes more hops to deliver locations to the cloud server. Table 5.4 shows the number of packets in the system. We can see that the number of control packets increase linearly. As shown in above experiments, our middleware supports not only mobile sensing itself but also analyzing the system performance. We believe that our approach can contribute to reduce whole cost of mobile sensing.

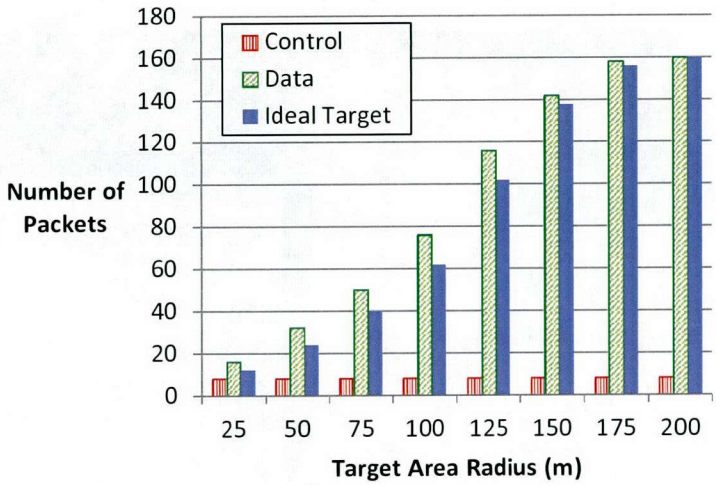


Figure 5.8: The number of L2 packets

5.5 Conclusion

In this chapter, we propose a middleware to support mobile phone cooperative sensing with a cloud server. We have designed a language to describe high-level specification of such systems where we can specify the whole system's behavior from developer-friendly viewpoint based on group of node concept, and the middleware to achieve the mobile phone cooperative sensing consists of apps on mobile phones and the server-side module. Our method automatically translates the given sensing query into a sequence of queries which are executed by the server and mobile phones. We provide a set of event sensing and communication primitives to achieve the given specification in the networks since we have designed in our previous work, a methodology to support design and development of collaborative WSN applications proposed in Chapter 4. However, it is very different from the mothod for WSN in terms of the target architecture where we need to take (i) cloud-server architecture and (ii) mobility into consideration, while the method in Chapter 4 assumes homogeneous, decentralized architecture without management by cloud-server. In this viewpoint, we believe this is the first approach to tackle such problems. We have shown some example descriptions of practical systems and have evaluated the quality of our proposed

