Technical Report

UCAM-CL-TR-739
ISSN 1476-2986

Number 739

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

State-based Publish/Subscribe for sensor systems

Salman Taherian

January 2009

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted
June 2008 by the author for the degree of Doctor of
Philosophy to the University of Cambridge, St John's College.

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

Recent technological advances have enabled the creation of networks of sensor devices. These devices are typically equipped with basic computational and communication capabilities. Systems based on these devices can deduce high-level, meaningful information about the environment that may be useful to applications. Due to their scale, distributed nature, and the limited resources available to sensor devices, these systems are inherently complex. Shielding applications from this complexity is a challenging problem.

To address this challenge, I present a middleware called SPS (State-based Publish/Subscribe). It is based on a combination of a *State-Centric* data model and a *Publish/Subscribe (Pub/Sub)* communication paradigm. I argue that a state-centric data model allows applications to specify environmental situations of interest in a more natural way than existing solutions. In addition, Pub/Sub enables scalable many-to-many communication between sensors, actuators, and applications.

This dissertation initially focuses on Resource-constrained Sensor Networks (RSNs) and proposes State Filters (SFs), which are lightweight, stateful, event filtering components. Their design is motivated by the redundancy and correlation observed in sensor readings produced close together in space and time. By performing context-based data processing, SFs increase Pub/Sub expressiveness and improve communication efficiency.

Secondly, I propose State Maintenance Components (SMCs) for capturing more expressive conditions in heterogeneous sensor networks containing more resourceful devices. SMCs extend SFs with data fusion and temporal and spatial data manipulation capabilities. They can also be composed together (in a DAG) to deduce higher level information. SMCs operate independently from each other and can therefore be decomposed for distributed processing within the network.

Finally, I present a Pub/Sub protocol called QPS (Quad-PubSub) for location-aware Wireless Sensor Networks (WSNs). QPS is central to the design of my framework as it facilitates messaging between state-based components, applications, sensors, and actuators. In contrast to existing data dissemination protocols, QPS has a layered architecture. This allows for the transparent operation of routing protocols that meet different Quality of Service (QoS) requirements.

To my home: my parents and my siblings

Acknowledgement

First of all, I would like to thank Jean Bacon, my supervisor, who has provided tremendous advice and support during the time of my PhD. Her insight helped my work go in the right direction, and she taught me the principles of research work. I am also grateful to Jon Crowcroft, my PhD advisor, who provided great guidance and feedback for my work.

I have had the pleasure of being a member of the Opera Group. In particular, I enjoyed discussing research with Daniel O'Keeffe, Sriram Srinivasan, David Evans, David Eyers, and Pedro Brandao. Many ideas stem from (often hour-long) coffee-breaks with colleagues that included Dan, David Evans, David Eyers, David Ingram, Eiko, Jat, Luis, Pedro, and Sriram. I would also like to thank Daniel O'Keeffe, David Evans, David Eyers, and Pedro Brandao for proof-reading parts of my thesis and suggesting improvements.

My research was funded by Microsoft Research (MSR) in Cambridge. I am also thankful to Helen Watson, my college tutor, Audrey Lanfear, Wendy Redgewell, and Melanie Prior (college secretaries) for supporting my studies at St. John's and providing endless letters for my stay, travel, accommodation, and research grants. Finally, I would like to thank my parents for their support and encouragement throughout the years, which made it all possible.

List of Publications

[TOB04] Taherian, S., O'Keeffe, D. & Bacon, J. (2004). Event dissemination in mobile wireless sensor networks. In *Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, pages 573–575, Fort Lauderdale, FL, USA, October 2004.

[TB07c] Taherian, S. & Bacon, J. (2007). State-filters for enhanced filtering in sensor-based publish/subscribe systems. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 346–350, Mannheim, Germany, May 2007. IEEE Computer Society.

[TB07a] Taherian, S. & Bacon, J. (2007). A publish/subscribe protocol for resource-awareness in wireless sensor networks. In J. Aspnes, C. Scheideler, A. Arora & S. Madden, eds., *Proceedings of the International Workshop on Localized Algorithms and Protocols for Wireless Sensor Networks (LOCALGOS)*, volume 4549 of *Lecture Notes in Computer Science (LNCS)*, pages 27–38, Santa Fe, NM, USA, June 2007. IEEE Computer Society, Springer-Verlag.

[TB07b] Taherian, S. & Bacon, J. (2007). SPS: A middleware for multi-user sensor systems. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC)*, pages 19–24, New York, NY, USA, November 2007. ACM.

[TB08] Taherian, S. & Bacon, J. (2008). Capturing High-Level Conditions, using a Publish/Subscribe Middleware, in Sensor Systems. In *Proceedings of the IET International Conference on Intelligent Environments (IE)*, Seattle, WA, USA, July 2008. IET. To Appear.

Contents

List of Figures

List of Tables

List of Acronyms

ACK Acknowledgment

ADT Abstract Data Type

ANPR Automatic Number Plate Recognition

AOA Angle of Arrival

API Application Programming Interface

CE Composite Event

CODD Cross-layer Opportunistic-sharing Data Dissemination

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance

CTS Clear to Send

DAG Directed Acyclic Graph

DBMS Database Management System

DCF Distributed Coordination Function

DCS Data-Centric Storage

DHT Distributed Hash Table

DK Detected Knowledge

DSMS Data Stream Management System

EA Event-Action

EB Event Broker

EC Event Client

EAI Electronic Application Integration

ECA Event-Condition-Action

ECG Electrocardiogram

EDT Event Dissemination Tree

FSA Finite State Automata

FSM Finite State Machine

FSMD Finite State Machine with Datapath

GHT Geographic Hash Table

GHTD GHT Dissemination

GPS Global Positioning System

GPSR Greedy Perimeter Stateless Routing

GS Geographical Scope

IEF Interval-based Event Filter

InfoS Information Space

JiST Java in Simulation Time

KP Knowledge Point

MAC Media Access Control

MANET Mobile Ad Hoc Network

NFA Non-deterministic Finite Automata

OSI Open System Interconnection

OSM Object State Model

P2P Peer-to-peer

Pub/Sub Publish/Subscribe

QE Query Expression

QPS Quad-PubSub

QT Quad-Tree

QoS Quality of Service

RFID Radio Frequency Identification

RSN Resource-constrained Sensor Network

RTS Request to Send

SF State Filter

SK Satisfying Knowledge

SMC State Maintenance Component

SPS State-based Publish/Subscribe

SQL Structured Query Language

SQTL Sensor Query and Tasking Language

SWANS Scalable Wireless Ad hoc Network Simulator

TDOA Time Difference of Arrival

WSN Wireless Sensor Network

Chapter 1

Introduction

This dissertation concerns support for high-level applications above sensor networks. I believe that large-scale sensor networks (with many users and diverse applications) demand suitable middleware solutions, and (as part of this thesis) investigate the development of an expressive Publish/Subscribe (Pub/Sub)-based middleware framework. This thesis concludes that a combination of Pub/Sub for scalability, abstraction, and openness, with state-centric data processing for expressiveness, can offer a suitable middleware framework for a large class of sensor network applications that are described as *smart environments*.

1.1 Sensor Systems

Sensor networks are composed of devices that are capable of measuring physical phenomena in a target environment. Recent technological improvements have enabled the production of advanced devices that are equipped with sensing, processing, and communication capabilities. In their most popular form, they are composed of low-power sensing components, a micro-controller, some limited amount of memory, a low-power radio, and a finite power supply. Although a single device has limited capability, when networked together, they can provide dense and accurate sensing about their environment.

Sensor networks, combining measurements with computation and communication, emerge as a promising technology that can be applied in a wide variety of application domains, for instance in the domain of control, actuation and maintenance of complex systems, fine-grained monitoring of indoor and outdoor environments, logistics, health care, and transportation. They are a reusable asset: they can be deployed for substantial periods of time, during which they can be used for various applications. Multiple users can share the infrastructure and run multiple applications concurrently - some of these applications may not even be known beforehand. In this dissertation, the term “sensor system” refers to the collection of sensor network infrastructure, the employed protocols and services, users and their applications. The next section describes the class of sensor network applications that are the focus of this dissertation.

1.1.1 Application Areas

Motivated by the increased availability of sensors and the accelerating trend toward ubiquitous environments, I envision many potentially complex applications that manipulate information derived from a large collection of heterogeneous sensors spread across a large geographical area. These applications can be classified according to their prior knowledge about the environment, as this knowledge often plays a key role in the type of interest in and manipulation of the sensor data by applications. I introduce three classes of pre-existing knowledge, and in each case detail the most likely form of data manipulation:

Zero Knowledge These applications often aim to understand an unknown or foreign territory (e.g. sensors deployed over Mars, sensors deployed to observe animal behavior). Their corresponding systems are strongly application-focused and often emerge as a result of application-driven deployment. Applications in this class are likely to receive all data that becomes available and store this for future analysis and/or processing. Frameworks that aim to assist these applications often explore data compression and efficient extraction methods that reduce the cost of data retrieval and prolong the duration of data collection.

Partial Knowledge This term, perhaps, defines the most common class of sensor network applications. Such applications have sufficient knowledge about the environment to perform data-driven operations with the received sensor data (e.g. automatic regulation of temperature, responses to forest fires, crop management, and traffic control in urban environments). Applications in this class often have specific interest in the data that can aid their operation. This often takes the form of detecting certain conditions, contexts, or situations from the sensor data. Frameworks that wish to assist these applications often abstract the sensor network complexities and provide high-level interfaces for easy and compact specification of interesting data and/or the programming of sensor devices.

Full Knowledge Where full knowledge about the environment (often a man-made structure or a machine like an airplane, an automobile, or a nuclear power plant) exists, applications often relate to safety, crisis monitoring, fault detection, or rescue. Sensed data are examined against known patterns and behavioral policies for deviations and anomalies that may indicate failures and/or unknown or uncommon phenomena in the environment. Frameworks that support these applications are often application-specific as they should allow for full specification of desired data behavior and should commonly process (or analyse) data with real-time guarantees.

This dissertation focuses on the second class of applications described above. Sensor networks underlying these applications are less application-specific and can often serve many users and applications with diverse interests. These applications often relate to *smart spaces* or *smart environments*. For example, a smart transportation system, comprised of many sensor and actuator devices (such as inductive loops, speed cameras, Automatic Number Plate Recognitions (ANPRs), Global Positioning System (GPS) devices, and traffic light signals) can serve many (potentially independent) users ranging from local traffic officers to individuals in possession of vehicles in the system. The type of interest in data may also be diverse, ranging from congestion and accident reports to information on bus arrival times or nearby taxi ranks. Supporting these applications over a large-scale sensor network is challenging.

1.2 High-level Application Support

In order to support the highlighted applications, a suitable middleware framework is needed that frees the application developer from the underlying (device-related or network-related) complexities and offers data (or information) through a high-level interface, providing simple processing and data manipulation capabilities. Such a framework needs to support many features, but this work confines itself to four key features that closely relate to data processing:

Abstraction Infrastructural details, consisting of sensing/actuation devices, network properties, and topological configurations, are heterogeneous and can change over time. These details describe the very low-level dynamics of the network and are often of little interest to the users. Users can instead benefit from higher level, data-centric abstractions that closely match their interests. Abstraction, in sensor systems, shifts the application/sensor network interaction from a fragile address-based communication to more robust data-centric communication.

Scalability Managing any large-scale network with many independent or collaborative devices and users is difficult. Managing a sensor network is more difficult as it contains a large volume of data that is produced by many networked devices (notably the sensors). A framework should support these devices and users without sacrificing efficiency or reliability. In most sensor networks using wireless (radio-based) communications, scalability implies support for efficient communications and managing client dynamics with low overhead.

Openness Openness in sensor systems allows users, devices, and applications to dynamically join or leave the system without centralized coordination or central management. It also supports the equal treatment of users and devices so that they can select one or more roles flexibly to suit their applications. For example, a user may operate as a consumer and receive sensor messages (data) in one application and simultaneously operate as a producer and send messages (commands) to actuators in another. Inter-device and inter-user interactions are also supported in an open system.

Expressiveness Data plays a key role in sensor systems. Thus, the expressiveness of a framework (in terms of the expressible interest in data) can strongly influence its usability. A framework is considered expressive if it can support a variety of interests that are useful to sensor network applications in a precise fashion so that irrelevant data are not delivered. In this dissertation, concrete application examples have been given at the start of each chapter in order to motivate and direct my design decisions.

Although many of the highlighted features have been investigated in past research, several sensor network characteristics prevent the re-use or easy migration to sensor systems of solutions that are developed in other contexts. I briefly touch upon these characteristics and their resulting challenges in the next section, and then proceed to describe a communication paradigm that is well suited to large-scale sensor networks.

1.2.1 Sensor Network Challenges

Sensor networks exhibit different characteristics from traditional networks. These characteristics divide into three classes:

Data. Data are often produced by the embedded sensing components. These components collectively observe an external shared entity called the "environment". As a result of this collaborative observation, sensor data are often *correlated* or even *redundant* in the information that it conveys to the user. This correlation and redundancy is useful when devices happen to fail or malfunction, but during normal operation it could be eliminated to reduce messaging. Sensor data are also *primitive*, meaning that it observes a simple phenomenon such as the temperature, light, sound, or humidity in the environment; it is primitive when the employed sensing components are small and low-powered. Data are thus typically too low-level to be meaningful to applications and often requires further internal processing in the framework. Switching our attention from the data to the observing entity, the environment also has temporal and spatial characteristics that need to be considered. The environment is *continuous* whereas sensor observations are discrete, i.e. are taken at discrete time points. This difference affects the capture of lasting and continuous conditions over the sensor data and leads to some correlation and redundancy (of observations) even at the level of a node. Environmental data has unique *type*, *time*, and *space* attributes which, if captured, can augment the meaning of the data. However, this challenges the internal data processing mechanism in the framework.

Scale. The scale of sensor networks relates to the *number of devices and users* that are in the network and their *dynamic behavior*. As the number of devices and users increases, the use of centralized solutions and algorithms becomes increasingly difficult and expensive. The scale of sensor networks calls for distributed solutions that can perform efficiently in the presence of large and changing numbers of clients (devices and users). The dynamic behavior includes two types of change in the set of clients: the intentional change (e.g. devices are added, removed, or replaced), and the unintentional change (e.g. nodes fail, are lost, or run out of power).

Resources. Resources that are available to the networked devices play a key part in determining the complexity of the code that can be executed over these devices. Without imposing numerical restrictions, a system developer must acknowledge that these resources are often *limited* and *heterogeneous*. The heterogeneity of resources not only corresponds to their types (e.g. power, processing, communication, and memory) but also to their amounts and to their costs of usage. For example, the communication resource in wireless (radio-based) sensor networks is considered far more expensive than the computational (processing) resource. On the other hand, user-specified policies can define *Quality of Service (QoS) restrictions* over the local and global expenditure of these resources. For example, one user may be interested in the reliability and accuracy of information and thus demand more frequent observations and data processing, while another could be interested in prolonged operation of the network and longer network lifetime. Since these QoS restrictions may be conflicting, I limit my focus to systems whose users have reached a common agreement over the cost and value of resources.

In the next section, I describe a communication paradigm whose implementation can offer many desired features and can address some of the discussed challenges.

1.2.2 Publish/Subscribe Paradigm

Publish/Subscribe is an asynchronous communication paradigm that supports many-to-many interactions between a set of Event Clients (ECs). An EC can be an information producer (publisher), an information consumer (subscriber), or both. EC interactions are data-centric: publishers describe their publishable events, subscribers express their interest in events, and the Pub/Sub protocol (also called the event service) delivers the published events to their corresponding event subscribers. This loose coupling of ECs aids scalability in dynamic environments where ECs and their roles can change frequently.

In this thesis I argue that Pub/Sub is a suitable communication paradigm for sensor networks; this paradigm is used to design a framework that exhibits suitable abstraction, openness and scalability for my target sensor systems. Because this communication paradigm primarily focuses on messaging, a set of complementary components have also been introduced which seamlessly interact with the Pub/Sub system and process data based on application-level requirements. These contributions are more comprehensively discussed in the next section.

1.2.3 Assumptions

The work presented in this dissertation assumes the following.

1. Sensor data are introduced to and processed within the middleware as a set of attributed tuples which have basic data type (e.g. numerical) values for processing. Thus, the internal manipulation of Abstract Data Types (ADTs) (e.g. image data) is not supported.

2. Sensor data are inclusive of environmental noise, whose distribution (model) is known to the applications and transparent to the middleware. Treatment of sensor noise depends strongly on the application semantics, which if modeled in a generic and accurate fashion could lead to significant operational and performance complexities at the middleware. Sensor noise can be treated as follows.

    - Application-defined components, like Virtual Devices [JAF+05] in HiFi [FJK+05] and data aggregation services in MIRES [SGV+04], can manipulate this noise in an accurate and efficient manner, independently. I support these components and label them as *services* in Figure 1.1. Nevertheless, since these components are external to the middleware, they can not benefit from the supported features and optimizations.

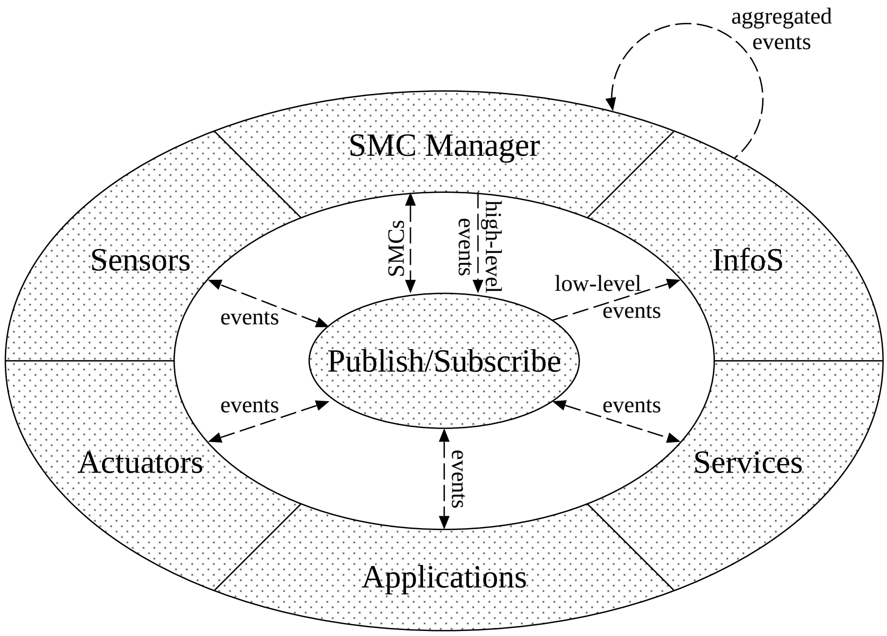


Figure 1.1: **SPS** components

- Expressive middleware interfaces can be used to perform simple sensor noise manipulations, internally. In the lightest component of my middleware, I confined this to *attribute-based computations*, where published data can have fields (meta-data), reflecting the accuracy and certainty of measurements, and can be manipulated as part of a Boolean expression. This manipulation induces little memory and computational overheads for resource-constrained devices. In the more expressive component of my middleware, I support *data aggregation functions* $\{max, min, sum, avg\}$. Although still limited in expressiveness, these functions are widely used (e.g. in DBMSs for WSNs, Section 2.4.1.1) for two reasons: (a) they are believed to suit a wide range of sensor network applications, (b) they can be implemented and processed efficiently.

3. Primitive data processing is possible within sensor networks. On resource-constrained sensor platforms, I limit this to data filtering, and on other sensor platforms, I bound the computational overhead by assuming basic data types (Assumption 1).

4. Where applications have high-level interests, low-level sensor data can be permanently discarded in favor of the high-level data. This assumption is necessary for the reduction of communication costs by in-network processing.

5. The network and the clients are co-operative and trustworthy. This work does not address any malicious behaviors and/or security concerns that may arise in real-world deployments.

1.3 Thesis Contribution

In this dissertation, I develop a **State-based Publish/Subscribe (SPS)** framework that is a generic middleware for high-level sensor network applications. **SPS** was designed to support the features discussed in Section 1.2, given the challenges discussed in Section 1.2.1. It consists of numerous cooperative components, where each distinct type of component provides unique functionality such as data processing, data dissemination, or data storage. Figure 1.1 shows a typical set of components that may reside on a node with **SPS** functionality. A Pub/Sub component centers the implementation and provides network-wide messaging. It also serves all the client components, which include sensors, actuators, applications, and services. These components may produce events as publishers, consume events as subscribers, or both. The Information Space (InfoS) and SMC Manager components are also clients of the Pub/Sub component, but implemented as part of the **SPS** framework. Low-level data are delivered to the InfoS component, processed and aggregated for the SMC Manager component, and finally examined for high-level interests (conditions) by the SMCs. If detected, high-level events are generated and published by the SMC Manager component for related subscribers. Components of the same type (homogeneous components) provide their functionality in a decentralized manner (e.g. by replication, by localized interactions, or by decomposition and distribution).

In developing the components of the **SPS** framework, I also had modularity and re-usability in mind. Sensor networks, as we shall see in Section 2.1.1, adhere to a wide design space, but also have some strong commonalities. Thus reusable components that can be used standalone or in conjunction with other protocols are valuable. I feel my contributions can be best described in terms of each component. Table 1.1 summarizes my contributions with respect to the required features and sensor network challenges.

The first two components in the table, SFs and SMCs, are data processing components, which process and evaluate data according to user-specified expressions, thereby providing expressiveness. InfoS components are closely related to SMCs: they allow data to be pre-processed and re-used for improved expressiveness in the **SPS** framework. Finally, a messaging component, QPS, implements the Pub/Sub communication paradigm that was motivated earlier in Section 1.2.2. These components are further described below.

State Filter (SF) SFs are lightweight event filtering components that are designed for RSNs. They extend the expressiveness of content-based Pub/Sub protocols by means of an enhanced subscription language. In SFs, the notion of *state* is used to capture lasting conditions over a set of discrete events. They also enhance the scalability of Pub/Sub protocols by filtering events that contain correlated or redundant information about the condition being observed. This filtering is achieved through *context-based* event processing, in which events are examined according to the current context of the condition being observed. SFs subsume the content-based filters of many content-based Pub/Sub protocols.

State Maintenance Component (SMC) SMCs are an advanced form of SF that are designed for more resourceful and heterogeneous sensor systems. The expressiveness of SFs

Table 1.1: Thesis contributions (in terms of components)

(a) Features list

Identification	Feature
f1	abstraction
f2	scalability
f3	openness
f4	expressiveness

(b) Challenges list

Identification	Challenge
c1	data are often correlated or redundant
c2	data are often primitive
c3	data are a discrete observation from a continuous environment
c4	data has type, time, and space attributes
c5	there exists a large number of clients
c6	clients are dynamic
c7	devices often have limited resources
c8	resources are heterogeneous
c9	user defines QoS restrictions over resources

(c) Designed components

Com.	Features				Challenges									Design Space
					Data				Scale		Resource			
	f1	f2	f3	f4	c1	c2	c3	c4	c5	c6	c7	c8	c9	
SF				√	√		√				√			constrained devices
SMC			√	√	√	√	√	√			√	√		part of **SPS**
InfoS	√				√	√	√	√						part of **SPS**
QPS	√	√	√					√	√	√		√	√	location-aware WSNs

is extended to allow the *fusion* of heterogeneous types of information into new (higher level and richer) types of information. They acknowledge the importance of the type, time, and space attributes of data and provide *temporal and spatial features* that allow fine-grained specification of high-level conditions. For additional state storage, these components can support greater expressiveness by allowing *memory-based* condition detection. Scalability concerns have also been considered in the design of SMCs: they operate *independently*, can be placed *flexibly* in the network, and are *decomposable* for distributed processing. SMCs are the data processing elements of **SPS**, which contribute towards its expressiveness and aid its scalability.

Information Space (InfoS) InfoS is a knowledge container for the **SPS** framework. It maintains data as in a relational Database table and performs basic operations that prepare data for processing by the SMCs. Notably, InfoS complements the SMC event processing capabilities with *contextual (time and space) awareness* and *aggregation*; it compacts a series of homogeneous data into a single data item of the same type that is more meaningful to the SMC. It also updates the stored data with knowledge that is received from a series of discrete events. As a result, InfoS can offer richer data (consisting of historic, continuous, contextual and/or aggregated data) to the SMCs for processing. As I shall describe later, the InfoS component and the SMC manager component, which hosts SMCs, are closely related in the **SPS** framework.

Quad-PubSub (QPS) QPS is a *distributed* Pub/Sub protocol for location-aware WSNs. It supports its clients through a *unified Pub/Sub interface*, and supports type, time, and space data attributes by implementing *topic- and location-based* event filtering capabilities. QPS addresses the dynamic behavior of ECs, and provides complete *time and location decoupling*. More important, however, is the *layered architecture* of QPS (which **SPS** also benefits from) that results in a modular and flexible system. In QPS, the layered architecture allows for the transparent operation of any location-based routing protocol that satisfies the user-defined QoS requirements. A common requirement, for example, is to prolong the lifetime of the network. As my evaluation confirms, this design decision does not result in performance losses and, in fact, reduces the number of nodes that must cooperate to provide Pub/Sub functionality. A dedicated layer in QPS ensures that the selected nodes have sufficient resources to perform their tasks whilst also providing *resource-awareness*. This functional layer actively relieves nodes from their tasks when their resources become depleted and finds suitable alternatives when nodes happen to fail. QPS can be used as a standalone Pub/Sub protocol in any location-aware WSN that has an underlying location-based routing protocol.

The **SPS** framework is a composition of the SMC, QPS, and InfoS components. Together, they provide all the desired features and address all the described challenges for supporting high-level sensor network applications.

1.4 Thesis Outline

The remainder of this dissertation is organised as follows:

Chapter 2 provides a survey of the background and related work.

Chapter 3 presents the State Filter (SF) component.

Chapter 4 describes the Quad-PubSub (QPS) Pub/Sub protocol.

Chapter 5 presents the State-based Publish/Subscribe (SPS) framework with emphasis on the State Maintenance Components (SMCs) and the Information Space (InfoS) components.

Chapter 6 gives a brief conclusion, summarising the work described in this thesis and outlining future work.

Chapter 2

Background & Related Work

In this chapter I provide an overview of previous research efforts to support high-level applications for sensor networks. I begin by describing sensor networks and their design space. I then explore three areas of work related to this dissertation. The first area, *Programming models*, empowers application developers with a series of tools that abstract the underlying network complexities and enable complete application development. The second and third areas provide higher level abstractions with a direct focus on sensor data. *Data delivery* is the second area and addresses challenges that arise when data needs to be relocated from sensors to application clients. *Data processing*, the third area, enables data processing according to application requirements.

