Scalable and Efficient Tasking for Dynamic Sensor Networks

by

Thanh Xuan Dang

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

Dissertation Committee:
Nirupama Bulusu
Wu-chang Feng
Wu-chi Feng
Yih-Chyun Jenq
Suresh Singh

Portland State University
© 2011

UMI Number: 3468959

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent on the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI 3468959

Copyright 2011 by ProQuest LLC.

All rights reserved. This edition of the work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Abstract

Sensor networks including opportunistic networks of sensor-equipped smartphones as well as networks of embedded sensors can enable a wide range of applications including environmental monitoring, smart grids, intelligent transportation, and healthcare. In most real-world applications, to meet end-user requirements, the network operator needs to define and update the sensors' tasks dynamically, such as updating the parameters for sensor data collection or updating the sensors' code.

Tasking sensor networks is necessary to reduce the effort in programming sensor networks. However, it is challenging due to dynamics and scale in terms of number of nodes, number of tasks, and sensing regions of the networks. In addition, tasking sensor networks must also be efficient in terms of bandwidth, latency, energy consumption, and memory usage.

This dissertation identifies and addresses the problems of scalability and efficiency in tasking sensor networks. The first challenge in tasking sensor networks is to define a mechanism that represents multiple tasks and sensor groups efficiently taking into account the heterogeneity and mobility of sensors deployed over a large geographical region. Another challenge in tasking sensor networks in general, and embedded sensor networks in particular, is to design protocols that can not only efficiently disseminate tasks but also maintain a consistent view of the task to be performed among inherently unreliable and resource-limited sensors.

We believe that a scalable and efficient tasking framework can greatly benefit the development and deployment of sensor network applications. Our thesis is that decoupling the task specification from task implementation using a spatial two-dimensional (2D) representation of a tasking region such as maps enables scalable, efficient, and resource-adaptive tasking over heterogeneous mobile sensor networks. In addition, reducing overhead in detecting inconsistencies across nodes enables scalable and efficient task dissemination and maintenance.

We present the design, implementation, and evaluation of Zoom, a multi-resolution tasking framework that efficiently encapsulates multiple tasks and sensor groups for sensor networks deployed in a large geographical region. The key ideas in Zoom are (i) decoupling task specification and task implementation to support heterogeneity, (ii) using maps for representing spatial sensor groups and tasks to scale with the number of sensor groups and sensing regions, and (iii) using image encoding techniques to reduce the map size and provide adaptation to sensor platforms with different resource capabilities.

We present the design, implementation, and evaluation of our protocol, DHV, which efficiently disseminates task content and ensures that all nodes have up-to-date task content in sensor networks. It achieves this by minimizing both the redundant information in each message and the number of transmitted messages in the networks. DHV has been included in the official distribution of TinyOS, a popular operating system for embedded sensor networks.

As sensor networks continue to develop, they will evolve from dedicated and single-purpose systems to open and multi-purpose large scale systems. Nodes in the network will be retasked frequently to support multiple applications and multiple users. We believe that this work is an important step in enabling seamless interaction between users and sensor networks and to make sensor networks more widely adopted.

To my parents, Truat Duong, Hong Tran, Cuc Le,

my wife, Han Tran,

and my children, Isabel and Jeff Dang

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Nirupama Bulusu, for her guidance throughout the program. Looking back over the years, Nirupama was the first person who introduced me to research when I worked with her on a research project in Australia as an undergraduate intern. Now, as I am almost at the end of the Ph.D. program, I feel extremely lucky that I have met her and worked under her guidance. She has patiently nurtured me to develop intellectually and professionally. I still remember her saying once that "advising a new student is like adopting a new child", a perspective that I will try to carry on in my career. I am indebted to her for all the great things she has done for me.

Similarly, I would like to thank Professor Wu-chi Feng, a very thoughtful mentor and skillful foosball coach. I was naturally drawn toward Wu-chi because of his openness and consideration. He always listens to me and gives me a broader view of research problems. My philosophical thinking and foosball skills are way better now than before I met Wu-chi.

I also would like to express my appreciation to Professor Suresh Singh, Professor Wu-chang Feng, Professor Feng Liu, and Professor Yih Chin Jenq for serving on my dissertation committee.

During my stay at Portland State University, I met many people with whom I had fruitful discussions. Among them are Akshay Dua, Phillip Sitbon, John Kassebaum, Francis Chang, Ed Kaiser, Chris Chambers, and many others.

I would like to thank Professor Sanjay Jha and Professor Chun Tung Chou from the University of New South Wales and Dr. Wen Hu from The Commonwealth Scientific and Industrial Research Organisation (CSIRO) for providing me feedback on various research projects.

I would like to thank Professor Antonio Baptista, Professor Yinglong Zhang, and Dr. Sergey Frolov at the National Science Foundation Science and Technology Center for Coastal Margin Observation and Prediction for providing me support during the time I worked there.

Especially, I would like to thank my wife, Han, for her patience and constant support throughout my graduate study. I am indebted to her for all her sacrifices. I just want you to know how much I appreciate it. I love you.

Last but most important, I thank God - Jesus Christ - for having a wonderful plan for me and my family. He showed me the peace, happiness, and the ultimate purpose of life. He brought many wonderful people like Hoa Nguyen, Tin Nguyen, Dao Le, the Lam's family, Triet Hue, Dat Tran, Vang Da, Dai Ngu, Thao Minh, Hao Tran, Dat Huynh, Dung Mai, Khanh Nguyen, Chau and Debi, Thu Huyen, Thang Hang, Long Tho, Huynh Nghia, Vu Tran, and many others who helped me and my family get through difficult times in life.

Thank you,

Thanh Dang

Portland, Oregon

May 2011

I would like to acknowledge the grants that supported my research at Portland State University. My research was supported by funding from the National Science Foundation (NSF) through grant 01-21475 (through the NSF Science and Technology Center for Coastal Margin Observation and Prediction) and grants 05-14818, 07-22063, and 0747442.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation: Multi-purpose Multi-user Sensing Systems

Recent advances in micro-electro-mechanical systems technology, wireless communication, and digital electronics have enabled the development of sensor networks which consist of many miniature sensor devices with integrated computation, communication, and sensing. These sensors can be attached to physical objects or deeply embedded into the environment to pervasively instrument the physical world. Each individual sensor can perform *tasks*, or a set of sensing operations. Collectively, tasks performed by multiple sensors accomplish the end-user's objectives. For example, a sensor node embedded in a car on a freeway can perform the task of reporting Global Positioning System (GPS) location traces to a base station. Collectively, data gathered from multiple such nodes can be used to estimate traffic flows. Sensor networks, therefore, can enable a wide range of useful applications including environmental monitoring [3], smart grids [4], intelligent transportation [5], and healthcare [6].

Since the late 1990s, many sensor networks have been deployed in a wide range of environments such as human bodies [6], buildings [7], volcanoes [8], and urban regions [9]. The networks have spanned in size from hundreds to thousands of stationary and mobile nodes [10]. Most sensor networks, however, have been deployed for a *single purpose* and are controlled only by network

operators. For example, in [3], a sensor network was deployed for the purpose of collecting ambient data for monitoring environmental conditions in a vineyard.

As sensor networking technologies continue to develop, the notion of creating *multi-purpose multi-user* sensing systems is becoming feasible. Technological advances enable sensor platforms to be more powerful in terms of computation and communication [11]. People have incentives to actively contribute sensing data and use sensor networks for multiple purposes [10, 12, 13]. For example, in the Mobile Millennium project [10], people contribute their GPS data for estimating traffic flows to make smart commuting decisions. Therefore, it is possible to share the same sensor network to support multiple applications for different users. For example, sensors deployed over large geographical urban regions [13] could be used by different users for different purposes such as noise monitoring and traffic monitoring.

To enable multi-purpose multi-user sensing systems, sensor networks must be able to update their tasks dynamically. The sensor networks will often need to perform different tasks over different spatial-temporal regions depending on the dynamics of the phenomena being studied. For example, using a mobile sensor network, the Oregon Department of Environmental Quality (DEQ) may want to collect air quality information such as CO_2 concentration in a specific region (e.g., Southeast region) in the city of Portland. The Oregon Intelligent Transportation Systems (ITS) may want to collect GPS data from all roads and freeways in the city of Portland. In this case, the network needs to perform two tasks simultaneously; collecting air quality data in Southeast Portland and collecting GPS data from nodes on all roads and freeways in Portland. In addition, the ITS may be receiving many GPS traces from nodes on US-26 but very little GPS data from nodes on Cornell Road. The ITS may want to signal the nodes on Cornell Road to increase reporting frequency, and signal the nodes

on US-26 to reduce the reporting frequency. This example illustrates how the end-user often needs to task the sensor network differently or to adapt its tasks differently across spatial-temporal regions.

While it may be possible to physically gather the sensor nodes, manually reprogram individual nodes, and redeploy them to update new sensors' tasks, it may be impractical to do so for several reasons. First, physical access to sensor nodes may not be available. Sensor networks can be deployed in unattended and hostile environments such as volcanoes. Physical access to these nodes can be life threatening. Second, data users may not have ownership of sensor nodes. For example, smartphones are personally owned and operated by users. Finally, it may not be scalable in terms of either the network size or the area over which sensors are deployed. Sensor networks can have many sensor nodes. Even if there are few nodes, they may be spread over a large area. It is labor intensive to manually collect, reprogram, and redeploy the nodes. Thus, tasking these sensor networks remotely through wireless communication is a convenient way of changing their operation.

Despite its critical importance, tasking has not been widely addressed in prior research. Previous approaches to tasking sensor networks either do not scale well to multiple tasks [14, 15] or do not scale well to large sensing regions [16], consuming significant network resources such as energy and bandwidth while incurring significant latency. In the next section, we describe the problem of scalability and efficiency in tasking sensor networks.

1.2 The Problem: Scalable and Efficient Tasking

We believe that there is a need for a scalable and efficient tasking framework to adapt the tasks performed by sensors according to user requirements in dynamic sensor networks. In particular, tasking sensor networks must be scalable

with the number of nodes, number of tasks, and sensing regions. The emergence of smartphones provides powerful generic sensor platforms that can be massively deployed over large geographical regions. Such sensing infrastructure can potentially support multiple sensing applications. However, it is important to recognize that such sensing infrastructure will be highly dynamic, in terms of when nodes may join or leave the network as well as how nodes are distributed and used over a sensing area.

Tasking sensor networks must be efficient in terms of bandwidth, energy, memory, and latency. Although advances in wireless technologies can allow nodes to achieve a high data rate in wireless communication, sensor nodes still share the communication medium. Therefore, the bandwidth per node is inversely proportional to the number of nodes in the network. With high node density, bandwidth becomes the limited resource. In addition, energy is limited in many scenarios. Sensors often run on batteries, which have a finite amount of energy, and are often deployed in unattended environments. In most cases, it is impractical to replace or recharge the batteries. Although there have been impressive technological advances in energy harvesting, the technologies must be used in conjunction with aggressive energy management on the device. Finally, during tasking, the network may become useless if some, but not all the nodes perform the new tasks. Therefore, tasking latency should be minimized to improve network performance.

The purpose of this dissertation is to *identify and address problems of scalability and efficiency in tasking sensor networks.* We believe this work constitutes an important step in making sensor networks more widely adopted.

1.3 Challenges

To help the reader understand the challenges in tasking sensor networks, we briefly present an overview of a tasking framework in the following section.

1.3.1 Tasking Framework Overview

There are often several steps in tasking a sensor network as illustrated in Figure 1.1. First, based on either the application requirements or detection of an event of interest, a user constructs a task, which can be a change in the network operation (e.g., set the light sampling rate to 10 Hz for one hour). The task is sent to a gateway device (either directly or through the Internet) which can communicate with both the sensor nodes and the user. The gateway disseminates the task to all nodes in the network. The nodes in the network, upon receiving the task information, perform the necessary operations (defined in the task implementation) to accomplish the task. The nodes also communicate with each other or with the gateway to ensure that all of them have the same up to date task[1].

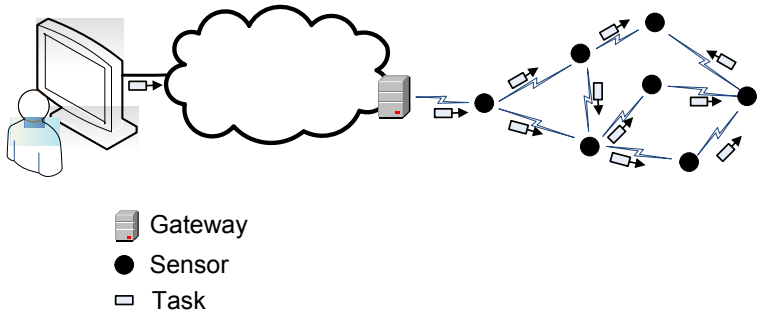


Figure 1.1: Key steps in tasking sensor networks: A user creates a task and sends the task information to a gateway device that can communicate with both the sensor nodes and the user. The gateway disseminates the task to all nodes in the network. The nodes in the network, upon receiving the task, perform the appropriate operations to accomplish the task. They also make sure that all nodes have the same up to date task.

[1]The communication could leverage a particular wireless technology, such as Zigbee [17], WiFi, 3G, and 4G LTE, depending on the capabilities of deployed sensor nodes.

A tasking framework often consists of four main components: *task implementation*, *task representation and encoding*, *task dissemination*, and *task maintenance*. *Task implementation* defines the set of operations that a sensor, upon receiving a task, performs. For example, the implementation of a task that has ID 1 (task 1) can set the GPS sampling rate to 10 Hz for one hour. Task implementation can be platform specific. *Task representation and encoding* defines the mechanism to represent specifications of tasks and their corresponding sensor groups. For example, a task specification can be collecting noise level at 10 Khz and the task is assigned to all nodes that are in a circle of 1 mile radius from a factory. One way to denote this is to have pre-defined attributes that indicate that the sensing region type is a circle, its origin is the factory, and its radius is 1 mile. *Task dissemination* provides communication schemes to efficiently transmit the tasks to the sensors. For example, the tasks are transmitted to all mobile sensors (e.g., smartphones) through a 3G network. *Task maintenance* ensures that sensors have a consistent view of the tasks being performed. For example, if a smartphone was turned off during the last task dissemination and now is turned on; it needs to update itself to the latest task that was disseminated.

Since the task implementation is platform specific, we assume that it exists. In this dissertation, we focus on task representation and encoding, and task dissemination and maintenance in sensor networks. To facilitate the research of this dissertation, we explore the problem of scalable and efficient tasking as follows. First, we study the problem of scalable and efficient task representation and encoding over large areas. We motivate and study this problem with the example of participatory networks of mobile smartphones (mobile sensor networks). Next, we study the problem of scalable and efficient task dissemination and maintenance with the example of embedded sensor networks which

consist of small form factor embedded sensor devices with limited resources such as energy, memory, and bandwidth. The following sections describe the main challenges of each problem as well as the justification for the network type chosen to study each problem.

1.3.2 Challenges in Task Representation over Large Areas

Mobile devices equipped with sensors can enable many applications. For example, GPS-enabled smartphones can report GPS location to estimate traffic flow [18]. Smartphones can be used to capture audio signals for monitoring noise pollution [9]. Multiple users such as the ITS and the DEQ might want to query information from a mobile sensor network to monitor traffic conditions and air pollution respectively. Hence, tasking mobile sensor networks must support multiple users and multiple applications. Basically, sensor nodes are segregated into multiple groups. Each group is assigned a task. Therefore, we focus on the key problem of *how to design a mechanism that can efficiently represent multiple tasks and multiple sensor groups over large areas*? A solution to this problem must address the following primary challenge.

Large Sensing Region

Mobile sensors such as smartphones can cover a large geographical region (e.g., a city). The number of nodes can be tens of thousands. The tasks can be performed either over a large region or a specific area with fine resolution. Therefore, the tasking framework should be flexible enough to task sensor nodes at different geographical resolutions depending on the application requirements.

Additionally, it must also address the following challenges in mobile sensor networks.

Mobililty

In mobile sensor networks, the nodes are often carried by humans or attached to physical objects like cars or trains. Hence, the nodes can be either stationary or moving at a speed of tens of miles per hour. For example, in the Mobile Millennium project [10], cell phones deployed on cars are leveraged to collect GPS data in the San Francisco Bay Area for traffic estimation. The cars can join and leave a sensing region frequently. These nodes may not be aware of the task associated with the sensing region. Therefore, it is important for a tasking framework to ensure that nodes know what they are supposed to do based on its current location.

Heterogeneity

Although mobile sensors such as smartphones have similar features, they differ vastly in the degree of their capabilities. For example, most smartphones have a camera, microphone, GPS and 3G capabilities. However, the processor speed can range from 400 MHz to 1 GHz. The camera can capture images from a resolution of 1.3 mega pixels to 12 mega pixels. The memory can range from 128 MB to 32 GB. The batteries also have different capacity ranging from 500 mAh to 3000 mAh. In addition, there are also a variety of operating systems supporting these platforms. The diversity in the smartphone platforms and their software requires that protocols adapt to devices with different capabilities.

1.3.3 Challenges in Disseminating Many Tasks Efficiently

We explore the problem of disseminating multiple tasks concurrently. Our example scenario is an embedded sensor network. Nodes in an embedded sensor network often perform the same task, but the task may change over time. The changes can be updating *task items* such as sensor firmware or sensing parameters. The update can happen periodically (e.g., daily or monthly) depending on the task items being updated. For example, updating sensor firmware might happen monthly but updating sensing parameters (e.g., light sampling rate) might happen hourly. Moreover, multiple tasks may need to be updated concurrently. Nodes in sensor networks often need to communicate with each other periodically to ensure that they all have the same up to date tasks. The cost in terms of energy and bandwidth of ensuring that nodes have the same updated tasks may overwhelm the cost of sensing itself [14]. Therefore, we focus on the key problem in tasking sensor networks; *how to disseminate multiple task items efficiently and make sure all nodes have the updated items?* A solution to this problem must address the following primary challenge.

Unreliable and Intermittent Operation

Embedded sensors' operations are often intermittent. In embedded sensor networks, sensors can be deployed in large numbers in various environments, including remote regions [3], hostile regions [19], and operating without human attendance. In many real deployments [20, 21], sensor nodes' operations are intermittent; nodes are on and off in an unpredictable way. That means even if tasking is successful in updating the new task to all the nodes that are on, some nodes that are off during the update may not be aware of the new task when they wake up. In addition, packet loss rates in wireless embedded sensor networks are high (e.g., from 5% to 10%) and distance dependent [22]. Fre-

quent sensor failure and intermittent communication make network behavior unpredictable. This makes task dissemination and maintenance in embedded sensor networks difficult. Additionally, in embedded sensor networks, we must address the following challenge.

Limited Resources

Energy is the scarcest resource of embedded sensors and it determines the lifetime of sensor networks. For example, a MicaZ sensor [23], which runs on two AA batteries, has a normal lifetime of up to only 30 days [24]. While energy harvesting is possible, the technologies are not yet applicable for small form-factor low power sensor platforms. Therefore, protocols in embedded sensor networks must conserve energy. For tasking protocols, one way to disseminate tasks and maintain consistency among sensors is to periodically broadcast messages containing the tasks' information. However, communication consumes a dominant amount of energy in embedded sensor networks. Hence, the energy consumed for maintaining task consistency in the networks may outweigh the energy consumed for sensing itself.

1.4 Solutions: Task Representation, Dissemination and Maintenance

We believe that a scalable and efficient tasking framework can greatly benefit the development and deployment of sensor network applications. Our thesis is that *decoupling the task specification from the task implementation using a spatial two-dimensional (2D) representation of the tasking region (e.g., maps) enables scalable, efficient, and resource-adaptive task representation over large areas and reducing overhead in detecting inconsistencies across nodes enables scalable and efficient dissemination and maintenance over large numbers of tasks.*

1.4.1 Solution #1: Spatial 2D Task Representation

Most prior approaches for representing tasks in sensor networks are declarative; they define precisely what a network should accomplish without describing the detailed instructions that the nodes should perform. These approach can be *attribute-based* or *rule-based*. In *attribute-based* approaches [25], the sensor network is typically regarded as a database. Queries defined using variants of the structured query language (SQL) are used to task sensors to report data and define sensor groups. For example, to select all light readings from sensors that have their IDs greater than 10, the complete query can be:

```
SELECT light
FROM sensors
WHERE id > 10
```

In *rule-based* approaches [26], sensor groups are defined by a set of rules, which can be considered as an *admission function* [11]. A sensor, whose state including sensing capability, location, or sensed data value, satisfies the rules is a member of the defined group. For example, the admission function returns true if the node energy level is greater than 500 mAh. What is missing in prior work [16] is the ability to specify tasks for multiple sensor groups efficiently while providing adaptation for sensor platforms with varying resource capabilities.

We believe that decoupling the task specification from the task implementation using a spatial 2D representation of the tasking region (e.g., maps) enables scalable, efficient, and resource-adaptive task representation over large areas.

The task specification contains the task ID, a unique number identifying the task, and task header which can contain grouping information. Task implementation is platform specific and contains specific instructions to be executed by sensor nodes. It can be preloaded into the sensor nodes or downloaded as needed.

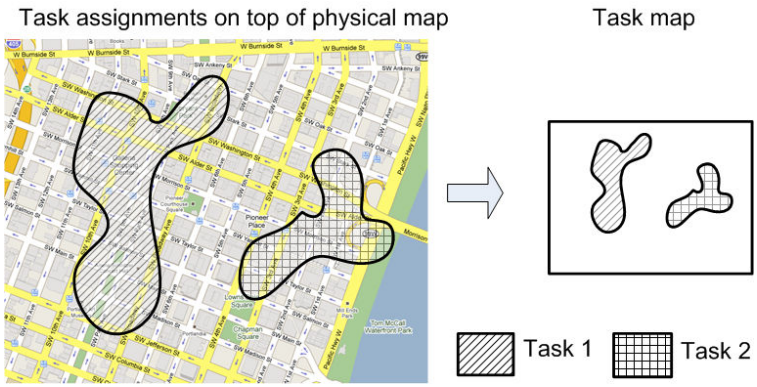


Figure 1.2: A task map overlaid on top of a physical topology: Physical groups of sensors can be viewed as a region in the image. Each pixel in the image represents a squared region in the physical map and the pixel value is the task ID. A node upon receiving a task map can calculate the value of the pixel in the map that corresponds to its physical location to know which task it should perform.

Task maps can be viewed as an image overlaid on top of a physical map as shown in Figure 1.2. Physical groups of sensors can be viewed as a region in the image. Each pixel in the image represents a square region in the physical map and the pixel value is the task ID. A node upon receiving a task map can calculate the value of the pixel in the map that corresponds to its physical location to know which task it should perform. This approach allows us to represent multiple groups and their tasks efficiently in one map. In addition, image encoding techniques can be applied to reduce the map file size as well as to provide resource adaptation for heterogeneous sensor platforms.

1.4.2 Solution #2: Task Dissemination and Maintenance

We focus on the problem of disseminating task items and making sure that all nodes have the updated items. This is often accomplished by dissemination and maintenance protocols that spread the items to all nodes efficiently and ensure that every node has the updated items. The key step is to find out which node has items that are different from the other nodes to trigger the update.