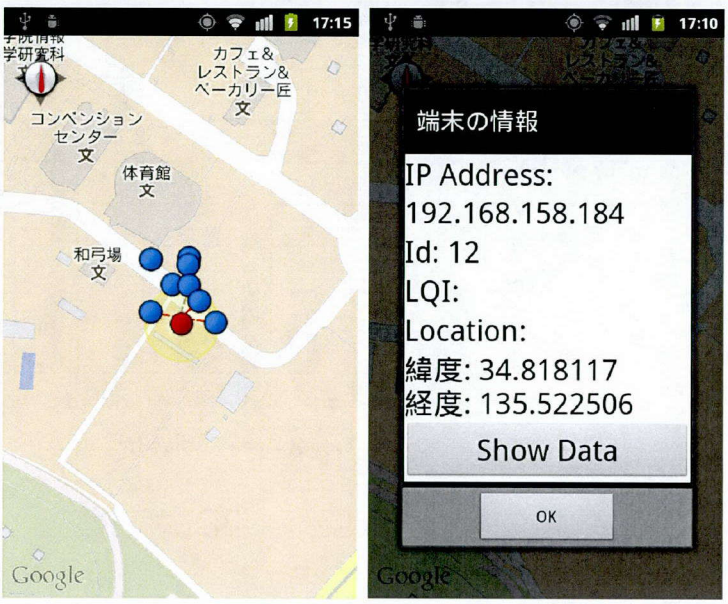


Figure 5.9: Screen Shot of An Example Application on Android

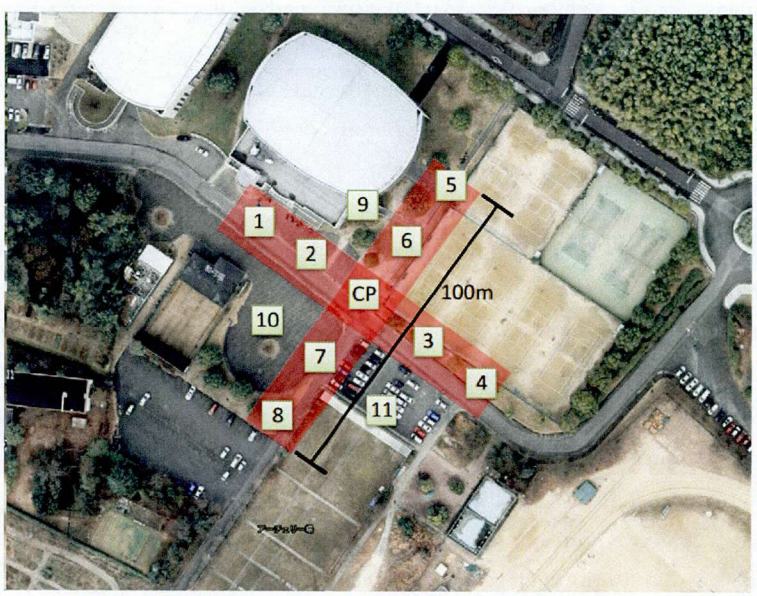


Figure 5.10: Field of Experimentation

method in the experiments.

Our ongoing work includes applying the proposed method to various situations such as pedestrian crowds, car traffic, train passengers, and mixture of them. In those platforms, we need to consider mobility, neighbor discovery and security issues keeping the architecture limitation in mind. Thus, we have to provide various methods corresponding to various situations.

Table 5.1: Predicates for Condition Part (Excerpt)

Type	Predinate	Description	Examined by
General	TestEach (v, exp)	true *iff* variable v satisfies exp at every node	Single
General	DurationTestEach (v, exp, t)	true *iff* variable v satisfies exp at every node for the duration t	Single
Location	InGeoCircle (c,r)	true *iff* all the nodes in the group are within the circle centered at c with radius r	Single
Topology	InFloatCircle (d)	true *iff* all the nodes in the group are within a circle with diameter d	Multi
Location	InGeoRectangle ($c1$, $c2$)	true *iff* all the nodes in the group are within the rectangle determined by two coordinates $c1$ and $c2$	Single
Topology	InFloatRectangle (w, h)	true *iff* all the nodes in the group are within the rectangle with width w and height h	Multi
Location	InGeoStreet (c,r)	true *iff* all the nodes in the group are in a street at c and within distance r from c	Single
Topology	InFloatStreet (d)	true *iff* all the nodes in the group are in a street and within distance d	Multi
Topology	Size (min, max)	true *iff* the number of nodes in the group is in $[min, max]$	Multi
Mobility	KeepUpWithCircle (d, t)	true *iff* all the nodes in the group keep up within a circle with diameter d for t seconds	Multi
Mobility	IsFollowingPath (p, t)	true *iff* all the nodes in the group follow the path p in this t seconds	Single
Opt-In	AllowsToProvideVideo (m, c)	true *iff* owners of all the nodes in the group show the caption c and allow to take and upload a movie m	Single

Table 5.2: Functions for Values and Actions (Excerpt)