2.1 Sensor Networks

Sensor networks are composed of devices that are capable of measuring physical phenomena in a target environment. Recent technological improvements have enabled the production of devices that are equipped with sensing, processing and communication capabilities. In their most popular form, they are composed of low-power sensing components, a micro-controller, some limited amount of memory, a low-power radio, and a finite power supply (battery).

Although a single sensor has limited capability, when deployed in large numbers, they provide dense sensing and have the ability to observe a given environment in great detail. They are intelligent compared with traditional sensors because they can process and communicate sensed information, and coordinate actions within the network. Technological advances have also improved the range of phenomena that can be captured by the small and low-power sensing components. Examples of these phenomena include vision, audio, ultra-sound, infra-red, temperature, humidity, noise, pressure, and vibration; [BKZD04] surveys a number of commonly used sensors and their application areas.

Sensor networks, combining measurements with computation and communication, are a promising emerging technology that can be applied in a wide variety of application domains, for instance in the domain of control, actuation and maintenance of complex systems, fine-grained monitoring of indoor and outdoor environments, logistics, health care and many more that are briefly touched upon in Section 2.1.2. They are a reusable asset; they can be deployed for substantial periods of time, during which they can be used for various applications. Multiple users could share the infrastructure and run multiple applications concurrently - some of these applications may not even be known beforehand.

Wireless Sensor Networks	vs	Wireless Ad Hoc Networks
hundreds to thousands of devices		tens to hundreds of devices
high network density		low network density
low communication bandwidth		high communication bandwidth
low (device) duty cycle		high (device) duty cycle
small and cheap devices		expensive and large devices
high device failure rate		low device failure rate
high data redundancy		low data redundancy
data centric interaction		address centric interaction
non-rechargeable and non-replaceable devices		rechargeable or replaceable devices

Table 2.1: WSNs vs Wireless Ad Hoc Networks

Wireless Sensor Networks (WSNs), which use radio-based communication, are one of the most common kinds of sensor networks. They can be deployed flexibly and maintained easily. The cost of installing, terminating, testing, maintaining, trouble-shooting, and upgrading a wired network has made wireless systems increasingly attractive for general scenarios. Research on WSNs is closely related to that on wireless ad hoc networks as almost all WSNs use ad hoc infrastructures and depend on organizing techniques (for establishing communication routes) that have been previously studied in the context of wireless ad hoc networks. However, there are some differences that must be considered when supporting applications on WSNs [IMK04]. Table 2.1 shows the main characteristics that often differentiate them.

There has been significant research attention on WSNs, much of which is directed towards power consumption issues because of nodes' power constraints. Other issues that have received relatively little attention in the past and are further explored in this dissertation are the external observation of sensor devices and temporal and spatial significance of sensed data. Sensor networks are different from traditional information-based networks, because sensors observe a common (shared) external entity, the physical environment. As a result, sensors may capture information that is correlated or redundant across time and/or space. In this dissertation, I develop a set of tools that aid the application developer in dealing with these correlation and redundancy issues. In the next section, I describe the sensor network design space.

2.1.1 Design Space

With the emergence of small sensor devices and WSNs, the quest for sensor network applications also began. Discussions at related conferences and workshops [RM04a] indicated that sensor networks have made their way into a wide range of applications with different requirements and characteristics. This not only complicates the classification of research on sensor networks, but also means that no single solution can benefit all sensor network applications. In an effort to scope and structure sensor network research, it was suggested [RM04a] that a *sensor network design space* be created that identifies the various dimensions of sensor networks. These dimensions not only characterize the properties of sensor networks, but also provide a coarse-grained classification of sensor network applications.

In the following sections I describe and discuss a number of design dimensions that are relevant to this dissertation. The aims of these discussions are two-fold; firstly to describe the range of sensor network systems that are of interest to me (and that can, therefore, benefit from this dissertation), and secondly to provide a global and comparative view of my contribution relative to the field of sensor networks.

Dimension	Classes
Deployment	random vs manual
	installed vs *ad hoc*
	one-time vs *iterative*
Network Size	small (10s) vs medium (100s) vs *large (1000s)*
	sparse vs covered vs *dense (redundant)*
Heterogeneity	homogeneous vs *heterogeneous*
	brick vs matchbox vs **grain**
Mobility	*immobile* vs partly vs **all**
	occasional vs **continuous**
	active vs passive
Communication	*radio* vs light vs inductive vs capacitive vs sound
Infrastructure	infrastructure vs *ad hoc*
Connectivity	connected vs intermittent vs **sporadic**
Role	sensor vs actuator vs relay vs user

Table 2.2: Sensor network design space

Table 2.2 lists the sensor network design dimensions and their classes. These dimensions closely relate to those considered by [RM04a], but have been slightly modified to reflect my personal view - [RM04a] has solely focused on sensor devices, yet I believe that sensor networks could embed other infrastructures such as actuators and relay nodes (explained shortly). For each dimension (or sub-dimension), the class that is most relevant to my work or contribution is highlighted in *italics*; sensor networks that occupy these design points can significantly benefit from this dissertation. On the contrary, items (classes) in **bold** have not been considered in this dissertation.

2.1.1.1 Deployment

The deployment of nodes may take several forms. Nodes may be scattered randomly or placed at chosen spots (manual). They may be individually set up (installed) or left in a global startup state (ad hoc) after their placement. Finally, deployment may be a one-time activity or a continuous process (iterative) where nodes are gradually added over time.

2.1.1.2 Network Size

Network size is often determined by the required coverage and connectivity. Coverage describes the geographical area of interest, as well as the degree of monitoring that is desired in the environment. If failures are frequent and sensor readings are imprecise, then a dense (redundant) coverage is preferred, otherwise a partial coverage (sparse) or full coverage (covered), depending on the desired Quality of Service (QoS), may suffice.

2.1.1.3 Heterogeneity (of platforms)

Networked nodes can have identical hardware (homogeneous) or different hardware (heterogeneous). Homogeneous networks are motivated by the low cost of manufacturing large quantities of identical hardware, but most prototype and deployed systems to date have consisted of a variety of hardware devices. A multi-tier (heterogeneous) network is favored due to its scalability and low per node scalability costs.

In relation to physical size of the devices, they may be as large as bricks or as small as grains; the size usually depends on the application environment where sensors are to be deployed. The size also affects the level of resources that could be available to a node. Thus, heterogeneous networks are often seen to have nodes of different sizes. To date, many hardware platforms have been developed that differ in hardware resources, size, reliability, and robustness. Some of these are *BEAN, Particles, BTnode, Rene, COTS, ScatterWeb, Dot, Sensinode, Ember, SHIMMER, Eyes, SquidBee, FireFly, SunSPOT, Fleck, Telos, IMote, TinyNode 584, Imote2, T-Mote Sky, KMote, T-Nodes, Mica, WeBee, Mica2, weC, Mica2Dot, XYZ, MicaZ, WINS, Mulle, WiseNet,* and *Nymph* - please refer to surveys [BBB+06; SNMa; MHS] for more details on these platforms.

2.1.1.4 Mobility

Sensors may change their location after initial deployment. Mobility can be active (i.e. automotive) or passive (i.e. as result of environmental influences like wind or water). It may apply to all, a subset, or none of the nodes in the network. The degree of mobility may also vary from occasional to constant (continuous) in time. Mobile Ad Hoc Networks (MANETs) are typically described by the active, all, and continuous mobility classes in this dimension. Sensor networks, however, are often passive and immobile.

2.1.1.5 Communication modality

The most common communication modality is radio, mainly because it does not require a free line of sight. Other communication modalities are light, sound, inductive, and capacitive. These may apply some restrictions, but have different characteristics that may suit alternative environments such as under water or underground. It is worth noting that most passive Radio Frequency Identification (RFID) systems use inductive coupling.

2.1.1.6 Infrastructure

Two common forms of constructing an actual communication network are infrastructure-based and ad hoc. In infrastructure-based networks, nodes can only directly communicate with so-called base station devices. Deployment and installation of these base stations, however, is expensive and often not feasible in target environments. Therefore, the alternative (ad hoc network) is preferred, where nodes can directly communicate with each other without an infrastructure. A combination of the two is also used sometimes, where clusters of nodes are interconnected by a wide area infrastructure-based network.

2.1.1.7 Connectivity

The connectivity of a network is defined by the physical location of sensor nodes and their communication ranges. If there is always a network connection (perhaps through multiple hops) between any two nodes, the network is said to be connected. Intermittent connectivity is where occasional network partitioning may exist, and sporadic is where nodes are isolated most of the time.

2.1.1.8 Device Roles

The majority of nodes in a sensor network are sensor devices (i.e. are equipped with sensing components). However, not all must be sensor devices. Some nodes may be attached to user applications, and reflect the user in the network. Others may be deployed to perform actuation (actuators) or just to ensure network connectivity (and prolong the network lifetime) by relaying packets (relay nodes). These roles (except relay) have their own sub-classifications which are shown in Table 2.3 and discussed below.

Sensors. These devices are identified by having a piece of hardware that monitors or observes the immediate environment. The phenomenon that is observed by these devices may be unique (homogeneous) or different (heterogeneous). The sensor may also be *scalar* or *discrete*, depending on the phenomenon that it observes. Scalar sensors observe a context that is continuously available for sampling such as temperature, humidity, light, and sound. The challenge here is to sample the environment so frequently that no important event is missed, but that also not much energy is dissipated over time. Discrete sensors, on the other hand, observe their phenomenon at discrete time points that are signalled by an external event such as user-entered information, a door-opened event, or a tag-read event.

Table 2.3: Classification of device roles

(a) Sensors

homogeneous
heterogeneous
scalar
discrete

(b) Actuators

internal
external

(c) Users

internal
external
one
many
collaborative
independent
learning
monitoring
checking

Actuators. Actuators can turn a passive sensor network into an active one; the network can react to changes that it senses from the environment. Actuation may manipulate the external environment or the internal sensor system.

Users. Users are often assumed to connect to the sensor network externally (via base stations), but in ad hoc networks it is more convenient to connect internally (i.e. via a chosen node in the network). There may be only one user, in which case the sensor network becomes strongly application-focused, or many users, in which case the sensor network is shared. If there are many users, they can operate collaboratively or independently; opportunities for resource sharing (sharing computations, data, and communications) may be lost if users operate independently. Finally, users either learn about, monitor, or check the environment, depending on their prior knowledge of the environment. Where the environment is foreign and unknown to the user, the user is mostly interested in observing and learning about the environment. Where some (partial) prior knowledge is available, the user may be keen to detect and monitor interesting events and/or situations. Where the user has full knowledge about the environment (as in man-made structures), then the user may be interested in checking and ensuring that the environment behaves (operates) as desired or designed.

2.1.2 Applications

As mentioned, sensor networks have found their ways into a wide range of applications. These applications occupy different points in the sensor network design space, and [RM04a] has illustrated this for a set of applications with dimensions that closely match those outlined in the previous section. In this section, I do not provide a classification and refer the reader to [RM04a]; instead, I list a number of areas and applications that have been considered for sensor networks in the literature. Table 2.4 provides a list of these applications, some of which may extend beyond the scope of this dissertation. Nevertheless, the technical chapters in this dissertation start with concrete and related application examples.

Area	Applications
industrial	monitoring/control of industrial equipment (LR-WPAN [GNC+01]). factory process control and automation [SSJ01]. manufacturing monitoring [SP04]. monitoring underground structures [LL07]. smart energy [RAF+01].
military	military and civilian surveillance [EGHK99]. military situation awareness [SSJ01]. sensing intruders on bases, detection of enemy units movements on land/sea, chemical/biological threats and offering logistics in urban warfare [DA02]. battlefield surveillance [SP04]. command, control, communications, computing, intelligence, surveillance, reconnaissance, and targeting systems [ASSC02]. target tracking [ZSR02].
location	location awareness (Bluetooth [GNC+01]). Person locator [SP04].
public safety	sensing and location determination at disaster sites [GNC+01; CGH+02].
automotive	tire pressure monitoring [GNC+01; CGH+02]. active mobility [RM04a]. co-ordinated vehicle tracking [SSJ01].
airports	smart badges and tags [GNC+01; CGH+02]. wireless luggage tags [GNC+01]. passive mobility (e.g., attached to a moving object not under the control of the sensor node) [RM04a].
agriculture	sensing of soil moisture, pesticide, herbicide, pH levels [GNC+01; CGH+02; BBB04].
emergency situations	hazardous chemical levels and fires (petroleum sector) [GNC+01]. fire/water detectors [DA02]. monitoring disaster areas [ASSC02].
rotating machinery	monitoring and maintenance (electric sector) [GNC+01].
seismic	warning systems [DA02].
commercial	managing inventory, monitoring product quality [SP04; ASSC02].
medical/ health	monitoring peoples locations and health conditions [SP04]. sensors for: blood flow, respiratory rate, Electrocardiogram (ECG), pulse oxymeter, blood pressure, and oxygen measurement [HMCP04]. monitor patients and assist disabled patients [ASSC02].
ocean	monitoring fish [SP04].
others	monitoring in-building energy usage [BBC]. fine-grain monitoring of natural habitats [CEH+01]. instrumented learning environments for children [SMP01]. measuring variations in local salinity levels in riparian environments [SBM+00].

Table 2.4: Some sensor network applications (partially from [GHIGGHPD07])

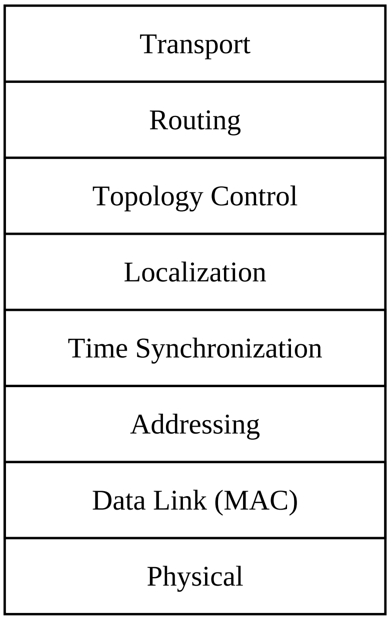


Figure 2.1: Communications protocol stack

2.1.3 Communication Protocols

Like traditional computer networks, sensor networks can also be analyzed in terms of Open System Interconnection (OSI) layers. The communications protocol stack, for sensor networks, can be described using eight layers [KW03; ASSC02]. These layers are *physical layer*, *data link (Media Access Control (MAC)) layer*, *addressing layer*, *time synchronization layer*, *localization layer*, *topology control layer*, *routing layer*, and *transport layer* (shown in Figure 2.1). These layers closely correspond to the *physical*, *data link*, *network*, and *transport* ISO OSI layers. Each layer may have zero or more protocols, depending on the sensor network design and application requirements, to support its functionality. In this dissertation, the communication layers below the routing layer are often referred to as the *network layer* for brevity. For a description of these layers and a survey of protocols that have been developed to support their functionality see [KW03; ASSC02]. Where directly related, a concise definition of the layers and discussion of their related protocols is presented within the technical chapters. In the next few sections, I discuss the related work on supporting high-level applications in sensor networks. This work either implements (parts of) the highlighted protocol stack and/or relies on it to support higher level application semantics.

2.2 Programming Models

Programming models are the foundations of sensor network middleware. They provide high-level programming interfaces to the application programmer and can be classified [SG08] into *node-level programming*, *group-level programming*, and *network-level programming*. Node-level programming takes a platform-centric view of the sensor network, and focuses on abstracting hardware and allowing flexible control of the nodes. The latter two models take an application-centric view and address how easily application logics can be programmed over a group or the entire network of nodes in the system. These classes are further described below.

2.2.1 Node-level Programming

One of the earliest node-level programming models, which has also become the *de facto* standard software platform (together with NesC [GLvB+03] programming language), is TinyOS [HSW+00a]. TinyOS is a component-based Operating System with modular programming that focuses on resource constrained devices; it offers a limited set of services, disallows dynamic allocation, and provides a simple concurrency model. Application programming at this low level is often very difficult. One way to tackle this complexity is to extend the event model.

Object State Model (OSM) [KR05] is a programming model that allows developers to specify their applications as Finite State Machines (FSMs). It extends the event paradigm with state and transitions, making actions a function of both the event and the program state. The authors claim that relaxing the tight coupling between events and actions in this way can ease application programming and support a more efficient style of programming. Once specified in terms of FSMs, the application can be transformed into native code through the OSM compiler.

OSM borrows the concepts of hierarchical and parallel composition of state machines from Statecharts [Har87], and adopts the concept of concurrent events from SyncCharts [And96]. It also introduces *state attributes* that allow information sharing among actions. Finite State Machine with Datapath (FSMD) [GR94] had earlier introduced similar attributes to FSMs. This reduced the number of states that had to be declared explicitly, but attributes had global scope and lifetime. In OSM state attributes are local and bound to a state hierarchy.

Another interesting node-level programming approach is to run a virtual machine on each node. Examples of this approach are frameworks like Maté [LC02], ASVM [LGC05], Melete [YRBL06], VMStar [KP05], Impala [LM03], and SensorWare [BHS03]. The main advantage of this approach is that programs can be expressed in smaller virtual machine byte code than native code; thus, program updates after deployment (re-programmability) can be performed more efficiently. The trade-off cost here is the interpretation of the virtual machine byte code that leads to some execution overhead. Please visit [SG08] for a survey of virtual machine frameworks for sensor networks.

2.2.2 Group-level Programming

Group-level programming aims to handle a collection of nodes and provide a set of language constructs that can be used to specify a desired group behavior. These models can be used to facilitate collaboration among a group of nodes. The groups can be formed according to topological connectivity or logical attributes; the former is called a *neighborhood-based group* and the latter is called a *logical group*.

Neighborhood-based Groups. The underlying motivation for this model is that sensor network algorithms often process data within a localized neighborhood [SG08]. This neighborhood can be specified according to the topology (n-hop neighbors) or the geographical distance (k-nearest neighbors), and is often assumed to remain static. Abstract regions [WM04] and Hood [WSBC04] are examples of this programming model, with the distinction that the latter is restricted to 1-hop neighbors as the sole group definition. This work supports a number of primitive operations, such as neighborhood discovery, variable sharing via a Linda-like tuple space and MPI-like data reduction. Although operations over groups can be efficiently implemented in these models, the translation of application logic into a network-dependent protocol may be difficult. In fact, in these models the network topology is not abstracted from the application developer but provided as the basis of collaboration between application nodes.

Logical Groups. This model is more expressive than the previous model and can even subsume the neighborhood abstraction by definition. Logical groups, however, are often defined at a higher level relating to sensor types, inputs and/or outputs. EnviroTrack [ABE+04] is a programming abstraction specifically designed for target-tracking applications. It differentiates itself from traditional localization systems in that it does not assume cooperation from the tracked entity (e.g. a user does not wear a beaconing device to aid localization and tracking). It assigns addresses to physical events in the environment, and defines groups that observe the same events. It provides similar services to the previously discussed programming abstractions, but as the group membership is more dynamic, it has a more sophisticated group management protocol. Another programming model proposes the SPIDEY [MP06] language, in which the static and dynamic attributes of nodes are exported and grouping predicates (conditions over attribute values) are specified to define the logical groups. Finally, PARC's PIECES framework presents a state-centric programming abstraction [LCRZ03] that eases the programming of collaborative signal and information processing applications. They employ the notion of "collaboration group" that is defined by a "scope" (which restricts the group membership) and a "structure" (which defines the member roles). They do not provide a rich set of communication operations, but provide sufficient expressiveness to allow for the implementation of high-level abstractions.

2.2.3 Network-level Programming

Network-level programming sees the sensor network as a whole and considers it as a single abstract machine. Its goal is to perform programming from a macroscopic viewpoint so that every node and data item can be accessed without considering low-level communications among nodes [SG08]. Regiment [NW04; NMW07] and Kairos [GGG05] are two programming models that allow global behavioral specifications. Regiment is a functional language with a syntax similar to Haskell; it hides the direct manipulation of program states, thereby providing flexibility for the compiler. Regiment programs are initially compiled into TML [NAW05] and then to NesC. Unlike Regiment, Kairos is language-independent and can be implemented as an extension to existing programming languages. Kairos focuses on providing a small set of constructs: reading and writing variables at nodes, iterating through 1-hop neighbors, and addressing arbitrary nodes. Furthermore, to reduce communication overhead, Kairos employs a weak consistency model (called "eventual consistency") for shared (remote) data access. Additional details, or other work related to this class of programming models can be found in [SG08].

2.3 Data Delivery

Data is the most important element in any sensor network; sensors are, after all, deployed to collect environmental data. I call the process of transferring data from sensors (i.e. producers) to users (i.e. consumers) *data delivery*. Data delivery may be *active* or *passive*. In the former, data is almost immediately routed and delivered to the user, while in the latter data is stored (either at the sensor node, at an intermediate node, or at the user node) and delivered only upon a subsequent pull request from the user. Note that *data routing*, which is an element of data delivery, is only required if the data producer and the data consumer do not co-exist on a node[1].

The distinction between active and passive delivery is also related to the *timeliness* requirements for data delivery; applications that favor timeliness and operate in a data-driven manner prefer active delivery, while others may prefer passive delivery. Passive delivery allows for some performance optimizations such as data aggregation and batched data routing. In the following sections, I confine myself to surveying data delivery for WSNs, and survey the suite of routing protocols that have been developed to support each type of data delivery.

2.3.1 Active Delivery

Most sensor network applications require some level of timeliness, therefore developing routing protocols for active data delivery has been a popular research topic. In this dissertation, unless otherwise specified, data routing generally refers to data routing for active data delivery.

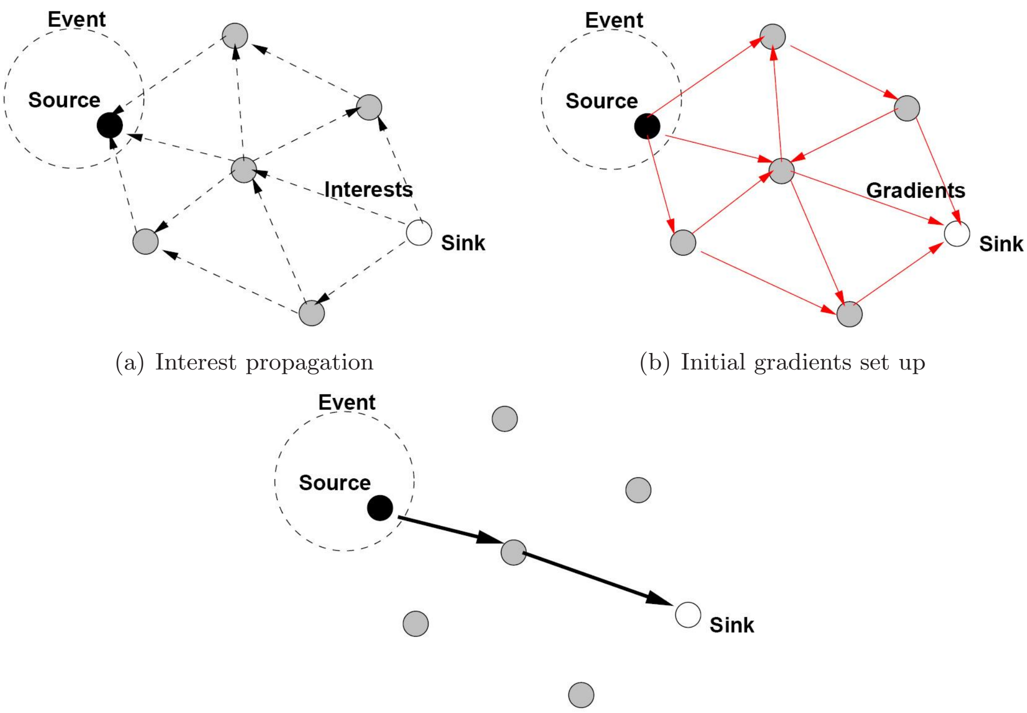
Data routing in WSNs is challenging for three reasons. Firstly, communication over wireless media is unreliable and failure-prone. Secondly, wireless communication is expensive - substantially more expensive than other resources such as processing [PK00]. Thirdly, routing is performed by devices which are themselves unreliable, primitive, and failure-prone. This has led researchers to investigate a whole range of wireless routing protocols that are optimized for different network and application settings.

Routing protocols can be broadly categorized into four classes: *data-centric*, *hierarchical*, *geographical* (*location-based*), and *QoS-based*. Apart from data routing, some protocols also support primitive data aggregation (e.g. filtering of duplicate data that is observed by different sensors). This form of aggregation (called *in-network aggregation*) has been shown [KEW02] to reduce the size of data that is communicated in the network, and therefore can reduce the overall communication cost. Table 2.5 shows the classification of some popular routing protocols, and support for data aggregation. I do not discuss these protocols here, but refer the reader to [AY05; AKK04] surveys for more information on them and their classifications. Instead, I

[1] They often don't!

Routing protocol	Data-centric	Hierarchical	Geographical	QoS	Aggregation
SPIN [HKB99]	√				√
Directed Diff. [IGE00]	√				√
Rumor routing [BE02]	√				√
GRAB [YZLZ05]	√				√
GBR [SS01]	√				√
MIRES [SGV+04]	√				√
COUGAR [BGS00]	√				√
CADR [CHZ02]	√				
ACQUIRE [SKA03]	√				
SAFE [KSS+03]	√		√		√
TTDD [YLC+02]		√	√		
GAF [XHE01]		√	√		
(AP)TEEN [MA01]	√	√			√
LEACH [HCB00]		√			√
PEGASIS [LR02]		√			√
(S)MECN [RM99]			√		√
GEAR [YGE01]			√		
SPEED [HSLA03]			√	√	
SAR [SGAP00]				√	
[TOB04]	√				
[CPR05]	√				
[HCRW04]	√				
[SR02]	√		√		
[LHZ04]			√		√
[YYA02]		√	√		
[SK00]		√			√
[AY03]		√		√	

Table 2.5: Classification of routing protocols for active delivery



(a) Interest propagation (b) Initial gradients set up

(c) Data delivery along reinforced path

Figure 2.2: Schematic diagram for Directed Diffusion

present a short description of one of the most popular routing protocols in WSNs, the Directed Diffusion protocol, and then present a communication paradigm that can serve future generations of sensor network applications.

Directed Diffusion [IGE00; IGE+03; HSE03]. Directed Diffusion is a data-centric routing protocol in which data generated by the nodes is named by *attribute-value* pairs. It is a destination-initiated reactive protocol in which routes are established when requested. An *interest* message is propagated throughout the network for named data (by a node) and data which matches this interest is then sent towards this node. The interest message and the data which is sent as a response to the interest contain a list of attribute-value pairs. The main Directed Diffusion protocol [IGE00; IGE+03] (also referred to as the *two-phase pull* protocol) operates as follows.