A tuple *(key, version number)* is often used to represent a task item where the *key* uniquely identifies the item and the *version number* indicates the freshness of the item; the greater the version number is, the more up-to-date the item is. Previous approaches [15, 27] advertise the whole item version numbers for comparison. The overhead in detecting inconsistencies across nodes in the network is high in terms of the number of transmitted messages. Hence, they are not efficient and do not scale well with the number of items and the number of nodes.

We believe that reducing overhead in detecting inconsistencies across nodes enables scalable and efficient dissemination and maintenance over large numbers of tasks.

We observe that in most cases, the two version numbers, if different, differ in only a few least significant bits. Therefore, instead of advertising the whole version numbers of all items to detect an inconsistency in the network, nodes carefully select bits that are likely to be different from other nodes and combine the bits into messages, and advertise them all together to reduce the number of unnecessary bits in each message. Nodes randomly advertise messages that contain information about item version numbers (e.g., a hash of all the version numbers) within each time interval to ensure that they have up to date task items. To further reduce the number of transmitted messages, a node upon receiving several messages with the same content, suppresses its own transmission. Together, nodes in the network can reduce both the number of transmitted messages and the number of unnecessary bits in each message.

1.5 Contributions

As described in Section 1.3, the problems of scalable and efficient tasking in sensor networks have distinct challenges in scaling to large areas and large

numbers of tasks. We recognize this and focus on addressing these key problems in tasking using the examples of mobile sensor networks and embedded sensor networks. For the problem of task representation, our work focuses on designing a mechanism that efficiently represents multiple sensor tasks and groups over large areas. We have applied this mechanism to a mobile participatory sensor network. For the problem of task dissemination and maintenance, our work focuses on developing a scalable dissemination and maintenance protocol to efficiently distribute task items and ensure that nodes have the updated items. We have applied this protocol to embedded sensor networks. Together, these solutions can be used for scalable and efficient tasking of multi-purpose multi-user sensor networks of the future. We also believe that this work is a step toward making sensor networks more widely adopted.

The contributions of this dissertation are:

Zoom – A multi-resolution tasking framework, applied to mobile sensor networks: This framework allows users to group and assign tasks to sensors in non-uniform, fine-grained ways across a large sensing region for heterogeneous mobile sensor networks. The key ideas in Zoom are (i) decoupling task specification and task implementation to support heterogeneity, (ii) using maps for representing sensor groups and the tasks to scale with the number of nodes and sensing regions, and (iii) using image encoding techniques to reduce the map size and provide adaptation to sensor platforms with different resource capability. Zoom is more intuitive, efficient and scalable compared to previous approaches. To the best of our knowledge, Zoom is the first multi-resolution, image based tasking framework for sensor networks.

DHV – A dissemination and maintenance protocol, applied to embedded sensor networks: DHV uses a bit-level information exchange scheme to scale with the number of items, and a gossip-based communication scheme to scale

with the number of nodes. Experimental results on both simulation and real testbeds show that DHV outperforms previous protocols by a factor of two in most cases. DHV has been included in the official distribution of TinyOS, a popular operating system for embedded sensor platforms, since version 2.1.1.

1.6 Dissertation Overview

The rest of this dissertation is organized as follows. In Chapter 2, we discuss prior work on embedded sensor networks and mobile sensor networks and describe related work in tasking these networks.

Chapter 3 then presents the design, implementation, and evaluation of Zoom, a multi-resolution tasking framework for mobile sensor networks. Chapter 4 describes the design, implementation, and evaluation of DHV, an efficient dissemination and maintenance protocol for embedded wireless sensor networks.

Finally, Chapter 5 summarizes the research contribution of this dissertation, discusses remaining challenges and outlines future research directions in tasking sensor networks.

Chapter 2

Background and Related Work

In this chapter, we present an overview of prior work on tasking sensor networks, an overview of mobile sensor networks, and an overview of embedded sensor networks. We also review related work that addresses specific tasking problems (described in Sections 1.3.2 and 1.3.3) in each type of network. We use the tasking model described in Section 1.3.1 as a reference model to layout common ground for understanding previous work in the field. Most previous work in sensor network programming [28, 25] provides some support for tasking. Therefore, we include work in the broad area of network programming that is related to tasking in this chapter.

2.1 Tasking Models for Sensor Networks

As described in Section 1.3.1, a tasking framework often has four main components. *Task implementation* defines the set of operations that a sensor, upon receiving a task, performs. *Task representation and encoding* define the mechanism to represent specifications of tasks and their corresponding sensor groups. *Task dissemination* provides communication schemes to efficiently transmit the tasks to the sensors. *Task maintenance* ensures that sensors have a consistent view of the tasks being performed. We describe each of the components in detail in the following sections.

2.1.1 Task Implementation

Task implementation defines the set of operations to accomplish a task so that a sensor, upon being assigned the task, knows what to perform. Task implementation can be specified in *sensor models*, *middleware*, or *executable images*. Sensor models [29, 30] provide hardware and software abstraction for sensor devices and services. On the other hand, the middleware [31] provides task implementation and makes it available via a set of application programming interfaces (APIs). Executable images contain the actual machine instructions to be performed by the sensors to accomplish the tasks.

Sensor models are useful to develop a standard specification of tasks but do not always provide the task implementation. SensorML [29], which was developed by the Open Geospatial consortium, aims at providing a full sensor model in XML format. SensorML provides abstractions for complex sensing systems including satellites and weather stations. Tasks in SensorML can be specified using modeling languages such as MathML [32]. Sensors must have an interpreter to translate these languages into the actual machine instructions to perform the task. sMAP [30], a simple measurement and actuation profile for physical information, aims at providing a simple and efficient method for accessing and controlling devices with limited resources. Tasks in sMAP can be specified by parameters. These parameters, however, are sensor specific.

There exists middleware to provide task implementation but it is often domain specific. Since the introduction of sensor networks, various middleware have been developed [31]. The middleware can be software libraries [33], toolkits [34], or virtual machines [35]. Each middleware provides a specific set of APIs that applications can be built on top of.

Executable images contain the actual machine instructions to be performed by sensors to accomplish the tasks. Executable images are platform specific.

Hence, for different platforms that run on different operating systems, programs must be developed and compiled specifically for these platforms. For example, applications for the MicaZ platform [23] are developed in TinyOS [14] using NesC language [36] and compiled for the MicaZ platform.

Due to the various options for task implementation, we assume that a task implementation exists. Each task has a unique task ID. But different sensor platforms can have different task implementations for the task.

2.1.2 Task Representation and Encoding

Task representation and encoding define a data structure to represent tasks and their corresponding sensor groups. There are two important aspects of task representation and encoding: (i) *scoping* - which sensors are affected and (ii) *idiom* - how the tasks are described.

Task Scoping

Task scopes can be at the *network level*, *group level*, or *node level*.

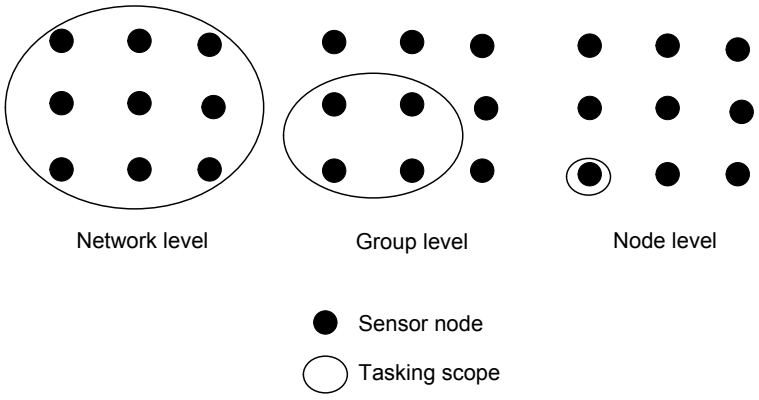


Figure 2.1: Scopes in tasking

- *Network Level*: Tasking at the network level assigns tasks to the whole network. This level of scoping often has network centric computation,

where the task execution is applied to the whole network. Examples in this category are TinyDB [25] and Cougar [37].

- *Group Level*: Tasking at the group level assigns tasks to groups. The groups can be physical [38], where the groups are defined based on physical proximity such as radio connectivity; or logical [28], where the groups are defined based on some properties such as sensor modality and sensor data. Nodes in a group, therefore, can be within a one-hop communication radius [38], multi-hop communication radii [39], or even non-connected [28]. This level of scoping often has group centric computation, where the task execution is applied to groups.

- *Node Level*: Tasking at the node level assigns tasks to individual nodes. This level of scoping often has node centric computation, where the task execution is applied to individual nodes. Examples in this category are ATaG [40] and GRA [41].

Task Idiom

The expression of tasks can be *imperative*, *declarative*, or a *hybrid of imperative and declarative*.

- *Imperative*: In this category, a task is expressed as a set of instructions that nodes must perform to accomplish the task. Example approaches in this category are Abstract Region [26] and Pleiades [42].

- *Declarative*: In this category, a task defines precisely what the nodes should accomplish without describing in detail the instructions that the nodes should perform. The task can be expressed in a domain-specific language (e.g., SQL) or in functional languages. Example approaches that use domain-specific languages are TinyDB [25] and FACTS [43].

For example, to collect all temperature readings from all sensors for 60 seconds, the user can create a query

```
SELECT temperature

FROM sensors

DURATION 60s
```

in TinyDB without specifying the details of the network routing. Example approaches that use functional languages are Regiment [44].

- *Hybrid*: In some cases [45], combining both imperative and declarative expressions not only states precisely what the tasks objective is but also provides detail on how to achieve it.

2.1.3 Task Dissemination

Task dissemination provides communication schemes to efficiently transmit the tasks from a gateway to the sensors. Task dissemination often relies on dissemination protocols that spread the tasks to all nodes or all groups of nodes in the network. In most cases [46, 26], the dissemination is not restricted in timing order. However, in some cases such as mobile agents where the code and the sensor data are able to migrate from one node to another to continue execution, tasks migrate to nodes in an order depending on the status of the execution [47].

2.1.4 Task Maintenance

Task maintenance ensures that sensors have a consistent view of the tasks being performed. This is necessary because nodes often do not have the updated tasks for several reasons. Sensor nodes might be highly mobile. Embedded sensors can be when they are unattended and exposed to harsh environments. There-

fore, some nodes may not be present in the network during the dissemination of the tasks.

Common approaches [27, 15] in task maintenance require nodes to periodically exchange messages that contain meta-information about their tasks (e.g., task version numbers) to identify which nodes do not have the updated tasks. The choice of message exchanging mechanisms as well as the meta-information can affect tasking performance significantly.

In the following sections, we describe related work in tasking mobile sensor networks and embedded sensor networks. To facilitate the understanding of tasking in each network type, we summarize background information about sensor platforms and applications of each network type. We then describe in detail prior work in tasking each network type specifically.

2.2 Mobile Sensor Networks

Mobile sensor networks consist of mobile hand-held smart devices. These devices are often carried by humans or attached to mobile objects such as cars. They can collect a wide range of sensing data including video, audio, GPS, and temperature. They can report the data to the Internet via WiFi or 3G/4G networks. Because of this, mobile phones offer an unprecedented opportunity to crowdsource sensor data collection to people. One of the challenges to doing this is the sheer diversity across mobile phones. For example, the phones can have a number of sensors including GPS, accelerometers, compass, pressure, proximity, gyroscope, and optional plug-ins for external sensors. Wireless communication technologies for smart phones range from long range and high bandwidth such as 4G to short range and low bandwidth such as near field communication (NFC).

There is a wide range of commercial operating systems for smart devices including Android, iOS, Nokia OS, Windows Phone, Symbian OS, and RIM OS. Depending on the operating system, applications can be developed in different languages. For example, Android applications are written in the Java language while iOS applications are developed in the objective-C language. Thus, the task implementation for a particular sensing task (e.g., noise pollution monitoring) would be different across different smartphone platforms.

2.2.1 Applications

In this section, we organize applications of mobile sensor networks into different domains. Table 2.1 shows that mobile sensor networks can enable many applications. However, a framework to selectively trigger each of these applications depending on the context (e.g., GPS location) is largely missing.

Application domain	Example Application	Description
Environmental monitoring	EarPhone [9] What's noisy [48] BudBurst [49]	Record noise levels and monitor plants as the seasons change
Urban monitoring	PetrolWatch [50] Bikestatic [51] Truckstop [52]	Capture gas prices, biketrack rating, and impact of stopping trucks
Health and well being	DietSense [53] Remote health monitoring [54]	Monitor food choices, provide remote access to health information
Transportation	VTrack [55] Google traffic [56] Mobile millenium [57]	Provide traffic information and estimate traffic delay
Social networking	CenceMe [58]	Capture and share personal activities on social networks

Table 2.1: Mobile sensor network applications

In the next section, we will review related work in tasking mobile sensor networks with a focus on task representation and encoding.

2.2.2 Task Representation and Encoding in Mobile Sensor Networks

Data structures for tasking mobile sensor networks must encapsulate groups of sensors and the task that each group must perform. Previous grouping approaches fall into two main categories: *attribute-based* and *rule-based* (Figure 2.2).

Cougar [37] and TinyDB [25] are examples of the attribute-based approach, wherein the sensor network is typically considered as a database. Nodes and data are named. Queries defined using variants of structured query languages (e.g., SQL) are used to task sensors to report data. The sensor groups are defined within the query. This is a preliminary approach for data collection and can only support limited in-network processing tasks. It is also difficult to define multiple groups of sensors at a fine spatial granularity.

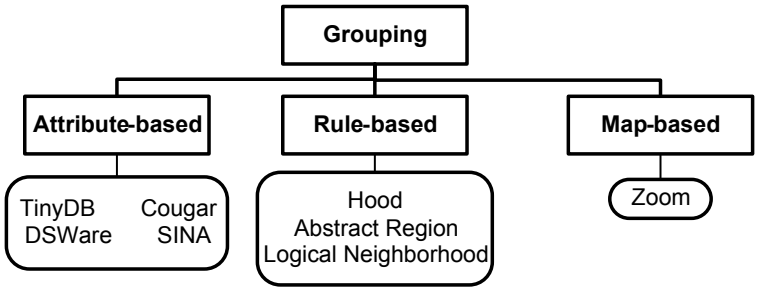


Figure 2.2: Grouping approaches

Hood [59], Abstract Regions [26], and Logical Neighborhood [28] are examples of the rule-based approach, wherein groups are often defined by a set of rules. A sensor, whose state including sensing capability, location, or sensing data, satisfies the rules is a member of the defined group. The rules may be defined based on physical parameters or logical parameters. For example, in Logical Neighborhood [28], logical sensors are specified by attributes and logical neighborhood are specified by the set of sensors satisfying a constraint on the sensors' attributes. The constraint is basically a predicate to determine if a sensor belongs to a logical group. The rule-based approaches offer greater flex-

ibility and capability in creating groups than the attribute-based approaches. However, it is still challenging to define multiple groups of sensors based on location information using the above approaches.

Zoom addresses this challenge by using a map-based approach. Essentially, the whole sensor network can be represented as a spatial map and groups can be defined on the map. The map can be viewed as an image overlaid on top of a physical map. Physical groups of sensors can be viewed as a region in the image. Each pixel in the image represents a squared region in the physical map and the pixel value is the task ID, which specifies the task being performed. This approach allows Zoom to task multiple spatial groups of sensors with varying granularity. In addition, image encoding techniques can be applied to reduce the map file size as well as to provide resource adaptation for heterogeneous sensor platforms.

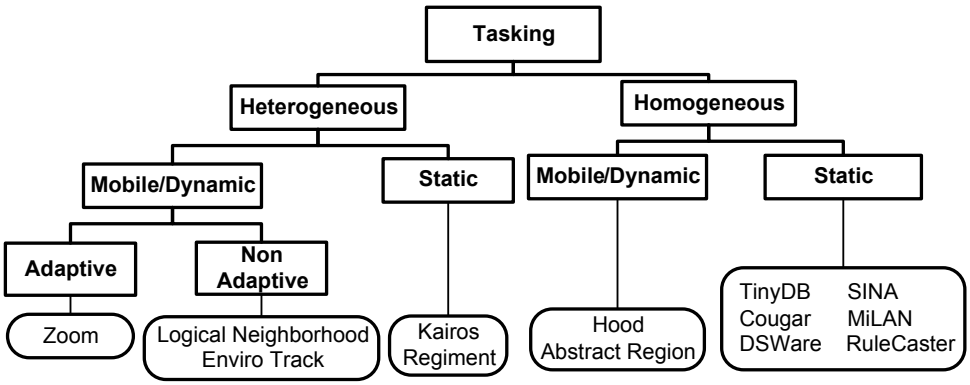


Figure 2.3: Tasking categories: None of the prior approaches provide resource adaption for heterogenous sensor platforms.

Figure 2.3 depicts an alternate view of prior work in tasking sensor networks, encompassing two main categories - supporting homogeneous networks and supporting heterogeneous networks. Many early approaches [59, 25, 26, 37] were designed for homogeneous networks. Only a few attempts [28, 60] support heterogeneous and mobile networks. Interestingly, no prior work provides adaptation to different platforms with a range of memory, computation, and

power capabilities, a distinct feature of heterogeneous networks. Our approach, presented in Chapter 3, not only supports heterogeneity and mobility but also provides resource adaptation techniques for different mobile sensor platforms.

2.3 Embedded Wireless Sensor Networks

Low-power sensor networks started attracting academic research and commercial interest in the late 90s. Since then, a wide range of hardware and software has been developed as both research prototypes [61, 62] and commercial products [63]. Most sensor platforms [23, 64, 65] use ultra low-power and low cost processors with limited memory. Much of the interest in such devices stems from their small form factor, the ability to embed them in almost any environment, ranging from redwood trees to nests of small birds. This means that the network designer must be cognizant of resource-constraints when designing network protocols. The two most popular processors are the 16-bit 25 MHz Texas Instruments MSP 430 processor and the 8-bit 8 MHz Atmel128 microcontroller. These processors are used in many sensor platforms such as MicaZ [23], Shimmer [64], and Telosb [65]. These platforms have RAM ranging from 4 KB to 64 KB, considerably lower than PCs and smartphones. The most common wireless communication is in the 2.4GHz ISM band. The physical layer and media access control standard is often IEEE 802.15.4 [66], which is specified for low-rate wireless networks (250 kbps, much lower than WiFi). Sensor nodes often run on batteries with capacity ranging from 500 mAh to 3000 mAh (if batteries can provide a current of 1 mA for 1 hour, the capacity of the batteries is 1 mAh). The lifetime of a node can range from 10 days to several months. Energy harvesting technologies [67, 68, 69] can help the nodes to harvest energy from the environment. However, the technologies are not yet suitable for small form factor devices.

There have been a number of operating systems developed for embedded sensor networks including TinyOS [70], Contiki [71], SOS [72], Mantis [73], RETOS [74], t-kernel [75], and NANO-rk [76]. These operating systems provide basic scheduling and concurrency mechanisms. The operating systems and other add-on packages [77, 78] provide a concurrency mechanism ranging from non-threading [70] to multi-threading and from event-driven to asynchronous message passing [72]. Applications can be developed and compiled with the operating system [70] or can be loaded dynamically into systems that have dynamic linking support [71]. However, mechanisms are required to disseminate the application to all nodes.

2.3.1 Applications

While new applications and use cases for embedded sensor networks continue to be explored, applications of embedded sensor networks encompass environmental monitoring, precision agriculture, smart environment, and healthcare. Table 2.2 shows the main application domains and their example applications. Currently, sensor networks are deployed for a single application. In future, the same sensor network could be used for multiple applications.

In the next section, we review related work in dissemination and maintenance protocols for low power sensor networks.

2.3.2 Dissemination and Maintenance Protocols for Embedded Sensor Networks

A sensor task consists of task items which can be configuration parameters [15, 90], code capsules [91, 92], or executable images [93, 94]. Each item is represented using a tuple *(key, version number, data)* where *key* is a unique identifier of the item, *version number* indicates the freshness of the item and

Application domain	Example application or deployment	Description
Environment monitoring	Volcano monitoring [79, 19]	Record seismic events
	Redwood [80]	Capture microclimate
Structure monitoring	Bridge monitoring [81]	Collect ambient vibration to monitor bridge
	Landslide [82]	Detect location changes to monitor landslide
Precision agriculture	Grapevine monitoring [3]	Record ambient conditions
Tracking	ZebraNet [83]	Study animal migrations and inter-species interactions
	Countersniper [84]	Track position of gun shooters
Habitat monitoring	Cane-toad monitoring [85]	Record and analyze acoustic signals to monitor cane-toads
	GreatDuck island [86]	Record ambient conditions
Smart environment	Eco-Sense Buildings [87]	Capture ambient conditions human activities to conserve energy consumption
	PecanStreet [88]	
Heathcare	Mercury [89]	Record vibration and heart-beat for health monitoring

Table 2.2: Embedded sensor network applications

is increased by 1 for each update, and *data* is the actual content of the item. Changing the task for the network basically involves updating the items for the sensors using dissemination and maintenance protocols.

Akdere et al. [95] developed a dissemination protocol for updating items based on epidemic algorithms, where each sensor upon receiving the items, retransmits the items to others until the whole network is updated. Epidemic algorithms, however, eventually terminate. A sensor that was turned off during the updating interval will not know that there was an update and it might not be updated until the next round. To know if there was an update in the network, a naive approach is for each node to query or advertise its keys and version numbers periodically. The network, as a whole, may transmit an excessive and unnecessary number of query and advertisement messages.

To reduce the number of transmitted messages, Levis et al. [27] developed the Trickle dissemination protocol based on polite gossip algorithms, where each sensor periodically transmits the items. In Trickle, each sensor periodically transmits the keys and version numbers of its items. However, if it hears several messages containing the same information as it has, it suppresses its transmission and increases the time interval. When a difference in the version numbers is detected, the sensor resets the period to the lowest preset interval. Trickle scales well with the number of sensors and has successfully reduced the number of messages in the network.

One limitation of Trickle is that it scales linearly with the total number of task items. Hence, Lin et al. [15] developed DIP, a dissemination protocol that scales logarithmically with the total number of items. In DIP, a node periodically broadcasts a summary message which contains hashes of its keys and version numbers. The use of hashes helps detect if there is a difference in O(1). But, once a difference is detected, DIP requires multiple search iterations

to identify the exact items that have different version numbers. The search is analogous to a binary search in a sorted array. Therefore, DIP has O(log(T)) complexity, where T is the number of items, in both time and the number of messages required to identify an item that needs an update. DIP uses a bloom filter to further improve the search but it requires extra bytes to be included in every summary message. If there are N new items, then the total number of messages is O(Nlog(T)).

Ideally, when there are N new items (N < T), we would like to transmit just enough information to identify these N items to update. Both Trickle and DIP transmit redundant information, O(T) and O(Nlog(T)) respectively, to identify the difference in version numbers. However, if two version numbers differ by even one bit in their binary representation, the two version numbers are different from each other.

Based on this fact, we develop the DHV protocol, elaborated in Section 4, which can detect the differences in O(T) complexity in both time and number of transmitted messages albeit with a very small factor that O(T) is almost a constant for most practical values of T. The number of messages required to identify which items have newer version numbers is O(NT) but also with a very small constant factor.