Function	Description
Average(v)	Calculate the average of variables p among all the nodes in the group
AverageSelect(v, n)	Calculate the average of variables v among randomly-chosen n or more nodes in the group
GetCentroid()	Calculate the centroid of the coordinates of nodes in the group
GetDiameter()	Calculate the maximum distance between nodes in the group
GetVelocity()	Calculate the average velocity of all nodes in the group
GetTrajectory(t)	Calculate the the past t trajectory of the centroid of the coordinates of nodes in the group
Sleep(t)	sleep in t
OutputData(d)	Let the server output d
ExecuteEach(f)	Let each node execute function f
GetVelocity()	Calculate the velocity of the centroid of the coordinates of nodes in the group f
OutputSamplingData (d, i, t)	Let the server output d for t every i
GetSamplingDataSelect (d, i, t, n)	Let n nodes in the group upload d for t every i and let the server to output the uploaded data

Table 5.3: Performance Evaluation in Real Environment Experimentation

R	20	30	40	50
Node Coverage Ratios	1.00	0.88	0.73	0.84
Delay (Sec)	13.963	17.812	21.443	20.598

Table 5.4: The Number of Generated Packets in Real Environment Experimentation

R	20	30	40	50
Control Packets	33	52	139	164
Data Packets	2	2	9	13

Chapter 6

Conclusion

In this thesis, we propose D-sense, a development support environment for sensor networks to support sensing system development comprehensively. The environment is designed according to three themes: (i) support for network-level programming and performance evaluation, (ii) node specification generation from application-level requirement for cooperative sensing of sensor node groups, and (iii) middleware to achieve cooperative mobile phone sensing with servers based on application-level queries.

The theme (i) abstracts details of node program codes and enables developers to implement network-level node programs such as routing protocols concentrating on their algorithm. It also supports performance evaluation of node programs on real devices and simulators by the code sharing mechanism. The theme (ii) derives a behavior specification of each node on WSNs from application-level requirement based on node groups which are defined by terms of real world such as location, topology, and sensor data. Developers can customize processes of the specifications. These support development of multiple layer of sensing systems comprehensively. The theme (iii) enables to achieve mobile phone sensing by giving requirements based on sensor node group in the same concept with the theme (ii). The cooperative approach in WSN cannot apply to mobile phone network because its topology is constantly changing because of its nodes' mobility. Thus, we add a term of time to the format of sensing requirement and design the middleware for mobile phone sensing by cooperation among the server and each mobile phone. The middleware achieves mobile phone sensing in hiding detail information of each mobile phone such as its location, mobility, and ID.

In Chapter 2, we have surveyed several researches of sensor network development support to show the features of our approaches. We have also surveyed researches about sensor node programming support, cooperative sensing development support, and moble phone sensing support to address the related work and show the features of our each methods.

In Chapter 3, we have designed and developed an integrated environment called D-sense for supporting development of WSNs. D-sense supports protocol design by high-level design APIs. Also it provides seamless collaboration of simulated and real networks for performance evaluation, and a powerful distributed debugging scheme. We have conducted performance evaluation of the SPEED protocol in simulation and real environment to show the effectiveness of D-sense. Our ongoing work includes developing a complete set of design/debug APIs and related tools, and opening them to public domain.

In Chapter 4, we have proposed a support methodology for cooperative wireless sensor network application development. We have designed a language to describe high-level specification of such applications where we can specify the whole system's behavior from developer-friendly viewpoint based on group of node concept, and have provided an algorithm to translate a given high-level specification into program codes for wireless sensor nodes. Our contribution compared with the existing work is that we focus on cooperative applications in WSNs and design a methodology to implement given applications in a fully-distributed way, assuming computing and communication capabilities of intelligent sensor nodes. In this viewpoint, we believe this is the first approach to tackle with such a problem. We have shown some example descriptions of practical applications and have evaluated the quality of generated programs in the experiments.