The *sink* (destination) requests data by broadcasting interests. An interest diffuses through the network hop by hop, and is broadcast by each node to its neighbors (see Figure 2.2(a)). As the interest propagates, *gradients* are set up to draw data towards the sink. Each node that receives the interest sets up gradients toward the nodes from which it receives the interest (see Figure 2.2(b)). When an interest arrives at a data producer, that *source* begins producing data. The first message sent from the source is marked as *exploratory* and is sent to all neighbors that have matching gradients. The exploratory data can reach the sink via one or more paths. The sink subsequently *reinforces* its preferred neighbor (this is defined by the application semantics) to establish a single reinforced path towards itself; the process is iterated (i.e. the reinforced neighbor reinforces its own preferred neighbor and so on) until the data source or sources are reached. Subsequent data messages are not marked exploratory, and are sent only on reinforced gradients (see Figure 2.2(c)). These gradients, however, are managed as soft-state; thus, both interests and exploratory data occur periodically to refresh this state. In addition, *negative reinforcements* are supported to eliminate erroneously reinforced gradients. Figure 2.2, taken from [IGE00], summarizes the protocol's basic operation.

In later work, [HSE03], the Directed Diffusion developers introduced two variants of the protocol, *one-phase pull* and *push diffusion*, that operate more efficiently than the two-phase pull protocol by eliminating one of the two broadcast stages. The one-phase pull protocol eliminates the propagation of exploratory data and implicitly reinforces gradients on the lowest latency paths. End-to-end flow identifications (*flow-id*) are used to ensure unique reinforced paths, and negative reinforcements are used to eliminate duplicates and/or path loops. The second variant, the push diffusion protocol, targets a rare application scenario in which sensors produce very little data and there are many sinks. It eliminates the interest message propagation, and uses exploratory data (by each source) to deliver initial data from sources to sinks. Sinks can then reinforce paths (as in the two-phase pull protocol) to receive non-exploratory messages.

2.3.1.1 Publish/Subscribe

Pub/Sub is a many-to-many asynchronous communication paradigm that loosely couples the data producers (*publishers*) and the data consumers (*subscribers*). It is data-centric in that the relationship between the publishers and the subscribers is solely defined by the data (*events*). Publishers actively introduce events (e) by means of invoking a *publish* operation, $publish(e)$; and subscribers independently describe their interests (s) through a *subscribe* operation, $subscribe(s)$.

The structure of events is described by an *event model* and often constitutes a set of attribute-value pairs. *Subscriptions* are filters that examine portions (or the entire contents) of events. They consist of a set of constraints that conform to a subscription language defined by a *subscription model*. An event e is said to *match* a subscription s if it satisfies all its declared constraints, $e \sqsubseteq s$. The expressiveness of the Pub/Sub system is determined by its subscription model. The most popular subscription models are *topic-based*, *content-based*, and *type-based*; readers are advised to consider [EFGK03] for more details about these subscription models. The remainder of this section deals with Pub/Sub in the context of WSNs, please consider [EFGK03; LP03; BV06] surveys for Pub/Sub systems in other contexts (e.g. wide-area networks and Internet-based environments).

WSN routing protocols as Pub/Sub. The importance of data in sensor networks has bridged the gap between the WSN routing protocols and the Pub/Sub communication paradigm. The data-centric routing protocols closely fit the description of a Pub/Sub protocol. The reinforced paths constructed by the Directed Diffusion protocol are equivalent to an Event Dissemination Tree (EDT) that directs events from publishers (sensors) to subscribers (sinks). A few differences, however, exist that are worthwhile discussing here.

Firstly, existing WSN routing protocols mainly focus on one-to-one or many-to-one communications where the consumer is often a fixed and known base station device. It is not clear if WSN routing protocols are suited (in terms of scalability) for a many-to-many communication environment, where there are many consumer nodes with dynamic behavior in the sensor network. Secondly, although in a Pub/Sub communication paradigm Event Clients (ECs) can flexibly adopt roles (e.g. base stations can become publishers and actuators can become subscribers to support a reversed flow of data), most WSN routing protocols to date have tightly coupled the role of the network nodes to that of a publisher and the gateway (or base station) to that of a subscriber. A Pub/Sub protocol can support actuation as noted earlier, and even provide inter-application messaging for collaboration and resource sharing.

Thirdly, the common Pub/Sub communication paradigm assumes publishers to be active (i.e. publishers invoke the publish operation independently of the subscriptions), whereas the WSN routing protocols prefer an initial pull style interaction where publishers are not required to publish any data until queried for - this allows for some energy saving as sensors may be turned off when there are no subscribers. Fourthly, data aggregation in Pub/Sub has not been supported to the same level as WSN routing protocols. MIRES [SGV+04], however, has shown how equivalent data aggregation can be achieved in Pub/Sub by introducing data aggregation services that subscribe to raw events and publish aggregated events. Finally, complete location and time decoupling in WSN routing protocols is not supported; this is further discussed below.

Decoupling in Pub/Sub. Pub/Sub supports many forms of decoupling between publishers and subscribers, two of which are *time* and *location* decoupling. Time decoupling means that interacting parties do not need to be active at the same time. More specifically, publishers do not need to be active when subscribers subscribe, and subscribers need not to be active when publishers publish to take part in the interaction. Many WSN routing protocols fail to update the EDT when a publisher joins the network *after* a subscription; thereby partially violating this time decoupling requirement. Soft-state subscriptions are a simplistic solution to this problem that may lead to event misses.

Location decoupling means that publishers and subscribers do not need to know about each other's location, and that the interaction between the two is independent of their location. This decoupling results in location-based filters over the data as opposed to the publisher's location. For example, the subscription "temperature events from sensors located in room A" imposes a constraint over the publisher's location (i.e. the publisher sensor needs to be located in room A) as opposed to the data values. I argue that a subscription of the form "temperature events with a location attribute value equal to room A" is preferable; in this case the constraint is imposed over the data and the publisher (sensor) may be located anywhere. A Pub/Sub protocol with complete location decoupling has numerous advantages over location-based WSN routing protocols that support location-based constraints over the publishers. For example, the following application settings can be supported by the Pub/Sub protocol but not the location-based routing protocols.

1. Sensors (e.g. camera) that observe a non-local environment remotely (i.e. *remote sensing*). The location attribute of the data does not match that of the publisher's.

2. Mobile sensors that publish data relating to a location visited in the past (i.e. *mobile sensing*). The location attribute of the data and the present location of the publisher may not match.

3. Aggregation of events at a node that is located outside the region of interest (i.e. *external aggregation*). In this case, the aggregated data is published by a node that is external to the area of interest.

Of course, Pub/Sub's main requirement is the decoupling of EC identities (i.e. publishers need not know about the subscribers, and vice-versa). In this dissertation, WSN routing protocols that are data-centric and satisfy this requirement are labelled as Pub/Sub protocols following the terminology of the research community. These protocols can operate and support a Pub/Sub-like interface. The term "proper Pub/Sub protocol", however, is used for a Pub/Sub protocol that provides a complete time and location decoupling. This dissertation presents QPS, a proper Pub/Sub protocol for WSNs, in Chapter 4.

2.3.2 Passive Delivery

Passive data delivery suits the class of applications that do not need to respond to sensed data immediately. These applications often benefit from a query-based communication paradigm. Sensed data, however, needs to be stored until queried for. In [REG+02], three storage mechanisms (source-side, consumer-side, and rendezvous-based) are compared and the rendezvous-based storage (also referred to as Data-Centric Storage (DCS)) has been shown to be more scalable than others. Following that analysis, significant research has focused on how to store data in a predetermined way so that queries can efficiently find their corresponding information. The initial work, Geographic Hash Table (GHT) [RKY+02], only supported named data (i.e. users could only query a name), but later efforts, DIMS [LKGH03], DIFS [GER+03], DIMENSIONS [GEH03], and [GGHZ04], support more expressive querying, allowing for range queries (over attribute values) or searching for features in sensor networks.

The highlighted work relies on location-awareness to build structured overlays (much like CAN-[RFH+01] or Chord [SMK+01]) in sensor networks. These overlays allow DCS points to be efficiently defined and located by independent entities (sensors and sinks) in the network. Furthermore, correlated data may be aggregated at the DCS points to reduce the size of data that is sent in response to a query. More recently, Seada and Helmy proposed *rendezvous regions* [SH04], a structured overlay in which each key is not mapped to a single node but to a region that contains multiple nodes. Nodes within each region collaboratively maintain their corresponding DCS point. The authors claim that this collaboration allows a higher level of reliability to be reached when nodes are failure-prone or mobile.

2.4 Data Processing

As sensor networks are made of large numbers of sensor nodes, data delivery from producers to consumers can easily congest the network. One solution to data congestion is to use the nodes' computational power and reduce the size of the data that is transmitted to the consumers. This mechanism, *data processing*, can also reduce the amount of energy that is consumed in the network, especially if radio-based communication is involved.

The advantages of data processing are two-fold. Firstly, the size of data is reduced, therefore a costly resource (communication) is partially traded with a cheaper resource (computation). Secondly, the meaning of data is improved. If user-guided data processing is performed, then the result is more meaningful and richer to the user. In this dissertation, I identify two classes of data processing: *data aggregation* and *data fusion*. Data aggregation is a process by which a homogeneous class of data (i.e. data of one type) is processed and reduced in size. The output data is of the same type as the input. This dissertation focuses on standard aggregation operations (similar to those found in the Structured Query Language (SQL)); there also exists a large body of work on *approximate aggregates* and *operator-specific aggregation algorithms* that is beyond the scope of this dissertation. An introduction can be found in [NLF07].

The second class of data processing, data fusion, is where heterogeneous data (i.e. data of different types) are processed into a new (and often more complex) type of data. This class of data processing is almost always user-defined and explored in two separate fields of study: the *Database-oriented abstraction* and the *event-based abstraction*. Compared to the discussed programming models (Section 2.2), these abstractions free the application developer from low-level coding which could significantly complicate the development of complex and correct programs. They provide suitable services (e.g. automated optimization, operator placement and ordering) with desirable guarantees (e.g. safety) at the expense of limiting the application developer to a restricted language semantics through a high-level (often declarative) interface. These abstractions are often an easy and compact way for end-users to write programs for sensor networks.

2.4.1 Database Abstraction

The Database is one of the earliest high-level abstractions proposed for sensor networks. Two approaches are typical in active treatment[1] of data produced by sensors. The first consists of viewing the sensor network as a distributed Database where data resides on spatially spread sensor devices and queries are pushed all the way down to the sensors from one or a few base stations. In the second form, the sensor network is seen as a non-interruptible source of data (data stream) whose data is archived and processed (according to user-expressed queries) at base stations. The first approach results in much less energy consumption if query-specific in-network aggregation can be used, while the second approach is useful for when applications do not know a priori what data processing to perform or if in-network processing is not possible. However, it does introduce the congestion problem noted earlier.

[1]Passive data treatment was discussed in conjunction with passive data delivery in Section 2.3.2

Realisation time	Longevity
ad hoc	one-time (transient)
pre-defined	continuous (persistent)

Table 2.6: DBMS Queries

Database queries can be broadly partitioned across two dimensions of *specification time* and *longevity* (see Table 2.6). Starting with the specification time, *pre-defined* queries are those which are known before any data is received or added to the Database table. This a priori knowledge allows a wide range of query optimizations and planning before data is processed. In contrast, *ad hoc* queries are specified at run-time and provide little opportunity for optimizations unless reconfiguration is possible. In order to avoid unnecessary complexities, most Database Management Systems (DBMSs) only allow ad hoc queries to examine the current state of the Database (i.e. ad hoc queries over the past data are not allowed).

Once a query is specified, the query may merely examine a snapshot of the Database; this query is referred to as a *one-time* or *transient* query, or examine the Database in a continuous or periodic fashion; this query is referred to as a *continuous* or *persistent* [TGN+92] one. Although traditional DBMSs focused on one-time queries, researchers feel that most emerging sensor network applications will demand continuous queries. This demand plus the unique characteristics of sensor networks have inspired a new body of research, surveyed below.

2.4.1.1 DBMS for WSNs

Since the emergence of WSNs, researchers have begun exploring potential applications for these networks, and have found many, some of which were listed in Table 2.4. They also discovered that the expected usage of sensor networks is that *users will query the network and obtain one or more responses*. With sensors producing data and users posing queries, a top-down architectural view resembling *distributed Databases* emerged quickly [GHH+02].

Studies, however, showed [YG03] that sensor networks are different from traditional distributed Databases. Firstly, sensor networks have communication and computation constraints that are very different from regular desktop computers or dedicated equipment in data centers; query processors have to be aware of these constraints. Secondly, the notion of the cost of a query plan is different as the critical resource in a sensor network is power, and query optimization and query processing have to be adapted to take this optimization criterion into account. Proposed solutions (e.g. COUGAR [BGS00; BGS01], TinyDB [MFHH03], SINA [SSJ01; SJS00], and ACQUIRE [SKA03]) have implemented a similar approach to this problem, but have pursued different enhancements and optimizations about the data model, the query model, and the processing model. Below, I provide a generic view of these frameworks' operations, with some emphasis on their distinctions and enhancements where appropriate.

At the high level, the operation of DBMSs for WSNs can be described in terms of their *data model*, *query model*, and *query processing*. The data model describes a single append-only Database for the entire sensor network. The sensor network is viewed as a Database table, whose columns contain all attributes of sensor devices in the network and rows specify the individual sensor data. This table provides a conceptual view of the sensor network for posing queries, as (in reality) data is not there at the query time - data is distributed among the sensor nodes and may not persist (be stored) forever. In TinyDB [MFHH03], the query itself drives the data acquisition, thereby unnecessary data sampling is avoided in order to conserve energy.

The query model describes the supported interface for endpoint users. These users specify declarative queries (written in SQL-like languages) that reflect their data processing needs. Most languages have support for periodic and continuous queries as sensor network applications are expected to operate over longer periods of time. *Aggregate operators*, *event processing capabilities*, *storage points*, and *lifetime queries* are also considered useful and implemented in TinyDB [MFHH03].

TAG [MFHH02] discusses the aggregation operators, and suggests how they may be evaluated (in a decentralized manner) in WSNs. Events are used as a mechanism for initiating data collection in response to some external stimulus. They are generated explicitly, either by another query, by some software in the operating system, or by specialized hardware on the nodes. Since in-network storage is limited, DBMSs for WSNs do not store data after processing; instead, users can explicitly store data by creating storage points (that accumulate a small buffer of data) and referencing this data in other queries.

In proposed DBMSs, a special type of node (called a *gateway node*) is distinguished to intermediate the connection between users and the sensor network. Queries are posed strictly at these gateway nodes. Queries are then planned, optimized and parsed into simple binary representations for distribution by the gateway node. The gateway node is a centralized entity that has accurate information about the network through *meta-data*. The meta-data, which is collected periodically from the nodes, consists of many attribute values such as the available sensor types (e.g. light, temperature, sound), acquirable values (the range of possible readings), cost of acquisition (in terms of power and time), change characteristics, triggering event types, extensible aggregate systems, and operating energy levels. The gateway node uses the meta-data to plan and optimize queries; the result is query fragments that can be evaluated on individual sensor nodes. The query processing goes as follows.

The gateway disseminates query fragments throughout the network. As the query propagates through the network, sensors organize into a routing tree[1] that allows data to be processed up the tree and towards the gateway. In every period (called an *epoch*), sensor nodes evaluate their query fragments and send their results up the tree, where the gateway node (as the root) then delivers the final result to the user. The epoch duration refers to the amount of time between successive data samples (sensor readings). Query processing, at individual sensor nodes, is

[1]Essentially, every node maintains a parent node that is one step closer to the root (the gateway) on the routing tree.

performed in two steps: *sampling with local operator execution* and *data propagation*. Every sensor node has its own query processor that processes and aggregates the sensor data; it then forwards its results to its parent (on the routing tree) which in turn does the same until the gateway node is reached. For some aggregate operators, partial aggregation is possible where parent nodes combine their results with those received from their children. This significantly reduces the cost of communications as well as the size of the data that is received at the gateway node. Table 2.7 highlights some DBMSs, that are developed for WSNs.

Project	Main features
SINA [SSJ01]	supports explicit tasking by implementing SQTL (an imperative language which allows users to embed scripts in SQL)
COUGAR [BGS00]	models sensors as ADTs and their output as time series
ACQUIRE [SKA03]	developed for one-time, complex queries for replicated data
TinyDB [MFHH03]	acquisitional query processor that focuses on efficient data acquisition and optimizes the routing tree
REED [AML05]	extends TinyDB with a static join operator (sensor data is joined with static tables)

Table 2.7: DBMSs for WSNs

The Database abstraction, provided by this body of work, is a powerful programing abstraction for sensor networks. The language is some form of SQL-like language, and mostly supports extensible aggregation functions. These efforts, however, are centered around a single design point: globally scoped queries issued from outside the network. Research in this area largely focuses on efficient query optimizations and evaluations, and overlooks issues concerning adaptation and reconfiguration. The gateway node is central to their operation, and leads to a centralized architecture.

In addition, DBMSs for WSNs have only considered relatively homogeneous sensor networks in which all nodes are equally powerful; the Berkeley MICA motes [HC02b] are often used to motivate and justify design decisions. Future sensor networks are likely to have several tiers of nodes with different performance characteristics. It is unclear how these DBMSs can take advantage of this heterogeneity. Such platform-driven research has also led to some shortcomings when implementations have been considered in the context of real-world applications. For example, TinyDB [MFHH03] developers realised the importance of the relational join operation only after they communicated with Intel engineers, who intended to deploy WSNs for condition based maintenance in their chip fabrication plants. This realisation led to the development of REED [AML05] that extends TinyDB with a static join operator. Free from such platform-driven constraints are Data Stream Management Systems (DSMSs) that will be discussed shortly. Finally, although actuation is supported by some of these DBMSs' languages, it is not immediately clear if a query-specification interface is best suited for actuation. Event-based architectures (discussed in Section 2.4.2) could perhaps provide a more suitable solution for applications' inter-communication and actuation.

2.4.1.2 Data Stream Management System

Recently DBMS researchers have recognized a new class of data-intensive applications (called *stream processing applications*), where data is best modeled as transient *data streams* - not persistent relations. These applications require continuous and low-latency processing of large volumes of data that may arrive at high rates. Examples of these applications include financial tickers [CDTW00; ZS02], network monitoring [GKMS01; SH98], on-line auctions [ABW02], security, manufacturing, and sensor data [BGS00; MF02]. The continuous arrival of data in multiple, rapid, time-varying, unbounded streams has yielded some new research problems. These applications have motivated a new class of Database-oriented systems, called Data Stream Management Systems (DSMSs) that differ significantly from traditional DBMS in terms of data model, query model, semantics, and implementation. Table 2.8 highlights some of these differences, and Table 2.9 provides a summary of main DSMS projects and their contributions.

DBMS	vs	DSMS
persistent relations		transient streams
one-time queries		continuous queries
random access		sequential access
only current state		includes historic data
low update rate		high data arrival rate
pull-based query plan		push-based query plan
unbounded disk space		bounded main memory
data at any granularity		imprecise/stale data
mostly exact answer		mostly approximate answer
no real-time requirements		real-time requirements

Table 2.8: DBMS vs DSMS

A *stream* is an infinite sequence of *(tuple, timestamp)* pairs, that arrive in append-only manner and may only be seen once [HRR99]. Each *tuple* is similar to a row in a Database table, consisting of a set of attributes that conform to a pre-defined stream schema. These tuples may be relation-based (as in STREAM [MWA+03], TelegraphCQ [CCD+03], and Borealis [AAB+05]) or object-based (as in COUGAR [BGS00] and Tribeca [SH98]). The *timestamp* defines a total order over the tuples in a stream. It may be *implicit* (set by DSMS when tuple arrives) or *explicit* (set by the source of data). Explicit timestamps are used when each tuple corresponds to a real-world event at a particular time. The distinction (between implicit and explicit timestamps) is similar to that between *transaction* and *valid* times in temporal DBMSs [SA85].

Project	Main features
Tapestry [TGN+92]	incremental evaluation of continuous (monotonic) queries for append-only Databases
Tribeca [SH98]	dataflow oriented query language in an Internet traffic monitoring tool
NiagaraCQ [CDTW00]	continuous queries over XML documents with dynamic grouping and shared execution of queries that are similar
OpenCQ [LPT99]	similar to NiagaraCQ (continuous queries over distributed persistent data). query processing is based on incremental view maintenance
TelegraphCQ [CCD+03]	considers a stream of data and a stream of queries, supports adaptive query processing and historical data processing
STREAM [MWA+03]	relation-based system with emphasis on memory management, and approximate query answering
Aurora [CcC+02]	workflow-oriented system in which users build query plans using *boxes* and *arrows*. uses timestamps to optimize the QoS.
Borealis [AAB+05]	distributed DSMS based on Aurora and Medusa [CBB+03], supports integration with sensor networks
Gigascop [CJSS03]	two-tiered architecture (low-level queries on source nodes, high-level queries on servers), compiles queries into C/C++ modules, and is designed for network monitoring applications
StatStream [ZS02]	computes on-line statistics across multiple streams
SMILE [JS03; SDBL07]	declarative monotonic continuous queries over Gryphon [SBC+98], supports fault-tolerance [Str04]
HiFi [FJK+05]	supports high fan-in infrastructures, proposes Virtual Devices [JAF+05] to clean filter and aggregate data

Table 2.9: Continuous query processors and DSMSs

DSMSs do not have powerful support for time-based operations, yet the time of observation is important in sensor networks. Timestamps, in DSMSs, are used for ordering more than timing purposes. Use of explicit timestamps give rise to total ordering issues and timestamp assignment problems when generating streams from a binary operator. Thus, DSMSs largely use implicit timestamps and devise simplistic solutions in the case of explicit timestamps. For example, they may assume bounded disorder or drop out-of-order tuples, and in the case of output streams have users specify the timestamps explicitly [BBD+02].

DSMSs use main memory, and cannot assume that the entire stream fits in the memory. They continuously receive new data and drop old data. This has serious implications on the query models and the operators as investigated in [LWWZ04], and discussed here briefly. To date, three querying paradigms have been proposed [GO03]: *Relation-based* languages (e.g. CQL [ABW06], StreaQuel [CCD+03; CKM+03], and Aquery [LS03]) that use SQL-like languages to query time-stamped relations, *Object-based* languages (e.g. COUGAR [BGS00]) that resemble SQL but include support for streaming Abstract Data Types (ADTs) and associated signal processing methods, and *Procedural* languages (e.g. Aurora [CcC+02]) that construct queries by defining data flow through various operators. Many of the operators are inspired by relational operators such as Select, Join, Project, Union, and Aggregates. These operators, however, assume and have been designed for data sets that are bounded in size. This creates a mismatch between DSMS requirements and operator capabilities which is formally studied in [LWWZ04]. Below, I have briefly categorized operators into three classes based on their execution semantics.

Monotonic Operators such as Select and Project are the simplest and most suited to data streams. They are *monotonic* as they can evaluate one element at a time, do not need to hold any state, and can produce results incrementally.

Blocking For some relational operators (e.g. Aggregates) one needs to process the entire input stream before producing a result. These operators are *blocking* because they are unable to produce the first tuple of the output before seeing the entire input stream. To address this shortcoming, most DSMSs propose the use of some type of *window* specification as a way to process tuples in groups. A window is a set of ordered tuples that at any evaluation time is bounded in size. Three types of window can be defined depending on the variability of window endpoints. A *fixed window* has only fixed endpoints, a *sliding window* has only variable endpoints, and a *landmark window* has one fixed and one variable endpoint. The width of a window can be described by a time interval (*time-based*), by the number of tuples (*tuple-based*), or by explicit *punctuation tuples* [TMSF03], which specify the end of a subset of data.

Stateful Other relational operators (such as Join) accumulate state that grows with the size of their inputs. Such state management is inappropriate as the input can be potentially unbounded in size. Windowed computations and approximations are used to reduce the memory requirement.

Processing and optimization of queries in DSMS is also different from DBMS. For a survey of related issues, please consider [BBD+02; GO03]. One area where strong differences emerge is the query and evaluation plans. Relational operators (in DBMSs) are pull-based: an operator requests data from its children in the plan tree only when needed. In contrast, stream operators consume data pushed to the system by the sources. Fjords [MF02] and STREAM [MWA+03] have proposed *queues* to reconcile these differences: sources push data into the queue and operators pull data as needed; but such approaches create new problems such as operator scheduling [BBMD03] and QoS maintenance [CcC+02]. This encouraged me to study a different class of system that is naturally push-based. The following section discusses data processing using an event-based abstraction.

2.4.2 Event Abstraction

There has been substantial research about event processing. In this section I focus on *Composite Event (CE) frameworks*, a branch of work that is closely related to my work. CEs first appeared in the context of active Database rules (or triggers) [WC94], but soon evolved beyond the context of Databases - [PSB04] proposes "Composite Event Detection as a Generic Middleware Extension".

Active DBMSs and CEs. Active DBMSs enhance traditional DBMSs with powerful rule processing (or *trigger*) capabilities [WC94]. This capability provides a uniform and efficient mechanism for many Database features and applications, including integrity constraints, views and derived data, authorization, statistics gathering, monitoring and alerting, knowledge bases and expert systems, and workflow management. Rules, in active DBMSs, specify the desired active behavior; they are defined by users, applications, or Database administrators. The most general form of a rule is an Event-Condition-Action (ECA), also called an *ECA rule*. The event causes the rule to be triggered, the condition is checked when the rule is triggered, and an action follows if the condition is true.

Operator	Semantic	Condition
conjunction	A, B	A and B occur in any order
disjunction	$A\|B$	A or B occurs
sequence	$A; B$	A occurs before B
negation	$\neg A$	A does not occur (usually within a time interval)
iteration	A^*	any number of A occurrences
selection	A^n	the n^{th} occurrence of A

Table 2.10: CE operators

In active DBMSs, sources [PD99] of events could be *structure operations* (e.g. insert, update, access tuple), *behavior invocations* (e.g. execution of some operation), *transactions* (e.g. abort, commit, begin), *abstract* (e.g. information entered by user), *exceptions* (e.g. unauthorized data access), *clocks* (e.g. day event), or *externals* (e.g. some pressed a button). An event is considered *primitive* when it is raised by a single low-level occurrence belonging to the described categories; otherwise, it is *composite*, raised by some combination of primitive or CEs using a range of operators that constitute the event algebra. The set of primitive or CEs that raise a CE are referred to as its *constituent events*. Some of the most common operators supported in active DBMSs are shown in Table 2.10. As evident, these operations examine patterns or sequences of *event occurrences*; therefore events almost always need to have timestamps. The timestamp is assigned either when the event arrives at the system (implicitly), or when it is generated at its source (explicitly). The type of the event is determined by its source, and if the event is attributed (contains values) then its attributes may be examined by the rule's condition or action. It is important to note that primitive and CEs only reflect the event component of an ECA rule.