2.4 Summary

In this chapter, we presented an overview of tasking sensor networks. We discussed the general structure of tasks for sensor networks as well as the four main components (task implementation, task presentation and encoding, task dissemination, task maintenance) of a tasking framework. We then provided overviews of mobile sensor networks as well as embedded sensor networks and pointed out that the emergence of these networks can create large scale, highly

mobile and multi-purpose sensor networks that can support multiple applications. This reinforces the need for tasking sensor networks. We then described tasking approaches in mobile sensor networks with a focus on task representation and encoding and summarized that the previous approaches were not scalable for mobile sensor networks deployed over a large geographical region. We also described tasking approaches in embedded sensor networks with a focus on task dissemination and maintenance and pointed out that the prior approaches were not scalable with the number of nodes and number of task items. Therefore, existing tasking frameworks are inefficient in terms of bandwidth, latency, and energy.

We briefly described how our work, Zoom and DHV, addressed limitations in prior work in tasking mobile sensor networks and embedded sensor networks respectively. We will describe Zoom and DHV in detail in Chapter 3 and Chapter 4 respectively.

Chapter 3

Zoom: A Multi-resolution Tasking Framework for Mobile Sensor Networks

This chapter describes the design, implementation, and evaluation of the Zoom tasking framework, which efficiently encapsulates multiple tasks and sensor groups for a mobile sensor network deployed in a large geographical region. It achieves this by using a map-based approach to represent and encode tasks and sensor groups as well as provide resource adaptation techniques for different sensor platforms.

3.1 Introduction

In Chapter 1, we motivated a key problem in scalable and efficient tasking of sensor networks, that is how to define a mechanism that can efficiently represent multiple tasks and sensor groups in a large geographical region. This problem is particularly relevant in mobile sensor networks, which are often distributed over a large geographical area. We also described the main challenges that a tasking framework must address. In particular, tasking mobile sensor networks must support heterogeneity, must take into account mobility, and must scale well with the number of sensors and sensing regions.

In this chapter, we describe the Zoom tasking framework and how it addresses the above challenges in detail. In Section 3.2, we describe the design goals as well as the assumptions we make in designing Zoom. We also discuss

the communication model in Zoom and present Zoom's key ideas. In Section 3.3, we provide an overview of Zoom as well as describe in detail task representation, task encoding, and resource adaptation techniques in Zoom. We discuss the limitations of Zoom in Section 3.4 and Zoom implementation in Section 3.5. Section 3.6 presents the evaluation of Zoom including metrics, methodology, and experimental results. Finally, we conclude this chapter with a summary in Section 3.7.

3.2 Overview

In this section, we present our design goals in Section 3.2.1 and describe key ideas in Zoom that can achieve the goals in Section 3.2.2 as well as state main assumptions in Zoom in Section 3.2.3.

3.2.1 Design Goals

A tasking framework for mobile sensor networks must have a data structure that satisfies the following requirements.

- *Multiple tasks and groups*: Mobile sensor networks are often deployed over a large geographical region and support multiple applications. Hence, it is important that the tasking framework is able to assign different tasks to different sensor groups.

- *Scalability*: Mobile sensor networks can also consists of thousands of mobile devices and span a large geographical region. Hence, the framework must scale with the number of nodes and sensing regions.

- *Efficiency*: The mobile devices can move at a speed of up to tens of miles per hour. Hence, they enter and leave a sensing region frequently. In

addition, as the number of nodes increases, the nodes also need to conserve bandwidth because they share the wireless communication medium.

- *Heterogeneity*: Mobile sensor networks consist of devices encompassing different hardware and software platforms. The devices have different capabilities in terms of bandwidth, computation power, energy, and memory. Therefore, it is necessary that the tasking framework support heterogeneous sensor platforms.

3.2.2 Key Ideas

To achieve the above design goals, we employ the following key ideas in designing Zoom. These ideas correspond to our thesis in Section 1.4.1.

- *Combining sensor tasks and sensor groups into one data structure*: A naive approach in tasking sensor networks is to assign tasks to individual nodes. This approach is, however, inefficient when the network needs to support multiple tasks. Hence, combining multiple tasks and groups into a data structure can potentially enable the tasking framework to scale with the number of nodes, number of tasks, and number of sensor groups. Zoom uses maps to represent multiple spatial sensor groups and tasks.

- *Minimizing file size and decoding time*: Minimizing the file size can conserve bandwidth and memory. In addition, during tasking, the network may become useless if all nodes are not performing the correct task. Hence, the decoding time should be minimized to enable efficient tasking. Zoom uses image encoding techniques to reduce map file size.

- *Providing resource adaptation*: Mobile sensor networks are heterogeneous. Hence, it is important for a tasking framework to provide resource adaptation to nodes that have different resource capabilities to ensure that

every node can be tasked. Zoom uses image encoding techniques to provide resource adaptation for different sensor platforms.

3.2.3 Assumptions

We make the following assumptions in developing the Zoom tasking framework.

- *Location-awareness*: Sensor nodes know their own location, which can be obtained from GPS receivers or other localization methods. This assumption is reasonable because most hand-held smart devices have built-in GPS.

- *Dissemination and maintenance support*: Once a node knows what task to perform and where to get the required information (e.g., task items) to perform the task, it can acquire the needed information using existing dissemination and maintenance protocols. This assumption is reasonable because mobile devices with reasonable networking capability (e.g., 3G, 4G) can acquire the needed information from a known source.

Before describing the key ideas in Zoom in detail, we discuss the communication model in Zoom.

3.2.4 Communication Model

A simple approach for a node to determine what task it should perform is to periodically *poll* a predefined server for tasks that match its context (e.g., location and sensing capability). This approach allows the server to assign exact tasks to individual node. The drawback is that the node has to disclose its private context information such as location to the server so that the server can assign the appropriate tasks based on the released context information. This communication scheme potentially violates the privacy of the node's owner. In

addition, the node has to actively poll the server, incurring high bandwidth usage. An alternative approach is a *push-based* approach – the server periodically broadcasts the network-wide task information to the network. A node, upon receiving the task information, can use its context information such as location to derive the corresponding task without releasing the context information. Although a message containing the task information for all nodes will have a larger size compared to a message containing the task information for a single node, this communication scheme preserves privacy and allows nodes to disseminate the task information within the network and keep the whole network updated using fewer transmissions. The key challenge now is to design a data structure that efficiently represents the task information.

In the next few sections, we will describe in detail the main components of Zoom.

3.3 Zoom Tasking Framework

Zoom has three main components as illustrated in Figure 3.1: *task representation*, *task encoding*, and *resource adaptation*. Task representation focuses on designing the data structure that can specify sensor groups and assigning tasks to the groups. Task encoding focuses on how to compress the task data structure to reduce the data size. Finally, resource adaptation enables tasking for sensor devices with different resource capabilities.

Task representation and encoding is performed at the back end where an operator can define geographical regions and assign tasks to each region. The task IDs with the location information are represented as a task map; a location on the map corresponds to a real physical location and the pixel value at a particular location on the map is the corresponding task ID, which specifies the task to be performed at that location. The map is then encoded as an image

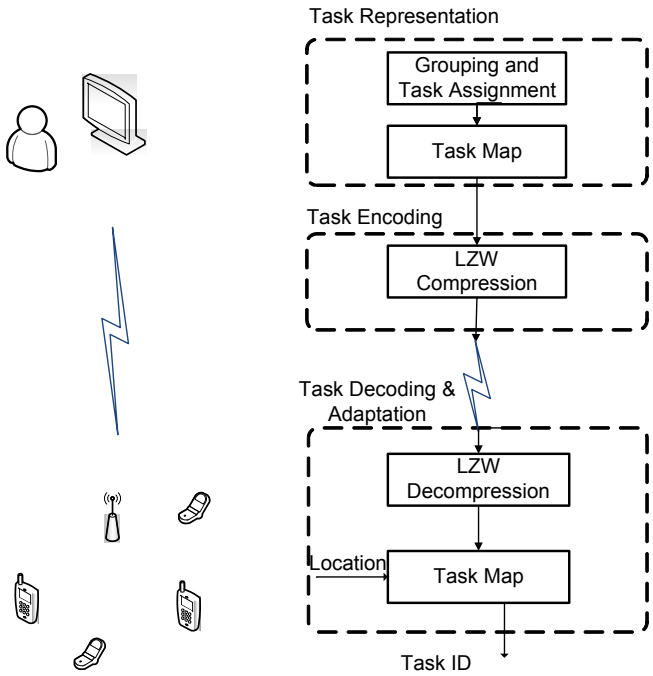


Figure 3.1: Zoom overview: Zoom has three main components: *task representation*, *task encoding*, and *resource adaptation*.

in *STIF format*, described in Section 3.3.2, and transmitted to the network.

Upon receiving the encoded task map, a node removes the image header and decompresses the task map. The node calculates the pixel in the image that corresponds to its physical location and retrieves the ID of the task it needs to perform.

The following three sections describe each of these components in detail.

3.3.1 Task Representation

Zoom decouples a task into task specification, which consists of a task ID and task header, and task implementation. Task ID is a unique number that can be used to identify the task. Task ID should be understandable by all sensor platforms. For example, task ID 5 is collecting temperature at 1 Hz. Task implementation is platform specific and can be a set of instructions or the required executable code to perform the task.

Spatial Grouping

A task map is used to represent geographical regions and the corresponding sensing tasks of each region. The task map can be viewed as an image overlaid on top of the physical map. A location on the task map corresponds to a real physical location. The pixel value at a particular location on the task map is the corresponding task ID which specifies the task to be performed at that location. Using this map-based representation, an operator can assign tasks to multiple spatial groups of sensors and transmit the task map to the network. Upon receiving the map, the node calculates the pixel in the task map that corresponds to its physical location to retrieve the task ID.

Figure 3.2 illustrates how Zoom works. To the left is the physical map of a region. An operator decides to measure noise pollution (task 1) in the left area and to measure the traffic conditions (task 2) in the right area. The operator defines the regions (e.g., by drawing on the map) and assigns them appropriate task IDs, indicated by pixel values. The corresponding task map (on the right of the figure) is then encoded and disseminated to the network. A node upon receiving the map determines the task it must perform by checking the corresponding pixel value.

If two regions overlap, the nodes in the overlapping region must perform both tasks. The overlapping region is assigned a new task ID, which in turn includes both the given tasks (Figure 3.3). The complexity of this operation is handled at the back end. Hence, the nodes themselves do not have to implement complex algorithms to interpret multiple tasks.

Other Grouping

Sensor groups can also be defined using other parameters such as sensor modality and sensor capacity. For example, one might issue a task to only nodes that

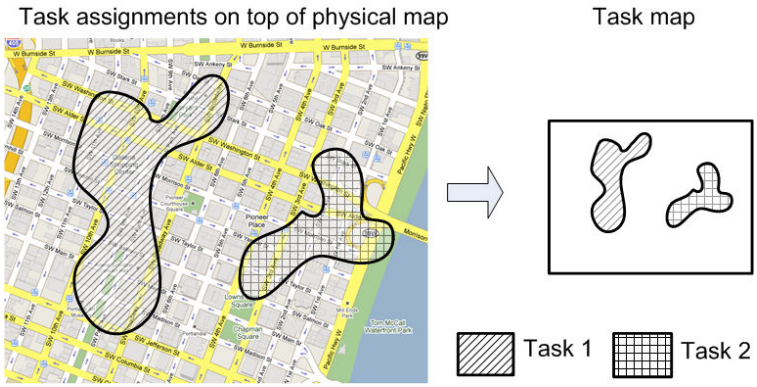


Figure 3.2: A task map overlaid on top of a physical map: A location on the task map corresponds to a real physical location. The pixel value at a particular location on the task map is the corresponding task ID which specifies the task to be performed at that location.

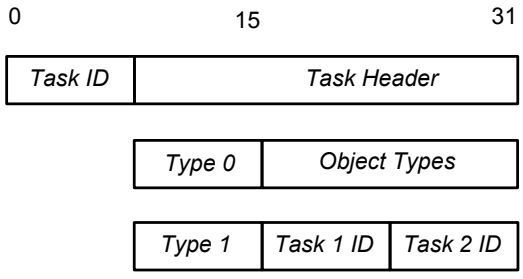


Figure 3.3: Task header: Depending on task type, the task header can contain IDs of the other tasks or physical object types that the sensor is attached to.

are attached to cars. These parameters can be included in the header of the map or in the task implementation that the node needs to download based on the task ID.

We divide other grouping parameters into two categories; static and dynamic. For static parameters, we consider the types of physical objects that the sensor is attached to. The types of the physical objects can be train, car, truck, bike, motorbike, human, or static objects. These types are defined in the task header. A node combines the spatial grouping information and the object type to determine if it belongs to a group. For dynamic parameters, we consider sensor types, sensor readings, and energy levels. These parameters are, however, defined in the implementation of the task.

3.3.2 Task Encoding

After surveying different encoding schemes, we find that an image-based format is suitable for Zoom for two reasons: it has the smallest file size and it enables resource adaptation for sensors with different resource capabilities. We have designed the Sensor Task Interchange Format (STIF) based on the graphic interchange format (GIF) [96], a common portable image format. GIF represents an image as a two dimensional array of 8-bit pixels. The pixel value is a reference to a color defined in the image header. GIF uses the Lempel-Ziz-Welch (LZW) [97] compression technique where it reduces the file size by maintaining a dictionary for sequences encountered in the data as it is encoded. STIF represents a task map as an image and uses the same LZW compression technique as GIF. We, however, replace the GIF header with a simple header containing the map identification (ID), the map version, the coordinates of the physical map, the height and width of the image, and the additional parameters if required. Figure 3.4 depicts the STIF header. The ID and version fields identify the map

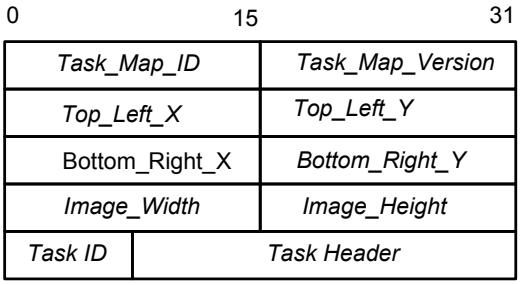


Figure 3.4: STIF header: The top left and bottom right coordinates scope the physical region to be tasked. The image width and height indicate the size of the encoded task map.

and the freshness of the map. The top left and bottom right coordinates indicate the scope of the physical region to be reprogrammed. The image width and height indicate the size of the encoded task map.

A node may not have enough resources to decode a high resolution map representing a large geographical region. At the same time, the node needs to know only the task IDs in a small geographical region around itself. Hence, instead of decoding the whole task map, nodes may just need to decode a small region in the map. We have developed three resource adaptation techniques, described in the next section that can help Zoom adapt to nodes that have different resource capabilities.

3.3.3 Resource Adaptation Techniques

The use of image encoding techniques allows Zoom to provide resource adaptation to sensors that have limited resources.

Multi-Resolution Encoding: Our first resource adaptation technique is to encode the task map at different resolutions, allowing nodes to download only the appropriate resolution that they need. Figure 3.5 shows a map encoded at three different resolutions. A lower resolution leads to a smaller image, requiring less memory and computational power to decode. There is a trade-off between the resolution and the ability to define the task at a fine granularity.

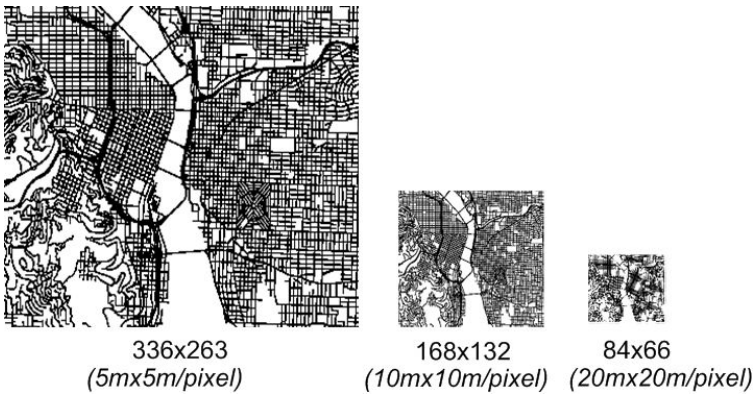


Figure 3.5: Multi-resolution encoding: A lower resolution leads to a smaller image, requiring less memory and computational power to decode.

Selected Region of Interest Encoding: In Figure 3.6, a node needs to know the task IDs of only a region large enough to cover its entire mobility (e.g. from home to work and back), rather than the task IDs of all regions in the map. Instead of encoding the entire task map, we can selectively encode only a small region within the map. Hence, the node can obtain a region of interest (ROI) in the task map. In addition, a node may need high granularity task IDs for a specific region such as a building to determine the appropriate task to perform when it is inside or outside the building. We can also selectively encode that region with a higher resolution. Hence, the node can obtain a higher resolution task map for the region of interest.

Region of Interest Cropping: Upon receiving an encoded map, a node does not necessarily decode the whole map, either because it is interested in only the task ID of its nearby region, or because it has limited resources and cannot decode the whole map. Zoom provides a technique called *region of interest cropping* that allows the node to quickly crop out only a region that is relevant to itself. This technique was originally developed to support region cropping in video streaming applications [98].

The main idea is to divide the task map into blocks and encode each block independently (see Figure 3.7). The encoded blocks are appended to each other.

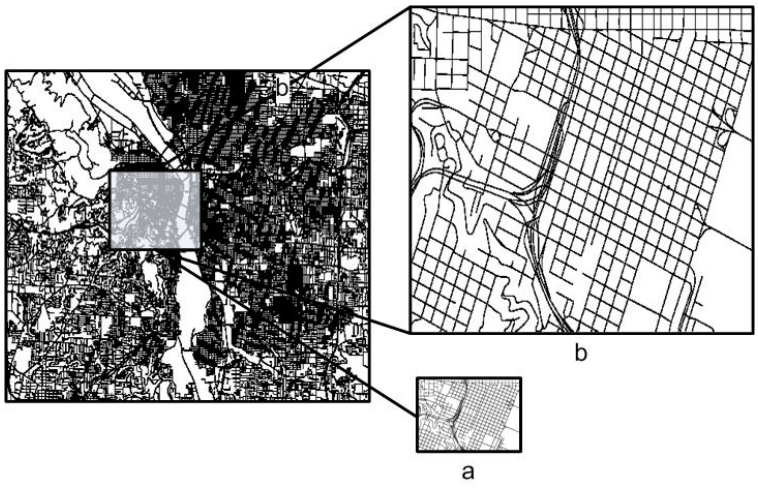


Figure 3.6: Selected region of interest encoding: A specific region can be encoded to reduce the image size to be transmitted. The region can also be encoded at a higher resolution to increase the identification accuracy.

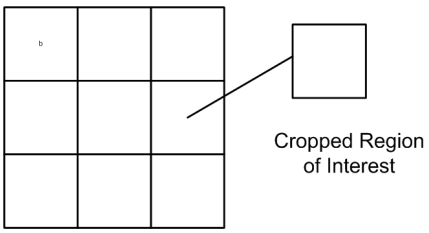


Figure 3.7: Region of interest cropping: The task map is divided into blocks and each block is encoded separately.

Upon receiving the encoded map, a node can determine the corresponding region of interest based on its location, then search for the start of that block, and decode only the found block. This technique does not conserve bandwidth but can help a node find its task ID quickly with fewer resources.

3.4 Limitations

The STIF format has two drawbacks. The number of task IDs is limited to 255 (8 bits/pixel). However, we believe that this is large enough for multiple concurrent tasks in a sensor network. We could increase the number of bits that represent a pixel, and consequently the number of task IDs, but at the cost of compression efficiency. Moreover, decoding STIF images may require slightly higher memory (albeit smaller than the image size itself) compared to other image formats. By carefully tuning the LZW compression parameters, we can overcome the memory problem. Indeed, Sadler et al. [99] have developed an LZW variant for resource poor embedded devices.

3.5 Implementation

We have implemented a complete system with both the task map decoder and networking support in our simulators. We have also implemented a task map encoder with support for Region of Interest (ROI) cropping in Matlab. We have implemented two variants of the task map decoder in C/C++; a basic task map decoder and a task map decoder with support for ROI cropping. The decoders are implemented in standard C. Hence they can be ported to different platforms using appropriate cross-compilers.

We have also implemented the task map decoder in Android (`http://www.android.com/`). Android is a software stack for mobile devices that includes an operating system, middleware, and key applications. Most of the Android

Components	Platform	Program size	RAM
Task map decoder	PC	14.5 KB	1.3 KB
Task map decoder with ROI cropping	PC	15.3 KB	1.5 KB
Task map decoder	Android	38.4 KB	2 KB

Table 3.1: Implementation detail

code is open source. The Android software development kit (SDK) allows us to develop applications in Java language. Table 3.1 shows the implementation details including program size and RAM usage. The programs are small in size and use less than 2 KB of RAM.

3.6 Evaluation

3.6.1 Goals and Metrics

Our evaluation goal is to answer the following questions:

1. Can the map-based approach in Zoom efficiently represent geospatial sensor groups and tasks?

2. How is Zoom's performance in terms of encoded map size affected by the number of groups and number of nodes in the network?

3. Is Zoom better than previous tasking approaches?

We analyze the file size of the encode task map, decoding latency, and the number of nodes that have incorrect tasks in different scenarios. We describe in detail how we setup the experiments in the following section.

3.6.2 Methodology

We investigate if the map-based approach in Zoom can represent geographical regions and tasks well. We consider how well STIF can encode road segments from a GIS file. We encode a geographical map using the STIF format and

analyze the number of pixels that contain more than one road segment. Figure 3.8 depicts possible cases where a pixel may contain only one road segment (a), two road segments (b), three road segments (c), or four road segments (d). In the ideal case, a pixel should uniquely identify a road segment, containing no more than one road segment. However, as the map resolution decreases, a pixel covers a larger geographical region and may contain more road segments. It is impossible to distinguish these road segments based on the pixel alone. In that case, STIF is unable to assign a distinct task to each road segment within a pixel. We refer to such a pixel as an error pixel. The definition of error also depends on the application. For example, in Figure 3.8(b), there is a clear error because the two roads do not intersect and are indistinguishable. Whereas the error pixel in Figure 3.8 (c) or (d), might be acceptable for some applications. In our evaluation, we consider (b), (c), and (d) as error pixels. We analyze the number of error pixels as a function of the map resolution.

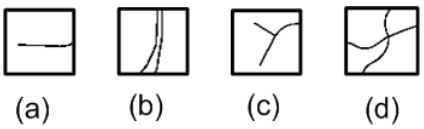


Figure 3.8: Number of roads within a pixel: (a) a pixel can uniquely identify a road segment. (b, c, d) a pixel can not uniquely identify a road segment.

We extract the GIS shapefile for Portland, Oregon and its nearby suburbs from the latitude and longitude coordinates of (7597010.859, 645515.097) to (7696218.347, 711876.59). This covers a 600 km^2 area. The shapefile is available at the US Census Bureau website and contains several records. Each record contains one or more street segments. Each street segment is defined by a set of points with corresponding longitude and latitude coordinates. We export this file into STIF files at different resolutions.

To investigate the second question, we use MobiReal [100], a realistic network simulator for mobile ad-hoc networks, to simulate a realistic traffic ap-

plication. MobiReal is built on top of GTNets [101], a full-featured network simulator. MobiReal allows separation of behavior and network simulation. Therefore, we can specify realistic behavior models for cars or pedestrians and integrate them with the network simulator.