In Chapter 5, we propose a middleware to support mobile phone cooperative sensing with a cloud server. We have designed a language to describe high-level specification of such systems where we can specify the whole system's behavior from developer-friendly viewpoint based on group of node concept, and the middleware to achieve the mobile phone cooperative sensing consists of apps on mobile phones and the server-side module. Our method automatically translates the given sensing query into a sequence of queries which are executed by the server and mobile phones. We provide a set of event sensing and commu-

nication primitives to achieve the given specification in the networks since we have designed in our previous work, a methodology to support design and development of collaborative WSN applications proposed in Chapter 4. However, it is very different from the mothod for WSN in terms of the target architecture where we need to take (i) cloud-server architecture and (ii) mobility into consideration, while the method in Chapter 4 assumes homogeneous, decentralized architecture without management by cloud-server. In this viewpoint, we believe this is the first approach to tackle such problems. We have shown some example descriptions of practical systems and have evaluated the quality of our proposed method in the experiments.

Our ongoing work includes applying the proposed method to various situations such as pedestrian crowds, car traffic, train passengers, and mixture of them. In those platforms, we need to consider mobility, neighbor discovery and security issues keeping the architecture limitation in mind. We have also working for supporting system optimization techniques such as prediction of running cost of the sensing system by application specification and environmental information. Therefore, developers may be able to design the system while verifying its performance. Although this is a big challenge, we believe it is beneficial for many service developers who wish to use smartphones for sensing purpose.

Acknowledgement

I would like to express my sincerely appreciation to continued support of my supervisor Professor Teruo Higashino of Osaka University through trials and tribulations of this Ph.D thesis. Again I express my heartiest gratitude to him for his encouragement and invaluable comments in preparing this thesis.

I am very grateful to Professor Koso Murakami, Professor Masayuki Murata and Professor Hirotaka Nakano of Osaka University for their invaluable comments and helpful suggestions concerning this thesis.

I am very grateful to Associate Professor Hirozumi Yamaguchi for the precious advices and technical discussions provided through out the research. I would like to thank Assistant Professor Takaaki Umedu of Osaka University for his valuable comments. I would like to express my thanks to Assistant Professor Akihito Hiromori for helpful support. I greatly thank Mr. Akihiro Inagaki an Mr. Yu-Chih Wang for their cooperation in the experiments.

Thanks go to everyone of Higashino laboratory for their feedback, encouragement and support.

Finally, I would like to thank my family and my friends for their help and understanding.

Bibliography

[1] G. Werner-allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, 2006.

[2] "Ist cruise project. flood detection using sensor networks," www.ist-cruise.eu/cruise/business- deck/wsns- applications/flood- detection- 1.

[3] A. P. R. da Silva, M. H. T. Martins, B. P. S. Rocha, A. A. F. Loureiro, L. B. Ruiz, and H. C. Wong, "Decentralized intrusion detection in wireless sensor networks," in *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks (Q2SWinet '05)*. New York, NY, USA: ACM, 2005, pp. 16–23.

[4] L.-A. Tang, X. Yu, S. Kim, J. Han, C.-C. Hung, and W.-C. Peng, "Tru-alarm: Trustworthiness analysis of sensor networks in cyber-physical systems," in *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1079–1084.

[5] C. Manzie, H. Watson, S. Halgamuge, and K. Lim, "On the potential for improving fuel economy using a traffic flow sensor network," in *Proceedings of the 3rd International Conference on the Intelligent Sensing and Information Processing(ICISIP 2005)*, jan. 2005, pp. 38 – 43.

[6] A. Deshpande, C. Guestrin, and S. R. Madden, "Resource-aware wireless sensor-actuator networks," *IEEE Data Engineering Bulletin*, vol. 28, pp. 40–47, 2005.

[7] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, Aug 2000, pp. 149–160.

[8] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A stateless protocol for real-time communication in sensor networks," in *Proceedings of the 23rd International Conference on Distributed Computing Systems(ICDCS 2003)*, May 2003, pp. 46–55.

[9] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "On the construction of energy-efficient broadcast and multicast trees in wireless networks," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, 2000, pp. 585–594.

[10] D. Braginsky and D. Estrin, "Rumor routing algorthim for sensor networks," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications (WSNA 2002)*, 2002, pp. 22–31.