Early efforts, in the context of CEs, focused on expressive event algebras and efficient CE detection models. Some of the detection models proposed to date are based on event graphs [CM94], Finite State Automata (FSA) [LGA96], Petri-Nets [GD93], λ-calculus [Cou02], and rules [FJLM05]. These models offer various formalisms, and attain performance gains through incremental CE detection (e.g. as in Snoop [CM94], ODE [LGA96], SAMOS [GD93]), and/or sharing of partial CE detection (e.g. as in EPS [ME01] and PADRES [FJLM05]). With the advance of networking technologies, however, the domain of applications that could benefit from events and active behavior grew beyond the scope of centralized active DBMSs. This led to the development of CE frameworks in a number of directions, which I will survey under *system environment*, *parameterization*, and *selection & consumption policies*.

System environment. With the separation of event component from the ECA rule, the context also moved from centralized active Databases. In particular, the shift from centralized systems to distributed systems led to research on timing issues (that addressed the lack of a global time in distributed systems) and spontaneous events (that were not possible in centralized systems). A common approach to assigning time and order to distributed events is the creation of *virtual clocks* at each site using the local hardware clocks [LLCB99]. Virtual clocks count real-time units (e.g. seconds) and map the reference time (a granular representation of dense physical time) to a clock-time that has a granularity by which its counter is incremented (e.g. seconds). An external clock synchronization aims at bounding the maximal time deviation between the virtual clocks and the reference time, and an internal clock synchronization aims at reaching a consistency between the virtual clocks. Several real-time mechanisms have been proposed to date; three widely used ones are: *network time protocol* [Mil89], *2g-precedence model* [KFG+93; Sch96], and *interval-based time systems* [LLCB99; PSB03].

Furthermore, event-based messaging paradigms (such as Pub/Sub) became increasingly popular as the infrastructural context for CE frameworks; their support for distributed event dissemination naturally suits CE frameworks in delivering primitive and CEs to corresponding CE detectors. Examples of projects that support CEs in the context of Pub/Sub middleware are REBECA [Mö01; FMG02; UMWG04], PADRES [FJLM05], and CEA [PSB03]. CE frameworks have also been studied in the context of sensor networks, though with limited expressiveness. DSWare [LSS03] supports local CEs by combining primitive events by the conjunction operator; the emphasis has been put mainly on timeliness and reliable detection of CEs over uncertain primitive events. COMiS [KVJ05] extends DSWare for detecting global CEs but retains its limited expressiveness to the conjunction operator. Finally, Mark Kranz (SENSID [Kra05]) has explored the feasibility of porting Amit [AE04], an expressive situation detector, over networks of Berkeley MICA motes [HC02b]. SENSID [Kra05] can detect expressive situations at the level of local nodes (i.e. detects local CEs). Tables 2.11 and 2.12 outline the main contribution of a number of CE-related projects.

Parameterization. With the evolution of events (from ECA rules) into model-independent CE frameworks, the need for condition specification became increasingly apparent. Condition

Project	Main features
Alert [SPAM91]	supports ECA rules in conventional Databases by means of continuous queries over append-only active tables
HiPAC [DBB+88]	active object-oriented DBMS with time-constrained data management
ODE [LGA96; GJ96]	CEs can be expressed as regular expressions (detection model uses FSA), condition can be examined as part of the event expression (supports Event-Action (EA) rather than ECA)
SAMOS [GD93; GD94]	the detection model uses colored Petri-Nets
Snoop [CM94; CKAK94]	model-independent event specification language, supports parameter contexts, the detection model uses an event tree
GEM [MSS97]	generic event monitor for distributed systems, rule-based language, detection model uses tree
EVE [GT96]	combines characteristics of active Databases and event-based architectures to execute event driven workflow
EPS [ME01]	implements a shared subscription tree for CE detection
[Cou02]	CE specification based on λ-calculus
CEA [PSB03; PSB04]	distributed CE detectors, extends a Pub/Sub middleware, language compiles into FSAs
REBECA [Mö01; FMG02]	provides programming abstractions for Pub/Sub middleware in object oriented languages, addresses distributed timing [LLCB99], CE language maps to the core language in CEA [PSB03]

Table 2.11: CE-related projects (part 1)

specification over the attribute values of individual or multiple events (that constitute the CE) is referred to as *parameterization*. Parameterization is often supported at the pre-CE and/or post-CE detection phase; this enables the re-use of efficient CE detection models that were previously developed in active DBMSs. Pre-CE detection parameterization is either encapsulated as part of the CE definition (e.g. as in Amit [AE04]) or defined as part of the type system (e.g. as in DSWare [LSS03], GENAS [HV02], and CEA [PSB03]). Pre-CE detection parameterization fits naturally in the context of content-based Pub/Sub systems, where candidate constituent events are received as a result of content-based subscriptions. On its own, however, pre-CE detection parameterization delivers limited expressiveness; for example, inter-event parameterization (cross-examination of constituent event attributes) is not possible. Post-CE detection parameterization provides higher expressiveness (at the expense of late event filtering) and is

Project	Main features
PADRES [FJLM05; LJ05]	integrates CE detection with content-based Pub/Sub, detection model uses event graphs that are mapped to rules, allows access to historic data [LCH+07]
GENAS [HV02]	supports flexible event selection and consumption for CE detection
Amit [AE04]	introduces lifespan (temporal context) during which situation detection becomes relevant, exhaustive support for lifespan definition, event instance override, selection, and consumption policies, and parameterization
SASE [WDR06; GC+07]	combines sequencing and declarative SQL, supports windows and negation, uses an event Database to support queries over history [GC+07], evaluates sequences using Non-deterministic Finite Automata (NFA)
Cayuga [DGH+06; DGP+07]	augments an SQL-like language with FILTER (unary), NEXT (binary), and FOLD (binary) constructs that support sequencing, supports multi-query optimization, detection model uses stateful NFA
DSWare [LSS03]	detects local CEs in sensor networks, support conjunction operator, addresses events' uncertainty by confidence values
COMiS [KVJ05]	detects global CEs in sensor networks, supports conjunction operator with restricted parameterization
[KBM04]	proposes an abstract event specification language based on temporal first order logic, abstract states are deduced using knowledge-base

Table 2.12: CE-related projects (part 2)

only implemented in a limited number of CE frameworks (e.g. Amit [AE04], SASE [WDR06], and Cayuga [DGH+06; DGP+07; BDG+07]). Interestingly, some CE frameworks that have approached expressive inter-event parameterization have resulted to the integration of declarative SQL-like languages (e.g. as in SASE [WDR06] and Cayuga [DGH+06]) into the event algebra.

Selection & Consumption Policies. Typically, when all constituent event types of a CE have occurred, there are many instances of a constituent event type that can be examined for event composition. Event selection and consumption policies define how these instances are treated in the scope of present and future CE detections. Event selection describes which qualifying events are taken into account for CE detection, and how duplicate events (events with matching type and timestamp) are handled. In order to reduce complexity and ease

CE specifications, most frameworks adopt a fixed event selection policy that either suits their application environment (e.g. EVE [GT96] selects a fixed (*chronicle*) policy that suits workflow management), or suits their detection model (e.g. Cayuga [DGH+06] has operator-dependent selection policies). Others, to achieve generality, either compute the most general case (e.g. as in PADRES [FJLM05]) or introduce parameters that can be set by the user (e.g. as in GENAS [HV02] and Amit [AE04]). Amit [AE04] supports one of the most expressive selection policies to date; it provides a separate *override* policy (for received events) and offers a number of event selection parameters that are detailed in Table 2.13. Table 2.13 also shows the four selection parameters that Snoop [CM94] considers to be most useful across a wide range of applications - most frameworks (with fixed policies) describe their selection policy in terms of these four. Note that the *continuous* and *cumulative* Snoop contexts have the same selection policy but differ in the consumption policy, discussed next.

Selection	Description	Snoop's contexts
first	selects the first instance that satisfies the conditions	chronicle
strict first	selects the first instance if it satisfies the conditions	
last	selects the last instance that satisfies the conditions	recent
strict last	selects the last instance if it satisfies the conditions	
each	selects all the instances that satisfy the conditions	continuous,cumulative

Table 2.13: Event selection parameters

Event consumption defines whether an event can be shared by multiple CEs (of the same type) or not (i.e. should be exclusively assigned to just one CE or not). In the former case the event is not consumed, while in the latter it is consumed; [ZU99] provides an overview of the selection and consumption policies for some of the earlier CE frameworks. The downside of this level of expressiveness, pursued in some CE frameworks like GENAS [HV02] and Amit [AE04], is that (for an ordinary application developer) it is difficult to predict how the choice of parameters can impact the system performance or resource usage, and yet some combinations (such as the *each* selection policy combined with the *shared* consumption policy) can be very unforgiving (resulting in a linear increase in memory usage and exponential increase in CE detections). Amit [AE04] introduces the notion of *lifespan* that confines the scope of CE detections to a window (of events) that is defined by an *initiator* and a *terminator*. This reduces the consequences of sloppy mistakes but also makes the CE specification language more complex. In fact, in Amit [AE04] many of the language construct values are redundant. This violates the minimality of language constructs that is a requirement of good language design [Cod71]. As an alternative, I believe that time and location attributes of data (in sensor networks) can provide an expressive and safe means of selecting and consuming events (data) for high-level information deduction. In this dissertation, I explore temporal and spatial selection and consumption policies beyond what has been proposed to date for sensor networks.

Chapter 3

State Filters

In this chapter I present State Filters (SFs) [TB07c] for capturing user interests in Resource-constrained Sensor Networks (RSNs). SFs aim to substitute the content-based filters used in Pub/Sub protocols (Section 2.3.1.1) for more expressive conditions and improved communication efficiency. They reduce the communication costs by exploiting the redundancy and correlation that is inherent in the sensor readings, reflecting a shared external environment. They also capture lasting conditions, mirroring lasting phenomena in a continuous environment, over a series of discrete events (data that is captured at discrete time points). The operational semantics of SFs are often coded as part of the application-level logic over sensor devices (Section 2.2), but in this chapter I formalize this operation as a component which lies within the Pub/Sub middleware and is independent of the clients layer (e.g. applications). In the design of SFs, compatibility with existing content-based filters has been considered, and the result has been a set of components that can subsume content-based filters and integrate well with the available Pub/Sub implementations that support attributed content-based subscription model. My evaluation, using real sensor data, demonstrates that SFs improve expressiveness and event filtering for RSN applications compared with the content-based filters.

This chapter opens with an introduction to RSNs, their characteristics and potential application areas. This is followed (in Section 3.2) by a discussion of Pub/Sub and its most widely used form of subscription language in RSNs. The review highlights some shortcomings of the content-based subscription model, due to data challenges in RSNs (Section 1.2.1) and motivates my work on SFs whose semantics are described in Section 3.3. Section 3.4 describes the distribution of SFs, and Section 3.5 evaluates the overall approach for expressiveness and effective event filtering. Related work is discussed in Section 3.6, and concluding remarks are made in Section 3.7.

3.1 Resource-constrained Sensor Networks

As sensor technology matures, wider ranges of platforms and sensor types have become available. A unique sensor type may have different hardware implementations, each of which offers a different level of reliability and accuracy in its readings. Platforms also vary as they adapt to

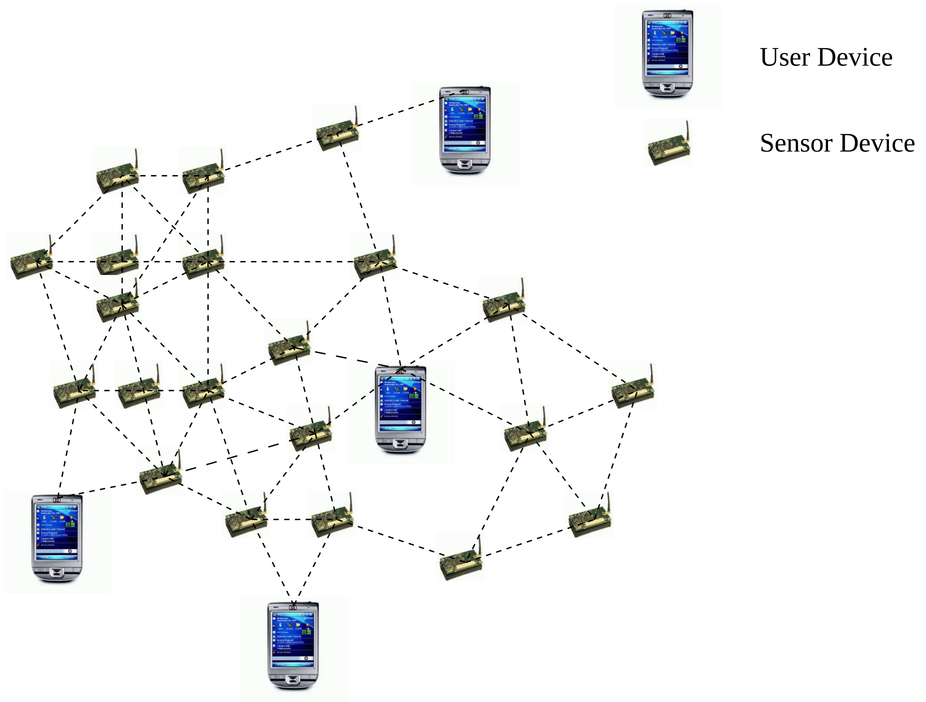


Figure 3.1: Ad hoc WSN topology

different sensor requirements: they may house small sensors such as temperature sensors, or large ones like inductive loops beneath roads. This diversification extends the range of applications that can be developed for sensor systems, and complicates general solutions. In this chapter, I focus on a group of sensor networks called *RSNs*.

RSNs are a subclass of sensor networks, where resource shortages prevent costly operations and protocol executions within the network. By costly operations I mean extensive memory usage, frequent communication, and/or complex computations. Such operations are often shifted to network boundaries, where devices with more resources (e.g. gateways or application nodes) are present.

The use of RSNs is motivated by low manufacturing costs and the small size of devices. They may be deployed in inaccessible areas where size matters and robust deployment is not possible. Habitat monitoring [CEH+01], for example, is not possible with large sensor devices, and robust wide-area deployment of temperature sensors, for example in a forest, is expensive (instead, primitive devices are used). In these networks, failures are anticipated and robustness is increased through redundancy. The next section takes a closer look at the common characteristics of RSNs.

3.1.1 Characteristics

Nodes in RSNs largely communicate over *wireless* media to benefit from low-cost deployment and ad hoc network formation; therefore RSNs can be considered as a special case of WSNs.

Low cost and ad hoc deployment requirements often result in nodes being *small* and *battery powered*, which in the case of unmanned operations and infrequent maintenance implies that they have limited lifetime. This reinforces the need for *primitive* resources (sensing, processing, storage, and communication components) that consume little energy. For example, nodes might only communicate within short ranges and form a *mesh-based* network topology with nearby nodes (see Figure 3.1). In addition, only *low-power sensing* is used, and the option of high-power sensing hardware that can provide more useful information is generally not available. Nodes are often *stationary*, unless moved by the environment (passive mobility in Section 2.1.1.4).

Size also takes a dominant role in restricting capabilities and yielding shared characteristics. Smaller devices are more vulnerable to environmental effects, and thus more *failure-prone*. Onboard size limitations result in *small memory space* and *basic processing core*. These restrict code space, and available dynamic memory, for protocol operations. In most instances, basic functionality focuses on communicating results toward a user or a base station; filtering and processing is done if practical. The next section highlights two application scenarios that can benefit from this class of sensor networks.

3.1.2 Application Scenarios

Applications for RSNs aim to exploit some of the more useful characteristics outlined above. Small size and low-cost production features allow environments to be monitored with ease and little cost. The application scenarios that I have selected to discuss here are *monitoring hazardous conditions in underground mines* and *monitoring office environments*.

Monitoring Hazardous Conditions in Underground Mines. When underground mine accidents occur, knowledge about the environmental conditions (along the rescue path) can significantly aid the rescue efforts. This knowledge may include the temperature, level of carbon monoxide, and presence of methane or other dangerous gases in the air. WSNs can be used to monitor these conditions when rescue operations are taking place. Prior knowledge about these conditions may also be used to prevent accidents.

In this scenario, sensor devices are scattered around the mines with sufficient density to ensure wireless connectivity. Base stations, located outside the mine, operate as gateways in-between the sensor network and external applications. Constraints over the observed data are defined by applications and partially injected into the network for in-network processing. In-network processing is basic, often confined to primitive filtering.

Monitoring Temperature and Humidity in Office Environments. Indoor temperature and humidity levels can have a direct effect on people's comfort, productivity, respiratory health, and well being. RSNs scattered within office environments can monitor the temperature and humidity levels in a ubiquitous way. These sensors report to base stations that are connected to user clients as well as related actuation devices (e.g. air conditioners). Significant temperature and humidity changes or deviations from the desired levels are reported to the base stations, which may notify the user or automatically trigger the appropriate actuators. The small and primitive nature of devices mean that numerous sensors may be deployed (per office environment) for increased robustness.

3.2 Publish/Subscribe

Pub/Sub provides data-centric communication between publishers and subscribers. Sensor devices are viewed as event publishers and sinks (user client or base station nodes) as event subscribers. Pub/Sub's loose-coupling means that sensors and sinks can interact without direct knowledge of each other. Instead, they relate to each other by data structures and values.

Data in Pub/Sub is represented by *Events*. An event model describes how an event is represented in the system. For RSNs, described above, I assume a flat, unstructured event space, $\mathbb{E}$.

Definition 3.1 (Event). *An event e consists of a tuple τ and belongs to the event space $\mathbb{E}$,*

$$e \in \mathbb{E}. \tag{3.1}$$

The tuple τ contains a number of attributes, *$\tau = (a_1, \cdots, a_n)$, where each a_i is a* name-value pair, *(n_i, v_i), with name n_i and value v_i. Attribute names are unique, i.e. $i \neq j \Rightarrow n_i \neq n_j$.*

Event dissemination in RSNs is challenging. Power constraints demand communication efficiency, while wireless (radio) communications pose serious unreliability and failure-proneness. Researchers have developed numerous routing and event dissemination protocols (see Section 2.3.1 or [HCRW04; CPR05] for Pub/Sub implementations) that target this goal under various network settings and environmental assumptions. The question as to which events match which subscribers relates to the subscription model, described next.

3.2.1 Subscription Model

Events are matched to subscribers according to their subscription expressions. These expressions, described by a subscription model, reflect the set of events that subscribers are interested to receive. The expressiveness of the subscription model determines the *usability* and the *computational overhead* of the Pub/Sub protocol. While usability is desired, computational overhead, in RSNs, must be restricted. *Topic-based* and *content-based* subscription models are the two most widely used subscription languages in RSNs.

A topic-based subscription model is suitable because event publishers are often typed according to their local sensing hardware. Use of topics alone, however, leads to the delivery of *all events* that are published under a certain topic by the corresponding sensors. A content-based subscription model can improve on this, allowing subscribers to more finely describe their data interests. In its simplest form, content-based subscriptions can examine event attributes individually. In a more general model, cross-examination of event attributes is also allowed.

Definition 3.2 (Event Subscription). *An event subscription s consists of a* filter F*, that is a stateless Boolean function.*

$$s = F : \mathbb{E} \to \mathbb{B}. \tag{3.2}$$

The filter F can be applied to an event $e \in \mathbb{E}$ to give a boolean value, $F(e) \mapsto x \in \{true, false\}$. In its simple form, the filter F consists of attribute predicates p_i *that are combined using the* conjunctive operator,

$$F = p_1 \wedge p_2 \wedge \ldots \wedge p_k. \tag{3.3}$$

An attribute predicate p is a tuple, $p = (n_p, o_p, v_p)$, where n_p is an attribute name, o_p is a boolean test operator, and v_p is an attribute value.

An event matches a subscription if it satisfies its filter F.

Definition 3.3 (Subscription Coverage). *An event e is covered by (or matches) a subscription s,*

$$e \sqsubseteq s, \tag{3.4}$$

if and only if

$$F(e) = true. \tag{3.5}$$

If $F = p_1 \wedge p_2 \wedge \ldots \wedge p_k$, then the above holds if and only if

$$\forall p \in F.\ \exists a \in e.\ a \sqsubseteq p \tag{3.6}$$

holds. An event attribute $a = (n_a, v_a)$ is covered by (or matches) an attribute predicate $p = (n_p, o_p, v_p)$,

$$a \sqsubseteq p, \tag{3.7}$$

if and only if

$$(n_p = n_a) \wedge o_p(v_p, v_a) \tag{3.8}$$

holds.

The above (content-based) subscription model can support topic-based subscriptions if the first attribute is set to topic name, $n_1 = topic$. The content-based subscription model provides four notable features:

Expressiveness Users can accurately describe their interests using attribute predicates.

Computational efficiency Events are examined once and in isolation (individually) against the attribute predicates.

Messaging efficiency Irrelevant (uncovered) events are filtered out.

Preservability Event operations are filter-only processes that leave the event structure intact.

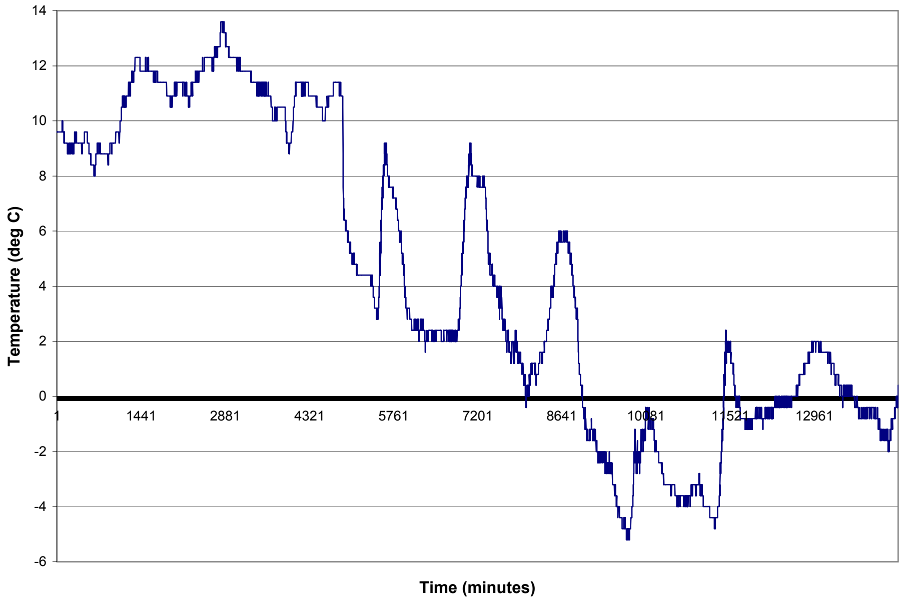


Figure 3.2: Temperature sensor readings (14400 minutes)

Discussion. Sensors observe a shared and continuous environment, where lasting phenomena are present. These observations are instantaneous and often periodic (see *Scalar Sensors* in Section 2.1.1.8). As a result, high correlation and redundancy is observed among the set of events that are published by one or more sensors that observe the same phenomena. Figure 3.2 shows a plot of temperature readings (by one sensor) for 10 days.

Content-based subscriptions filter those events that do not match subscribers' interests, but pass all events that match the subscription. These passed events include the correlated and redundant observations that match the subscription. Therefore, following a subscription match, subscribers may receive many more events that indicate much the same information as others (e.g. many events may indicate that the temperature is now below 0 °C). These events impose an $O(n)$ communication overhead in theory[1], and vary in numbers according to the sampling granularity (event publishing rate) and deployed sensor redundancy. The performance of the content-based subscription model could be significantly improved if redundant and correlated events were also to be filtered.

Another weakness of the content-based subscription model is that it does not provide an accurate knowledge about the duration of conditions or phenomena of interest to the subscribers. Every event that is delivered to the subscriber signals the continuation and persistence of the related phenomena, but not its termination (ending). The time that follows every event delivery represents an uncertainty period where the lack of event publication or the filtering of events can not be distinguished at the subscriber. The subscriber is continuously doubtful as to whether he/she will receive the next observed data (event) or not. In fact, the subscriber can benefit

[1]The run-time overhead may be even higher due to network interference and congestion.

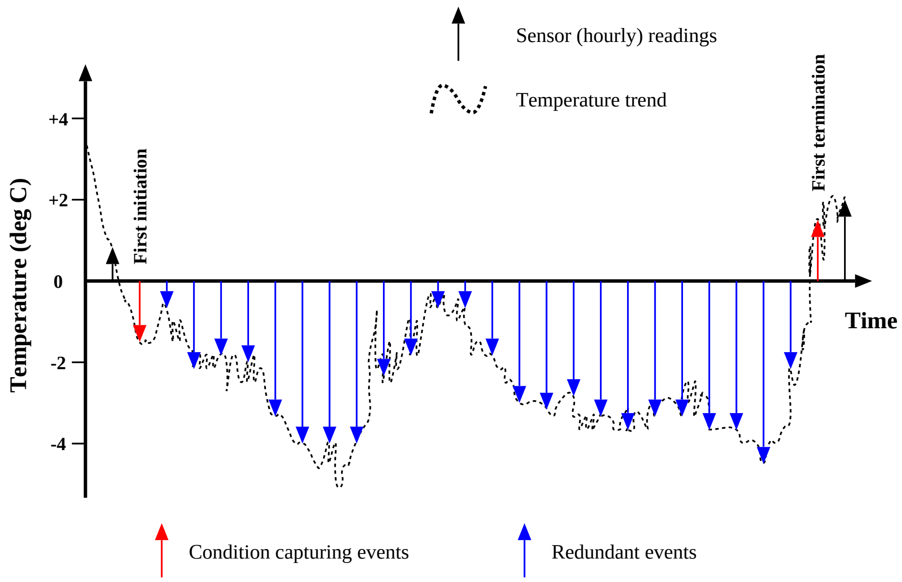


Figure 3.3: Capturing temperature below 0 °C condition

from some knowledge that indicates the definite termination of its phenomenon of interest (e.g. an event that indicates that the temperature is no longer below 0 °C). This *incomplete capture of lasting conditions* and the *inability to filter correlated and redundant events* are concerns that I address by introducing State Filters in this chapter.