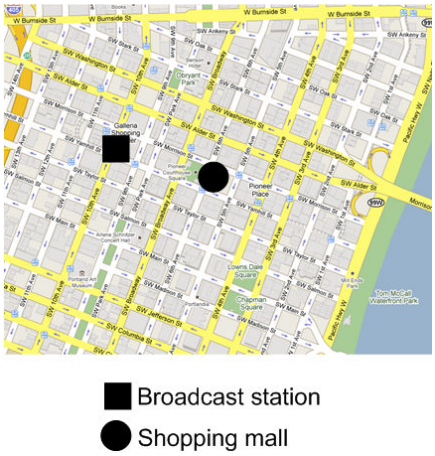


Figure 3.9: Simulation area: 1 km × 1 km area in downtown Portland, OR.

We simulate a *Tasking* application in a 1 km × 1 km area in downtown Portland (Figure 3.9). The main components of the simulation are:

- *Tasking Application*: We arbitrarily define regions in the map and assign different task IDs for those regions. A base station (marked as a black circle in Figure 3.9) broadcasts the encoded task map to the network every 15 seconds. A car, upon receiving the map, decodes the map and updates its task index. The car also schedules rebroadcast of the map. Together with the task map, the base station also broadcasts a sale advertisement for a nearby shopping mall (marked as a black square in Figure 3.9).

- *Mobility Behavior*: Cars are generated based on predefined road density that is close to the real density. A random entry point and a random destination are generated for each car at initialization. Cars travel to their destinations on the shortest path routes. However, upon receiving a sale advertisement, a car may add the mall address as an intermediate

destination with a predefined probability (10% in our simulation). A car arriving at the mall stays at the mall for several minutes before leaving for the final destination. After arriving at the destination, the car is removed from simulation and a new car will be generated randomly. This mobility behavior allows us to simulate dynamic behavior inspired by a real scenario.

- *Networking*: Table 3.2 shows the full network stack used in the simulation. Cars communicate with each other and with the base station using IEEE 802.11. The communication ranges are set to 100 meters for the cars and 300 meters for the base station. Both the cars and the base station use UDP as their transport protocol and IP as the network layer protocol. A car upon receiving the task map schedules periodic map rebroadcasts with an interval of 5 seconds. However, if it hears a broadcast of the same task within this interval, it suppresses its transmission and doubles the broadcast period interval. The maximum interval is set to 150 seconds.

Layer	Class	Description
Application	Advertisement	
Presentation	STIF format	Modification of GIF
Transport	L4Protocol	Wrapper class for UDP
Network	L3Protocol	IP V4
Routing	MyRoutingDSR	Dynamic source routing
Link (MAC)	L2Proto80211	802.11
Physical	DynamicWirelessLink	

Table 3.2: Description of the networking stack used in simulation.

We analyze the (i) update latency versus number of nodes (cars) in the network and (ii) the average number of nodes with incorrect task indices versus map resolution. We also encode task maps with different number of blocks and analyze the map size and the time to decode a specific block. We also define different number of regions and assign them different task IDs and analyze the

encoded map size versus the number of tasks. Experimental results are also shown in Sections 3.6.3.

In addition, we also conduct experiments on a real handheld smart device and analyze the decoding time for task maps of different resolutions. The device is a HTC Google G2 smartphone with an 800 MHz Qualcomm Snapdragon MSM7230 processor running Android 2.2. The image resolutions range from 90×60 to 5712×3833. The results are also shown in Section 3.6.3.

Finally, answering the third question is somewhat tricky. It is not really possible to quantitatively compare Zoom to previous tasking approaches because Zoom addresses a different problem and provides slightly different features. The work closest to Zoom is Logical Neighborhood [28]. We compare the size of Zoom encoded maps to the size of Logical Neighborhood predicates that define an equivalent spatial group of nodes.

We define a number of regions on a map and encode the map using Zoom. The number of regions is varied from 1 to 10. Using Logical Neighborhoods, we define the region boundaries and embed them in the predicate. Nodes with locations satisfying the predicate, i.e., their locations lie inside the regions' boundaries, are members of the corresponding groups. We compare the size of the Zoom STIF files to the size of Logical Neighborhood predicates to find out which approach requires fewer data transmissions.

We have described our methodology in setting up experiments to evaluate Zoom. In the next section, we present the experimental results showing that Zoom is scalable and efficient for tasking mobile sensor networks.

3.6.3 Experimental Results

This section shows the results of the experiments described above. The results show that Zoom can efficiently represent multiple sensor tasks for multiple

sensor groups in large geographical regions. Zoom data size is smaller than Logical Neighborhood, the state-of-the-art approach.

Error Pixels versus Resolution

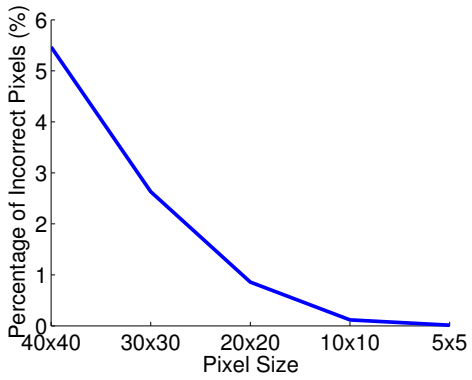


Figure 3.10: Total error pixels versus resolution: With size of 321 KB at resolution 2857 x 1917, a STIF image can uniquely describe every road segment.

Figure 3.10 plots the percentage of error pixels (pixels containing more than one road segment) versus the map resolution. The number of error pixels decreases when the map resolution increases. With resolution 2857 x 1917, which is equivalent to a 10 m x 10 m square per pixel, the percentage of error pixels is almost zero. The map is only 321 KB in size. This is much smaller than the shape file, which is 7 MB. With resolution 715 x 479, which is equivalent to a 40 m x 40 m square per pixel, the percentage of error pixels is around 5.5% while the encoded map size is only 34.5 KB. Hence, the map-based approach in Zoom is suitable for representing geographical regions and tasks.

Figure 3.11 shows the distribution of error pixels. As expected, the higher the map resolution, the lower the error. Nevertheless, most error pixels contain 2 to 5 road segments. Figure 3.12 plots the distribution of error pixels for a map of resolution 90 x 60. The whiter the color, the higher the number of roads colliding within the pixel. Most high error pixels are distributed near the downtown and freeway intersection areas. This error distribution map is useful

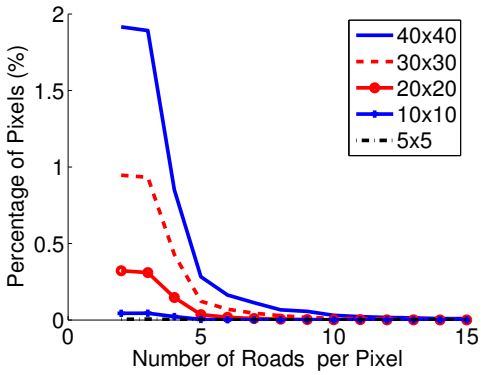


Figure 3.11: Distribution of error pixels: Most error pixels contain 2 to 5 road segments.

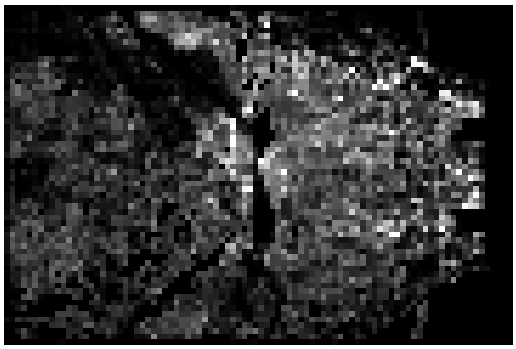


Figure 3.12: Spatial distribtion of error pixels (Portland, OR): High error pixels (white color) are distributed near the downtown and freeway intersection areas.

because we can increase the map resolution to decrease the identification error, when deploying a task map over a high error region.

Update Latency for Different Number of Nodes

Figure 3.13 plots the number of nodes with incorrect task indices versus time for different network sizes. The simulation scenario is a *Tasking* application (Section 3.6.2) in a 1 km $\times$ 1 km area in Portland, OR. Each pixel in the task map represents a 10 $\times$ 10 square region in the physical map. The base station broadcasts the encoded task map at the 15^{th} second. In the first 15 seconds, no node has the right task IDs. After the base station broadcasts the map, nodes update their task IDs and rebroadcast the map. Hence, the error rate decreases quickly. However, due to nodes joining and leaving the network dynamically, we can not achieve a zero error rate. The error rate reaches a stable threshold after 50 seconds. Also, the higher the density, the lower the error.

Update Latency for Different Map Resolutions

Figure 3.14 shows the percentage of nodes with incorrect task IDs versus time, with the task map encoded at different resolutions. The simulation scenario is 1 km $\times$ 1 km square in downtown Portland, Oregon. The task map is encoded at resolutions of 191 $\times$ 155, 96 $\times$ 78, and 48 $\times$ 39 pixels resulting in sizes of 1.48 KB, 1.15 KB, and 0.975 KB respectively. Each pixel in the task map represents a 5 m $\times$ 5 m, 10 m $\times$ 10 m, and 20 m $\times$ 20 square region in the physical map respectively. The smaller the encoded map size, which corresponds to a larger geographical area per pixel, the higher the error.

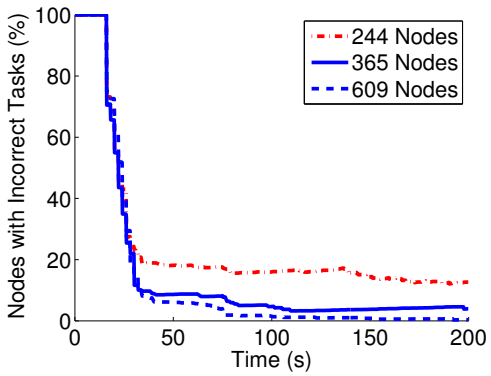


Figure 3.13: Percentage of nodes with incorrect task IDs versus time: The higher the node density, the lower the error.

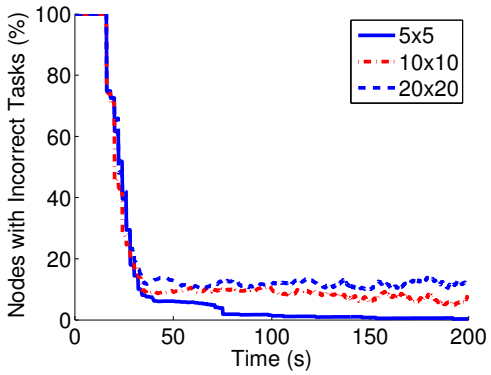


Figure 3.14: Percentage of nodes with incorrect task IDs versus time: Each pixel in the task map represents a 5×5, 10×10, or 20×20 squared meter region in the physical map. The higher the map resolution or the smaller square region each pixel represents, the lower the error.

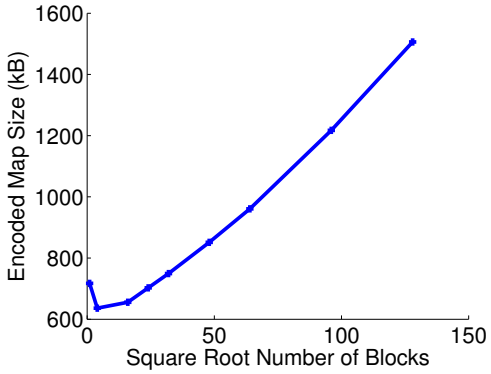


Figure 3.15: Encoded map size versus number of blocks.

Trade-off Between Compression and Decoding Speed in ROI Cropping

Figure 3.15 plots the encoded map size of the same resolution versus number of blocks within the map. The original map size is 717 KB with resolution 1536 × 1536. The map is divided into several blocks, with each block encoded independently. The encoded map size decreases at first as the map is divided into 4 blocks. After that, the map size increases with the number of blocks. The peak at 4 blocks is because in the original map, the limited size dictionary of repeated patterns in LZW does not optimally capture the most frequent patterns over the entire map.

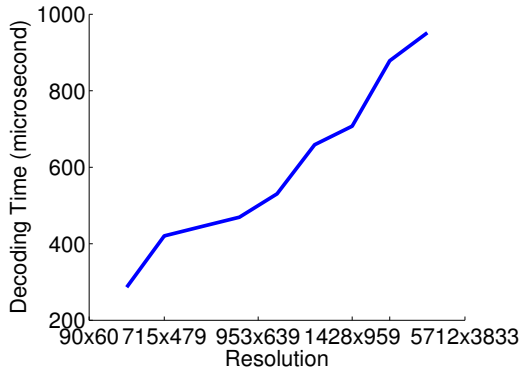


Figure 3.16: Decoding time versus map resolutions on the Google G2 smartphone: The decoding time is less than 1 millisecond even for maps with high resolutions.

Figure 3.16 shows the decoding time versus task map resolutions on the Google G2 platform. As the resolution increases, the decoding time increases. However, even for maps with a high resolution of 5712 × 3833 pixels, the decoding time is less than 1 millisecond.

Task Map Size versus Number of Regions

Figure 3.17 plots the encoded map size versus the number of regions. The map size grows in proportion to the number of regions. This is reasonable as the

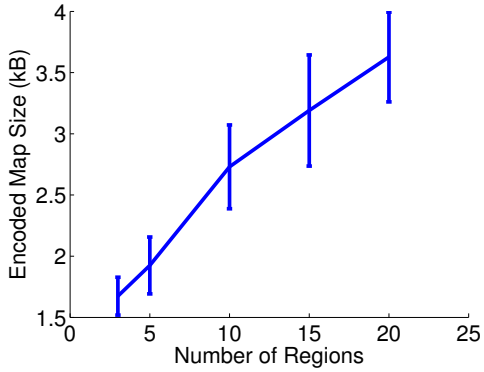


Figure 3.17: Encoded map size versus number of regions: Encoded map size increases proportionally with the number of regions.

number of recurrent patterns in the map usually decreases when the number of distinct pixel values in the map increases.

Encoded Map Size versus Logical Neighborhood Predicate Size

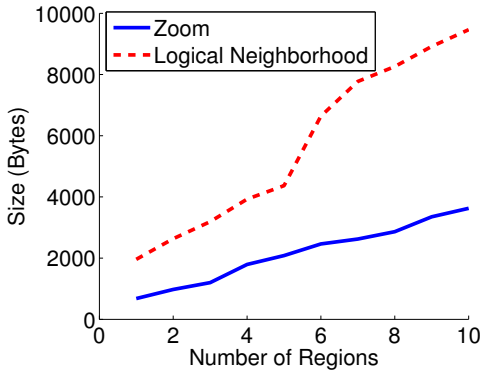


Figure 3.18: Encoded map size versus number of regions. STIF maps always have a smaller size compared to Logical Neighborhood predicates.

Figure 3.18 plots the size of the STIF maps and the Logical Neighborhood predicates when encoding different numbers of regions. The regions are selected randomly and the number of regions are varied from 1 to 10. STIF maps always have a smaller size compared to Logical Neighborhood predicates. This implies that Zoom potentially uses less bandwidth than Logical Neighborhood and sensors in Zoom use less memory than in Logical Neighborhood.

3.7 Summary

In this chapter, we presented Zoom, a multi-resolution tasking framework for mobile sensor networks. Zoom's innovation is to support heterogeneous devices by decoupling the task specification from the task implementation. Zoom uses maps to represent task specification and groups and encodes them in our proposed Sensor Tasking Interchange Format (STIF), making tasking intuitive for network operators. The use of maps also allows a sensor to quickly obtain its task identification without running complex geometric algorithms to determine whether it belongs to a region or not. We have also presented three resource adaptation techniques to reduce memory, bandwidth and CPU usage in Zoom.

Our evaluation shows that Zoom is capable of tasking arbitrary groups of sensors in a large geographical network. With an encoded map of size only 34.5 KB, Zoom can task a region of 600 km^2 with only 2% error. In addition, simulation of a realistic traffic application over a region of 1 km^2 with a task map of size 1.48 KB shows that more than 90% of nodes are tasked correctly. Finally, for the same number of tasks, Zoom's encoded map size is always 50% smaller than the predicate size in the state-of-the-art Logical Neighborhood approach. To the best of our knowledge, this is the first work to propose a map based approach for mobile sensing systems. We believe that Zoom's tasking capability is a step toward providing structure in increasingly unstructured mobile geo-spatial sensing systems.

Chapter 4

DHV: A Dissemination and Maintenance Protocol for Embedded Sensor Networks

This chapter describes the design, implementation, and evaluation of the DHV protocol, which efficiently disseminates task items and ensures that nodes have the up-to-date items in embedded sensor networks. It achieves this by minimizing both the redundant information in each message and the total number of transmitted messages in the networks.

4.1 Introduction

In Chapter 1, we showed that the key problem in tasking embedded sensor networks is to design a dissemination and maintenance protocol that can efficiently spread task items and ensure that all nodes have the updated items. We also described the main challenges that the dissemination and maintenance protocol must address. In particular, the protocol must take into account distinct characteristics of embedded sensor networks, such as intermittent node operation and limited node resources.

In this section, we describe in detail DHV, a scalable and efficient dissemination and maintenance protocol for embedded sensor networks. DHV is efficient in terms of bandwidth, energy, memory, and latency. It also scales well with the number of nodes and number of task items.

The rest of this chapter is organized as follows. In Section 4.2, we specify the requirements for a dissemination and maintenance protocol, our design goals, our assumptions, and the key ideas in the DHV protocol. We describe in detail the DHV protocol in Section 4.3 and its suppression mechanism in Section 4.4. We discuss limitations of DHV in Section 4.5. Section 4.6 presents theoretical analysis of the DHV protocol and shows that DHV has a better theoretical performance than other protocols. Section 4.7 and Section 4.8 describe the implementation and evaluation of DHV respectively. We present protocol selection guidelines in Section 4.9. Finally, we conclude this chapter with a summary in Section 4.10.

4.2 Overview

In this section, we present our design goals in Section 4.2.1 and describe key ideas in DHV that can achieve the goals in Section 4.2.2 as well as state main assumptions in DHV in Section 4.2.3.

4.2.1 Design Goals

The main requirements for a dissemination and maintenance protocol (DMP) are:

- *Convergence*: A DMP must ensure that all nodes will *eventually* have the same updated task items. This is an important requirement, especially for embedded sensor networks wherein the nodes are distributed and unreliable.

- *Scalability*: A DMP must scale with both the number of nodes and the number of task items.

- *Efficiency*: A DMP must enable a node with an old task item to discover a newer task item and update it with low latency. It must also conserve energy, memory, and bandwidth.

4.2.2 Key Ideas

To achieve the above goals, we design DHV with the following key ideas:

- *Minimizing unnecessary information in each message*: We observe that to detect if two version numbers are different, it is not necessary to compare all the bits in the version numbers. Indeed, if the two version numbers are different, only one different bit is adequate to conclude that they are different. In DHV, a sensor transmits only the most probable bits of the version numbers instead of transmitting the whole version numbers to identify items with different version numbers, thus reducing the amount of redundant information in transmission.

- *Detecting multiple inconsistencies simultaneously*: Previous protocols exchange messages to detect if items in the network have different version numbers serially. In contrast, we design DHV to *concurrently* detect multiple items that have different version numbers to reduce detection latency.

- *Minimizing redundant transmissions*: In wireless communication, the medium is shared among nodes. DHV uses a *gossip-based communication scheme* to scale with the number of nodes. A node advertises information about its items (e.g., a hash of version numbers of all items) at a random point in time within each time interval. However, if it receives messages with the same information as it has, it suppresses its own transmission, thus reducing the total number of transmissions in the network.

4.2.3 Assumptions

Before describing DHV in detail, we list the following assumptions:

- *Unit increment of the version number per update*: The version number is incremented by one for each update. This assumption is reasonable and indeed is implicitly made in both DRIP [27] and DIP [15]. It allows us to infer that, if two version numbers are different, they mostly differ in a few least significant bits. DHV exploits this assumption to restrict the comparison scope to only a few least significant bits of the version numbers.

- *Unique item ordering*: The order of (key, version number) tuples is the same for all nodes. The same ordering can be achieved by using the same sorting algorithms based on the items' keys at all nodes. This assumption allows DHV to identify which items need updates from the indices of the different version numbers.

- *Low update frequency*: The updates are relatively infrequent (e.g., every hour or every day). This assumption is reasonable because the network task changes infrequently in most practical applications [81, 21].

The next section describes an overview of DHV, which was designed based on the above key ideas and the above assumptions.

4.3 The DHV Protocol

DHV views the set of all task item version numbers as a two dimensional binary matrix (Figure 4.1) where the number of rows is the number of task items and the number of columns is the number of bits in each version number. DHV has five main phases as shown in Figure 4.2.

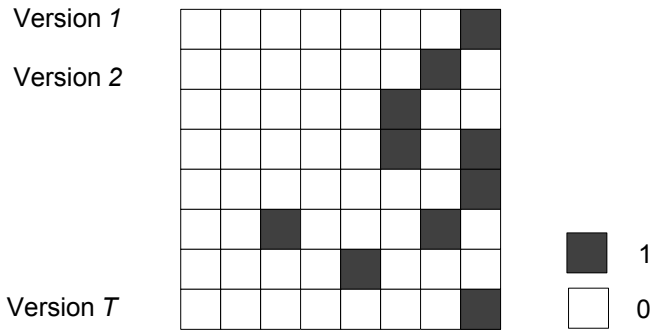


Figure 4.1: Versions as a two dimensional binary matrix

- *Detecting difference*: A node broadcasts a hash of all its version numbers to its neighbor nodes, nodes that can directly communicate with the broadcasting node. Upon receiving the hash, a neighbor node compares the received hash with its own hash. If the two hashes are the same, it is likely that the broadcasting node and its neighbor have the same set of version numbers. If the two hashes are different, the node and its neighbor have at least one item that has a different version number.

- *Identifying location of different bits using horizontal search*: If a node receives a hash that is different from its own hash, the node broadcasts a checksum of all version numbers, the exclusive OR of all the rows in the matrix to its neighbor nodes. A neighbor node, upon receiving the checksum, compares the checksum with its own checksum to identify which bits are different. The locations of the different bits in the checksums indicate the columns in the binary matrix that are different from the broadcasting node.

- *Identifying different version numbers using vertical search*: Once a node knows the location of the different bits in the checksum, the node broadcasts a bit slice of all the version numbers, which is a column in the matrix, at the location of different bits found in the horizontal search. If the bit slices are similar, but the hashes differ, the node broadcasts a bit

slice of index 0 (index of the least significant bit) and increases the bit index to find the different locations until the hashes are the same. Upon receiving a bit slice, the node compares it to its own bit slice to identify the locations corresponding to the differing (key, version number) tuples.

- *Identifying version ordering*: A node broadcasts the (key, version number) tuple to its neighbor nodes. A neighbor node, upon receiving the tuple, compares it to its own (key, version number) tuple to decide who has the newer item. If the node has a higher version number, it transmits the item data. Otherwise, the node transmits its (key, version number) to notify other nodes that it has older items.

- *Updating*: A node with a higher version number broadcasts its item data to nodes having the item with lower version numbers.