[11] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999)*, 1999, pp. 90–100.

[12] W. Wei and A. Zakhor, "Multiple tree video multicast over wireless ad hoc networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 1, pp. 2–15, January 2007.

[13] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Annual Hawaii International Conference onSystem Sciences (HICSS-33)*, January 2000, pp. 1–10.

[14] Y. Liyang, W. Neng, Z. Wei, and Z. Chunlei, "GROUP: A grid-clustering routing protocol for wireless sensor networks," in *Proceedings of the 2nd International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2006)*, September 2006, pp. 1–5.

[15] Scalable Network Technologies, Inc., "Qualnet simulator," http://www.scalable-networks.com/.

[16] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, Aug. 2008.

[17] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *Proceedings of Wireless Communications and Networking Conference (WCNC 2005)*, March 2005, pp. 1664–1669.

[18] C. Technology, "Micaz motes."

[19] R. Murty, G. Mainland, I. Rose, A. Chowdhury, A. Gosain, J. Bers, and M. Welsh, "Citysense: An urban-scale wireless sensor network and testbed," in *Proceedings of the 2008 IEEE International Conference on Technologies for Homeland Security (HST '08)*, May 2008, pp. 583 –588.

[20] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: a wireless sensor network testbed," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, Apr 2005, pp. 483–488.

[21] CitySense project, "CitySense - An Open, Urban-Scale Sensor Network Testbed," http://www.citysense.net/.

[22] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921 – 960, 2007.

[23] C. Baker, K. Armijo, S. Belka, M. Benhabib, V. Bhargava, N. Burkhart, A. Der Minassians, G. Dervisoglu, L. Gutnik, M. Haick, C. Ho, M. Koplow, J. Mangold, S. Robinson, M. Rosa, M. Schwartz, C. Sims, H. Stoffregen, A. Waterbury, E. Leland, T. Pering, and P. Wright, "Wireless sensor networks for home health care," in *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW '07)*, vol. 2, may 2007, pp. 832 –837.

[24] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system,"

in *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*. New York, NY, USA: ACM, 2004, pp. 1–12.

[25] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks," in *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality (REALMAN '06)*. New York, NY, USA: ACM, 2006, pp. 63–70.

[26] E. Ertin, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, and M. Nesterenko, "Kansei: a testbed for sensing at scale," in *Proceedings of the 5th international conference on Information processing in sensor networks (IPSN '06)*. New York, NY, USA: ACM, 2006, pp. 399–406.

[27] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 3, pp. 16:1–16:26, July 2009.

[28] A. Boulis, C.-C. Han, R. Shea, and M. B. Srivastava, "Sensorware: Programming sensor networks beyond code update and querying," *Pervasive and Mobile Computing*, vol. 3, no. 4, pp. 386–412, Aug. 2007.

[29] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys '05)*. New York, NY, USA: ACM, 2005, pp. 255–267.

[30] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys '03)*. New York, NY, USA: ACM, 2003, pp. 1–13.

[31] C. Borcea, C. Intanagonwiwat, P. Kang, U. Kremer, and L. Iftode, "Spatial programming using smart messages: Design and implementation," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 690–699.

[32] K. Römer, C. Frank, P. J. Marrón, and C. Becker, "Generic role assignment for wireless sensor networks," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop (EW 11)*. New York, NY, USA: ACM, 2004.

[33] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "Programming wireless sensor networks with the teenylime middleware," in *Proceedings of the 8th ACM/IFIP/USENIX international conference on Middleware (MIDDLEWARE 2007)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 429–449.

[34] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[35] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 52 –59, Aug 2001.

[36] G. Mainland, L. Kang, S. Lahaie, D. C. Parkes, and M. Welsh, "Using virtual markets to program global behavior in sensor networks," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop (EW 11)*. New York, NY, USA: ACM, 2004.

[37] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Computing Surveys (CSUR)*, vol. 43, no. 3, pp. 19:1–19:51, Apr. 2011.