3.3 State Filters

State Filters (SFs) are stateful content-based filters that extend the content-based subscription model in expressiveness and filtering efficiency. The extension comes with minimal state storage and maintenance costs that are detailed in Section 3.3.1. I call these components State Filters as opposed to State Detectors because the output event type is always the same as the input event type. This property distinguishes SFs from detectors (such as Composite Events) that produce events of different type than the input events. SFs examine events individually and according to some subscriber-specified expression that matches the content-based subscription filters. They match content-based filters in simplicity and implementation, and can replace content-based filters in Pub/Sub implementations that are developed for RSNs.

The SF subscription model is designed to capture lasting conditions over discrete events. A *lasting condition* is defined as an environmental phenomenon, whose observation (through sensors) corresponds to a set of sequential events that all match a user's subscription. Lasting conditions are captured using a pair of events, that signal the start and the end of the condition, respectively. For example, consider Figure 3.3 where the temperature trend is shown and hourly sensor readings are marked by arrows. Let's assume that the user is interested in being notified when the temperature is below 0 °C. This is a lasting condition. I say that a lasting condition is

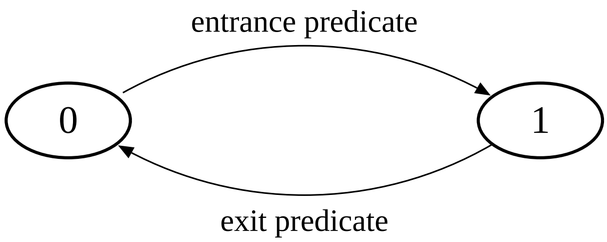


Figure 3.4: FSA representation of an SF

captured accurately, if one can isolate the first instance of an observation (event), that *suggests the condition*, and the first instance of an event that *disproves the condition* (see Figure 3.3). The accuracy of capture is dependent on the sampling rate.

Events that fall in-between the start and the end of the capture deliver little new information (about the occurrence of the condition) and are regarded as *redundant*. These events may be realised across one or more sensors, and often originate from periodic sampling in scalar sensors, and dense deployments in sensor networks (Section 2.1.1). SFs use the notion of state to maintain persistent knowledge about the condition being observed, and to filter out events that are irrelevant or redundant. Events that pass through SFs are highly informative, containing *unique* and *transitive* knowledge about the conditions. These events deliver knowledge more efficiently and effectively than their counterpart events that pass through a simple content-based filter. The next sections describe my SF model, and explain how lasting conditions are specified and captured using SFs.

3.3.1 State

SFs use the notion of state, borrowed from FSM and event calculus, to maintain knowledge about the observing phenomenon. This knowledge indicates whether the condition of interest has been detected or not. With this persistent knowledge, SFs can examine the observed data (events) more effectively.

Every SF defines two states: a state of *null*, which is the initial state and reflects the unknown, or absence of the condition, and a state of *detection*, which reflects knowledge about the existence or occurrence of the desired condition. A *status bit* is used to maintain information about the current active state - zero for null and one for detection. Every SF also has a pair of boolean predicates that govern the state transitions (see Figure 3.4).

Definition 3.4 (State Filter)**.** *An SF, s, consists of a tuple,*

$$s = (P^n, P^x, b), \tag{3.9}$$

where P^n and P^x are the entrance and exit predicates, respectively, and b is the status bit. The predicates guard transitions between two states of null *and* detection*, and b maintains information about the current active state ($b = 0$ when active state is* null*, and $b = 1$ when active state is* detection*).*

Events are examined against entrance and exit predicates to accurately capture lasting conditions. Initially, the entrance predicate is examined for condition initiation, and then the exit predicate is examined for condition termination. A SF can only detect conditions that are separated in time, i.e. two conditions cannot overlap in time or start one-another[1]. A SF can also be formally described using *Event Calculus* [KS86].

Let us label the SF's *null* state as fluent *S0*, the SF's *detection* state as fluent *S1*, the set of events that satisfy the SF's entrance predicate (P^n) as *E0*, and the set of events that satisfy the SF's exit predicate (P^x) as *E1*. The status of the condition being observed, denoted by fluents *S0* and *S1*, can thus be determined by deductive event calculus using the following predicates that describe the effects of events *E0* and *E1*. In the following predicates, t denotes the time points and notation has been borrowed from [Sha97].

$$InitiallyP(S0) \wedge InitiallyN(S1) \tag{3.10}$$

$$\forall t\ Happens(E0, t) \wedge HoldsAt(S0, t) \rightarrow Terminates(E0, S0, t) \wedge Initiates(E0, S1, t) \tag{3.11}$$

$$\forall t\ Happens(E1, t) \wedge HoldsAt(S1, t) \rightarrow Terminates(E1, S1, t) \wedge Initiates(E1, S0, t) \tag{3.12}$$

In addition, at every time point t at most only one event can happen in the system. This is because a SF cannot examine multiple events simultaneously. The next section describes the expressiveness of the predicate language, used to describe condition occurrence and termination constraints.

3.3.1.1 Predicate Language

Predicates are *boolean* expressions that can have either a *true* or a *false* value. Event attribute names and data values are used as operands, and a set of operators are used to process and constrain the event attribute values.

The set of supported operators are divided into three classes:

Mathematical These operators (symbols: $+, -, *, /, ||$(ABS)) join event attribute values and/or data values.

Comparative These operators (symbols: $>, <, \geq, \leq, ==, !=$) form boolean values from ordered data type comparisons.

Logical These operators (symbols: && (AND), || (OR), ! (NOT)) combine several boolean expressions to form a single compound expression. The unary operator NOT is an exception to the above, and negates a boolean expression.

A predicate may be examined against an event, in which case the event attribute names are substituted with their associated attribute values, and the entire expression is evaluated as a boolean function to output either *true* or *false*. A predicate is said to have been *satisfied* if its value is *true*.

[1] An SMC, introduced in Chapter 5, can detect concurrent conditions.

3.3.2 Subscription Model

Subscribers define their conditions of interest using SF subscriptions.

Definition 3.5 (Event Subscription)**.** *An event subscription, s, of class* State Filters*, consists of two boolean predicates P^n and P^x, and a set of* scopes *S,*

$$s = \{P^n, P^x, S\}. \tag{3.13}$$

Each scope $S_i \in S = \{S_1, S_2, \ldots, S_n\}$ indicates a group of event publishers that may observe a condition independently (discussed later in Section 3.4.2). The predicates define an SF, and may be examined against events e to give boolean values, $P(e) \mapsto x \in \{true, false\}$.

Definition 3.6 (Subscription Coverage)**.** *An event e, from an event publisher e_p, is covered by (or matches) a subscription s,*

$$e \sqsubseteq s, \tag{3.14}$$

if and only if

$$\exists u \in S.\ e_p \in u.\ (b = 0\ \wedge\ P^n(e) = true)\ \ \vee\ \ (b = 1\ \wedge\ P^x(e) = true), \tag{3.15}$$

where b is the status bit of the associated SF. If e is covered by s, $e \sqsubseteq s$, then the status bit b is toggled,

$$e \sqsubseteq s \Rightarrow b = \begin{cases} 1 & for \quad b = 0 \\ 0 & for \quad b = 1 \end{cases} \tag{3.16}$$

When the condition is not detected, events are examined for condition detection (the P^n predicate is evaluated), and when it is detected, events are examined for condition termination (the P^x predicate is evaluated). This introduces *context-based* data processing, where events are examined against different predicates according to the current status of the condition.

A subscriber receives a pair of events per captured condition. The first event signals the detection of the condition, and the second event signals its termination. Other events are filtered, as they convey either irrelevant or redundant knowledge about the condition. Using SFs and assuming reliable event services, a user can hold firm knowledge about a condition's continuous presence for a period that is bounded by a pair of capturing event notifications.

Not all conditions of interest may be lasting though. Some may be *momentary*, and the content-based subscription model captures these best. The SF subscription model can accommodate these filters as follows. Let us label a content-based filter expression as F, then the equivalent SF subscription is one that has both predicates set to F, i.e. $P^n = P^x = F$.

3.4 Distributed Filtering

So far, the placement of SFs has not been discussed. This section investigates the consequences of shifting SFs into the network, away from the subscribers and towards the publishers.

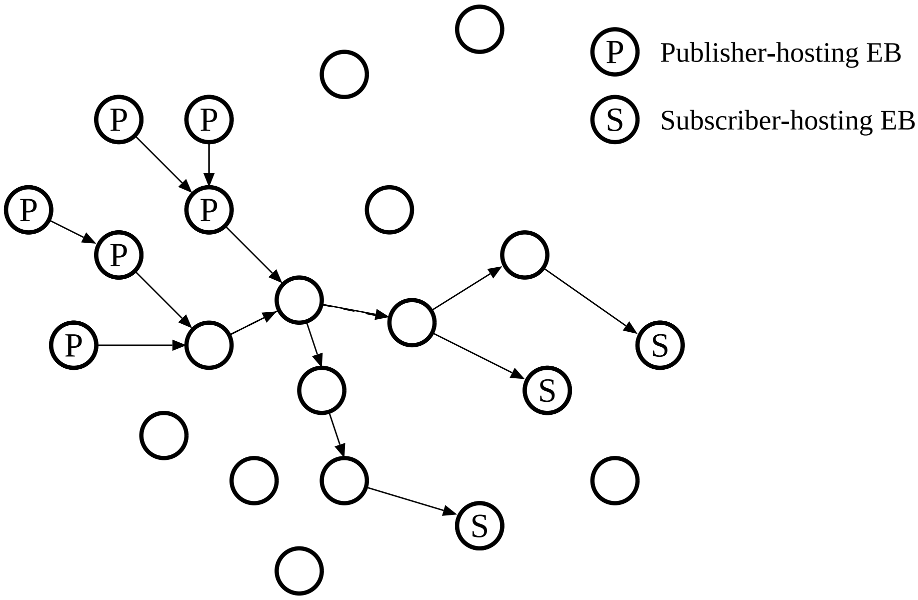


Figure 3.5: An EDT (involving six publishers and three subscribers)

The lightweight design of SFs allows them to operate over most resource-constrained sensor devices. Distribution of SFs results in load-balancing, where the storage and computational costs of SFs are spread across many devices. Where this distribution is applied strategically, communication costs in the network may also be reduced. An added functionality, *condition scoping*, may also be achieved if SFs are distributed over Event Dissemination Trees (EDTs).

Many Pub/Sub protocols, developed for RSNs, construct an EDT for event dissemination. Publisher-hosting Event Brokers (EBs) define the roots of the EDT and subscriber-hosting EBs define the leaves. The EDT then disseminates events, which are published by the publishers, from the roots to the leaves (see Figure 3.5). The arrows in Figure 3.5 indicate the event forwarding paths in the network. I continue my discussions with reference to an EDT Pub/Sub model, that is commonly used in stationary sensor networks.

An SF may be hosted at an event forwarding node (forwarding EB) that is part of the EDT. The forwarding node would then examine events that are received on the EDT, against the hosting SF, and only forwards the events if they pass through the SF. The next section discusses event ordering that becomes important when SFs are distributed in the network.

3.4.1 Detection Policies

The proposed SF model is order sensitive. This means that the order in which events are processed (filtered) has impact on results (matched events), and the accuracy of captured conditions (i.e. whether the first instances of detection and termination are captured or not).

This ordering concern is negligible when SFs are hosted on publisher nodes, but becomes substantial when filtering is performed in conjunction with event forwarding in the network.

Even if I assume that the network does not re-order events, events may arrive out-of-order at the SF-hosting nodes due to different network delays that are involved in routing events from different publishers to an SF-hosting node. Resolving this concern introduces a trade-off between the *event delivery latency* and the *accuracy of condition capture*. In this trade-off, the application requirements become important. I thus propose different detection policies that provide alternative trade-offs.

Best-Effort Detection Policy The best-effort detection policy states that events are processed (at forwarding nodes) as they arrive, without delay. It targets *timeliness*, where events are processed and delivered as quickly as possible to the subscribers. This policy may result in missed detections when conditions occurring over short durations follow one-another. The accuracy of condition capture may also be reduced as delivered events may not reflect the first instances of condition initiation and termination. The loss of accuracy may not be important for applications where causality is not an issue. As discussed before, subsequently occurring events convey the same information, albeit with slightly inaccurate timestamps. The receipt of any of these events that initiate or terminate the condition may be sufficient.

Guaranteed Detection Policy In guaranteed detection policy I process events in total order. This ensures that the first instances of condition initiation and termination, are identified and delivered to the subscribers. This policy is usually expensive to achieve in RSNs. A lightweight approach that can offer acceptable performances is to buffer events for a time interval, t, and then re-order them according to their timestamps[1]. The time interval t is adjustable, but initially set to the maximum network round-trip period. This approach is vulnerable to the distributed clock drift problem, as well as abrupt changes in the network delay (due to link failures or network partitions). A time synchronization protocol (see Sections 2.1.3 and 2.4.2) may be used to reduce clock drift to acceptable levels.

3.4.2 Detection Scoping

Subscription SFs may be replicated and distributed within a network. The subscription's detection scopes set indicates the set of events that are examined by each SF.

Definition 3.7 (Detection Scope)**.** *A scope S, for an SF F, is the set of publishers, whose events are examined by F,*

$$S = \{e_p | F(e)\}, \tag{3.17}$$

where e_p is the publisher of event e.

A detection scope S is expressed by the subscriber (as part of its subscription) and indicates the scope of data correlation and redundancy across multiple sensors (publishers) with respect to its condition of interest. Publishers that belong to the same detection scope are believed to

[1]Events are timestamped at their publisher-hosting EBs.

capture a *unique* condition in the environment; thus events that are published by these sensors can at most signal the presence of a single condition in the environment. For example, consider the use of outdoor light sensors to capture a lasting condition known as the "night time". For this specific condition, many sensors are believed to report correlated or redundant values because they observe the same condition.

Where multiple independent conditions can occur concurrently, the user should define multiple detection scopes accordingly. For example, each (closed) room in a building could separately and independently satisfy a "brightness" condition (a lasting condition that is detected by indoor light sensors when the lights are on). Multiple detection scopes (belonging to the same detection scopes set) are often disjoint and capture concurrent conditions independently, but they can also contain one-another (i.e. $S_1 \subseteq S_2$) in which case *nested scoping* (discussed shortly) is achieved. Partial overlap (without containment) between detection scopes is prohibited, i.e. $S_1 \cap S_2 \neq \emptyset \Rightarrow (S_1 \subseteq S_2) \vee (S_2 \subseteq S_1)$.

Subscribers do not need to specify detection scopes by individual publisher identities. Instead, they may use a *high-level abstraction* to indicate these publishers indirectly. In this thesis, I support regional abstractions, but other abstractions (e.g. type-based, energy-based, reference-based, etc.) may equally be implemented (see [SFCB04]). With regional abstractions, subscribers may specify detection scopes by closed spatial regions. When a spatial region r is specified, then any publisher that falls within the region is considered to be a member of the detection scope. The region r may be specified using location coordinates or a location name that is meaningful to a location service. Chapter 4 discusses a Pub/Sub protocol that supports regions defined by absolute location coordinates.

3.4.2.1 Placement Policies

The problem as to how SFs are distributed to cover their detection scopes relates to the EDT. If one considers the EDT upside-down (such that the subscriber is at the root and the publishers are at the leaves), then every SF is placed on a branch whose leaves strictly cover the set of publishers that fall within its detection scope. Figure 3.6 shows an example, where an SF is replicated and placed on the EDT to cover three distinct detection scopes: x, u, and v. The arrows indicate the event forwarding paths (on the EDT), the P indicates the publisher-hosting EB, and the S indicates the subscriber-hosting EB.

Of course, knowledge of detection scopes must be taken into account when constructing the EDT, otherwise this SF placement policy may not be applicable. When a detection scope relates to a single publisher, the SF may be shifted down as far as the publisher-hosting EB. This results in source-side filtering, where events are filtered with zero messaging cost. Source-side filtering also ensures totally ordered events, as only one source is involved. If subscriptions happen to overlap, SFs may be shared[1]. In this case an event that passes through one SF may be delivered to multiple subscribers, and the overall computation is reduced.

[1] This may need some coordination functionality at the event service

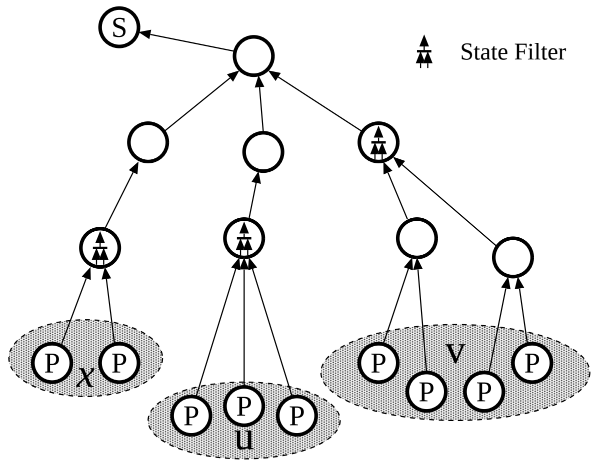


Figure 3.6: SF placement on the EDT

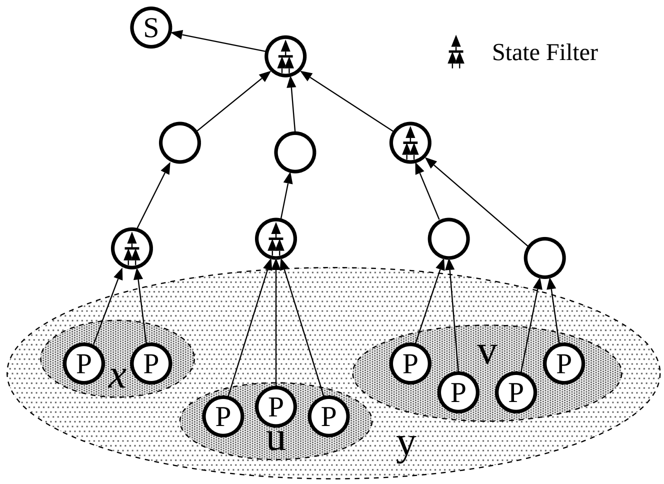


Figure 3.7: Nested Scoping

3.4.2.2 Nested Scoping

In some cases, the monitored phenomenon (or condition) can be detected by examining data from only one or a few sensors that are included in a single detection scope. Recalling an earlier example, the "night time" condition can be accurately detected by examining data from only a single outdoor light sensor; yet data from multiple sensors is also highly correlated and redundant for this condition. Another example is the detection of the "high temperature" condition in a room where redundant temperature sensors are deployed. In fact, these cases almost always appear when redundant sensors are deployed in an environment. *Nested scopes* can be used to deal with these cases effectively and efficiently.

When using nested scopes, the subscriber includes all of the smaller detection scopes (e.g. x, u, and v in Figure 3.7), as well as the larger detection scope (e.g. y in Figure 3.7) in the detection scopes set. SFs that correspond to the small detection scopes can capture the same condition independently. This provides reliable detection, such that if sensors belonging to one scope happen to fail, sensors in another scope can still capture the same condition. These SFs can also be more effectively pushed towards the publishers to attain earlier event filtering.

The larger detection scope (scope y) describes a larger detection area, which the condition is likely to span (i.e. sensors in this area produce correlated and redundant data about the condition). This larger detection scope filters (redundant) events that emerge from nested detection scopes. The combination is effective because the condition is independently monitored by multiple SFs (related to the x, u, and v detection scopes), and is efficient because events are processed close to their source, and further processed at a higher SF that eliminates redundant events across multiple nested detection scopes.

3.4.3 Fault-Tolerance

Failures are frequent in RSNs. The unreliable nature of wireless communications and primitiveness of sensor devices (see Section 3.1.1) are common sources of failure. Such failures affect the operation of system protocols, and demand robust solutions. I discuss these failures under two categories: *link failures* and *node failures*.

Link Failures. The unreliable nature of wireless communication often results in packet losses, and link-layer disconnections. For example, events at the Pub/Sub layer may be lost during event routing. This can significantly impact the system reliability as events often deliver unique and important information to the subscribers. This concern is amplified with SFs, where fewer, more informative, and unique events are delivered to the subscribers.

Link failures are most efficiently detected and resolved at the lower communication layers. MAC protocols can ensure reliable delivery on hop-level basis, and network layer or Pub/Sub protocols can ensure reliable delivery at an end-to-end level. Link failure detection at the Pub/Sub layer results in formation of new event dissemination links that repair the existing EDT. This may result in SF misses, which are discussed later in Section 3.4.3.1.

Hop-by-hop and end-to-end reliable delivery operations may assert high communication costs when link failures are frequent. This cost, however, is justified by the assurance that the data (event) is eventually delivered. Alternatively, one may rely on the correlation and redudancy of sensor data to opportunistically increase chances of data delivery to the user. This option results in (a) high communication costs even when link failures are rare, and (b) no assurance of eventual data delivery. If one decides to select this option, either due to design simplicity or low operational overhead, then SFs could be used to control the degree of data redundancy.

Node Failures. Sensor devices are subject to frequent failures, either due to environmental conditions or loss of resources (e.g. power depletion). These failures can affect the Pub/Sub protocol if EBs that are part of the EDT happen to fail; this may also result in SF losses, if SFs were hosted at these failing EBs. Pub/Sub protocols can have persistent storage of EB data to resume operation and repair the EDT using an alternative EB. SFs, however, may not

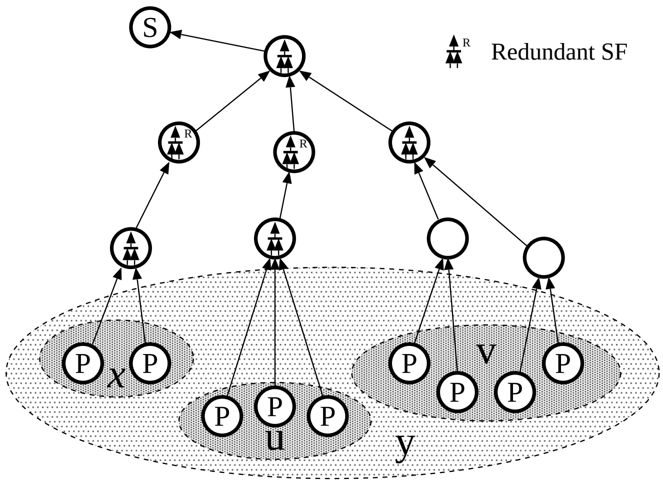


Figure 3.8: Redundant SFs

be protected as part of this persistent storage; thus I propose an alternative and independent solution (for SFs) as follows.

3.4.3.1 Redundant SFs

In this section, I propose the operation of redundant SFs that can minimize the impact of link and node failure on SF operations. This approach increases computation but does not affect the communication cost — recall that computation is a significantly cheaper resource in WSNs than the communication resource [PK00]. Redundant placement of SFs increases the chance of event filtering when a link or node fails and the EDT is repaired locally. This mechanism presents a trade-off between *computation cost* and *SF reliability*.

The redundant SF approach exploits the fact that events from a detection scope may be subject to multiple independent SF replicas without affecting the end result. A formal proof for this is presented in Appendix A. Thus, multiple SFs may be placed on nodes that are part of the same EDT branch without affecting the end result (see Figure 3.8). The only restriction here is that the SFs' status bits must be synchronized prior to operation. This can be ensured if placements are performed prior to event delivery (with initial status bits reflecting the *null* state). Note that once an SF is removed from an EDT, it can not resume operation at any time in the future. Since a transactional status bit synchronization process can introduce substantial complexity and overhead, it is best to resort to a soft-state subscription model where SFs are renewed periodically as a result of subscription refreshments.

3.5 Evaluation

In this section I evaluate the proposed SFs for expressiveness and effective event filtering. These evaluations are with respect to the motivated application scenarios at the beginning, and include use-cases that might emerge in those systems. At first, I evaluate the expressiveness of SFs by discussing a few conditions that are easily described and captured using SFs, but are found to be tedious and complex when described in a content-based subscription model. Following this, I will examine the performance of SFs, using real sensor data, and study the impact of SFs on events, communication costs, and subscribers' experience.

3.5.1 Expressiveness

The expressiveness of a subscription model is determined by its ease of use in describing conditions in sensor systems, and the set of expressible interests in the language. In the following two sections, I highlight example use-cases for each of the motivated application scenarios, and discuss the use of SFs against the content-based subscription model.

3.5.1.1 Detecting Hazardous Conditions in Mines

RSNs may be deployed in underground mines to detect and monitor hazardous conditions, as described in Section 3.1.2. The following describes a condition that needs to be monitored when accidents occur in mines. I first describe the condition of interest and then attempt to efficiently capture it using the described subscription models.

C-to-CO Reaction.[1] The presence of methane gas (above 500ppm) operates as a catalyst for the transformation of carbon to carbon monoxide (a toxic gas) when the temperature is above 20 °C. Rescuers need to know if this reaction is present as it may affect the available rescue time.

I assume the presence of the following sensor devices in the environment.

Temperature Sensor publishes events containing a single attribute name *temp* and a single attribute value v that indicates the measured temperature value (in degrees Celsius).

Methane Sensor publishes events indicating the level of methane gas in the environment (attribute name: *methane*, attribute value: v - sampled methane concentration (in ppm)).

As one may notice, carbon monoxide sensors are not used. This is because carbon monoxide sensing hardware is expensive (i.e. low-power carbon monoxide sensors do not exist). Thus, one must use the knowledge of methane gas and temperature to deduce information about the C-to-CO reactions. This condition is difficult to capture reliably, because it involves two distinct event types and therefore requires data fusion. Data fusion is often not supported on resource-constrained platforms because it increases the code complexity, slows the data filtering process, and requires dynamic memory allocation. Nonetheless, the following SF subscription can be used to capture this condition efficiently.

$$s = \{methane > 500, (methane < 500) \,||\, (temp < 20), \mathbb{U}\} \tag{3.18}$$

[1] In reality, this reaction may be monitored in different ways. Here, I present an artificial version for the sake of discussion.

The entrance predicate, $methane > 500$, detects the presence of methane gas that can accelerate the C-to-CO reaction. It should ideally be expressed as $(methane > 500)\&\&(temp > 20)$, but since SFs can neither store events nor fuse events, only one of the two conditions may be specified. In this example, the first expression is used as one expects it to be less frequently matched by an event. Events, published by the methane sensor, may satisfy this predicate. The exit predicate, $(methane < 500) \,||\, (temp < 20)$, detects the disappearance of methane gas or the low temperature value that restricts C-to-CO reactions. The predicate may be satisfied by events from the methane sensor or the temperature sensor. The detection scopes set, $\mathbb{U}$, describes a single universal set that covers all publishers (sensor devices) in the mine. A finer detection scope may be used if the accident is confined to a limited area.