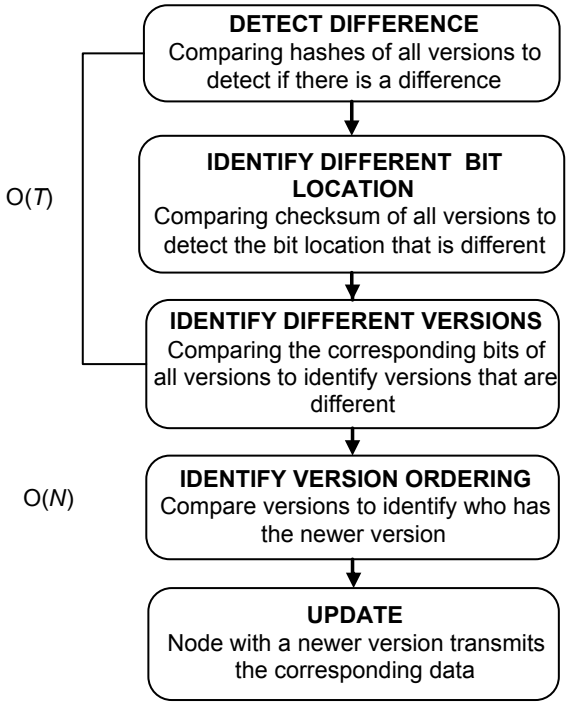


Figure 4.2: Five main phases in DHV: (Source [1]: modified with permission from Springer Verlag)

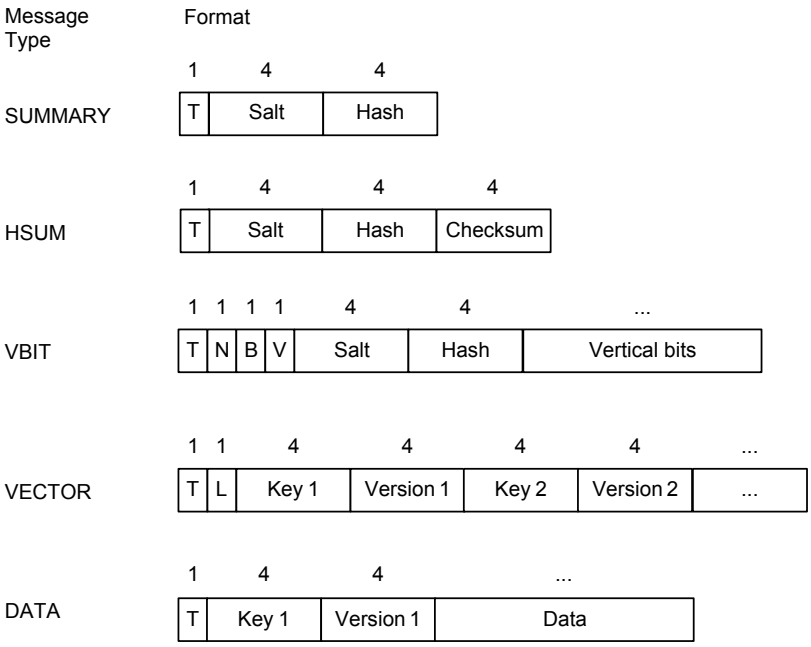


Figure 4.3: DHV message formats: (Source [1]: Used with permission from Springer Verlag)

There are five message types that DHV uses as shown in Figure 4.3.

SUMMARY: This message type contains the hash of all version numbers as well as the random seed used for hashing. It contains the least amount of information compared to other message types. A node can detect only if there is a difference using this message type.

HSUM: This message type contains the checksum of all version numbers, the hash of all version numbers, and the random seed used for hashing. It is used to identify the different bit indices in the version numbers between two nodes.

VBIT: This message type contains selected bits of version numbers and is used to identify version numbers that are different between two nodes. The bits corresponding to the differing indices of the VBIT messages are identified from HSUM messages. If two HSUM messages are similar but the SUMMARY messages differ, VBIT messages for the least significant bit indices are compared because the version numbers mostly differ in a few least significant bits.

DATA: This message type contains the actual item data to be updated. DATA messages can contain item data such as configuration parameters, code capsules, or executable images.

VECTOR: This message type contains one or more (key, version number) tuple. When a node receives a VECTOR message advertising older version numbers compared to itself, the node broadcasts a DATA message containing its version numbers of the task items. When a node receives a VECTOR message advertising newer version numbers compared to itself, the node broadcasts its own VECTOR message so that other nodes having newer items will transmit the corresponding DATA.

Figure 4.4 illustrates how DHV works. Node 1 and Node 2 have a set of item keys and item version numbers, in which the item with the key number 2 has different version numbers. Node 1, first, broadcasts its SUMMARY hash of all the version numbers. Node 2 receives *hash 1* and detects that *hash 1* is different from *hash 2* of node 2. Hence, node 2 broadcasts its *HSUM* message, which is a checksum of all version numbers. Node 1 receives the HSUM message 2 and compares it to its own checksum. Node 1 identifies that the 2^{nd} bits differ. Hence, node 1 copies the 2^{nd} bit of all the version numbers into one or more *VBIT* messages and broadcasts them. Node 2 receives a VBIT message, compares it to its own VBIT message and detects that the 2^{nd} bits are different from each other. Hence, node 2 knows that the item with key index 2 is different from its neighbors. Node 2 broadcasts a *VECTOR* message containing (key 2, version 2). Node 1 receives (key 2, version 2) from node 2 and sees that node 1 has a newer version of this item. Hence, node 1 broadcasts the *DATA* of key 2.

Due to the distributed nature of sensor networks, nodes in the network still transmit redundant messages. The next section is going to describe how DHV, based on Trickle [14], minimizes the number of redundant transmissions.

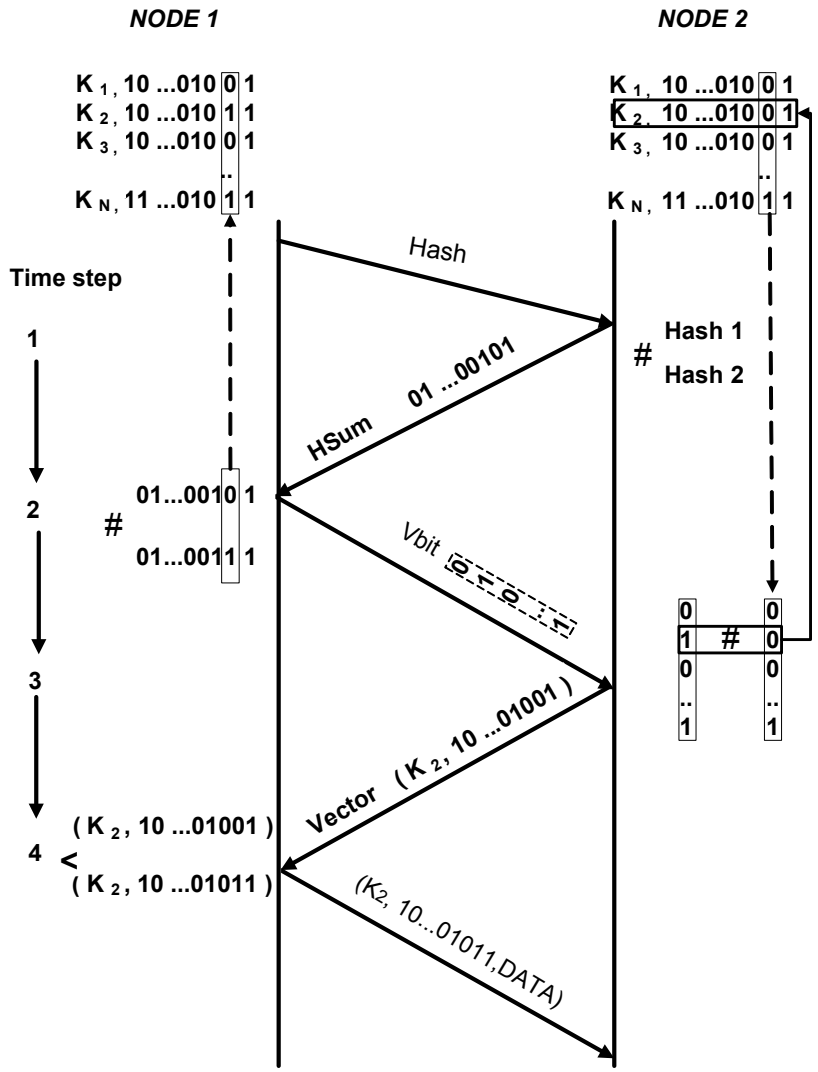


Figure 4.4: DHV flow diagram: Node 1 broadcasts its SUMMARY message which contains the hash of all the version numbers. Node 2 receives *hash 1* and detects that *hash 1* is different from *hash 2* of node 2. Node 2 broadcasts its *HSUM* message which contains the checksum of all version numbers. Node 1 receives the HSUM message 2 and compares it to its own checksum. Node 1 identifies that the 2^{nd} bits differ. Node 1 copies the 2^{nd} bit of all the version numbers into one or more *VBIT* messages and broadcasts them. Node 2 receives a VBIT message, compares it to its own VBIT message and detects that the 2^{nd} bits are different from each other. Node 2 broadcasts a *VECTOR* message containing (key 2, version 2). Node 1 receives (key 2, version 2) from node 2 and sees that node 1 has a newer version of this item. Node 1 broadcasts the *DATA* of key 2. (Source [1]: Used with permission from Springer Verlag)

4.4 Suppression Mechanism and Transmission Scheduling

The next section briefly describes a suppression mechanism used in Trickle [14]. Then, we describe how we extend Trickle to develop the transmission scheduling algorithm in DHV.

4.4.1 Trickle Suppression Mechanism

In Trickle, time is divided into intervals. At a random point of time in each interval, a node considers broadcasting its message (e.g., a SUMMARY message). If the node has already received several messages with the same content in this interval, the node suppresses its transmission.

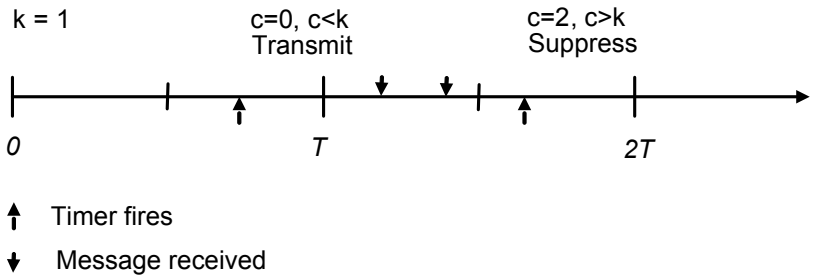


Figure 4.5: Suppression mechanism in Trickle: If $c < k$, the node broadcasts its message. Otherwise, the node suppresses its own transmission and doubles the value of τ.

Formally, each node maintains a timer t in the range of $[0, \tau]$ to schedule when the node decides to transmit a message, a counter c to keep track of the total number of received messages that have the same content as the node has, and a suppression threshold k to decide if the node needs to transmit a message. c is incremented by 1 whenever the node receives a message that has the same content as it has. When the timer t fires, the node decides whether to transmit a message based on the values of c and k. If $c < k$, the node broadcasts its message. Otherwise, the node suppresses its transmission and doubles the value of τ. For example, for $k = 1$, in the first interval in Figure 4.5, $c = 0$ and $c < k$. Hence, the node broadcasts its message. However, in the second interval in

Figure 4.5, $c = 2$ and $c > k$. Hence, the node suppresses its transmission. If the node receives a message that contains different information from its own, it resets the value of τ to the predefined minimum value. To make sure each node listens to other nodes long enough before making the transmission decision, t is actually constrained to be in the range of $[\tau/2, \tau]$. When the time interval $[0, \tau]$ completes, the node starts a new interval where c is reset to 0 and t is randomly selected from $[\tau/2, \tau]$.

The following section describes how DHV uses Trickle to efficiently schedule transmissions in wireless sensor networks.

4.4.2 DHV Transmission Scheduling

A decision on whether to send out a message is made when the Trickle timer fires after a specified time interval. Figure 4.6 shows the flow chart of how the decision is made at a node. The node keeps a counter c for the number of received messages that have the same content as it has. If c is greater than a threshold k, the node suppresses its transmission and doubles the next interval period. Otherwise, it checks if there are pending messages to send. If the node has DATA messages to send, it will send one message out. If there are no DATA messages to send, the node will check if there are other pending messages in the following order: VECTOR, VBIT, and HSUM to be sent out. If there are no pending messages, the node will send a SUMMARY message and double the next time interval period as it expects that the network is stable.

4.5 Limitations

DHV has several limitations.

- *DHV is not optimized for subsets of nodes.* DHV aims at disseminating task items to all nodes in a network. In some cases where task items

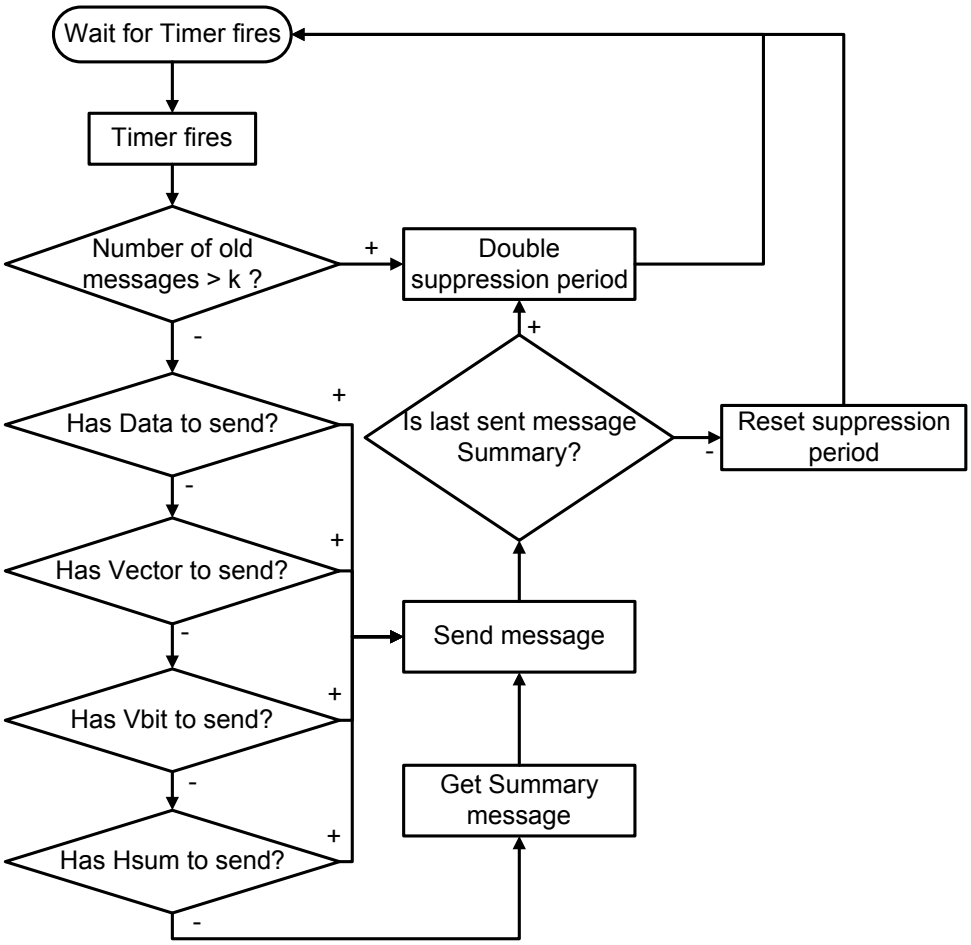


Figure 4.6: Transmission scheduling flow diagram: The node keeps a counter c for the number of received messages that have the same content as it has. If c is greater than a threshold k, the node suppresses its own transmission and doubles the next interval period. Otherwise, it checks if there are pending messages to send.

need to be disseminated to only a small number of nodes, DHV might not perform well. However, DHV can be combined with grouping information described in Chapter 3 to improve dissemination and maintenance performance.

- *DHV is not optimized for subsets of items.* One of the assumptions in DHV is that all nodes have the same set of task items. DHV maintains consistency of all the task items. In some scenarios where nodes only need to maintain consistency of a subset of task items (e.g., items of an active task), DHV might not perform well. However, a mask can be used to indicate which task items are under consideration. This approach requires an additional field in DHV messages for storing the mask information.

- *Hash collisions can occur.* DHV relies on hashes of version numbers to detect if there are inconsistencies among nodes in a network. Theoretically, it is possible that there are inconsistencies among nodes in a network but the hashes are the same. The probability of collision depends on the number of items and the number of bits representing hashes. In practice, nodes can send the complete items after a certain number of update rounds to ensure that all nodes have the same updated items.

The next section will provide performance analysis of different protocols in different scenarios.

4.6 Theoretical Analysis

To gain an insight into the performance of different dissemination and maintenance protocols, we describe theoretical analysis of DRIP [27], DIP [15], and DHV. For simplicity, we use a network of two nodes with a zero packet loss rate.

One node called the *updated node* has the updated items where the other node, *outdated node*, has old items. We calculate the number of messages that need to be transmitted in the network to detect and identify which items have different version numbers compared to the other node. Let assume that the two nodes have a total of T items. We will first consider a simple case where the network updates one item at a time. Then, we generalize the analysis for the case where the network updates multiple items.

4.6.1 Updating One Item

In this case, the network updates one item at a time. The updated node has an item whose version number is greater than the other node. However, the two nodes do not know which item has a different version number. Each protocol allows nodes to exchange messages to find out which item has a different version number and which node has a newer item.

DRIP

In DRIP, a node broadcasts a tuple (key, version number) randomly within an interval. Hence, the best case performance is achieved when a node advertises the exact item that has a different version number and identifies the difference in $O(1)$. The worst case performance is achieved when the nodes advertise all tuples (key, version number) for T items. The worse case performance hence is $O(T)$. The average case performance is also $O(T)$.

DIP

In DIP, nodes perform binary search to determine which items have different version numbers. Hence, the best case, worst case, and average case performance is $O(log(T))$.

DHV

In DHV, nodes perform three search steps: detecting difference, horizontal search, and vertical search. Each step requires one message to be transmitted. Hence, the best case, worst case, and average case performance is 3 messages.

Figure 4.7 shows the theoretical average case performance in terms of number of transmitted messages of the three protocols updating one item in a simple network of two nodes. DHV has communication overhead for searching which item has a different version number. Therefore, DHV performs worse than DIP and DRIP when the total number of items is less than 10. However, when the total number of items increases, DHV uses a constant number of messages to update the network. Theoretically, for updating a single item, DHV outperforms both protocols as the number of item increases.

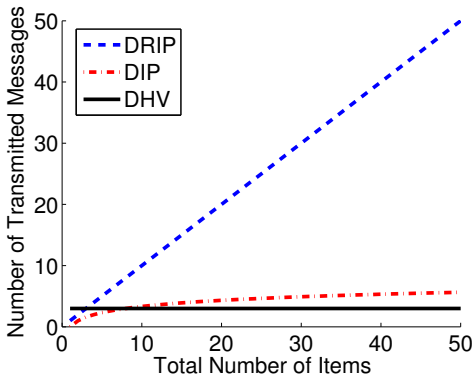


Figure 4.7: Number of transmitted messages versus total items

4.6.2 Updating Multiple Items

Let p $(0 \leq p \leq 1)$ be the probability that an item is updated within one updating round (e.g., a time interval). Let k be the number of rounds that the outdated node does not get updated. Again, we calculate the average number of messages transmitted to identify which item needs to be updated.

DRIP

In DRIP, the average case performance is still $O(T)$.

DIP

In DIP, nodes perform binary search to identify which items have different version numbers. Hence, in the best case, worst case, and average case performance for one item is $O(log(T))$. The probability that an item is not changed in one round is $1-p$. The probability that an item is not changed after k rounds is $(1-p)^k$. Hence, the probability that an item is updated after k rounds is $q = 1-(1-p)^k$. The average number of items that are updated after k rounds is $q \times T = (1-(1-p)^k) \times T$. The average case performance for multiple items is $O(log(T) \times (1-(1-p)^k) \times T)$.

DHV

The average number of times an item is updated after k rounds is $p \times k$. That means, the location of the most significant bit in the version number that is changed can be smaller than $log(p \times k + 1) + 1$. The total number of messages in the average case is hence smaller than $O(log(p \times k + 1) + 1)$.

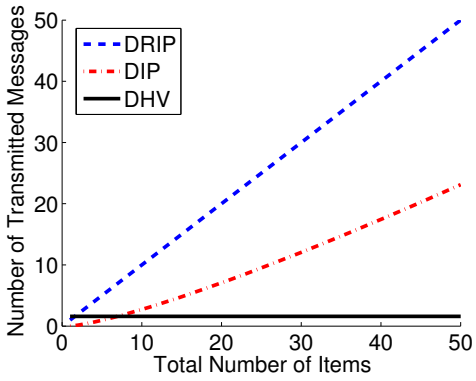


Figure 4.8: Total transmitted messages versus total items: $p = 0.1$ and $k = 5$.

Figure 4.8 shows the performance of different protocols in terms of number of transmitted messages versus total number of items T. There are $k = 5$ rounds that a node misses the updates. Within each round, each item may be updated with a probability of $p = 0.1$. DHV uses a constant number of messages, 3–5 messages, to detect which items have different version numbers.

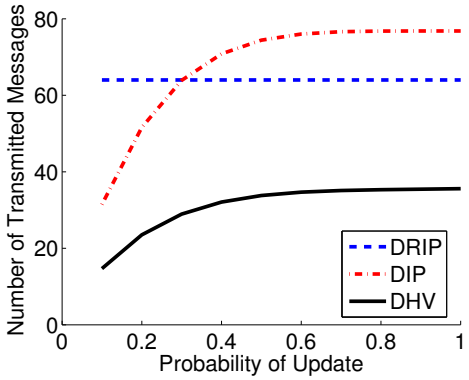


Figure 4.9: Total transmitted message versus update probability: $T = 64$ and $k = 5$.

Figure 4.9 shows the performance of different protocols in terms of number of transmitted messages versus the probability of update p. There are $k = 5$ rounds wherein a node misses the updates and total $T = 64$ items. Within each round, each item may be updated with a probability ranging from $p = 0.1$ to $p = 1$. DHV uses a constant number of messages, 3–5 messages, to detect which items have different version numbers.

Figure 4.10 shows the performance in terms of number of transmitted messages versus number of rounds k of different protocols. Within each round, an item gets updated with a probability of $p = 0.1$. The total number of items is $T = 64$. DHV uses a relatively constant number of messages to detect which items have different version numbers compared to DRIP and DIP.

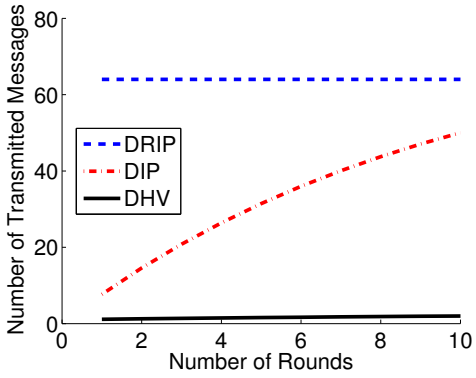


Figure 4.10: Total transmitted messages versus update rounds: $T = 64$ and $p = 0.1$.