[38] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 122–173, March 2005.

[39] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Proceedings of the 2nd International Conference on Mobile Data Management (MDM 2001)*, January 2001, pp. 3–14.

[40] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in *Proceedings of the 10th International Conference on Ar-*

chitectural Support for Programming Languages and Operating Systems (ASPLOS-X 2002), 2002, pp. 85–95.

[41] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004, pp. 29–42.

[42] T. Imieliński and S. Goel, "Dataspace—querying and monitoring deeply networked collections in physical space," in *Proceedings of the 1st ACM international workshop on Data engineering for wireless and mobile access(MobiDe 1999)*, 1999, pp. 44–51.

[43] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood, "Envirotrack: towards an environmental computing paradigm for distributed sensor networks," in *Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, 2004, pp. 582–589.

[44] F. Zhao and L. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, July 2004.

[45] J. Liu, M. Chu, J. Reich, and F. Zhao, "State-centric programming for sensor-actuator network systems," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 50–62, Oct.-Dec. 2003.

[46] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proceedings of the 1st international conference on Mobile systems, applications and services (MobiSys 2003)*, 2003, pp. 187–200.

[47] R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using kairos," in *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2005)*, 2005, pp. 126–140.

[48] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan, "Reliable and efficient programming abstractions for wireless sensor networks," *ACM SIGPLAN Notices*, vol. 42, no. 6, pp. 200–210, June 2007.

[49] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, "The abstract task graph: a methodology for architecture-independent programming of networked sensor systems," in *Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services (EESR '05)*. Berkeley, CA, USA: USENIX Association, 2005, pp. 19–24.

[50] G. Mainland, G. Morrisett, M. Welsh, and R. Newton, "Sensor network programming with flask," in *Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys '07)*. New York, NY, USA: ACM, 2007, pp. 385–386.

[51] K. Terfloth and J. Schiller, "Efficient configuration and control of sanets using facts," in *Proceedings of the 1st ACM international workshop on Heterogeneous sensor and actor networks (HeterSanet '08)*. New York, NY, USA: ACM, 2008, pp. 9–16.

[52] U. Bischoff and G. Kortuem, "Life cycle support for sensor network applications," in *Proceedings of the 2nd international workshop on Middleware for sensor networks (MidSens '07)*. New York, NY, USA: ACM, 2007, pp. 1–6.

[53] M. J. Ocean, A. Bestavros, and A. J. Kfoury, "snBench: programming and virtualization framework for distributed multitasking sensor networks," in *Proceedings of the 2nd international conference on Virtual execution environments (VEE '06)*. New York, NY, USA: ACM, 2006, pp. 89–99.

[54] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz, "A distributed coordination framework for wireless sensor and actor networks," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2005)*. New York, NY, USA: ACM, 2005, pp. 99–110.

[55] B. J. Bonfils and P. Bonnet, "Adaptive and decentralized operator placement for in-network query processing," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 389–409, 2004.

[56] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: a wireless sensor network testbed," in *Proceedings of the 4th international symposium*

on Information processing in sensor networks (IPSN '05). Piscataway, NJ, USA: IEEE Press, 2005.

[57] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: A robotic wireless and sensor network testbed," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, Apr 2006, pp. 1 –12.

[58] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06).* New York, NY, USA: ACM, 2006, pp. 15–28.

[59] J.-H. Huang, S. Amjad, and S. Mishra, "Cenwits: a sensor-based loosely coupled search and rescue system using witnesses," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys '05).* New York, NY, USA: ACM, 2005, pp. 180–191.

[60] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han, "Dustminer: troubleshooting interactive complexity bugs in sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08).* New York, NY, USA: ACM, 2008, pp. 99–112.

[61] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: tracking energy in networked embedded systems," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation (OSDI'08).* Berkeley, CA, USA: USENIX Association, 2008, pp. 323–338.