The content-based subscription model requires two filters to support the same condition.

$$s_1 = methane > 500 \tag{3.19}$$

$$s_2 = (methane < 500) \,||\, (temp < 20) \tag{3.20}$$

The first detects the presence of methane gas and the second detects the absence of methane or low temperature value, much like the discussed SF predicates.

Discussion. The SF and the content-based filter expressions are very similar. The described condition is captured using a single SF, or two content-based filters. Results, however, differ widely.

The independent, but complementary semantics of the two content-based filters means that almost all published events (by the methane sensor) are delivered to the subscriber, $\forall e \in \mathbb{E}.\, e = ((n_1, v_1)).\; n_1 = methane \wedge v_1 \neq 500 \Rightarrow e \sqsubseteq s_1 \vee e \sqsubseteq s_2$. This renders the content-based subscription model ineffective for events published by the methane sensor, as almost all events match either of the two content-based filters. The SF subscription model, however, delivers an event only when the presence of methane gas is detected, and suppresses all subsequent methane gas readings until its concentration is lowered or the temperature value falls below 20 °C.

Both subscription models operate poorly when the temperature value is less than 20 °C. The content-based subscription model performs worse as s_2 is continuously satisfied and events are delivered to the subscriber. The SF subscription model cycles through states when methane gas is present and the temperature is below 20 °C. This problem arises because neither subscription model can fuse data. Chapter 5 presents SMCs that can overcome this and other limitations.

3.5.1.2 Regulating Office Temperature

Office environments may be equipped with RSNs to monitor working environments and the air quality (see Section 3.1.2). Sensors may be used to capture user interests and drive actions automatically. One such example may be the regulation of heat in office environments during the summer season. Let's describe our condition of interest as follows.

Automated Temperature Regulation. Users may specify desired temperature ranges for their office environments. A low temperature value T_l defines the lowest acceptable temperature value, while T_h defines the highest. A preferred temperature value T_p that falls between the highest and lowest values, $T_l < T_p < T_h$, may also be specified - otherwise $T_p = avg(T_l, T_h)$. When the office temperature t rises above T_h, the air conditioning unit must be activated and the office air is cooled until t reaches T_p. The temperature t must reach $T_p < T_h$, otherwise the two counter forces (environmental heat and the air conditioner) would cause the temperature value t to oscillate about the T_h value.

I assume the presence of the following sensor devices in the environment.

Temperature Sensor publishes events containing a single attribute name *temp* and a single attribute value v indicating the measured temperature value (in degrees Celsius).

The following SF subscription would capture this condition as desired.

$$s = \{temp > T_h, temp \leq T_p, \{region : \text{office}_1\}\} \tag{3.21}$$

The entrance predicate, $temp > T_h$, detects undesirable (high) temperature in the office. The matched event may be delivered to an air conditioning unit to signal *start*. The exit predicate, $temp \leq T_p$, detects when the temperature is lowered to a preferred value T_p. The event that satisfies this predicate may also be delivered to the air conditioning unit to signal *stop*. The detection scopes set, $\{region : \text{office}_1\}$, describes a single detection scope that relates to the user's office (office_1).

Again, the content-based subscription model requires two filters to support the same condition.

$$s_1 = temp > T_h \tag{3.22}$$

$$s_2 = temp \leq T_p \tag{3.23}$$

Discussion. The SF subscription model results in the delivery of event pairs to the subscriber (e.g. air conditioning unit). These events relate to the start and the end of the desired condition, making automatic reactions easy and efficient. Although the content-based subscription model captures the condition, the received set of events still need to be processed. Repetitive events after the detection of rising temperature need to be suppressed, and redundant events with temperature values below T_p should be ignored. In summary, the content-based subscription model is only effective in respect of a small set of events, whose temperature values fall in-between the preferred and high threshold value, $T_p < temp \leq T_h$. These events are filtered by the content-based subscription, otherwise all events are passed through the filter.

3.5.2 Event Filtering

In this section, I investigate the performance of SFs with respect to condition capturing accuracy and incurred communication costs. Reducing communications in RSNs is desirable, as wireless communications are considered the main source of power consumption in these networks. In order to compare the effectiveness of SFs against the content-based subscription model, I describe a simple condition that can be equivalently described in both subscription models. I use the following two metrics for evaluation.

Capture Accuracy Capture accuracy is the level of knowledge that is conveyed to the subscriber about a condition's start and ending.

Messaging Efficiency The number of events that are suppressed by a filter. This has a strong impact on the resulting communication costs.

The next section describes my simulation environment, in which I have implemented and examined subscription models with respect to the outlined evaluation metrics.

3.5.2.1 Simulation Environment

I adopted the Scalable Wireless Ad hoc Network Simulator (SWANS), built on top of Java in Simulation Time (JiST), as my base simulation environment. A multi-hop WSN was modeled as my test platform, over which the subscription models were implemented for evaluation.

The radio model was configured according to the CC1000 radio parameters [CC1] that is in use on the BTnode platform [BT], on Mica motes [MIC], and several other platforms. The MAC layer implements Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) by a sequence of Request to Send (RTS)-Clear to Send (CTS)-Data-Acknowledgment (ACK) messages. The network layer addresses nodes according to their location in the simulation environment.

Two Pub/Sub protocols were implemented to support the SF and the Interval-based Event Filter (IEF) subscription models, respectively. They share significant code, as they use a common EDT model for event dissemination. I implemented the Directed Diffusion [IGE00; IGE+03] protocol as my underlying Pub/Sub scheme. Directed Diffusion is designed to operate over multi-hop WSNs, supports a Pub/Sub-like interface, and constrains subscriptions using *rectangles* that are similar to detection scopes for SFs. The following two subscription models where implemented over Directed Diffusion for evaluation.

Interval-based Event Filters IEFs are content-based filters, whose matched events have predefined *validity intervals* (proposed by DSWARE [LSS03]) to represent lasting conditions. Validity intervals are assigned according to the underlying environment and the known characteristics of the observing phenomenon. An IEF filters all events that follow a matched event during its predefined validity interval, T, i.e. if $e_1 \sqsubseteq s$ then $\forall e_2 \in \mathbb{E}.\, e_2^t - e_1^t \leq T \Rightarrow F(e) = \mathit{false}$. This was implemented by accompanying every IEF with a timer, that starts after an event matches the subscription and filters all subsequent events until the matched event's validity interval times out.

State Filters SFs were implemented as described in Section 3.3. SF predicates were stored as two independent filters, and a single status bit (stored in the memory) was used to determine which filter needs to be applied over the received event. The detection scopes were supported through the notion of regions, and mapped to the underlying Pub/Sub (subscription) rectangles.

3.5.2.2 Experimental Setup

A two-dimensional outdoor environment was simulated, comprising sixteen equisize regions. Each region was allocated one to three temperature sensors that monitored the local region's temperature. Temperature sensor devices were programmed to report regional temperature values (in the form of events) every three minutes. Published events contained a single attribute name *temp* and a single attribute value v indicating the measured temperature value (in degrees Celsius). The number of temperature sensors totalled 35 devices (an average of just over 2 sensors per region). These sensors were supported by 55 additional wireless nodes in the simulation environment to ensure wireless network connectivity.

Real sensor data, collected from the Cambridge Weather Station, were used to model temperature in each simulated outdoor region. When temperature sensors sampled their regions, these values were provided by the underlying simulation engine (I assumed a uniform temperature distribution across each simulated outdoor region).

Ten distributed subscribers, $S = \{S_1, S_2, \ldots, S_{10}\}$, were simulated in the environment, with similar (but non-identical) interests over temperature changes. Subscribers wished to be notified when a certain threshold temperature value $T_{x \in S}$ had been exceeded in their chosen regions, i.e. the subscription predicate was $temp > T_x$. All threshold values were in the vicinity of 10 °C, but different for each subscriber, $\forall x \in S,\ |T_x - 10| < \epsilon.\ \forall y \in S\ \ x \neq y \Rightarrow T_x \neq T_y$.

After analysing the environment and the condition of interest I decided to set the event validity interval to *thirty minutes* in the IEF model. Subscribers' regions of interest were defined as detection scopes in the SF model and subscription rectangles in the IEF model. Nested detection scopes were also used in the SF model to place SFs over individual publishers in each region.

3.5.2.3 Performance Results

Simulation results, excluding sensor failures and relating to thirty hours of real data, are shown in Table 3.1. In Table 3.1, the publisher-scoped filters refer to the SFs/IEFs placed on individual temperature sensors (also referred to as the source-side filters). Region-scoped filters refer to SF detection scopes where SFs were imposed over events emerging from individual outdoor regions.

From a total of 21000 published events in the system, only 6600 events related to the subscribers' regions of interest. A condition capturing resolution of *three minutes* (in the case of SFs) against the *thirty minutes* interval period of the IEF[1] demonstrated the increased capturing accuracy of SFs against IEFs. With the IEF subscription model, a trade-off is realised between efficiency and accuracy of condition capture, such that a larger validity interval increases efficiency but also compromises the accuracy by an even larger value.

Only 22 events were passed through the publisher-scoped SFs (from a total of 6600 events), contributing to a filter ratio of 0.9966. This figure compares to the 0.9672, that corresponds to

[1] the largest observed inaccuracy with IEFs in this experiment was 18 minutes.

Statistics	SF	IEF
publishers	35	35
subscribers	10	10
subscriptions	10	10
covered publishers	11	11
publisher-scoped filters	27	27
region-scoped filters	10	N/A
published events	21000	21000
covered events	6600	6600
publisher-scoped filter's filter ratio	0.9966 (22#)	0.9672 (216#)
shared events before duplicate suppressions	16	192
duplicates suppressed	14	0
shared events after duplicate suppressions	6	192
delivered events	20	620
capturing resolution	3mins	30mins

Table 3.1: Simulation Results

the publisher-scoped IEFs. With higher source-side filtering and delivery of 600 fewer events to (the subscribers), SFs demonstrate high messaging efficiency in comparison to IEFs.

Table 3.1 shows that out of the 22 events (which passed through the publisher-scoped filters), 14 events were further filtered at the region-scoped SFs. These 14 events were redundant. The remaining 8 events were those which were delivered to the ten distributed subscribers in the system — 20 events were disseminated to the subscribers in total (i.e. some events were delivered to multiple subscribers). In comparison, IEFs had a lower source-side filtering and (without filtering redundant events over each region) delivered a total of 620 events to the subscribers.

In this experiment, SF failure was not considered as it does not lead to erronous or missed events, but simply overwhelms the user with all the data that is generated by the sensors. If SF replication is used (as discussed in Section 3.4.3.1), then results corresponding to the number of published and delivered events remain the same and the number of filters increases by the degree of replication. Finally, it should be noted that only a single set of data (collected from the Cambridge Weather Station) was used in this experiment, and although temperature data from other sensor platforms is expected to demonstrate similar trends, the discussed results are confined to this unique experiment.

3.6 Related Work

Content-based Pub/Sub is more expressive than topic-based Pub/Sub, and can result in higher communication efficiency (due to more effective event filtering) in sensor systems. An extensive comparison between the content-based and the introduced SF subscription models is presented throughout this chapter. I therefore extend my comparison, in this section, to two other classes of related work: *CE frameworks* and *Database-oriented approaches*.

Composite Event Frameworks. CE frameworks (discussed in Section 2.4.2) are related to this work, as they can support similar features through complex event patterns and operators. Although they are designed around heavy-weight components where processing and memory resources are not a concern, e.g. active DBMSs and Electronic Application Integration (EAI) brokers, the basic principle can be compared in the context of a sensor system. The proposed SFs can be closely implemented, using the *sequence* operator [CM94], in CE frameworks. Essentially two event types, A and B, are defined to reflect the events that match the (entrance and exit) predicates of an SF, respectively. Two subscriptions of the form $B; A$ (B followed by A) and $A; B$ (A followed by B) are also expressed (with the *recent* consumption policy) to capture the condition initiation and termination events, respectively. This CE-based implementation can be compared against the state-based design (of SFs).

The preservability feature of SFs means that the filter preserves the input event type at the output. This contrasts with the CE-based implementation, where output event types are different to the input event types. In terms of reliable detection, the proposed CE-based implementation cannot detect the first condition initiation — alternative expressions can be more complex and result in higher operational complexity. Finally, the shared context of entrance and exit predicates in an SF can yield different results than the two independent contexts for the CE expressions stated earlier. I believe SFs provide a more natural way of expressing lasting conditions than multiple independent CE expressions.

Database-oriented Approaches. The Database is one of the earliest examples of high-level abstractions for sensor network programming. COUGAR [BGS00] and TinyDB [MFHH03] fall within this category. They allow users to issue queries in a declarative SQL-like language. To achieve energy efficiency, COUGAR pushes selection operations to the sensor nodes so that they can reduce the amount of data to be collected. For the same objective, TinyDB focuses on the acquisitional issues: where, when and how often to sample and deliver data. Although these support more expressive computations than are achievable by SFs, they fail to support context-based data processing. Since user queries can not change (independently) according to the output data, cf. SFs, the application programmer needs to express its context-based data processing requirements in a single non-trivial and complex query expression. Although TinyDB's data storage points can store data, much as how context is maintained in SF's status bit, it is unclear if this can be used to implement context-based data processing (as in SFs) within these frameworks.

3.7 Summary

In this chapter, I presented SFs [TB07c] that extend content-based filters with capabilities to capture lasting conditions, and to filter correlated and redundant events that emerge from one or more sensor devices. These contributions were motivated by the continuous nature of the observed environment and the realisation that many interesting phenomena have temporal continuity. Aside from expressiveness, the proposed SFs also reduce significant communications overhead by filtering those events that deliver correlated and redundant information about the monitored conditions. Detection scopes were also introduced to allow fine-grained specification of data correlation and redundancy boundaries (about a condition) over many sensor devices. These contributions have come with negligible state storage, and high compatibility. SFs can subsume existing content-based filters, and even substitute them in relevant Pub/Sub protocols that are designed for sensor systems.

Chapter 4

Quad-PubSub

In this chapter I present Quad-PubSub (QPS) [TB07a], a topic- and location-based Pub/Sub protocol for location-aware Wireless Sensor Networks (WSNs). QPS is a distributed Pub/Sub protocol that supports its Event Clients (ECs) through a unified Pub/Sub interface, and provides complete time and location decoupling (Section 2.3.1.1). The interaction between the publishers and the subscribers is defined by events that have topic and location attributes. Since the majority of sensor network applications can benefit from location coordinates defined in 2-D space, QPS focuses on 2-D geographical space and partitions it into hierarchical quadrants which form Quad-Trees (QTs) - hence the name Quad-PubSub.

QPS uses Event Broker (EB) functionality at different nodes to disseminate events globally. It selects a limited number of nodes (EBs), using a localized subscription resolving algorithm, and uses these to disseminate events corresponding to certain topic and location values across the network. A dedicated layer in QPS provides resource-awareness: it ensures that the selected nodes have sufficient resources to perform their tasks and actively relieves them from their duties when their resources become depleted.

Key to the design of QPS is a layered architecture. This allows for the transparent operation of location-based routing protocols that satisfy user-defined Quality of Service (QoS) requirements (e.g. prolonged network lifetime, resource-aware routing, timely data delivery, near-optimal routing, etc). I motivated this design decision by observing [SR02] how dynamic and probabilistic routing can extend network lifetime when compared to data dissemination protocols such as Directed Diffusion [IGE00; IGE+03] that use fixed optimal paths. To allow such dynamic routing, the event dissemination and the event routing operations must be separated. This separation, however, can prevent some performance optimizations such as the formation of shared event forwarding paths that are key to scalability. QPS exploits location-awareness to achieve mutual separation of operations and path sharing. In its design, an ϵ factor is used to manipulate a trade-off between the two.

In this chapter I initially discuss the location-aware WSNs that form the basis of this work. A discussion of cross-layer data-dissemination protocols, and Pub/Sub protocols in particular, is presented in Section 4.2. The discussion motivates QPS, which is formally presented in Section 4.3. In Section 4.4 I evaluate the performance and contributions of the proposed protocol.

I follow this with a review of the related work in Section 4.5 and a summary of the chapter in Section 4.6.

4.1 Location-aware WSNs

The notion of *location* is often used to describe the geographical relation of objects or entities in a system. In location-aware WSNs, devices are augmented with a notion of location that describes their geographical position within the network (and the environment). A coordinate system is defined, and locations are described in absolute terms using the coordinate system. The notion of location enhances the meaning of data, which is observed or captured by the sensing devices, and aids the subsequent operations (aggregation or fusion) that are performed over this data. It also provides an added dimension for indexing data, such that spatial constraints can be imposed by end-users. As we shall describe shortly, location can also benefit some system operations such as data routing.

The problem of identifying nodes' spatial coordinates in some coordinate system is referred to as *localization*. Extensive research has been done on localization; a general survey can be found in [HB01]. Localization approaches mostly differ in their assumptions about the network deployment and hardware capabilities. Distributed localization methods, which do not require centralized computations, can be divided into *range-based* and *range-free* methods. The former uses distance or angle estimations in calculating locations, while the latter just uses the received message contents from nodes that know their locations (called *anchors*). Time of arrival [HWLC97; Dan97], received signal strength [PI03; BP00], Time Difference of Arrival (TDOA) [SHS01], and Angle of Arrival (AOA) [NN03a] have been used for range-based localizations, and single-hop (e.g. Centroid method [BHE00]) and multi-hop (e.g. DV-HOP [NN03b]) beaconing (from anchors) have been used in range-free localizations [HHB+03]. A quantitative comparison [LR03] shows that no single algorithm performs best, and performance depends on many conditions such as the range errors, network connectivity, and anchor fraction.

A location coordinate is sufficient for describing the location of a sensing device, but not for the data that describes a spatially continuous environment. The observed data often has a meaning beyond a single location coordinate (*point*), and could reflect a *region* that encompasses the point of sensing. This region mirrors a *sensing coverage* that depends on the monitoring context and environmental physiography, the latter of which is often variable and difficult to evaluate. As a result, a conservative approach is often adopted where the location of data is strongly tied to the location (point) of sensing, and increased coverage is pursued by increased spatial sampling (i.e. higher WSN density). The following two sections describe how the location information affects system operations and impacts the range of WSN applications.

4.1.1 Location-based Routing

Event dissemination in large-scale ad hoc networks is difficult. The problem demands a global search that identifies the matching publishers and subscribers, and needs some state storage (at the appropriate nodes) to intermediate the connection. Additional costs may be incurred if topology-based routing is used in location-free WSNs. Where location information is available, however, routing performance can be improved.

Location-based routing protocols require nodes to know their own location, the location of their one-hop neighbors, and the location of the destination. These protocols conserve memory and bandwidth since discovery floods and state propagation are not required beyond a single hop; thus they perform better than topology-based routing protocols.

Location-based routing protocols can be divided into three classes [MWH01]: *restricted flooding*, *geographic forwarding*, and *hierarchical routing*. The first class of protocols set up a region (using the location information of the source and the destination), and then flood the region with the packet that is intended for the destination. Examples of this class are DREAM [BCSW98] and LAR [KV00]. Although reliable and simple in operation, these protocols consume much bandwidth and can result in serious network congestion, thus they are most commonly used for route discovery rather than route forwarding.

The second class of protocols are more efficient as they only forward packets to one neighbor (lying in the general direction of the destination) at a time. For a fixed transmission range, MFR [TK84] (also known as *greedy forwarding*) is an efficient protocol that sends a packet to the neighbor that is closest to the destination. When the transmission range is adjustable other strategies have been shown [HL86] to perform better. Greedy Perimeter Stateless Routing (GPSR) [KK00] is a popular protocol, that uses a combination of greedy forwarding and planar graph traversal to overcome the local maxima (*hole*) problem. WSN protocols, such as GAF [XHE01] and GEAR [YGE01], extend these strategies with energy awareness.

Finally, the third class of protocols (e.g. Terminodes [BBC+01] and Grid [GRI]) use a combination of strategies for different stages of the forwarding. For example, proactive distance vector routing is used at the local level and geographic forwarding is used at the global level. For more details please consider the [MWH01] survey.

4.1.2 Potential Applications

The use of location information strengthens two classes of applications that are otherwise constrained or infeasible. The first class relates to the set of applications where *location aids data*. In this class, it is imperative to tag data with location to enable meaningful processing, data correlation, and subsequent actuation. This class of applications often considers the environment's physiography to be unique and expresses a homogeneous interest across the sensor network. For example, in target tracking, identification of the target is the only interest across the sensor network.

The second class is where *location aids query (or task)*. In this class, applications do not consider the environment's physiography to be unique, and exploit location-awareness to finely express their interests with respect to various parts of the sensing environment. For example, in a smart transportation environment, different speed limits may be imposed on different monitoring roads and highways (identified by their location). The majority of location-based applications, however, relate to both as they utilize features from both classes. The design and implementation concerns that follow each class though, are different. Below, I have highlighted two examples that reflect each class individually.

Forest Fire Detection WSNs can be deployed to detect forest fires in their early stages, or monitor their progress thereafter [YWM05; Hef07]. A large number of sensing devices is deployed, each of which monitors a certain context such as the temperature or humidity of its local environment. Sensor readings are then reported to a base station if alarming values or patterns are detected. The base station examines the data, received from multiple sensors, and determines the likelihood of a real fire. This analysis and the subsequent action strongly depends on the location of observations: data location is needed to accurately aggregate data and to direct dispatched teams to the right location in the forest. In this application, location aids data.

Crop Management WSNs can be used to monitor climatic conditions, weather and crop data in agricultural fields [WCS+07; Bag05; HFH+05]. These networks may span multiple indoor/outdoor environments with different fruits and vegetables. Farmers monitor for different conditions or diseases at each crop field, and aim to minimise the use of chemical treatments in each field. These conditions may be monitored through observation of the humidity, temperature, and moisture on the leaves. Location information is vital for monitoring the appropriate condition at each field. For example, potato fields are monitored for phytophtora (a fungal disease) and, if necessary, treated with fungicide in the affected areas, while in rice fields, similar data is used to predict rice blast (a rice disease).

4.2 Cross-layer Pub/Sub Protocols

Data routing and data dissemination are different communication paradigms. The former supports a one-to-one communication model, where a packet (or a message) is routed from a *source* to a *destination*, while the latter describes a many-to-many communication model, where data is disseminated from many information producers to many information consumers. In Pub/Sub, the relationship between the information producers (*publishers*) and consumers (*subscribers*) is determined by the structure and contents of the data (*events*) itself, and thus Pub/Sub is also a *data-centric* communication paradigm. Traditionally, data dissemination protocols (e.g. multicast and Pub/Sub) focused on end-to-end level interactions and used primitive communication models, such as that provided by the data routing protocols, for low-level node-to-node level interactions. The close relationship between the two communication paradigms, however, has encouraged many researchers to explore cross-layer designs, where data dissemination and data routing are performed as part of a unified protocol.

Cross-layering is the practice of accessing other layers' protocol stacks or, at its extreme, the practice of unifying their implementations into one larger (more complex) protocol [CCMT04]. In the case of data dissemination and data routing, cross-layering enables the data dissemination service to examine the routing tables, which are maintained by the data routing protocol for node-to-node level packet forwarding. Cross-layer implementations are often more compact and efficient. For example, common concerns (such as node failures and topological changes) can be jointly addressed rather than separately. Disadvantages mainly relate to the reduced flexibility in the system architecture, tight-coupling, and mutual dependencies in operations.

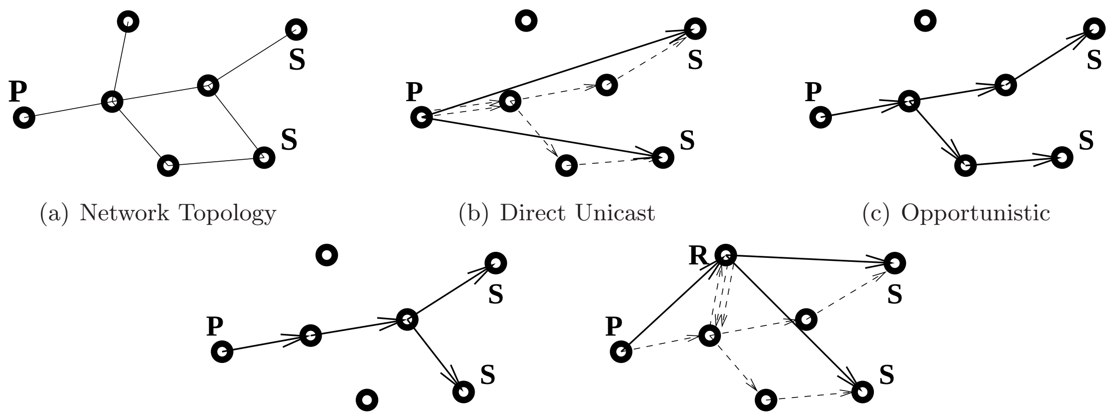
One optimization that is commonly pursued in cross-layer data dissemination protocols is the formation of *shared data (event) forwarding paths*. Shared paths are common data routes that multiple event forwarding paths (for different subscribers) use in the network. Forwarding a single event along the shared path can benefit multiple event subscribers. These routes can be identified in cross-layer designs, where knowledge about the overlapping routes and destinations (subscribers) are apparent to the protocol.

In decentralized Pub/Sub protocols, shared paths are formed at the subscription resolution stage. Existing event forwarding paths are detected and shared paths are formed when subscriber interests happen to match. Shared paths offer numerous advantages, two most notable of which are increased communication savings, and synchronized forwarding. Communication costs are reduced when an event is shared over a communication route for multiple subscribers. The alternative (event replication and forwarding along multiple routes) has a communication cost that is at best linear to the number of subscribers. Shared paths also offer synchronized forwarding, where data for multiple subscribers is forwarded in synchronization. This narrows the time window in which different subscribers receive the data.

Although shared paths reduce communication costs, they result in *in-network state storage* and *fixed paths*, both of which (if neglected) can reduce the WSN lifetime.

In-network States Cross-layer data dissemination protocols store in-network states to guide data from publishers to subscribers. These states pose storage costs that if neglected can exceed nodal resources in the case of a large number of subscriptions. Where localized interactions are used [IGE00; IGE+03], these states are stored on a *per-hop* basis, guiding data from every node to the next until subscribers are reached. Where globally unique addresses (e.g. location-based addresses) are available, these states still need to be stored to reflect knowledge about the existing event forwarding paths.

Fixed Paths The formation of a shared path entails a merge between an existing event forwarding path and a new one. Where decentralised solutions are used, this merge happens within the network and relies on fixed event forwarding paths that can be used without further resolving a subscription. Sometimes, however, these fixed paths are not wanted. For example, [SR02] shows that dynamic routing performs better than fixed path routing, even if optimal routes are used, when prolonged network lifetime is desired. They have shown that a dynamic routing approach can extend the energy savings by 21.5% and the



(a) Network Topology (b) Direct Unicast (c) Opportunistic

(d) Greedy Sharing (e) GHT-based

Figure 4.1: A comparison of four event forwarding techniques

network lifetime by 44%, when compared to the optimal paths used in Directed Diffusion [IGE00; IGE+03]; they maintain a set of sub-optimal paths, chosen by means of a probability function, from which they select a single path randomly to deliver data.