4.7 Implementation

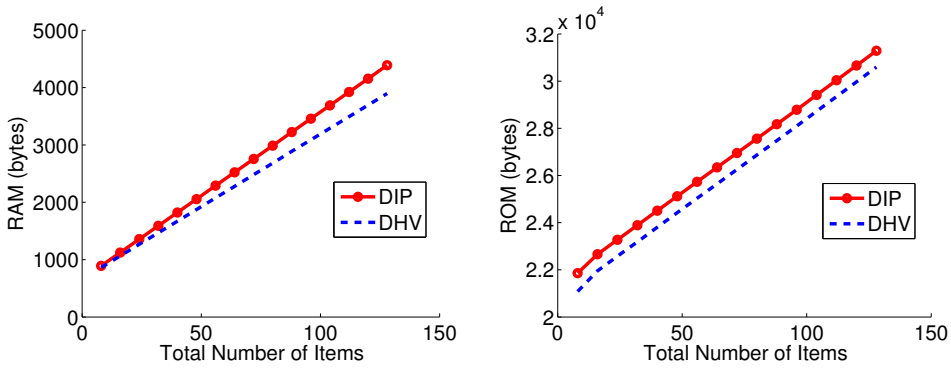
We have implemented DHV in TinyOS 2.1.1 [14] and tested the protocol with MicaZ [23] and Tmote [102] platforms. We chose TinyOS because of its popularity and strong support for different sensor platforms. TinyOS currently supports 18 embedded sensor platforms. TinyOS applications are written in nesC [36], a dialect of the C language optimized for sensor platforms with limited resources. TinyOS programs are built out of software components, some of which present hardware abstractions. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation, and storage.

Similar to DIP, both item keys and version numbers in DHV are 4 bytes. The number of bytes in the keys and the version numbers can be adjusted for different applications. DHV is now part of the official TinyOS 2.1.1 core library and can be downloaded from the official TinyOS website `http://tinyos.net`. Other background information and tutorials can be accessed from the TinyOS wiki webpage `http://docs.tinyos.net`.

Metric	DRIP	DIP	DHV
ROM (Byte)	15676	18686	17760
RAM (Byte)	364	405	402
Code size (Byte)	43178	51454	48918

Table 4.1: Implementation statistics for the TestDissemination application (*tinyos-2.x/apps/tests/TestDissemination*).

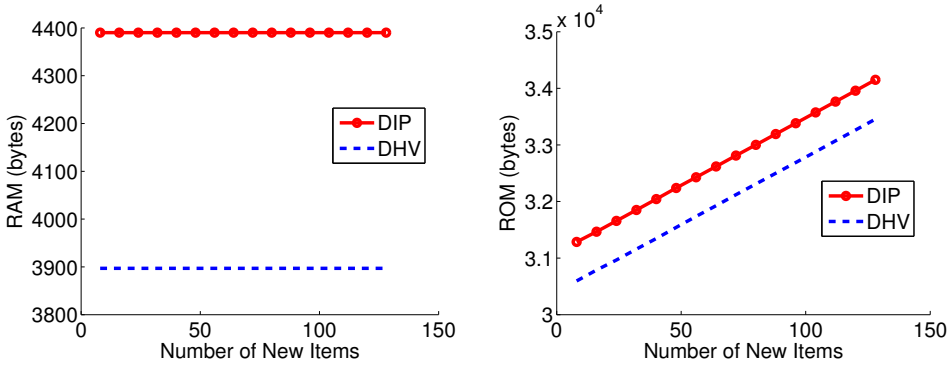
Table 4.1 compares DHV memory usage to DRIP and DIP when compiling with a standard TestDissemination application with only two items. They are largely similar. DRIP has the smallest code size and uses the least memory. This is reasonable because DRIP is the least complex protocol. The differences between DIP and DHV in terms of RAM and ROM are insignificant. DHV uses slightly less ROM and RAM compared to DIP.



(a) RAM (b) ROM

Figure 4.11: Code size versus total number of items: Number of new items is 8. The total number of items varies from 8 to 128. The ROM and RAM usage increase proportionally with the total number of items. However, DHV always uses slightly less ROM and RAM than DIP.

Figure 4.11 compares the code size and memory usage on the MicaZ platform of DIP and DHV as a function of the total number of items. The number of new items is fixed and equal to 8. The ROM and RAM usage increase proportionally with the total number of items. DHV always uses slightly less ROM and RAM than DIP.



(a) RAM (b) ROM

Figure 4.12: Code size versus total number of new items: The total number of items is 128. The number of new items varies from 8 to 128. The ROM usage increases proportionally with the total number of new items. However, DHV always uses slightly less ROM and RAM than DIP.

Figure 4.12 compares the code size and memory usage in the MicaZ platform between DIP and DHV versus the total number of new items. The total number of items is fixed and equals to 128. The ROM usage increases proportionally with the total number of new items. DHV always uses slightly less ROM and RAM than DIP. We also observe similar results on the TelosB platform.

4.8 Evaluation

4.8.1 Goals and Metrics

Previous work has shown that DIP outperforms other dissemination and maintenance protocols (DMP) [15]. Therefore, our experimental goals are to study if DHV performs better than the state-of-the-art DIP protocol. We use the

following metrics to evaluate the performance, with a lower value indicating better performance for all metrics.

- Total latency to update new items. This metric indicates how fast a DMP can help the network converge.
- Total numbers of transmitted messages and transmitted bytes to update new items. These metrics indirectly represent the energy and bandwidth consumption of a DMP protocol.
- Total energy consumed for updating a network. This is the energy consumed by a network measured from the time when new task items are disseminated to the time when all nodes in the network are updated with the new task items.

4.8.2 Methodology

We conducted experiments with five different parameters and three different network topologies. The experiments were in both simulation and real testbeds. One or more nodes with newer items update a network with older items. This scenario occurs in practice when a node or a base station reprograms a network or disseminates events to all nodes in the network.

We varied four different parameters including the total number of items, number of new items to be updated, number of nodes in the network, and packet loss rate. The notations and values for the parameters used in both simulation and real testbeds are described in table 4.2.

The three different network topologies (as shown in figure 4.13) are listed below.

- *Single-hop clique* where every node in the network can communicate with every other node,

Notation	Meaning	Range	Evaluation Purpose
T	Total number of items	8 to 128	Scalability in terms of total number of items
N	Number of new items	8 to 128	Scalability in terms of number of new items
D	Number of nodes	8 to 64	Performance with different network density
L	Packet loss rate	5% to 45%	Performance with different link quality

Table 4.2: Notation and ranges of parameters used in evaluation.

- *Multi-hop chain* where nodes are arranged in a line and a node can communicate with only adjacent nodes, and
- *Multi-hop grid* where nodes are arranged in a grid and a node can communicate with some nearby nodes.

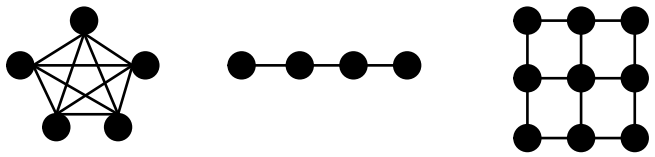


Figure 4.13: Example network topologies used for evaluation.

We purposely chose these three topologies because they cover the whole topology spectrum. *Single-hop clique* and *multi-hop chain* topologies are two extreme cases where a nodes can communicate with the maximum number of neighbor nodes and minimum number of neighbor nodes respectively. *Multi-hop grid* topology is a representative for most multi-hop networks.

The experiments were conducted in both simulations using TOSSIM [103], a discrete event simulator tool for wireless sensor networks and on two real sensor network testbeds; PSU SynLab MicaZ testbed at Portland State University and Motelab Tmote testbed at Harvard University [104]. Table 4.3 describes in detail the different networks we used for evaluation.

Testbed	Number of nodes	Networking
TOSSIM	128	One-hop Multi-hop
PSU-SynLab	64 MicaZ [23]	One-hop
MoteLab	121 Tmote [102]	Multi-hop

Table 4.3: Network description.

In the following sections, we describe in detail how experiments were setup in both simulation and real testbeds.

TOSSIM Simulation

We use TOSSIM [103] because its popularity and accuracy. It is a simulator for TinyOS [14], one of the most popular operating systems for embedded sensor platforms. TOSSIM simulates entire TinyOS applications by replacing TinyOS components with simulation implementations. TOSSIM is a discrete event simulator; when it runs, it pulls events from the event queue (ordered by time) and executes them.

TOSSIM does not allow simulation of packet loss directly. Instead, the packet loss depends on several parameters like receiving gain, noise, and clear channel access threshold. As a first step, we studied the effect of these parameters on packet loss. We simulated a two-node network. The noise is simulated using the state-of-the-art closest pattern matching approach [105] with noise traces from the Stanford Meyer library and are available in the TinyOS source code. Due to memory limitations, we only use the first 1000 entries in the trace, which is well above the recommendation of 100 entries. Figure 4.14 (Right) shows the packet loss rate versus receiving gain. The receiving gains corresponding to packet loss rates of 5, 10, 15, 20, 25, 30, 35, 40, and 45% are -70, -74, -76, -78, -81, 84, -87,-88, -89 dBM, respectively. Based on this result, we can select different gain values for different packet loss rates in simulation.

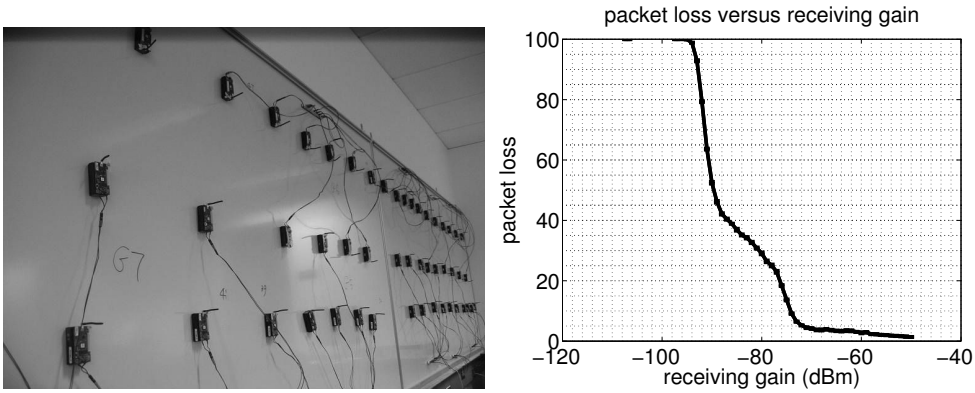


Figure 4.14: (Left) Real MicaZ testbed (Right) Packet loss versus receiving gain using TOSSIM simulation.

Gain (dBM)	-70	-74	-76	-78	-81	-84	-87	-88	-89
Packet loss (%)	5	10	15	20	25	30	35	40	45

Table 4.4: Packet loss rates versus receiving gain using TOSSIM simulation.

For both *single-hop clique* and *multi-hop chain* topologies, we evaluate how DHV and DIP performance is impacted by different parameters including the total number of items T, the total number of new items N, the packet loss rates L, and the density D. Density refers to the number of radio communication neighbors. We compare DHV and DIP in a clique network. The default setting is $D = 32$ (nodes) (10 nodes for a multi-hop chain), $T = 64$ (keys), $N = 8$ (keys), $L = 5\%$. We vary D, T, N, and L.

For a *multi-hop grid* topology, there are two experiments with medium and high density networks. The total number of nodes is 225. The topology and link configurations are extracted from example files in TOSSIM (15-15-medium-mica2-grid.txt and 15-15-tight-mica2-grid.txt in tossim/topologies directory).

We characterize *density* by the average link gains. The smaller the gain is, the higher the packet reception rate of the link. Figure 4.15 shows the histogram of the link gains of medium and high density networks. For the medium density network, the majority of links have gains ranging from -120dB to -100dB while the high density network has the gain distribution around -100dB to -80dB.

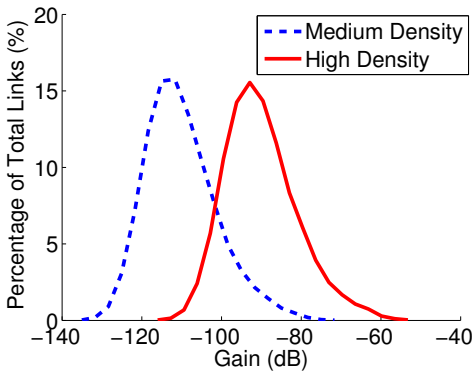


Figure 4.15: Link gain histogram: For medium density network, the majority of links have gain from -120dB to -100dB while the high density network has the gain distribution around -100dB to -80dB.

The send and receive activities are logged into log files. Based on the log files, we analyze the total number of transmitted messages and the the dissemination time. Each experiment is repeated 10 times to account for randomness in timing. Like DIP, DHV uses 2 (key, version number) tuples per VECTOR message to ensure comparability.

Experiments with Real Testbeds

We evaluated DHV and DIP on two real world testbeds; PSU SynLab - a one-hop MicaZ testbed at Portland State University and MoteLab - a multi-hop Tmote testbed at Havard University.

PSU-SynLab Testbed

As part of the research, we setup a embedded wireless sensor network testbed that we refer to as the Portland State University (PSU) SynLab testbed. The testbed has 64 MicaZ [106] wireless sensors deployed in the Systems and Networking Lab in the Department of Computer Science. Due to the small physical deployment area, the nodes in the testbed are within the communication range of each other. Hence, the testbed basically has a clique topology; any node can communicate directly with other nodes.

We use one MicaZ node to capture all the messages transmitted in the network. The node is connected to a MIB510 programming board and reports the received messages to a computer using an RS-232 serial communication. The messages are logged on the computer for analysis. Figure 4.14 shows the layout of the testbed.

We varied the number of sensor nodes D from 8 to 56, the total number of items from 8 to 128, and the number of new items N from 8 to 64. We observed the total number of transmitted messages and total time required to complete updating the whole network.

Energy Consumption Measurement on PSU-SynLab Testbed

To evaluate the performance in terms of energy consumption for dissemination and maintenance protocols in embedded sensor networks, we measure the total energy consumption by the PSU SynLab testbed in different scenarios.

Figure 4.16 shows our setup for collecting power measurements. We use an Agilent 6651A power supply to provide a consistent DC power source (4.5V/3A) to all nodes in the network. The Agilent 6651A is controlled from a computer using an IEEE-488-2 GPIB interface. An Agilent 34411A digital multi-meter is placed between the DC power supply and the sensor network to measure the DC current drawn by the network. The digital multi-meter is also controlled from a computer using Python scripts based on the PyVISA package [2]. The measurements are transferred to the computer via a TCP/IP connection. With this setup, we can collect about 130 samples per second.

MoteLab Testbed

In addition to the PSU SynLab testbed, we also conduct experiments with the MoteLab testbed [104], a multi-hop embedded wireless sensor network testbed at Harvard University. MoteLab has 190 Tmote sensor nodes, deployed across three floors in the Maxwell Dworkin Laboratory at Harvard University.

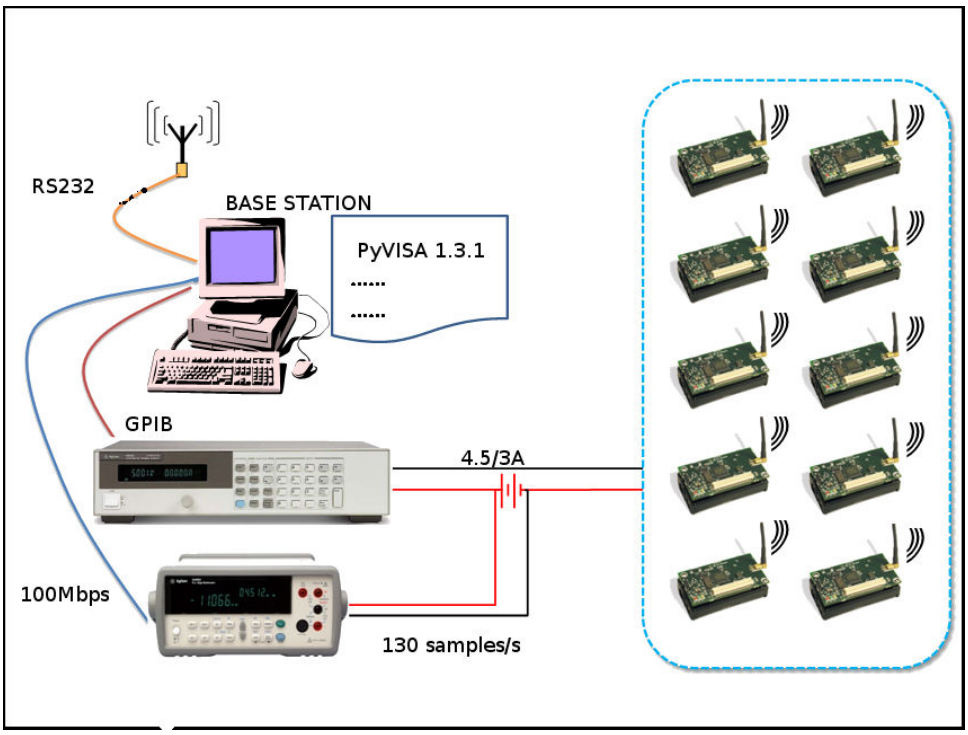


Figure 4.16: Power measurement setup: An Agilent 34411A digital multi-meter is placed between the DC power supply and the sensor network to measure the DC current drawn by the network. The digital multi-meter is also controlled from a computer using Python scripts based on the PyVISA package [2]. The measurements are transferred to the computer via a TCP/IP connection.

The Tmote sensor nodes have a TI MSP430 8 MHz processor with 10 KB of RAM, 1MB of Flash memory, and a Zigbee Chipcon CC2420 radio operating at 2.4 GHz. All nodes are connected directly to the Ethernet. Hence, the nodes can record their activities and send the information to a computer via the Ethernet connection.

We varied the number of items and observed update progress and time required to complete updating the whole network. The number of active nodes at the time of experiment was 121. The number of items was $T = 128$. The number of new items was $N = 8$ and $N = 120$ in each experiment.

We have described the experimental methodology we used to evaluate the DHV protocol. The next section describes the experimental results in detail.

4.8.3 Experimental Results

In this section, we describe the experimental results which we classify into two categories: simulation results and testbed results. First, we present results from TOSSIM simulations on a single-hop clique network, multi-hop chain network and multi-hop grid network respectively.

Single-hop Clique Network

Performance versus Total Number of Items: Figure 4.17 shows the comparison between DHV and DIP in a single-hop clique network when we vary the total number of items T. The other parameters are $D = 32$, $N = 8$, and $L = 5\%$. DHV performance in terms of total number of transmitted messages and tasking latency is relatively constant with T. Meanwhile, the number of transmitted messages and tasking latency of DIP increase as T increases. As an example case, when $T = 64$, nodes using DHV transmit only about 40% of the total number of messages and complete reprogramming within 45% of the time taken by nodes using DIP.

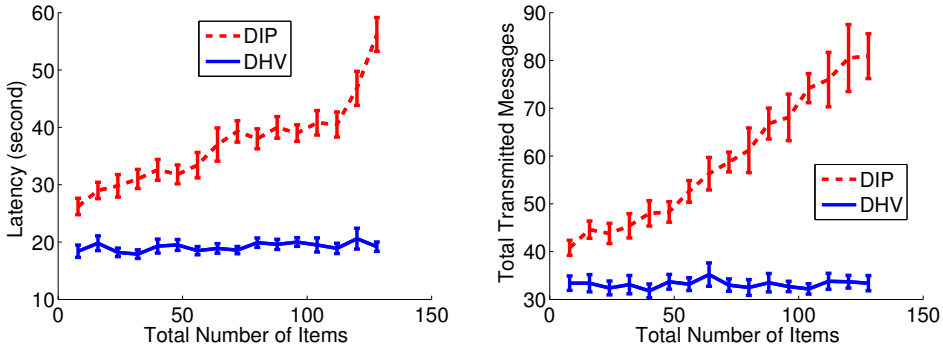


Figure 4.17: Tasking latency versus total items: $D = 32$, $N = 8$, $L = 5\%$. T varies from 8 to 128. DHV performance in terms of total number of transmitted messages and tasking latency is relatively constant with T. Meanwhile, the number of transmitted messages and tasking latency of DIP increase as T increases.

Performance versus Total Number of New Items: Figure 4.18 shows the comparison between DHV and DIP when we vary the number of new items. Nodes using DHV always transmit fewer messages than nodes using DIP. The other parameters are $D = 32$, $T = 64$, and $L = 5\%$. Nodes using DHV also use only half the time to complete updating the network compared to nodes using DIP. For example, when $N = 32$, nodes using DHV transmit about 70% of the messages transmitted by nodes using DIP and complete reprogramming in 50% of the time.

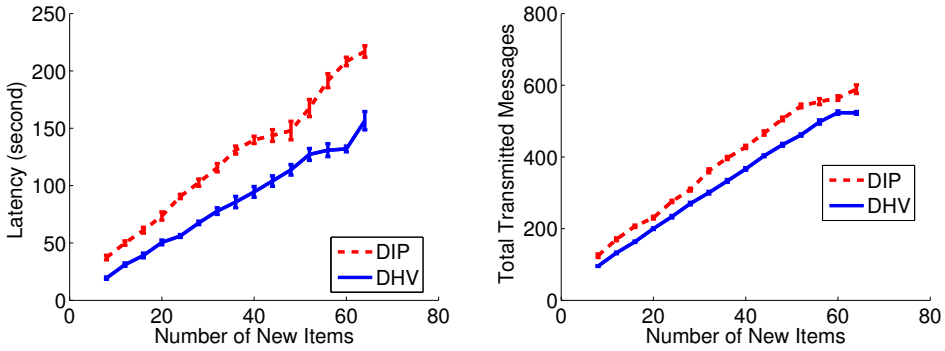


Figure 4.18: Total latency versus total new items: $D = 32$, $T = 64$, $L = 5\%$. N varies from 8 to 64. Nodes using DHV also use only half the time to complete updating the network compared to nodes using DIP.

Performance versus Number of Nodes: Figure 4.19 shows the comparison of DHV and DIP when we vary the number of nodes in a clique from 8 to 64. The other parameters are $T = 64$, $N = 8$, and $L = 5\%$. Nodes using DHV complete updating the network in 33% of the time and uses 50% fewer messages than nodes using DIP.

Performance versus Packet Loss Rate: Figure 4.20 shows the comparison of DHV and DIP when we vary the packet loss rate from 5% to 45%. The other parameters are $D = 32$, $T = 64$, and $N = 8$. DHV completely outperforms DIP in terms of latency. Nodes using DHV complete updating task items twice faster than nodes using DIP. Nodes using DHV transmit about 70% of number of messages to complete updating compared to nodes using DIP.

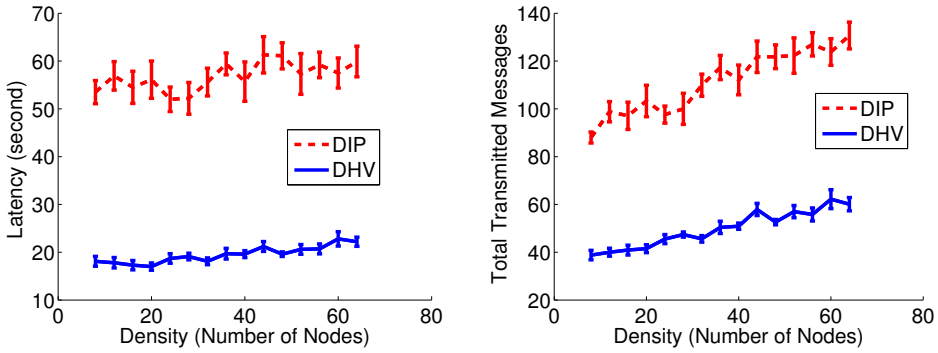


Figure 4.19: Total latency versus network density: $T = 64$, $N = 8$, $L = 5\%$. D varies from 8 to 64. Nodes using DHV complete updating the network in 33% of the time and uses 50% fewer messages than nodes using DIP.