[62] M. Woehrle, C. Plessl, J. Beutel, and L. Thiele, "Increasing the reliability of wireless sensor networks with a distributed testing framework," in *Proceedings of the 4th workshop on Embedded networked sensors*, ser. EmNets '07. New York, NY, USA: ACM, 2007, pp. 93–97.

[63] H. Liu, T. Roeder, K. Walsh, R. Barr, and E. G. Sirer, "Design and implementation of a single system image operating system for ad hoc networks," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services (MobiSys '05).* New York, NY, USA: ACM, 2005, pp. 149–162.

[64] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: a vector-based macroprogramming framework for cyber-physical systems," in *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08)*. New York, NY, USA: ACM, 2008, pp. 225–238.

[65] A. Pathak and V. K. Prasanna, "Energy-efficient task mapping for data-driven sensor network macroprogramming," in *Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems (DCOSS '08)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 516–524.

[66] R. Newton, G. Morrisett, and M. Welsh, "The regiment macroprogramming system," in *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN '07)*. New York, NY, USA: ACM, 2007, pp. 489–498.

[67] D. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica, "The design and implementation of a declarative sensor network system," in *Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys '07)*. New York, NY, USA: ACM, 2007, pp. 175–188.

[68] K. Whitehouse, F. Zhao, and J. Liu, "Semantic streams: a framework for composable semantic interpretation of sensor data," in *Proceedings of the Third European conference on Wireless Sensor Networks (EWSN'06)*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 5–20.

[69] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: a neighborhood abstraction for sensor networks," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services (MobiSys '04)*. New York, NY, USA: ACM, 2004, pp. 99–110.

[70] L. Mottola and G. P. Picco, "Logical neighborhoods: a programming abstraction for wireless sensor networks," in *Proceedings of the Second IEEE international conference on Distributed Computing in Sensor Systems (DCOSS'06)*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 150–168.

[71] M. Hossain, A. Alim Al Islam, M. Kulkarni, and V. Raghunathan, "μsetl: A set based programming abstraction for wireless sensor networks," in *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN 2011)*, Apr 2011, pp. 354 –365.

[72] P. Ciciriello, L. Mottola, and G. P. Picco, "Building virtual sensors and actuators over logical neighborhoods," in *Proceedings of the international workshop on Middleware for sensor networks (MidSens '06)*. New York, NY, USA: ACM, 2006, pp. 19–24.

[73] F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, and M. Sgroi, "Spine: a domain-specific framework for rapid prototyping of wbsn applications," *Software - Practice and Experience (SPE)*, vol. 41, no. 3, pp. 237–265, Mar. 2011.

[74] T. Bourdenas and M. Sloman, "Starfish: policy driven self-management in wireless sensor networks," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '10)*. New York, NY, USA: ACM, 2010, pp. 75–83.

[75] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, "Ear-phone: an end-to-end participatory urban noise mapping system," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '10)*. New York, NY, USA: ACM, 2010, pp. 105–116.

[76] N. D. Lane, Y. Xu, H. Lu, S. Hu, T. Choudhury, A. T. Campbell, and F. Zhao, "Enabling large-scale human activity inference on smartphones using community similarity networks (csn)," in *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. New York, NY, USA: ACM, 2011, pp. 355–364.

[77] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application," in *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08)*. New York, NY, USA: ACM, 2008, pp. 337–350.

[78] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Soundsense: scalable sound sensing for people-centric applications on mobile phones," in *Proceedings of the 7th international conference on Mobile systems, applications, and services (MobiSys '09)*. New York, NY, USA: ACM, 2009, pp. 165–178.

[79] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, "Peir, the personal environmental impact report, as a platform for participatory sensing systems research," in *Proceedings of the 7th international conference on Mobile systems, applications, and services (MobiSys '09)*. New York, NY, USA: ACM, 2009, pp. 55–68.