In order to allow dynamic routing, forwarding paths must be relaxed and *path freedom* (the opposite of fixed paths) should be allowed. Path freedom allows data to take arbitrary routes from sources to destinations. The semantics of this freedom can be subject to resource-awareness, timeliness, near-optimal routing, or other user-specified QoS policies. Examples of routing protocols that can benefit from this path freedom are [SR02; GTS06; TGS06; SZHK04; HHKV01; KRKI04]. The next section explores the relationship between shared paths and path freedom in WSNs with reference to an example. In Section 4.3 I present a Pub/Sub protocol that supports a combination of shared paths and path freedom. This combination is achieved by controlling the level of in-network state storage and the fixed paths that are formed in the network.

4.2.1 Path sharing vs Path freedom

Let's consider Figure 4.1 as a case study, where event forwarding paths for two subscribers and a single publisher are shown; the solid arrows are Pub/Sub links, and the dashed arrows are shortest forwarding routes. The subscribers, denoted by `S`, have a common data interest. This interest is fulfilled by the single publisher, denoted by `P`. If we abstract the link-layer functionality, the communication cost of disseminating events from the publisher `P` to the subscribers `S` can be examined by the number of communication hops that is taken by an event to reach the subscribers (Figure 4.1(a) shows the network topology). In this regard, I examine four known techniques for setting up the event forwarding paths from `P` to `S`. These techniques are discussed with emphasis on the achieved *shared event forwarding paths*, support for *path freedom*, and the *overall communication* that is induced for delivering an event notification.

Direct Unicast When resolving subscriptions, direct unicast links can be set up at the publisher P that point to each subscriber, see Figure 4.1(b). In this, maximum support for path freedom is achieved; event notifications can take any route from the publisher to the subscribers, guided by the routing protocol. No paths are shared, as events are replicated at the publisher and forwarded independently towards the subscribers. The lowest notification delivery cost is 6 hops, and a total of 2 states are stored in the network (at P).

Opportunistic Sharing When a cross-layer data dissemination protocol is used in location-aware WSNs, opportunistic shared paths [IEGH02] can be formed as in Figure 4.1(c). States are stored at every intermediate hop, and used to merge overlapping lowest latency paths. The event delivery cost is now reduced to 5 hops - interestingly the savings are made at the publisher's region, which help to extend the lifetime of the publisher and its surrounding nodes. Path freedom in this setup has been diminished, as events must propagate through the selected set of intermediate nodes that reflect a fixed path. These fixed paths direct events on a hop-level basis from the publisher to the subscribers. A total of 5 states are now stored in the network.

Greedy Sharing At the expense of higher communication costs (e.g. network broadcast), more effective shared paths [IEGH02] can be formed, see Figure 4.1(d). Greedy sharing involves a search for existing event forwarding paths, of which the most suitable path is selected for optimal results. The notification delivery cost is now reduced to 4 hops, but subscription resolutions now have a communication cost that is proportional to that of network broadcast. Like opportunistic sharing, path freedom is lost and fixed paths are formed that guide events on a hop-level basis; a total of 4 states are stored in the network.

GHT-based Using Geographic Hash Table (GHT) [RKY+02] (a Distributed Hash Table (DHT)-like protocol for location-aware WSNs), one can construct shared paths while also attaining some path freedom. In this setup, subscriptions are joined and shared paths are set up at some defined rendezvous nodes, denoted by R in Figure 4.1(e). Support for path freedom exceeds the previous two approaches, but is still lower than the direct unicast approach as events still need to route through the rendezvous node. A total of 3 states are stored in the network (one state at the P, and two states at the R). This approach has a number of disadvantages that outweigh its benefits. Firstly, the rendezvous nodes are selected statically (predefined) according to some location coordinates as opposed to their level of resources. Secondly, they are subject to high event handling and dissemination costs that can deplete their resources (e.g. battery power) and lead to their failure. Thirdly, the resultant event dissemination paths (even in the case of shortest-distance routing) are often quite expensive; the cost of event dissemination, in Figure 4.1(e), is 8 hops.

4.3 Quad-PubSub

QPS is a topic-based Pub/Sub protocol that supports *shared paths* as well as *path freedom* in location-aware WSNs. It separates event dissemination from event routing, and implements the former as a functional layer. An underlying routing protocol is required, which may be customized according to the characteristics of the deployed sensor network, and may satisfy user-defined QoS requirements. The amount of path freedom that QPS provides for forwarding events from publishers to subscribers determines how much flexibility the routing layer has for meeting its QoS requirements. This amount is tunable by an ϵ factor that users provide as part of their subscriptions.

QPS uses the location-awareness property of the network to build an overlay of logical EBs (in the form of QTs), over which it constructs Event Dissemination Trees (EDTs) to interconnect the publishers and the subscribers. The ϵ factor manipulates a trade-off between shared paths and path freedom in the construction of EDTs at this overlay. A localized subscription resolving algorithm is used (at each of the logical EBs on the overlay) to form event forwarding paths, which constitute the EDTs, according to the user-specified ϵ factors and in a decentralized manner.

In the next section I outline my event model that defines how data is represented in QPS. Section 4.3.2 outlines the system architecture, and discusses the components and operational layers of QPS. The QPS dissemination model is presented in Section 4.3.3. This model is executed by a set of routing algorithms that is outlined in Section 4.3.5. A resource-awareness model (Section 4.3.6) and a reliability model (Section 4.3.7) complement these operational features, and address changing levels of nodal resources and nodes' failure-proneness in WSNs.

4.3.1 The Event Model

In general, data in Pub/Sub is represented as *event publications* (or *events*). Event publications are manifested for routing and processing as *event publication messages* (or *event notifications*). These are asynchronous messages that are transfered from the event publishers to the event subscribers. Prior to event publication, however, publishers need to advertise their set of publishable events via *event advertisements*. This provides prior knowledge about the event publications, which QPS uses to operate more efficiently and fulfill event subscriptions. The interconnection between publishers and subscribers is drawn when subscribers express their interests via *event subscriptions*. The next three sections describe event publications, subscriptions, and advertisements, as used in QPS.

4.3.1.1 Publications

In location-aware WSNs, supported by QPS, events are assumed to have notions of *topic* and *location* assigned to them. The topic describes the type of information that is contained in the event, and the location maps the information to a certain (point-based) location coordinate within the geographical space. The use of topics leads to a structured event space $\mathbb{E}$ that is easier to manage, and more naturally suited to sensor systems where sensor hardwares and readings are often typed.

Definition 4.1 (Event Notification). *An event notification e consists of a tuple τ and belongs to the event space $\mathbb{E}$,*

$$e \in \mathbb{E}. \tag{4.1}$$

The tuple τ contains an event topic, *t_e, a* location, *l_e, and a set of* attributes,

$$\tau = (t_e, l_e, \{a_1, \cdots, a_n\}). \tag{4.2}$$

An event topic t is a member of a pre-defined set of event topics T, $t \in T = \{t_1, \cdots, t_k\}$. Event location l_e is also a member of the geographical space S, $l_e \in S$. Each attribute, a_i, is a name-value pair, *(n_i, v_i), with name n_i and value v_i. Attribute names are unique, i.e. $i \neq j \Rightarrow n_i \neq n_j$. Every event e corresponds to a unique combination of a publisher, e^p, and a timestamp, e^t, in the system, i.e. $\forall e_1, e_2 \in \mathbb{E}$ if $e_1^p = e_2^p \wedge e_1^t = e_2^t$ then $e_1 = e_2$.*

4.3.1.2 Advertisements

Event advertisements are pre-announcements that indicate what events, from the event space $\mathbb{E}$, are likely to be observed in the system. Event publishers (such as sensors) advertise their events prior to event publications. This increases QPS' knowledge about the granularity of events that may be realised about the event space, and helps to fulfill event subscriptions (described later in Section 4.3.3.2).

Definition 4.2 (Event Advertisement). *An event advertisement d consists of a tuple τ_d,*

$$d = \tau_d, \tag{4.3}$$

that contains an advertisement topic *t_d and an* advertisement region *$r_d \subseteq S$,*

$$\tau_d = (t_d, r_d). \tag{4.4}$$

The tuple describes a set of events, E_d, whose event topic match t_d and location fall within the region r_d,

$$E_d = \{e \in \mathbb{E} \mid t_e = t_d \wedge l_e \in r_d\}. \tag{4.5}$$

The set E_d describes the set of publishable events from the event space $\mathbb{E}$.

4.3.1.3 Subscriptions

Event consumers describe their event interests through subscriptions. Event subscriptions, like advertisements, govern a subset of the event space $\mathbb{E}$ that consumers hold interests over. Similarly, subscriptions have associated event topics and regions of interest.

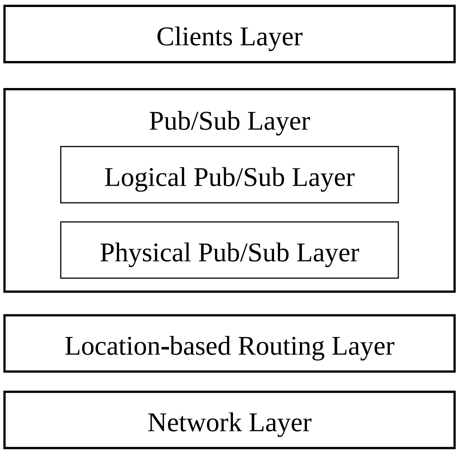


Figure 4.2: QPS Architecture

Definition 4.3 (Event Subscription)**.** *An event subscription s consists of a tuple τ_s,*

$$s = \tau_s, \tag{4.6}$$

that contains a subscription topic t_s*, a* subscription region $r_s \subseteq S$*, and a* subscription epsilon factor ϵ_s*,*

$$\tau_s = (t_s, r_s, \epsilon_s). \tag{4.7}$$

The tuple described a set of events, E_s, whose event topic match t_s and location fall within the region r_s,

$$E_s = \{e \in \mathbb{E} \mid t_e = t_s \wedge l_e \in r_s\}. \tag{4.8}$$

The subscription ϵ factor relates to the event forwarding path and is discussed later in Section 4.3.3. An event notification e may be examined against a consumer's subscription s to determine if the subscriber is interested in the event or not.

Definition 4.4 (Subscription Coverage)**.** *An event notification $e = (t_e, l_e, \{a_1, \cdots, a_n\})$ matches a subscription $s = (t_s, r_s, \epsilon_s)$,*

$$e \sqsubseteq s, \tag{4.9}$$

if and only if

$$e \in E_s. \tag{4.10}$$

The above can only hold true if and only if

$$t_e = t_s \wedge l_e \in r_s. \tag{4.11}$$

4.3.2 Architecture

The architecture of a sensor system that employs QPS is shown in Figure 4.2. Each layer builds on top of the functionality provided by the layer underneath and exports a clearly defined interface to the layer above. Apart from that, the layers are independent of each other. A layered

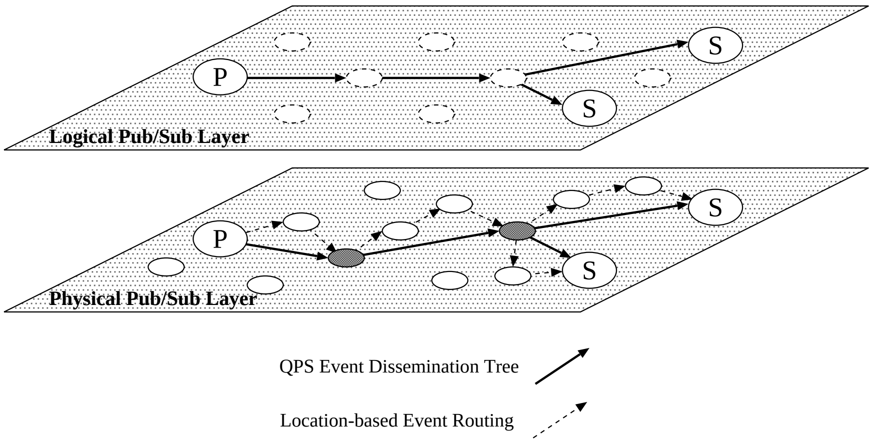


Figure 4.3: QPS Layers

architecture has the advantage that each layer may have its own independent implementation, which can easily be replaced by a different implementation that supports the same interface. This extends the use of QPS, such that different (customized and/or efficient) networking and routing protocols can be (transparently) used in different implementations of QPS to suit different sensor network deployments or target environments. I have described the role of each layer with reference to an example (see Figure 4.3) below.

Clients Layer The highest layer in the architecture is the clients layer. It consists of independent components, who produce and/or consume events in the system. Components in this layer benefit from the underlying data-centric messaging that is provided by the Pub/Sub layer. Figure 4.3 shows two subscribers (denoted by S) which are served by a single publisher (denoted by P).

Logical Pub/Sub Layer This layer provides the core functionality of the Pub/Sub. It benefits from the reliability and abstraction that is provided by the physical Pub/Sub layer, and focuses on the interconnection of related information producers and consumers in the network. The top layer in Figure 4.3 shows the operation of this layer, which constructs an EDT via some abstract (logical) EBs (denoted by dashed ovals). The EDT directs event notifications from publishers (P) to subscribers (S). A balance between path sharing and path freedom is controlled at this layer (according to the subscribers' ϵ factor specifications).

Physical Pub/Sub Layer The physical Pub/Sub layer reflects the resource- and network-aware operations of the Pub/Sub layer. It does not implement any Pub/Sub functionality, but only addresses the network and nodal concerns. This layer ensures the selection and involvement of a suitable set of nodes for Pub/Sub functionality, and provides three services, *resource-aware mapping*, *proactive hand-over*, and *fault-tolerance*. The resource-aware mapping service selects nodes that have sufficient resources for participation on the EDT. Figure 4.3 shows two filled ovals which have been selected to operate as the selected

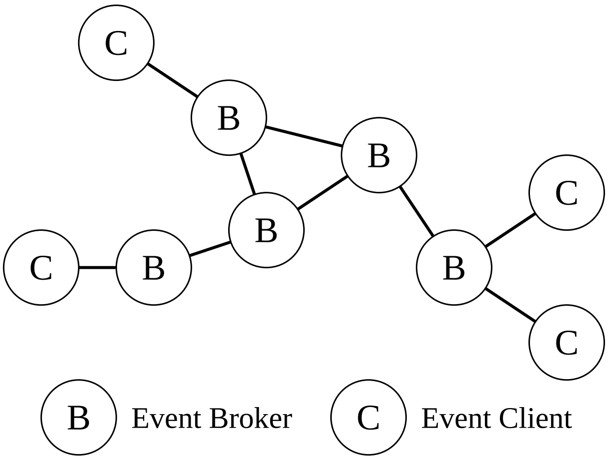


Figure 4.4: QPS Components

EBs at the logical layer. The hand-over service actively monitors these selected nodes for sufficient resources, and relieves them from their operations when their resources fall short. Finally, fault-tolerance is supported to combat abrupt node failures. QPS replicates its data structures across nearby nodes to independently recover from these failures.

Location-based Routing Layer The location-based routing layer implements a reliable unicast messaging service that delivers a message from a source node to a destination node whose location-based address is known. Figure 4.3 illustrates the operation of this layer by a series a dashed arrows which forward the event notifications on hop-by-hop basis along the constructed EDT. When the destination address is set to `ANY`, the routing protocol is assumed to deliver the packet to all one-hop neighbors either by a localized broadcast or by series of unicast messages. The implementation may follow some user-defined QoS requirement such as increased network lifetime or timely event delivery.

Network Layer The network layer ensures globally unique addresses for nodes in the sensor system. It uses nodal locations to assign location-based addresses, and provides addresses when nodes join the network.

4.3.2.1 Pub/Sub components

A Pub/Sub protocol needs to be decentralized to support scalability and fault-tolerance. A distributed implementation entails the operation of many components that operate together to achieve Pub/Sub functionality. These components reside on different nodes, and have different roles that define their purpose and operation. These roles and operations are described by a *component model*.

In my component model, I introduce two kinds of components, *Event Brokers (EBs)* and *Event Clients (ECs)*. EBs implement the entire functionality of the Pub/Sub layer and provide a service to the ECs. To use the Pub/Sub, ECs must connect to at least one EB. ECs come in

Returns	API Call	Parameters
void	send	(Destination destination, MessageType type, Message message)
void	register_handler	(MessageType type, Boolean peek, Callback callback)

Table 4.1: The routing protocol's API

Returns	API Call	Parameters
void	receive	(Message message)
Message message	peek	(Message message)

Table 4.2: The QPS EB callback API

two flavours, *event publishers* that publish events and *event subscribers* that subscribe to events. A Pub/Sub protocol with EBs and ECs is shown in Figure 4.4.

Event Brokers. EBs are the main components of the QPS. A single EB constitutes a complete implementation of the Pub/Sub, but usually multiple EBs are deployed together. These components reside on every node that wishes to support Pub/Sub functionality, and cooperate with each other to form an EDT. EBs use the interface that is exported by the underlying location-based routing protocol to achieve their functionality. The exported routing protocol interface is shown in Table 4.1. The interface allows EBs to send messages and register handlers for Pub/Sub messages in the system. Subsequently, EBs receive messages that are either addressed to them or addressed to some location-based address that is closest to them. In addition, EBs may peek and modify contents of related messages that are handled by the routing protocol. The exported interface for message handling by the EBs is shown in Table 4.2.

An EB that has one or more event publishers connected *locally* (at the same node) is called a *publisher-hosting EB*. Similarly, an EB becomes a *subscriber-hosting EB* if it is maintaining a *local* connection to one or more event subscribers. An EB that is situated on the EDT and intermediates the connection is called a *forwarding EB*. An EB may be all, some, or none of the above.

Definition 4.5 (Event Broker (EB))**.** *An EB $b \in \mathbb{B}$ from the set of all EBs $\mathbb{B}$ maintains a tuple,*

$$b = (C_P, C_S), \tag{4.12}$$

where C_P is a set of locally connected event publishers and C_S is a set of locally connected event subscribers.

Returns	API Call	Parameters
void	advertise	(Publisher pub, EventTopic topic, EventRegion region)
void	publish	(Publisher pub, Event event)
void	subscribe	(Subscriber sub, EventTopic topic, EventRegion region, EpsilonFactor epsilon, Boolean guaranteed_coverage_fulfillment, Callback callback)
void	unadvertise	(Publisher pub, EventTopic topic, EventRegion region)
void	unsubscribe	(Subscriber sub, EventTopic topic, EventRegion region)

Table 4.3: The QPS EB's API

Returns	API Call	Parameters
void	notify	(Event event)
void	failed_coverage_fulfillment	(EventTopic topic, EventRegion region)

Table 4.4: The QPS event subscriber callback API

Event Clients. ECs are components that reside on the clients layer of the architecture. They maintain a connection to their local EBs, and do not possess any Pub/Sub functionality themselves. An EC uses the interface that is exported by its *local* EB to request Pub/Sub functionality, such as publishing or subscribing to events. Since this interface only handles the communication of the EC with its local EB, it may be synchronous or asynchronous. This interface conforms to the standard Pub/Sub interface and is listed in Table 4.3.

ECs are tied to the application components of a sensor system, such as sensors, actuators, services, and users. An event publisher is a client component that produces event publications and passes them to the Pub/Sub protocol for dissemination. An event subscriber subscribes to events and consumes event publications. These events are subsequently passed to the attached clients (e.g. users or actuators). Unlike event publishers, event subscribers receive asynchronous notifications from their local EBs whenever an event is published that matches one of their subscriptions. They may also receive failure reports regarding their guaranteed subscription coverage request. For this they export an asynchronous callback interface, shown in Table 4.4, to the local EB.

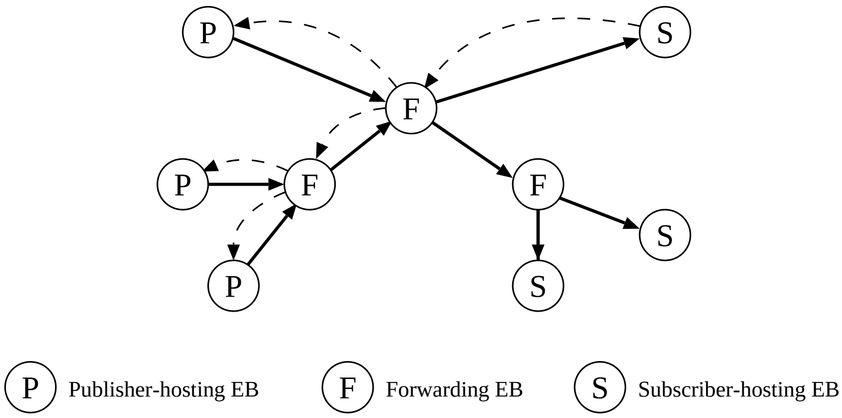


Figure 4.5: QPS's EDT

Definition 4.6 (Event Client (EC))**.** *An EC $c \in \mathbb{C}$ from the set of all ECs $\mathbb{C}$ maintains a tuple,*

$$c = (b_c, t_c, r_c, \epsilon_c), \tag{4.13}$$

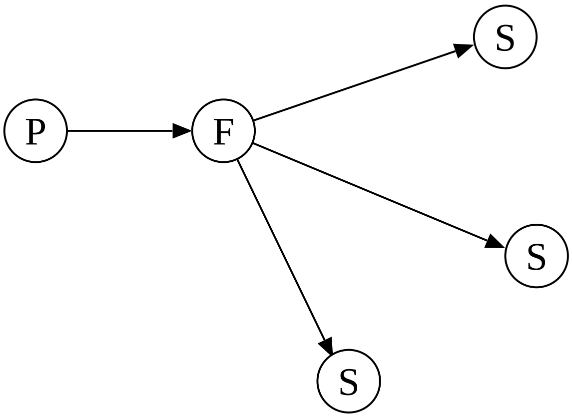
where $b_c \in \mathbb{B}$ is the local EB that c is connected to, $t_c \in T$ and $r_c \subseteq S$ are the related event topic and event region that c has advertised/subscribed, and ϵ_c is the desired event forwarding path-length ratio *when c is a subscriber, otherwise $\epsilon_c = \emptyset$.*

4.3.3 Dissemination Model

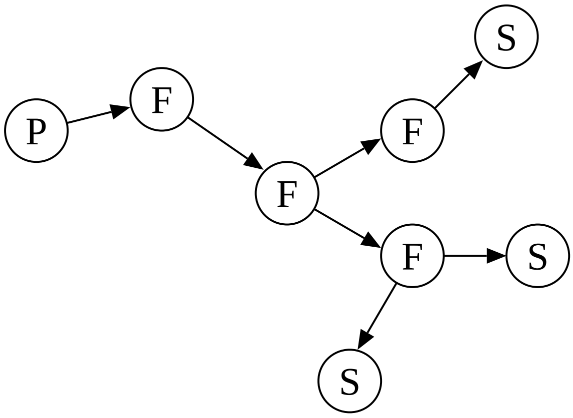
The dissemination model describes the QPS EDT that disseminates events from publisher-hosting EBs to subscriber-hosting EBs. The EDT interconnects publishers and subscribers, whose advertisements and subscriptions overlap, and examines subscription coverages over event notifications that propagate through the tree. Publisher-hosting and subscriber-hosting EBs are interconnected through a set of one or more intermediate EBs, referred to as the *forwarding EBs*, see Figure 4.5 (the EDT is shown by solid arrows). QPS actively maintains this EDT as ECs join and leave the system.

An event forwarding path between any publisher-hosting EB and any subscriber-hosting EB is always intermediated by at least one forwarding EB. This is to ensure correct Pub/Sub functionality, and impose minimal load on publisher-hosting and subscriber-hosting EBs when they are not co-located. Forwarding EBs are dynamically selected from a pool of potential forwarding EBs (defined by the logical Pub/Sub layer).

The interconnection between any neighboring (pair of) EBs on the EDT is defined by a *publish-subscribe link* (the solid arrows). In effect, an EDT is composed of many publish-subscribe links that operate independently but achieve an overall goal of disseminating events from publishers to subscribers. Each link delivers event notifications from one EB to the next EB that is closer to the subscriber-hosting EBs on the EDT. These links are noted by the subscription entries that reside at the downstream EBs (tails of the arrows) and point to their



(a) EDT with fewer forwarding EBs (small ϵ)



(b) EDT with more forwarding EBs (large ϵ)

Figure 4.6: Impact of the number of forwarding EBs on the EDT

upstream EBs (heads of the arrows). The EB at the head of a link, whose tail is at a publisher-hosting EB, is an "immediate forwarding EB to the publisher-hosting EB"; and the EB at a tail of a link, whose head is at a subscriber-hosting EB, is an "immediate forwarding EB to the subscriber-hosting EB".

Publish-subscribe links are formed as part of constructing event forwarding paths in the network. The EDT shown in Figure 4.5 is a composition of three event forwarding paths, each of which relates to a single subscriber-hosting EB in the diagram. Subscription messages, reflecting event subscribers' interests, propagate through the network from each subscriber-hosting EB (dashed arrows in Figure 4.5 show this for one) and construct (reversed) event forwarding paths that resemble tree structures with subscriber-hosting EBs rooted at their tops. These subscription messages are handled by a localized subscription resolving algorithm at the receiving EBs. The algorithm impacts a trade-off between path sharing and path freedom as follows.

4.3.3.1 Path sharing vs Path freedom

Publish-subscribe links are formed at the logical Pub/Sub layer. These links may be shared among multiple event forwarding paths, in which case they help to achieve path sharing within the EDT. The location-based routing protocol governs how events are routed along these links, and benefits from path freedom, such that it may employ arbitrary routing policies when delivering events from the tails of the publish-subscribe links to their heads.