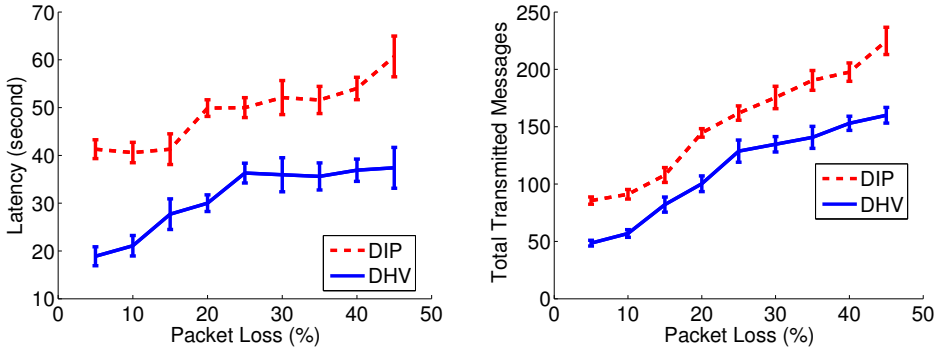


Figure 4.20: Total latency versus packet loss: $D = 32$, $T = 64$, $N = 8$. Packet loss rate L varies from 5% to 45%. Nodes using DHV complete updating task items twice faster than nodes using DIP. Nodes using DHV transmit about 70% of messages to complete updating compared to nodes using DIP.

We have just described the experimental results of DHV and DIP in a single-hop clique network. In the next section, we describe the results of the protocols in a multi-hop chain network.

Multi-hop Chain Network

Performance versus Total Number of Items: Figure 4.21 compares the performance of DHV and DIP in a 10-hop chain network when we vary the total number of items, T, from 8 to 128. The other parameters are $D = 10$, $N = 8$, and $L = 5\%$. DHV's performance is again relatively constant with T. Meanwhile, the number of transmitted messages and latency in DIP increase as T increases. As an example case, when $T = 64$, nodes using DHV transmit only about 40% of the total number of messages and complete reprogramming within 45% of the time taken by nodes using DIP.

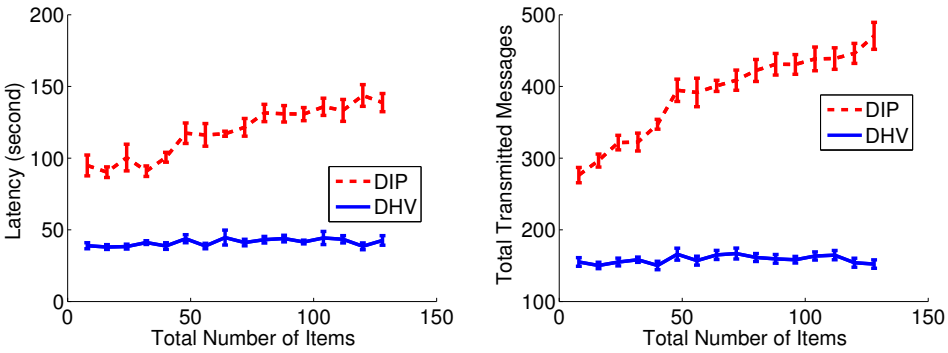


Figure 4.21: Total latency and transmitted messages versus total items: $D = 10$, $N = 8$, $L = 5\%$. T varies from 8 to 128. DHV's performance is again relatively constant with T. Meanwhile, the number of transmitted messages and latency in DIP increase as T increases.

Performance versus Total Number of New Items: Figure 4.22 shows the comparison of DHV and DIP in a 10-hop chain network when we vary the number of new items N from 8 to 64. The other parameters are $D = 10$, $T = 64$, and $L = 5\%$. Nodes using DHV always transmit fewer messages than nodes using DIP to complete updating the network. Nodes using DHV also spend

only 50% of the time to complete updating the network compared to nodes using DIP. For example, when $N = 32$, nodes using DHV transmit about 70% of the messages and complete tasking the network in 50% of the time compared to nodes using DIP.

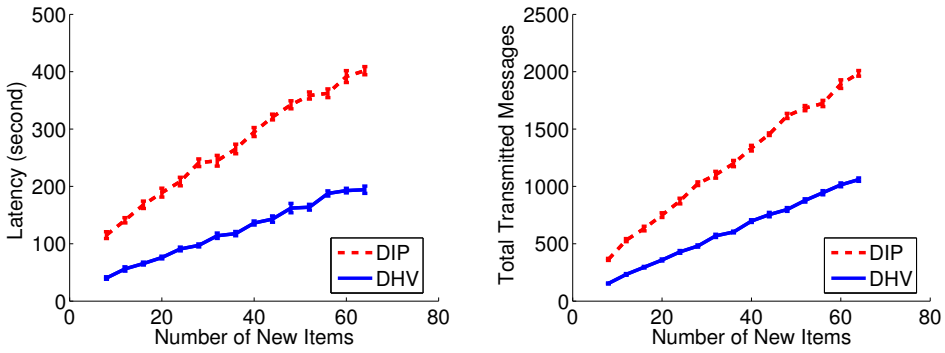


Figure 4.22: Total latency and transmitted messages versus total new items: $D = 10$, $T = 64$, $L = 5\%$. N varies from 8 to 64. Nodes using DHV always transmit fewer messages than nodes using DIP to complete updating the network. Nodes using DHV also spend only 50% of the time to complete updating the network compared to nodes using DIP.

Performance versus Number of Hops: Figure 4.23 compares DHV and DIP when we vary the number of nodes in the chain from 2 to 20. The other parameters are $T = 64$, $N = 8$, and $L = 5\%$. DHV completes updating in 33% of the time and uses 50% fewer messages than DIP.

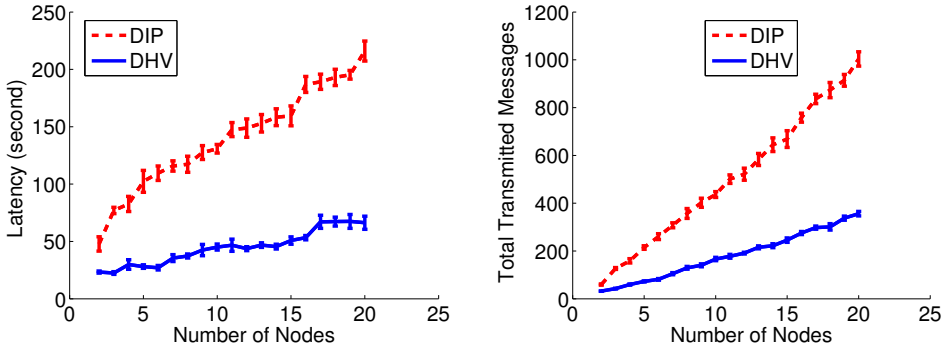


Figure 4.23: Total latency and transmitted messages versus network density: $T = 64$, $N = 8$, $L = 5\%$. D varies from 2 to 20. DHV completes updating in 33% of the time and uses 50% fewer messages than DIP.

Performance versus Packet Loss Rate: Figure 4.24 shows the comparison of DHV and DIP when we vary the packet loss rate L from 5% to 45%. The other parameters are $D = 10$, $T = 64$, and $N = 8$. In terms of total transmitted messages, DHV outperforms DIP at low packet loss rates. However, as the packet loss rate increases, DHV gets closer to DIP and exceeds DIP when the packet loss rate is greater than 35%. Similarly, nodes using DHV complete updating the network earlier than nodes using DIP when the packet loss rate is smaller than 25%. This is a surprise because DHV performs better than DIP under high packet loss rates in single-hop clique networks. One possible explanation is that in a clique network, a node can communicate with all other nodes and hence can still communicate with at least some nodes over lossy links and DHV can still perform searching for items that have different version numbers in approximately $O(1)$. Whereas, in a multi-hop chain network, a node can communicate with only two adjacent neighbor nodes. Therefore, it is likely that the node cannot communicate with any node in a broadcast over lossy links and the search restarts again and again.

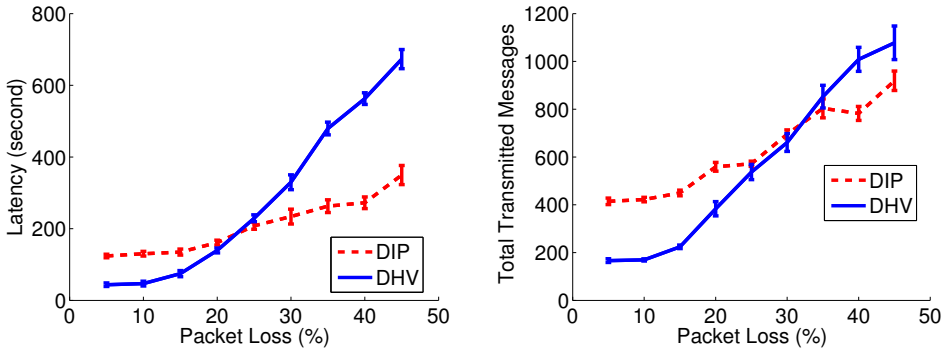


Figure 4.24: Performance versus number of nodes: $D = 10$, $T = 64$, $N = 8$. Packet loss rate L varies from 5% to 45%. DHV outperforms DIP at low packet loss rates. However, as the packet loss rate increases, DHV gets closer to DIP and exceeds DIP when the packet loss rate is greater than 35%.

Multi-hop Grid Network

Figure 4.25 plots the update time versus the number of completed nodes for DHV and DIP. The total number of items is $T = 128$. The number of new items is $N = 8$. It takes DHV about 50% and 70% of the time compared to DIP to update medium density networks (shown in the left figure) and tight density networks (shown in the right figure) respectively. In the medium density network, DHV and DIP update time grows linearly with the completed nodes because in each transmission, only a few nodes can receive the messages. The update progresses from the node with the new items and spreads out to the whole network. Hence, the number of completed nodes grows linearly with time. In contrast, in the high density network, when a node broadcasts, most other nodes receive the message. Hence, the network converges quickly. The inflection point when the number of completed nodes is around 200 can be explained by the fact that some nodes always receive messages with high noise. It takes much longer to complete updating these nodes.

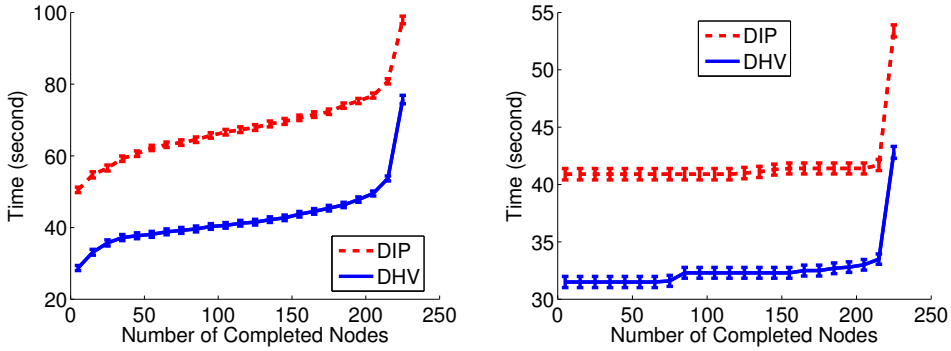


Figure 4.25: Convergence time for multi-hop networks: T=128 and N=8. It takes DHV about 50% and 70% of the time compared to DIP to update medium density networks (left) and tight density networks (right) respectively.

We have described our evaluation of DHV and DIP based on simulations. In the following section, we describe experimental results of DHV and DIP in real world testbeds.

Real Test-bed Results

We will organize the results based on two real world testbeds: the PSU-SynLab single-hop clique network and the MoteLab multi-hop network.

PSU-SynLab Single-hop Clique Network

Performance versus number of nodes: Figure 4.26 shows the total number of transmitted messages and the update time on the PSU-SynLab testbed. There are a total of $T = 64$ items and $N = 8$ new items. The number of nodes D is varied from 8 to 56 nodes. Nodes using DHV transmit 30% fewer total messages and complete updating earlier compared to nodes using DIP.

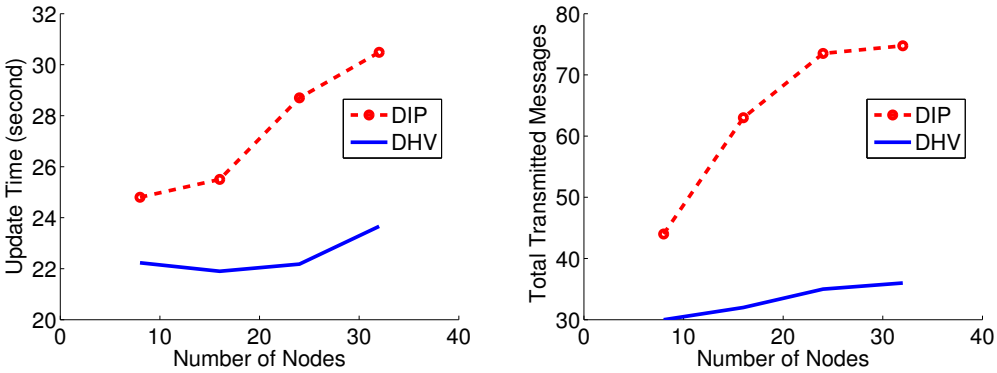


Figure 4.26: Total transmitted messages versus network density: $T = 64$, $N = 8$, D is varied from 8 to 56 nodes. Nodes using DHV transmit 30% fewer total messages and complete updating earlier compared to nodes using DIP.

This result confirms that DHV outperforms DIP in both simulation and real testbed. However, the performance of both DHV and DIP is slightly different from the simulation result in Figure 4.19. In particular, both DHV and DIP send fewer messages compared to the simulation scenario with 5% packet loss rate. It is likely that packet loss in our small testbed was much lower than in the simulation, hence, the improved performance of both protocols.

Energy Consumption versus Network Density: Figure 4.27 shows the total energy consumed for updating the whole network. The number of nodes in the network D is varied from 8 to 32 nodes. Nodes using DHV consume less energy than nodes using DIP. In particular, nodes using DHV consume around 70% of energy consumed by nodes using DIP to update the whole network.

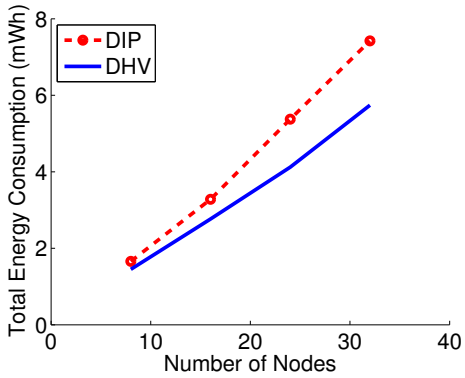


Figure 4.27: Energy consumption: D varies from 8 to 32 nodes. Nodes using DHV consume around 70% of energy consumed by nodes using DIP to update the whole network.

MoteLab Multi-hop Network

Performance versus Total Number of Items: Figure 4.28 shows the update time versus the total number of items, N = 8 and T varies from 8 to 128. DHV shows a relatively constant update time versus the total number of items. In contrast, DIP update time increases with the total number of items. Although, it is not very clear in the figure, DIP update time has logarithmic behavior with the total number of items.

Update Progress: Figure 4.29 shows the updating progress in terms of the fraction of the network that is updated versus time for two cases: updating 8 new items in 64 items and updating 120 new items in 128 items. DHV outperforms DIP, especially, when the total number of new items is high (e.g., 120 new items).

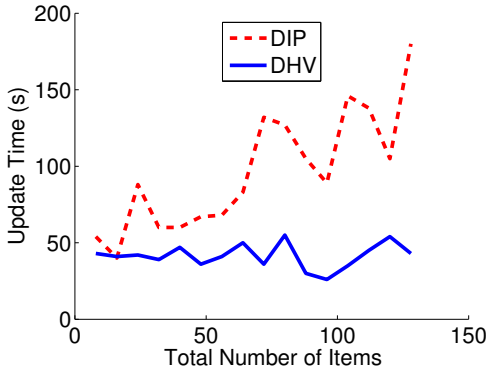
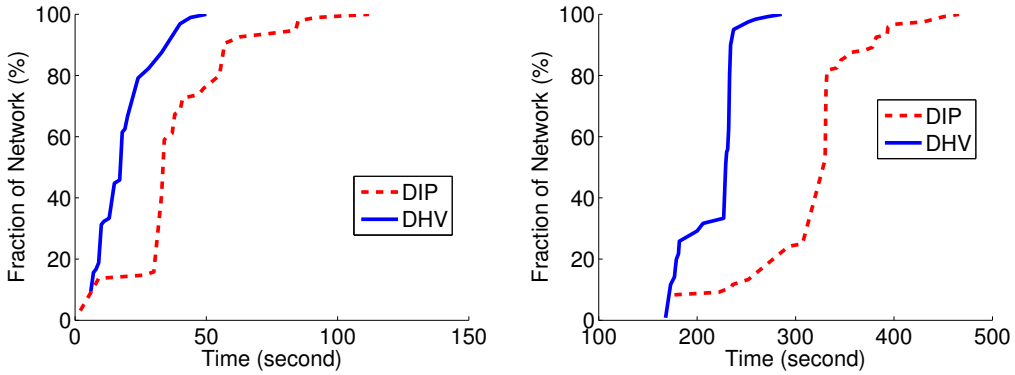


Figure 4.28: Tasking latency versus number of items: DHV shows a relatively constant programming time versus T while DIP updating time increases with T. DHV shows a relatively constant update time versus the total number of items. In contrast, DIP update time increases with the total number of items.



(a) (b)

Figure 4.29: Update progress: a) T=64, N = 8: DHV completes updating the network in 50% of the time compared to DIP. b) T = 128, N = 120: DHV completes updating in 50% of the time compared to DIP.

4.9 Protocol Selection Guidelines

In most cases, DHV outperforms existing dissemination protocols. However, DHV performs worse than DIP in a lossy multi-hop network with a chain topology. Hence, we, in general, recommend developers and operators to use DHV for dissemination in wireless sensor networks. If there are only a few items (fewer than 10 total items) to be disseminated in the network, a simple dissemination protocol such as DRIP might be suitable.

4.10 Summary

In this chapter, we designed, implemented, and evaluated the DHV protocol for dissemination and maintenance in embedded sensor networks. The key innovation in DHV is that it reduces the number of transmitted bits in the network by carefully selecting and transmitting only necessary information at the bit level to detect and identify which task items need updates. Together with a carefully designed suppression mechanism, DHV is able to reduce the total number of messages significantly. Theoretically, DHV can identify differences with $O(1)$ complexity in the total number of items instead of logarithmically compared to DIP. Simulations and real-world experiments validate that DHV performs better than the state-of-the-art DIP protocol in most scenarios. We believe that DHV can not only be used in embedded sensor networks but also in other distributed applications that require data consistency. DHV has been included as a core network library in the official release of TinyOS 2.1.1 and can be downloaded from `http://tinyos.net`.

Chapter 5

Conclusion and Future Work

We conclude this dissertation with a summary of our contributions and lessons learnt as well as directions for future work.

5.1 Summary

In this dissertation, we have identified and addressed the problem of scalable and efficient tasking, adapting the operations of nodes in a sensor networks to achieve the end users objectives. We explored tasking solutions for mobile sensor networks and embedded sensor networks. We believe this work is an important step to make sensor networks more widely adopted in many application domains. Due to the inherently different characteristics of embedded sensor networks and mobile sensor networks, we focused on different aspects of tasking for each network.

First, we identified the need for a task representation and encoding scheme that can efficiently encapsulate multiple sensor groups and their assigned tasks. This problem is particularly critical in mobile sensor networks that may be deployed over a large area. Therefore, we focused on addressing the problem of scalable and efficient task representation and encoding for mobile sensor networks. We designed and implemented the Zoom framework that allows users to group and assign tasks to sensors in non-uniform, fine-grained ways across a large sensing region for mobile sensor networks. The key ideas in Zoom are (i)

decoupling task specification and task implementation to support heterogeneity, (ii) using maps to represent sensor groups and their tasks, and (iii) using image encoding techniques to reduce the map size and provide adaptation to sensor platforms with different resource capabilities. Zoom is more intuitive, efficient and scalable compared to previous approaches.

Next, we identified the need for a scalable dissemination (spreading the task items to all nodes) and maintenance (making sure all nodes have the updated task items) protocol for sensor networks. It is a critical problem in embedded sensor networks, wherein dissemination can consume a significant amount of energy. Hence, we focused on addressing the problem of scalable and efficient dissemination and maintenance in embedded sensor networks. We designed and implemented the DHV protocol that allows nodes to disseminate and maintain tasks with fewer transmitted messages, lower latency, and less energy consumption compared to the state-of-the-art protocols.

DHV employs two key ideas. First, it uses a bit-level information exchange scheme to make DHV efficient and scalable with the number of task items and number of nodes. The key observation is that if the two item version numbers are different, they likely differ in a few least significant bits. Hence, a sensor should transmit only the most probable bits of the version numbers instead of transmitting the whole version numbers to identify task items that have newer version numbers. Second, DHV uses a gossip-based communication scheme to scale with the number of nodes. A node broadcasts information about its version numbers (e.g., hash of all the version numbers) randomly within each interval. However, if it receives messages with the same content as it has, it suppresses its transmission. Experimental results on both simulation and real testbeds show that DHV outperforms previous protocols by a factor of two in most cases. DHV has been included in the official distribution of TinyOS, a

popular operating system for embedded embedded sensor platforms with an average of 35,000 downloads a year, since version 2.1.1.

There are two main lessons that we have learned while doing the research of this dissertation.

Information-driven processing: In some situations, by utilizing information from an algorithm's inputs, we can improve the performance of the algorithm significantly. In DHV, by exchanging only the bits in the item version numbers that are likely to be different from the others, we can reduce the number of transmitted messages and hence reduce the maintenance latency and conserve energy and bandwidth. Using the same approach, we rearrange sensor data based on the data values to achieve better compression performance of the sensor data [107]. We believe this approach is also useful in many other cases.

Map-based representation: Maps by themselves represent information on some coordinates. They have been used in several applications. Binary maps are used to keep track of unused sectors in hard disks [108]. Maps are also used to present activities in computer vision [109]. In this dissertation, we also demonstrate that maps are efficient to represent sensor tasks and groups. We believe that maps are also useful in other cases such as summarizing spatial sensor data.

5.2 Impact

The DHV protocol implementation is included in the TinyOS distribution version 2.1.1 as open source software since August 2009. The source code of TinyOS can be downloaded from

`http://code.google.com/p/tinyos-main/`.

The specific source code for DHV is in the `/tos/lib/net/dhv/` directory and the test application is in the `/apps/tests/TestDHV/` directory.

The source code for Zoom is also available at

`http://sys.cs.pdx.edu/home/projects/zoom`.

5.3 Future Directions

The work presented in this dissertation is a step toward enabling wide adoption of sensor networks. There are several interesting future directions that are motivated by either our work or the general problem of tasking sensor networks.