[80] D. Chu, N. D. Lane, T. T.-T. Lai, C. Pang, X. Meng, Q. Guo, F. Li, and F. Zhao, "Balancing energy, latency and accuracy for mobile sensor data classification," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*. New York, NY, USA: ACM, 2011, pp. 54–67.

[81] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *Proceedings of the 7th international conference on Mobile systems, applications, and services (MobiSys '09)*. New York, NY, USA: ACM, 2009, pp. 179–192.

[82] R. Fakoor, M. Raj, A. Nazi, M. Di Francesco, and S. K. Das, "An integrated cloud-based framework for mobile phone sensing," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12)*. New York, NY, USA: ACM, 2012, pp. 47–52.

[83] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: a programming framework for crowd-sensing applications," in *Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12)*. New York, NY, USA: ACM, 2012, pp. 337–350.

[84] Y. Lee, Y. Ju, C. Min, S. Kang, I. Hwang, and J. Song, "Comon: cooperative ambience monitoring platform with continuity and benefit awareness,"

in *Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12)*. New York, NY, USA: ACM, 2012, pp. 43–56.

[85] "PhoneGap," http://phonegap.com/.

[86] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "Prism: platform for remote sensing using smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*. New York, NY, USA: ACM, 2010, pp. 63–76.

[87] T. Kaler, J. P. Lynch, T. Peng, L. Ravindranath, A. Thiagarajan, H. Balakrishnan, and S. Madden, "Code in the air: simplifying sensing on smartphones," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*. New York, NY, USA: ACM, 2010, pp. 407–408.

[88] X. Bao and R. Roy Choudhury, "Movi: mobile phone based video highlights via collaborative sensing," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*. New York, NY, USA: ACM, 2010, pp. 357–370.

[89] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell, "Darwin phones: the evolution of sensing and inference on mobile phones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*. New York, NY, USA: ACM, 2010, pp. 5–20.

[90] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*. New York, NY, USA: ACM, 2010, pp. 255–270.

[91] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08)*. New York, NY, USA: ACM, 2008, pp. 323–336.

[92] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, "Micro-blog: sharing and querying content through mobile phones and social participation," in *Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys '08)*. New York, NY, USA: ACM, 2008, pp. 174–186.

[93] H. Ishizuka, S. Fukumoto, T. Nishimoto, R. Fukuhara, T. Morita, K. Sugo, N. Thepvilojanapong, S. Konomi, K. Sezaki, R. Shibasaki, and Y. Tobe, "Kitokito: supporting impromptu collaboration in participatory sensing using smart camera phones," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*. New York, NY, USA: ACM, 2010, pp. 377–378.

[94] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "Anonysense: privacy-aware people-centric sensing," in *Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys '08)*. New York, NY, USA: ACM, 2008, pp. 211–224.

[95] H. Lu, N. D. Lane, S. B. Eisenman, and A. T. Campbell, "Bubble-sensing: A new paradigm for binding a sensing task to the physical world using mobile phones," in *Proceedings of International Workshop on Mobile Devices and Urban Sensing (MODUS '08)*, 2008.

[96] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 6–28, December 2004.

[97] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A geographic hash table for data-centric storage in sensornets," in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, September 2005, pp. 78–87.

[98] "D-sense web." http://www-higashi.ist.osaka-u.ac.jp/software/WSN/D-sense/.

[99] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Models and solutions for radio irregularity in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 2, pp. 221–262, 2006.

[100] S. Clarke, "Measuring API usability," *Dr. Dobb's Journal*, vol. 29, pp. S6–S9, 2004.

[101] Y. Ko and N. Vaidya, "Location-Aided Routing (LAR) in mobile ad hoc networks," *Wireless Networks*, vol. 6, pp. 307–321, 2000.

[102] Space-Time Engineering, "Scenargie base simulator," http://www.spacetime-eng.com/.

[103] S. Mori, T. Umedu, A. Hiromori, H. Yamaguchi, and T. Higashino, "Data-Centric Programming Environment for Cooperative Applications in WSN," in Press.