Real-time Tasking of Sensor Networks: Guarantees in tasking latency are very important in time critical applications. For example, the network operator needs to know when the network has finished updating new tasks to start making use of the sensor data. Hard real-time is difficult to achieve. Soft real-time is often useful for networking planning and management. Tasking latency depends on many internal and external factors such as the number of nodes in the network, link quality, routing protocols, and medium access mechanisms. These factors make it challenging to understand and manage tasking latency. There has been prior work [110, 111] that explores the possibility of providing timing guarantees in routing. However, most of the other factors are not well understood.

Task and Sensor Data Representation: We believe there are numerous applications of maps in representing spatial information for sensor networks. In this dissertation, we apply standard compression techniques to compress the task maps. We can reduce the compressed file size by carefully assigning the task numbers that result in the best compression performance. However, it is not well understood how to do so dynamically with different number of tasks and sensor groups. Spatial sensor data can also be summarized using maps to reduce the file size. For example, to monitor temperature in a forest using sensors attached to animals, the temperature values with their locations can be

summarized into a map where the locations of the pixels on the map correspond to the physical locations and the pixels' values are the temperature values.

Tasking Nano Scale Sensing Networks: In this dissertation, we addressed the problem of scalable and efficient tasking of mobile sensor networks and embedded sensor networks. The sensor platforms we considered have small form factor, some computation capability, and wireless communication. We have not considered other sensor and actuator platforms such as programmable matter [112] and nano-sensors [113]. Such platforms can be deeply embedded into physical space or as part of the physical objects themselves. Such platforms have different computation and communication mechanisms compared to traditional computers. Hence, tasking these devices also requires new methods and approaches.

References

[1] Thanh Dang, Nirupama Bulusu, Wu-Chi Feng, and Seungweon Park, "Dhv: A code consistency maintenance protocol for multi-hop wireless sensor networks," in *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN'09)*, Cork, Ireland, February 2009, pp. 327–342.

[2] Torsten Bronger, "Python gpib etc. support with pyvisa," June 2006.

[3] Jenna Burrell, Tim Brooke, and Richard Beckwith, "Vineyard computing: Sensor networks in agricultural production," *IEEE Pervasive Computing*, vol. 3, pp. 38–45, January 2004.

[4] V.C. Gungor, Bin Lu, and G.P. Hancke, "Opportunities and challenges of wireless sensor networks in smart grid," *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 10, pp. 3557 –3564, October 2010.

[5] Paolo Costa, Geoff Coulson, Richard Gold, Manish Lad, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco, Thirunavukkarasu Sivaharan, Nirmal Weerasinghe, and Stefanos Zachariadis, "The runes middleware for networked embedded systems and its application in a disaster management scenario," in *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, Washington, District of Columbia, USA, 2007, pp. 69–78, IEEE Computer Society.

[6] Heribert Baldus, Karin Klabunde, and G. Msch, "Reliable set-up of medical body-sensor networks," in *Proceedings of the First EuropeanWorkshop (EWSN' 04)*, Berlin, Germany, February 2004, pp. 353–363.

[7] Murat Demirbas, "Wireless sensor networks for monitoring of large public buildings," *Computer Networks*, vol. 46, pp. 605–634, 2005.

[8] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proceedings of the 7th symposium on Operating systems design and implementation*, Berkeley, California, USA, 2006, OSDI '06, pp. 381–396, USENIX Association.

[9] Rajib Kumar Rana, Chun Tung Chou, Salil S. Kanhere, Nirupama Bulusu, and Wen 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)*, Stockholm, Sweden, 2010, pp. 105–116, ACM.

[10] Saurabh Amin, Steve Andrews, Saneesh Apte, Jed Arnold, Jeff Ban, Marika Benko, Re M. Bayen, Benson Chiou, Christian Claudel, Coralie Claudel, Tia Dodson, Juan carlos Herrera, Ryan Herring, Quinn Jacobson, Toch Iwuchukwu, James Lew, Xavier Litrico, Lori Luddington, Jd Margulici, Ali Mortazavi, Xiaohong Pan, Tarek Rabbani, Tim Racine, Erica Sherlock-thomas, Dave Sutter, and Andrew Tinka, "Mobile century using gps mobile phones as traffic sensors: A field experiment," in *15th World Congress on Intelligent Transportation Systems 2008*, 2008, p. 18.

[11] James Horey, Arthur B. Maccabe, and Angela Mielke, "Kensho: a dynamic tasking architecture for sensor networks," *SIGBED Rev.*, vol. 4, pp. 19–24, July 2007.

[12] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *Workshop on World-Sensor-Web (WSW06)*, 2006, pp. 117–134.

[13] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, and Ronald A. Peterson, "People-centric urban sensing," in *Proceedings of the 2nd Annual International Wireless Internet Conference (WICON' 06)*, Boston, Massachusetts, 2006, p. 18, ACM.

[14] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler, "The emergence of networking abstractions and techniques in tinyos," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, San Francisco, California, 2004, pp. 1–1.

[15] Kaisen Lin and Philip Levis, "Data discovery and dissemination with dip," in *Proceedings of the 2008 International Conference on Information Processing in Sensor Networks (IPSN 2008)*, Washington, District of Columbia, USA, 2008, pp. 433–444.

[16] Luca Mottola and Gian Pietro Picco, "Programming wireless sensor networks: Fundamental concepts and state-of-the-art," *ACM Computing Surveys (Accepted)*, , no. 1, pp. 1–57, 2009.

[17] Zigbee, "Low-rate wireless personal area networks," May 2011.

[18] Daniel B. Work, Olli-Pekka Tossavainen, Quinn Jacobson, and Alexandre M. Bayen, "Lagrangian sensing: traffic estimation with mobile devices," in *Proceedings of the 2009 conference on American Control Conference*, St. Louis, Missouri, USA, 2009, ACC'09, pp. 1536–1543, IEEE Press.

[19] Wen-Zhan Song, Renjie Huang, Mingsen Xu, Andy Ma, Behrooz Shirazi, and Richard LaHusen, "Air-dropped sensor network for real-time high-fidelity volcano monitoring," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, Kraków, Poland, 2009, MobiSys '09, pp. 305–318, ACM.

[20] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet," in *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*. 2002, ASPLOS-X, pp. 96–107, ACM.

[21] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, New York, NY, USA, 2002, WSNA '02, pp. 88–97, ACM.

[22] Jerry Zhao and Ramesh Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, Los Angeles, California, USA, 2003, SenSys '03, pp. 1–13, ACM.

[23] Crossbow, "Micaz mote platform," in *MicaZ Mote Platform*, San Jose, CA, Oct Oct 2009, pp. 1–1.

[24] Hailun Tan, "Maximizing network lifetime in energy-constrained wireless sensor network," in *Proceedings of the 2006 international conference on Wireless communications and mobile computing*, Vancouver, British Columbia, Canada, 2006, IWCMC '06, pp. 1091–1096, ACM.

[25] Samuel Madden, Micheal J.Franklin, Joseph Hellerstein, and Wei Hong, "Tinydb: An acquisitional query processing system for sensor networks," *ACM Transaction on Database System*, vol. 30, no. 1, pp. 122–173, March 2005.

[26] Matt Welsh and Geoff Mainland, "Programming sensor networks using abstract regions," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, San Francisco, California, 2004, pp. 3–3, USENIX Association.

[27] Philip Levis, Neil Patel, David Culler, and Scott Shenker, "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, San Francisco, California, 2004, pp. 2–2, USENIX Association.

[28] Luca Mottola and Gian Pietro Picco, "Programming wireless sensor networks with logical neighborhoods," in *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, Nice, France, 2006, InterSense '06, ACM.

[29] Open Geospatial Consortium, "Sensor model language (sensorml)," 2010.

[30] Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler, "smap: a simple measurement and actuation profile for physical information," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, Zurich, Switzerland, 2010, SenSys '10, pp. 197–210, ACM.

[31] Karen Henricksen and Ricky Robinson, "A survey of middleware for sensor networks: state-of-the-art and future directions," in *Proceedings of the*

international workshop on Middleware for sensor networks, Melbourne, Australia, 2006, MidSens '06, pp. 60–65, ACM.

[32] World Wide Web Consortium, "Mathematical markup language (mathml) version 3.0," June 2010.

[33] Ben Greenstein, Eddie Kohler, and Deborah Estrin, "A sensor network application construction kit (snack)," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Baltimore, MD, USA, 2004, SenSys '04, pp. 69–80, ACM.

[34] Phillip Sitbon, Wu chi Feng, Nirupama Bulusu, and Thanh Dang, "Sensetk: A multimodal, multimedia sensor networking toolkit," in *Proceedings of the 14th Annual Multimedia Computing and Networking Conference*, San Jose, CA, 2007, pp. 60–65.

[35] René Müller, Gustavo Alonso, and Donald Kossmann, "A virtual machine for sensor networks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, Lisbon, Portugal, 2007, EuroSys '07, pp. 145–158, ACM.

[36] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler, "The nesc language: A holistic approach to networked embedded systems," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, San Diego, California, USA, 2003, PLDI '03, pp. 1–11.

[37] Yong Yao and Johannes Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, 2002.

[38] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler, "Hood: a neighborhood abstraction for sensor networks," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, Boston, MA, USA, 2004, MobiSys '04, pp. 99–110, ACM.

[39] Liqian Luo, Tarek F. Abdelzaher, Tian He, and John A. Stankovic, "Envirosuite: An environmentally immersive programming framework for sensor networks," *ACM Transactions on Embedded Computer Systems*, vol. 5, pp. 543–576, August 2006.

[40] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating Systems Review*, vol. 36, pp. 131–146, December 2002.

[41] Christian Frank and Kay Rómer, "Algorithms for generic role assignment in wireless sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (Sensys' 05)*, San Diego, California, USA, 2005, pp. 230–242, ACM.

[42] Nupur Kothari, Ramakrishna Gummadi, Todd Millstein, and Ramesh Govindan, "Reliable and efficient programming abstractions for wireless sensor networks," in *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, San Diego, California, USA, 2007, PLDI '07, pp. 200–210, ACM.

[43] Kirsten Terfloth, "Facts - a rule-based middleware architecture for wireless sensor networks," in *Proceedings of the first international conference on communication system software and middleware (COMSWARE06*, New Delhi, India, January 2006, pp. 1–8.

[44] Ryan Newton, Greg Morrisett, and Matt Welsh, "The regiment macroprogramming system," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN' 07)*, Cambridge, Massachusetts, USA, 2007, pp. 489–498, ACM.

[45] Amol Bakshi, Viktor K. Prasanna, Jim Reich, and Daniel 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*.

[46] Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan, "Macro-programming wireless sensor networks using kairos," in *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS' 05)*, Marina del Rey, California, 2005, pp. 126–140.

[47] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, Columbus, Ohio, 2005, ICDCS '05, pp. 653–662, IEEE Computer Society.

[48] CENS, "What's noisy," May 2011.

[49] Sandra Henderson, "Project budburst: Timing is everything," May 2011.

[50] Nirupama Bulusu, Chun Tung Chou, Salil Kanhere, Yifei Dong, Shitiz Sehgal, David Sullivan, and Lupco Blazeski, "Participatory sensing in commerce: Using mobile camera phones to track market price dispersion," in *Proceedings of the International Workshop on Urban, Community, and*

Social Applications of Networked Sensing Systems, Raleigh, NC, 2009, pp. 1–5.

[51] CENS, "Bikestatic: Bike what's good," May 2011.

[52] Ashley, "Boyle heights: Truck stop," May 2011.

[53] Donnie Kim, Nicolai Petersen, Mohammad Rahimi, Jeff Burke, and Deborah Estrin, "Smartphone data acquisition and analysis for monitoring food choices," April 2011.

[54] Miguel A. Laguna, Javier Finat, and José A. González, "Remote health monitoring: A customizable product line approach," in *Proceedings of the 10th International Work-Conference on Artificial Neural Networks: Part II: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Berlin, Heidelberg, 2009, IWANN '09, pp. 727–734, Springer-Verlag.

[55] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson, "Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, Berkeley, California, 2009, SenSys '09, pp. 85–98, ACM.

[56] Google, "Availability of real-time traffic," May 2011.

[57] Daniel B. Work, Olli-Pekka Tossavainen, Quinn Jacobson, and Alexandre M. Bayen, "Lagrangian sensing: traffic estimation with mobile devices," in *Proceedings of the 2009 conference on American Control Conference*, St. Louis, Missouri, USA, 2009, ACC'09, pp. 1536–1543, IEEE Press.

[58] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew 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*, Raleigh, NC, USA, 2008, SenSys '08, pp. 337–350, ACM.

[59] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler, "Hood: a neighborhood abstraction for sensor networks," in *Proceedings of the Mobile Systems (MobiSys' 04)*, Boston, MA, 2004, pp. 99–110.

[60] 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 the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan, 2004, pp. 582–589.

[61] Ralph Kling, "Intel research mote," June 2010.

[62] Oracle Labs, "Sun spot world," June 2010.

[63] Arch Rock Corporation, "Arch rock's energy optimize," June 2010.

[64] Shimmer, "Shimmer sensor platform," May 2011.

[65] Crossbow, "Telosb sensor platform," May 2011.

[66] Pat Kinney, "Eee 802.15 wpan task group 4 (tg4)," May 2011.

[67] F. Ayazi and K. Najafi, "A harpss polysilicon vibrating ring gyroscope," *Microelectromechanical Systems, Journal of*, vol. 10, no. 2, pp. 169 –179, jun 2001.

[68] Lizzie Tang and Chris Guy, "Radio frequency energy harvesting in wireless sensor networks," in *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*, Leipzig, Germany, 2009, IWCMC '09, pp. 644–648, ACM.

[69] Jaeseok Yun, Shwetak Patel, Matt Reynolds, and Gregory Abowd, "A quantitative investigation of inertial power harvesting for human-powered devices," in *Proceedings of the 10th international conference on Ubiquitous computing*, Seoul, Korea, 2008, UbiComp '08, pp. 74–83, ACM.

[70] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister, "System architecture directions for networked sensors," *SIGPLAN Not.*, vol. 35, pp. 93–104, November 2000.

[71] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, Washington, District of Columbia, USA, 2004, LCN '04, pp. 455–462, IEEE Computer Society.

[72] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava, "A dynamic operating system for sensor nodes," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, Seattle, Washington, 2005, MobiSys '05, pp. 163–176, ACM.

[73] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han, "Mantis: system support for multimodal networks of in-situ sensors," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, San Diego, California, USA, 2003, WSNA '03, pp. 50–59, ACM.

[74] Hojung Cha, Sukwon Choi, Inuk Jung, Hyoseung Kim, Hyojeong Shin, Jaehyun Yoo, and Chanmin Yoon, "Retos: resilient, expandable, and threaded operating system for wireless sensor networks," in *Proceedings of the 6th international conference on Information processing in sensor networks*, Cambridge, Massachusetts, USA, 2007, IPSN '07, pp. 148–157, ACM.

[75] Lin Gu and John A. Stankovic, "t-kernel: providing reliable os support to wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, Boulder, Colorado, USA, 2006, SenSys '06, pp. 1–14, ACM.

[76] Anand Eswaran, Anthony Rowe, and Raj Rajkumar, "Nano-rk: An energy-aware resource-centric rtos for sensor networks," in *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, Washington, District of Columbia, USA, 2005, pp. 256–265, IEEE Computer Society.

[77] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali, "Protothreads: simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, Boulder, Colorado, USA, 2006, SenSys '06, pp. 29–42, ACM.

[78] Kevin Klues, Chieh-Jan Mike Liang, Jeongyeup Paek, Răzvan Musăloiu-E, Philip Levis, Andreas Terzis, and Ramesh Govindan, "Tosthreads: thread-safe and non-invasive preemption in tinyos," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, Berkeley, California, 2009, SenSys '09, pp. 127–140, ACM.

[79] Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz, and Jonathan Lees, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[80] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong, "A macroscope in the redwoods," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005, SenSys '05, pp. 51–63, ACM.

[81] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of the 6th international conference on Information processing in sensor networks*, Cambridge, Massachusetts, USA, 2007, IPSN '07, pp. 254–263, ACM.

[82] A. Sheth, K. Tejaswi, P. Mehta, C. Parekh, R. Bansal, S. Merchant, T. Singh, U. B. Desai, C. A. Thekkath, and K. Toyama, "Senslide: a sensor network based landslide prediction system," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, San Diego, California, USA, 2005, SenSys '05, pp. 280–281, ACM.

[83] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi, "Hardware design experiences in zebranet," in *Proceedings of the second ACM Conference on Embedded Networked Sensor Systems (SenSys' 04)*, Baltimore, Maryland, 2004, pp. 227–238.

[84] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton, "Sensor network-based countersniper system," in *Proceedings of the 2nd inter-*

national conference on Embedded networked sensor systems, Baltimore, MD, USA, 2004, SenSys '04, pp. 1–12, ACM.

[85] Wen Hu, Van Nghia Tran, Nirupama Bulusu, Chun Tung Chou, Sanjay Jha, and Andrew Taylor, "The design and evaluation of a hybrid sensor network for cane-toad monitoring," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, Los Angeles, California, 2005, p. 71, IEEE Press.

[86] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler, "An analysis of a large scale habitat monitoring application," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Baltimore, MD, USA, 2004, SenSys '04, pp. 214–226, ACM.

[87] Milan Milenkovic, "Eco-sense buildings," May 2011.

[88] Brewster McCracken, "Pecan street project," May 2011.

[89] Konrad Lorincz, Bor-rong Chen, Geoffrey Werner Challen, Atanu Roy Chowdhury, Shyamal Patel, Paolo Bonato, and Matt Welsh, "Mercury: a wearable sensor network platform for high-fidelity motion analysis," in *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, Berkeley, California, 2009, pp. 183–196, ACM.

[90] Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta, and David Culler, "Marionette: using rpc for interactive development and debugging of wireless embedded networks," in *Proceedings of the fifth international conference on Information processing in sensor networks (IPSN 06)*, Nashville, Tennessee, USA, 2006, pp. 416–423, ACM.

[91] Philip Levis and David Culler, "Maté: a tiny virtual machine for sensor networks," *SIGPLAN Not.*, vol. 37, no. 10, pp. 85–95, 2002.

[92] Omprakash Gnawali, Ki-Young Jang, Jeongyeup Paek, Marcos Vieira, Ramesh Govindan, Ben Greenstein, August Joki, Deborah Estrin, and Eddie Kohler, "The tenet architecture for tiered sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys 06)*, Boulder, Colorado, USA, 2006, pp. 153–166, ACM.

[93] Jonathan W. Hui and David Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of ACM Sensys 04*, Baltimore, Maryland, USA, 2004, pp. 81–94, ACM.

[94] Sandeep S. Kulkarni and Limin Wang, "Mnp: Multihop network reprogramming service for sensor networks," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS 05)*, Washington, District of Columbia, USA, 2005, pp. 7–16.

[95] Mert Akdere, Cemal Çagatay Bilgin, Ozan Gerdaneri, Ibrahim Korpeoglu, Özgür Ulusoy, and Ugur Çetintemel, "A comparison of epidemic algorithms in wireless sensor networks," *Computer Communications*, vol. 29, no. 13-14, pp. 2450–2457, 2006.

[96] Larry Wood, "Graphics interchange format(sm) version 89a," *Graphics Interchange Format Programming Reference*, vol. 1, no. 1, pp. 1–34, 1990.

[97] Abraham Lempel and Jacob Ziv, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 1, pp. 337–342, 1977.

[98] Wu chi Feng, Thanh Dang, John Kassebaum, and Tim Bauman, "Supporting region-of-interest cropping through constrained compression," in *Proceeding of ACM Multimedia 2008)*, Vancouver, British Columbia, 2008, pp. 745–748.

[99] Christopher M. Sadler and Margaret Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proccedings of ACM Sensys 06)*, Boulder, Colorado, Nov. 2006, pp. 265–279.

[100] Kazuki Konishi, Kumiko Maeda, Kazuki Sato, Akiko Yamasaki, Hirozumi Yamaguchi, Teruo Higashino, and Keiichi Yasumoto, "Mobireal simulator evaluating manet applications in real environments," in *Proceedings of the MASCOTS 2005*, Atlanta, Georgia, September 2005, p. 537.

[101] George F. Riley, "Simulation of large scale networks ii: large-scale network simulations with gtnets," in *Proceedings of the 35th Winter Simulation Conference (WSC' 03)*, New Orleans, Louisiana, 2003, pp. 676–684, Winter Simulation Conference.

[102] Crossbow, "Telosb mote platform," in *TelosB Mote Platform*, San Jose, CA, Oct Oct 2009, pp. 1–1.

[103] Philip Levis, Nelson Lee, Matt Welsh, and David Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems (Sensys 03)*, Los Angeles, California, USA, 2003, pp. 126–137, ACM.

[104] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh, "Motelab: a wireless sensor network testbed," in *Proceedings of the 4th international*

symposium on Information processing in sensor networks, Los Angeles, California, 2005, IPSN '05, IEEE Press.

[105] HyungJune Lee, Alberto Cerpa, and Philip Levis, "Improving wireless simulation through noise modeling," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN 2007*, Cambridge, Massachusetts, April 25-27, 2007, pp. 21–30.

[106] Marc Kramer and Alexander Geraldy, "Energy measurements for micaz node," in *Technical Report, Technical University Kaisers Lautern,GI/ITG KuVS, (2006)*, pp. 1–7.

[107] Thanh Dang, Nirupama Bulusu, and Wu-chi Feng, "Rida: A robust information-driven data compression aorchitecture for irregular wireless sensor networks," in *Proceedings of the Forth the European conference on Wireless Sensor Networks, EWSN 07*, Delft, The NetherLands, January January 2007, pp. 133–149.

[108] Jeff Bonwick, "Space maps," *Sun Blog*, vol. 38, pp. 393–394, 2010.

[109] D. Demirdjian, K. Tollmar, K. Koile, N. Checka, and T. Darrell, "Activity maps for location-aware computing," in *Proceedings of the Sixth IEEE Workshop on Applications of Computer Vision*, Washington, District of Columbia, USA, 2002, WACV '02, pp. 70–, IEEE Computer Society.

[110] Tian He, John A. Stankovic, Chenyang Lu, and Tarek Abdelzaher, "Speed: A stateless protocol for real-time communication in sensor networks," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, Providence, Rhode Island, 2003, ICDCS '03, pp. 46–, IEEE Computer Society.

[111] Jingyuan Li, Yafeng Wu, Krasimira Kapitanova, John A. Stankovic, Kamin Whitehouse, and Sang H. Son, "Run time assurance of application-level requirements in wireless sensor networks," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, Berkeley, California, 2009, SenSys '09, pp. 367–368, ACM.

[112] Seth Copen Goldstein, Todd C. Mowry, Jason D. Campbell, Michael P. Ashley-Rollman, Michael De Rosa, Stanislav Funiak, James F. Hoburg, Mustafa Emre Karagozler, Brian Kirby, Peter Lee, Padmanabhan Pillai, J. Robert Reid, Daniel D. Stancil, and Michael Philetus Weller, "Beyond audio and video: Using claytronics to enable pario," *AI Magazine*, vol. 30, no. 2, July 2009.

[113] Carlos Adolfo Piña García, Ericka-Janet Rechy-Ramírez, and V. Angélica García-Vega, "Comparing three simulated strategies for cancer monitoring with nanorobots," in *Proceedings of the 7th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence*, Berlin, Heidelberg, 2008, MICAI '08, pp. 1020–1030, Springer-Verlag.