The number of publish-subscribe links, used in an EDT, impacts the achieved path freedom. As the number of links increases so does the number of forwarding EBs that events need to pass through, see Figure 4.6. This reduces path freedom and often leads to segmented, winding event forwarding paths that are composed of many short publish-subscribe links. In addition, the involvement of many forwarding EBs that each reside on a separate network node increases the vulnerability of an EDT to node failures. On a positive note, however, an increased number

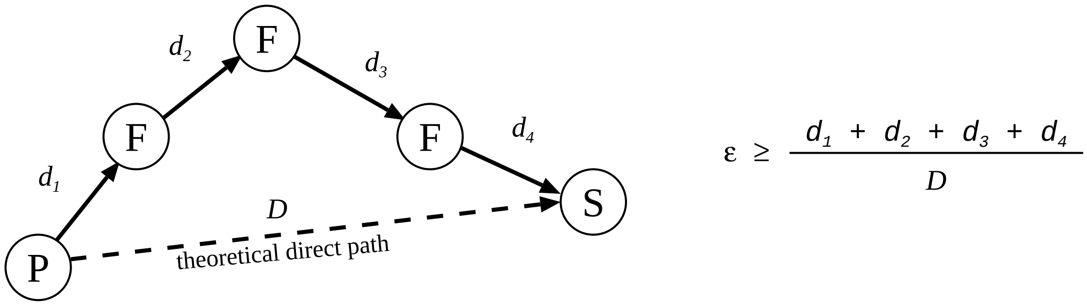


Figure 4.7: Subscriber-specified ϵ factor

of publish-subscribe links increases chances of link selection and path sharing among multiple event forwarding paths.

The subscriber-given $\epsilon \in \mathbb{R} : \epsilon \geq 1$ factor indicates the desirable event forwarding path length relative to a theoretical direct path. In other words it is the ratio of the longest permissible path length to the theoretically shortest (direct) path length, see Figure 4.7. The ϵ factor indirectly manipulates the number of forwarding EBs that intermediate the connection between a publisher-hosting EB and a subscriber-hosting EB. It empowers subscribers to control the following attributes of the formed event forwarding path.

Path Freedom A lower ϵ value would decrease the number of selected EBs, thus increases *path freedom*.

Event Delivery Latency A lower ϵ value would shorten the event forwarding path, such that the lower bound of the *event delivery latency* is reduced.

Path Sharing In contrast to the previous two attributes, path sharing can only be promoted by higher ϵ values that allow event forwarding paths to be stretched for more overlap and path sharing.

In most cases, the subscriber sets the ϵ factor according to its desired event delivery latency or path freedom if a specialized routing protocol is involved. If the subscriber has no interest of the event delivery latency or the path freedom, then he/she can specify an arbitrary ϵ value - my evaluations (in Section 4.4.4.3) show that a larger ϵ value is preferred, though beyond a certain threshold value performance is unaffected.

4.3.3.2 Dissemination Policies

QPS conforms to two event dissemination policies that enhance its usability and reliability in sensor systems.

Real-time Coverage This policy extends the standard subscription coverage with time decoupling (Section 2.3.1.1). It states that an event publication $e \in \mathbb{E}$, published by an EC, $e^p \in \mathbb{C}$, is delivered to every subscriber, whose subscription s matches the event publication, $e \sqsubseteq s$, including when e^p advertised its events after the time of the event subscription.

Single Delivery This policy asserts that events are *unique* in QPS, and that every event publication can be delivered at most once to any subscriber. It formally states that an event publication $e \in \mathbb{E}$, published by e^p at time e^t, can be at most delivered once to any subscriber $u \in \mathbb{C}$: if $\texttt{notify}_u(e)$ denotes a *distinct* event delivery operation (event e is delivered to an event subscriber u), then $\forall e_1, e_2 \in \mathbb{E}$ if $\texttt{notify}_u(e_1) \wedge \texttt{notify}_u(e_2) \Rightarrow e_1 \neq e_2 \Rightarrow (e_1^p \neq e_2^p) \vee (e_1^t \neq e_2^t)$.

In addition to the above policies, QPS introduces a number of operational policies that impact the formation of EDTs and event subscribers' experience in the system. The first policy determines the importance (weight) of the subscription ϵ factor in constructing event forwarding paths, and the second provides a QoS for ECs' interests (subscriptions).

Epsilon Compliance. Subscription ϵ factors impact the formation of event forwarding paths, but also pose a communication overhead if they are to be enforced over shared paths. QPS empowers the system designer to select a global policy that influences a trade-off between ϵ factor compliance and increased communication savings.

Guaranteed ϵ Compliance Guaranteed ϵ compliance dictates that the subscriber-given ϵ factors must be asserted over all event forwarding paths. When subscription trees merge, this policy enforces the examination of all ϵ factors over the shared path. The shared path is reconstructed if it is not in compliance with the smallest ϵ factor.

Best-effort ϵ Compliance When subscription trees merge, this policy prioritises communication savings over the ϵ compliance. It neglects to assert the ϵ factors over the shared paths, and leads to best-effort ϵ compliance, where event forwarding path formations comply to ϵ factors only until shared paths are reached.

Coverage Fulfillment. Event subscribers are limited to receiving events that are published by the event publishers in the system. The relation between a subscriber's interests and event publishers' publishable events can be described by a *subscription coverage fulfillment* policy in QPS. The level of subscription coverage that may be attained for a given subscription, s, against a set of realised event advertisements, A, determines the coverage. A choice of two policies are available, on per subscription basis, as follows.

Best-effort Coverage Fulfillment The best-effort coverage fulfillment policy asserts no more conditions than what the real-time coverage policy already asserts in QPS. With this policy, there may be zero or more event publishers at any time that serve (can publish events for) an event subscriber.

Guaranteed Coverage Fulfillment This policy asserts that complete subscription coverage should be achieved. In other words, all events that the event subscriber has interests over must be publishable by some set of event publishers at all times. This requires a complete coverage of the event subscription by the set of event advertisements, $E_s \subseteq \bigcup_{d \in A} E_d$, where E_s is the set of interested events (by a subscriber s), and E_d is the set of publishable events that is described by an advertisement d from the set of all advertisements A. In sensor systems, the guaranteed coverage fulfillment policy is useful for applications that wish to have their phenomenon (or condition) of interest under continuous and complete surveillance by one or more sensors (event publishers). Under this policy, event subscribers are notified about any changes (e.g. publisher failure) that affect their subscription coverage by the `failed_coverage_fulfillment` callback function.

4.3.4 Event Service

QPS event service is decentralized for scalability, fault-tolerance, and increased load balancing. Every node in the network is assumed to possess Pub/Sub functionality, and hosts an EB component. The EDT, discussed above, is formed in a decentralized manner and by some localized operations at these EBs.

Every forwarding EB is assigned a role that is defined at the logical Pub/Sub layer. The logical layer defines a pool of distinct roles, that can serve any subscription request either individually or in combination. While distinct roles promote the distribution and involvement of many EBs on the EDT, a localized subscription resolving algorithm and the physical Pub/Sub layer control this distribution. The former restricts the selection of EBs for participation on the EDT, and the latter ensures that only resourceful EBs are involved in the EDT.

The two Pub/Sub layers have separate, but related, views and operations about the EB components. The following sections describe these layers separately, and present a set of notations that are used in subsequent sections for describing their correspondence in attaining Pub/Sub functionality.

4.3.4.1 Logical layer

The logical layer has a network-independent view of QPS EBs. I call these components, *logical EBs*, because they are defined at an abstract level. The logical Pub/Sub layer provides complete Pub/Sub functionality at this abstract level, as it constructs and maintains EDTs over the logical EBs.

The logical layer defines a graph, $G_{T,L} = (G_{T,LV}, G_{T,LE})$, whose vertices $G_{T,LV}$ define the set of logical EBs and directed edges $G_{T,LE}$ describe the EBs' parent-child relationships (discussed shortly) for the set of all event topics T. This graph is known globally; EBs can locally compute the graph using a single *hash* function and the set of all event topics T. Each logical EB can only forward a subset of the event space $\mathbb{E}$ on an EDT. The logical layer systematically partitions the event space and assigns subsets to these logical EBs. These subsets define the roles that are associated with the logical EBs.

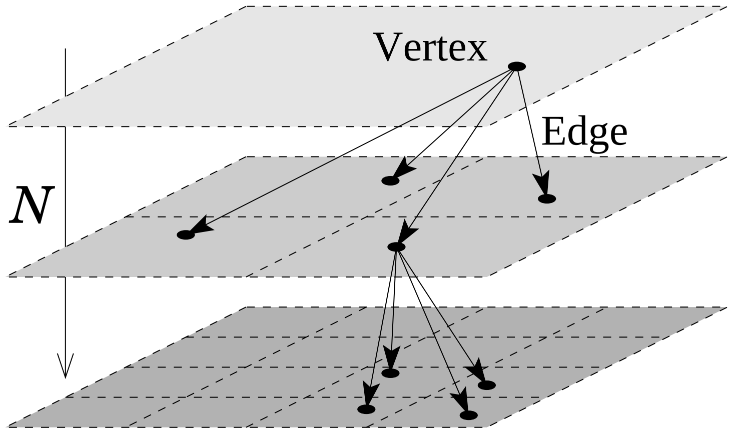


Figure 4.8: Geographical Scopes

In order to determine the subsets, the event space $\mathbb{E}$ is partitioned across two domains: the event topics and the event location. The set of event topics T is a discrete set which is partitioned along every topic member. The event location, however, is a much larger set that needs a partitioning policy; I assume a two dimensional (2-D) geographical space[1] and describe my partitioning policy as follows.

The logical layer encloses the entire sensor network's coverage area in a region, referred to as the *network space*, S. The location attribute of all publishable events is assumed to fall within this network space, $\forall e \in \mathbb{E}\ l_e \in S$. Although sensor network coverage may be dynamic and dependent on nodes' join or leave operations, the network space is assumed to be static and can be arbitrarily defined larger than the network's coverage area. The logical layer partitions S into a hierarchy of Geographical Scopes (GSs), in which the number of hierarchy levels N is pre-defined by the system designer. Since the location-based routing protocol is responsible for geographically broadcasting messaging within the lowest level GSs, it is sensible to select N such that nodes contained in the lowest level GSs are within direct transmission range of one-another. In this partitioning, the first (highest) hierarchical level constitutes the entire network space S as a single GS. Subsequent partitions divide every GS on the top level into four (six if 3-D space is used) equisized GSs to form a hierarchical structure. This partitioning is iterated until N hierarchical levels are achieved, see Figure 4.8. The GSs are fixed, and total $\frac{4^N-1}{3}$ scopes.

For every combination of an event topic $t \in T$, and a GS $g \subseteq S$, a logical EB $u \in G_{t,LV}$ is defined that is responsible for events matching the event topic t and holding a location attribute parameter $l \in g$. If one interconnects the EBs related to an event topic $t \in T$, from the highest GS to the lowest GSs, a *Quad-Tree (QT)* is formed (see Figure 4.8). The parent-child relationship on this tree is described by the directed edges of the logical layer graph, $G_{T,LE}$. The lowest level vertices, that have no children, are referred to as the *leaf vertices* (or *leaf EBs*) of the QT. The operation of the logical layer QT is independent for each event topic. Hence, I study this layer

[1]The Pub/Sub mechanism presented for QPS can similarly be applied for 3-D space with the notable differences that *Oct-Trees* and an *Oct-PubSub* would be realised.

(for the remainder of this chapter) from the perspective of a single event topic, $t \in T$, and its corresponding QT, $G_{t \in T,L} \equiv G_L = (G_{LV}, G_{LE})$.

Decentralized EDT maintenance. The main function of the logical layer is to maintain interaction between the publishers and the subscribers through an EDT. Logical EBs are the only EBs that can become forwarding EBs (i.e. form the EDT). They can join (cover) subscriptions if they overlap, decompose and relay them to child EBs (on the QT) for more direct event forwarding paths, or register them and serve their corresponding (subscriber-hosting) EBs as immediate forwarding EBs. A localized subscription resolving algorithm directs each logical EB to perform one or more of the above actions when it receives a subscription request. Actions are either driven by messages or by local state changes.

At the global level, operations are often subscriber-initiated, and go as follows. A subscriber-hosting EB, in order to receive events for its subscriber, dispatches a request to the nearest logical EB that is responsible for the events of interest. The request is handled by the addressed logical EB (as described above) and resolved over the QT. This subscription resolution connects the subscriber (at one or more points) to the EDT. At the other end of the EDT, advertisements are used to connect publishers to the EDT. Publisher-hosting EBs are almost always connected to the EDT via leaf EBs, i.e. the leaf vertices of the QT. Although the EDT's forwarding EBs are selected from the QT, the EDT itself may have a structure different from the QT.

4.3.4.2 Physical layer

The physical Pub/Sub layer maps the EDT, that is constructed by the logical layer, on to the real network. It describes the real network by a physical layer graph $G_P = (G_{PV}, G_{PE})$, whose vertices G_{PV} represent the deployed network nodes, and directed edges G_{PE} describe the asymmetric link-layer connections between them. Since every node is assumed to house an EB component, the vertices could also be considered as real EBs, i.e. $G_{PV} \equiv \mathbb{B}$.

Resource-awareness and network maintenance. The physical layer is responsible for mapping logical EBs (Pub/Sub roles) to physical (real) EBs. It initially maps the logical EBs to real EBs on demand basis. The initial interaction with a logical EB is always message-based, and the physical layer exploits this plus the multi-hop nature of the routing process to search for a suitable EB at the routing stage. At the end, it directs the message to the most resourceful EB that is found during the search process. Subsequent mappings are performed proactively, when the EB's resources fall short or drop rapidly.

A useful feature of the physical layer is that its operations are entirely based on message contents and/or local states. This, assuming a trusted environment, allows EBs to invoke operations *for* other EBs. For example, when an EB fails, another EB can unsubscribe the failed EB from the EDT to save some messaging; this is achieved by issuing an unsubscribe message with the failed EB's address as its source field[1]. QPS EBs use this feature to transparently maintain the EDT through standard Pub/Sub operations.

[1]Note that this and other fields, described later, are independent of those introduced by the routing or network layer protocols.

4.3.4.3 Notation

In line with my earlier discussion, the notation that is presented in this section is for a single event topic $t \in T$.

- $G_L = (G_{LV}, G_{LE})$ and $G_P = (G_{PV}, G_{PE})$ denote the logical and physical Pub/Sub graphs, respectively.
- $u \in G_{LV}$ denotes a vertex on the logical graph, that represents a logical EB on the QT.
- $u \in G_{PV}$ denotes a vertex on the physical graph, that represents a real node in the network. The node houses one EB and zero or more EC components.
- $(u, v) \in G_{LE}$ denotes a directed edge that represents the parent-child relationship between u and v on the QT (see Figure 4.8).
- $(u, v) \in G_{PE}$ denotes a directed edge that represents an asymmetric link-layer connection from u to v. A connection is symmetric if and only if $(u, v) \in G_{PE} \Leftrightarrow (v, u) \in G_{PE}$.
- $loc(u \in (G_{LV} \cup G_{PV})) \mapsto p \in S$ is a function that maps a vertex u to a point, p, on the network space, S. Let $l(u \in G_{LV}) \equiv loc(u)$ and $p(v \in G_{PV}) \equiv loc(v)$ be short-hand notations. $p(v \in G_{PV})$ reflects the location of a node that is determined by the network layer (localization algorithm), and $l(u \in G_{LV}) = hash(s(u))$ statically defines the location mapping of any logical EB u on to S (the *hash* function and its parameter, $s(u)$, are defined shortly).
- $hash_{t \in T}(r \subseteq S) \mapsto p \in r \subseteq S$ denotes a geographical hash function, that when given a key (event topic t) and a region r, outputs a unique location p within the given region r. Since p is bounded by r, the vertices of a QT probabilistically converge towards their GS centroids as GSs shrink from the top to the bottom.
- $cov^{u \in G_{PV}}(v \in G_{PV}) \mapsto r \subseteq S$ is a function that returns the subscription region r, for which v has u registered in its subscription routing table (routing tables are discussed later in Section 4.3.5.2). Also, $cov(v \in G_{PV})$ denotes the *stable Pub/Sub subscription coverage* that is registered at v's subscription routing table (explained in Section 4.3.5.2).

Logical layer notations

- $c(u \in G_{LV}) = \{v \in G_{LV} | (u, v) \in G_{LE}\}$ denotes the children of u on the QT; $c(u) = \emptyset$ if and only if u is a leaf vertex.
- $s(u \in G_{LV}) \mapsto s \subseteq S$ is an inverse function that returns the GS of vertex u. Also, $s_{i \in \{1, \cdots, 4\}}(u \in G_{LV}) \equiv s(c_i(u))$ denotes the GS of u's children.
- $c_{i \in \{1, \cdots, 4\}}(u \in G_{LV})$ denotes the ith child of u, that is located in the ith sub-GS of $s(u)$, starting from the quadrant with the minimum coordinate values and counting clockwise.

- $K(c \in \mathbb{C}, s = (t_s, r_s, \epsilon_s)) \mapsto \{u \in G_{LV} | \bigcup_{x \in K(c,s)} cov^b(x) = r_s,\ \forall v \in K(c,s),\ u \neq v \Rightarrow cov^b(u) \cap cov^b(v) = \emptyset\}$ describes the overall operation of the logical layer, in which an event subscription s from an event subscriber c that is connected to its local EB $b \in G_{PV}$ is resolved over the QT. An EB $u \in G_{LV}$ is an immediate forwarding EB if and only if $u \in K(c,s)$. The asserted conditions $\bigcup_{x \in K(c,s)} cov^b(x) = r_s$ and $cov^b(u) \cap cov^b(v) = \emptyset$ describe the QPS's *real-time coverage* and *single delivery* policies.

- $R\ (u \in G_{LV}, v \in G_{PV}, r_v \subseteq S, \epsilon_v \in \mathbb{R}) \mapsto \{(y \in G_{LV}, r_y \subseteq S, \epsilon_y \in \mathbb{R}) | y \in c(u) \cup \{u\}, r_y \subseteq s(y)\}$ is a *Localized Subscription Resolving Algorithm* that the logical Pub/Sub layer implements to handle incoming subscription messages (detailed later in Section 4.3.5.4).

Physical layer notations

- $map(u \in G_{LV}) \mapsto v \in G_{PV}$ denotes a one-way mapping function, that maps every vertex on the logical layer on to a vertex on the physical layer. The function is a resource-aware mapping function, implemented by the physical Pub/Sub layer.

- $suit(u \in G_{PV}, v \in G_{LV}) \mapsto y \in \mathbb{R}$ is a suitability function that indicates how suitable a physical node u is for operating in the role of a logical EB v. The physical Pub/Sub layer uses this function to perform resource-aware mapping.

- $compare(x \in \mathbb{R}, y \in \mathbb{R}) \mapsto h \in \{true, false\}$ is a suitability compare function that indicates whether a node's workload should be reduced following the change of suitability value from x to y. This function is used by the physical Pub/Sub layer to perform active hand-overs.

4.3.5 Routing

Routing algorithms govern the inter-EB messaging that occurs in QPS. They direct actions following receipt of messages and designate destinations for messages that are generated by the EBs. These actions and destinations are largely based on the message types, and messaging end-points. I first discuss the message types in QPS, then outline the data structures that are maintained at the EBs, and finally present the routing algorithms that control the propagation of each message type in the system. Induced operations at the senders and receivers are also explained as part of this presentation.

4.3.5.1 Message Types

Four types of messages are realised at the Pub/Sub layer in QPS: *advertisement messages*, *subscription messages*, *coverage fulfillment messages*, and *event publication messages*. The first type distributes information about possible event publications in QPS. The second type distributes requests for events that are of interest to an event subscriber. These messages result in the formation of publish-subscribe links and selection of a set of forwarding EBs. The third type transfers knowledge about covered and uncovered subscription regions. It is only used when guaranteed coverage fulfillment is requested. Finally, event publication messages envelope published events, that need to be disseminated across the network to reach the corresponding subscribers.

In addition to these message types, there are unadvertisement and unsubscription messages, which are inverses of the corresponding messages described above. The routing algorithms use them to remove state from EBs, but for all practical purposes they behave in the same manner as their positive counterparts. Next I will explain the structure of each message type.

Advertisement Messages. An event publisher that is willing to publish events may cause its hosting EB to send *advertisement messages*. Advertisement messages are routed to corresponding EBs that do not know about the publisher. These messages create states in EBs' advertisement routing tables, that may be used later to form publish-subscribe links. Messages contain an `eventTopic` and an `eventRegion`, which define the set of publishable events $E_d \subseteq \mathbb{E}$ for an advertisement d. The `source` reflects the publisher-hosting EB, and the `destination` contains the address of the target logical EB. Three fields (`candidateEB`, `suitabilityFactor`, and `handoverEB`) relate to the physical Pub/Sub layer functionality that is discussed later in Section 4.3.6.

source	destination	eventTopic	eventRegion
candidateEB	suitabilityFactor		handoverEB

Subscription Messages. A *subscription message* may be sent, by a subscriber-hosting EB, when an event subscriber makes a subscription call. These messages are routed to one or more EBs that can serve subscribers' requests. Where these requests are met, new entries are added to the subscription routing tables of some EBs, which denote active publish-subscribe links. These links direct subscription matching events to the subscriber-hosting EB. Subscription messages contain an `eventTopic`, an `eventRegion`, and an `epsilonFactor` field, that define the set of desirable events $E_s \subseteq \mathbb{E}$ for a subscription s. The `source` indicates the subscribing EB, and the destination points to an EB that can serve the subscription. The Q_{bit} (`quadBit` field) is used to indicate whether the source (subscriber) is on the QT or not, and the `fulfillBit` reflects subscriber's preference about the coverage fulfillment.

source	destination	eventTopic	eventRegion	epsilonFactor
quadBit	fulfillBit	candidateEB	suitabilityFactor	handoverEB

Coverage Fulfillment Messages. A *coverage fulfillment message* is sent by a publisher-hosting EB to indicate a covered subscription region, or by a forwarding EB to indicate an uncovered subscription region. A `fulfillBit` indicates which of the two is implied, and `eventTopic` and `eventRegion` fields highlight the details of the (un)covered subscription. These messages are only produced if the guaranteed coverage fulfillment policy is requested.

source	destination	eventTopic	eventRegion	fulfillBit

Event Publication Messages. An *event publication message* is generated by a publisher-hosting EB, when an event is received from a publisher. They are routed to one or more forwarding EBs that further disseminate the event to the related subscribers. The forwarding of event publication messages is controlled by subscription routing tables that reside at the EBs. These messages contain an `eventTopic`, an `eventLocation`, and a series of event attributes `eventAttribute` (name-value pairs) that reflect the event. In addition, a Q_{bit} (`quadBit` field) is used to prevent cyclic message forwarding along the EDT.

source	destination	quadBit	eventTopic
eventLocation	eventAttribute$_1$	eventAttribute$_2$	···

4.3.5.2 Data Structures

QPS functionality depends on three data structures that are maintained at all EBs. Two routing tables implement the Pub/Sub functionality, and an EB mapping table maintains information about the EBs that have taken the role of one or more logical EBs. These data structures are managed locally and contain information that is received via Pub/Sub messages or from the local ECs.

Routing Tables. An *advertisement routing table* records information about advertisements and a *subscription routing table* does the same for subscriptions. The routing tables have a similar form and are thus sub-instances of the same data structure. Their purpose is to maintain information about the set of publishers and subscribers whose advertisement and subscription messages have been registered at the EB. These registrations (and de-registrations) mirror publish-subscribe link formations (and eliminations) in QPS. Insertion or deletion of entries to and from these routing tables trigger handlers that assess the impact of these link formations or eliminations on the EDT, and may lead to subsequent independent subscribe, unsubscribe, or coverage fulfillment message generations by the EBs.

Definition 4.7 (Subscription Routing Table)**.** *A subscription routing table RT_{sub} contains a set of routing table entries, RT_E,*

$$RT_E \in RT_{sub}. \tag{4.14}$$

A subscription routing table entry RT_E is a tuple,

$$RT_E = (sub, q, f, b), \tag{4.15}$$

where sub is a subscription, q is the Q_{bit} that was contained in the received subscription message, f is the `fulfillBit`, and $b \in G_{PV}$ is the broker that sent the subscription (the `source` field of the received subscription message).

The coverage function $cov^{u \in G_{PV}}(v \in G_{PV})$ (introduced earlier) is a short-hand notation for u's registered subscription region in v's subscription routing table, $cov^{u \in G_{PV}}(v \in G_{PV}) = r_s$, where $r_s \in sub \in RT_E = (sub, q, f, b) \in RT_{sub} : b = u$. Also, the 'stable Pub/Sub subscription coverage' at $v \in G_{PV}$, $cov(v \in G_{PV})$, is the accumulated subscription region of all subscription routing table entries that have been fully resolved in the network, and have subscription ϵ factor non-equal to zero, $cov(v \in G_{PV}) = \bigcup_{r \in R} r$, where $R = \{r_s | \exists sub = (t_s, r_s, \epsilon_s) \in RT_E = (sub, q, f, b) \in RT_{sub}.\ \epsilon_s \neq 0\}$.

The action that follows a subscription insertion or deletion (to or from an EB's, u's, subscription routing table) depends on the change in the EB's accumulate subscription coverage, $cov(u)$. I label the coverage prior to change as $cov_{\text{before}}(u)$, and the one after as $cov_{\text{after}}(u)$. The two may then be compared as follows.

$cov_{\text{after}}(u) > cov_{\text{before}}(u)$ u independently subscribes to the added subscription region $cov_{\text{after}}(u) - cov_{\text{before}}(u)$ to compensate for the change. The `source` fields of the newly generated subscription messages are set to u.

$cov_{\text{after}}(u) = cov_{\text{before}}(u)$ No action is necessary.

$cov_{\text{after}}(u) < cov_{\text{before}}(u)$ u independently unsubscribes the region $cov_{\text{before}}(u) - cov_{\text{after}}(u)$. The `source` fields of the generated unsubscription messages are set to u.

Definition 4.8 (Advertisement Routing Table)**.** *An advertisement routing table RT_{adv} contains a set of routing table entries, RT_E,*

$$RT_E \in RT_{adv}. \tag{4.16}$$

An advertisement/subscription routing table entry RT_E is a tuple,

$$RT_E = (adv, b), \tag{4.17}$$

where adv is an advertisement and $b \in G_{PV}$ is the broker that sent the advertisement (the `source` field of the received advertisement message).

When an advertisement $d = (t_d, r_d)$ is registered or removed (to or from the table), the registered subscriptions $\{s = (t_s, r_s, \epsilon_s) | s = sub \in RT_E \in RT_{sub}\}$ are examined. For insertion, those with overlapping regions, $\{s | r_s \cap r_d \neq \emptyset\}$, are forwarded to the advertisement sender $b \in G_{PV}$; this action connects the newly found publisher to the EDT. For deletion, i.e. when a publisher leaves, the set of subscribers who requested guaranteed coverage fulfillments are notified if their subscription region is affected, i.e. if $r_s \not\subseteq cov_{\text{after}}(u)$. Coverage fulfillment messages are generated and dispatched to these subscribers, as instructed in Section 4.3.5.5.

EB Mapping Table. An *EB mapping table* records information about logical EBs mapped to the physical EBs. This information is used to direct messages that are addressed to the logical EBs to their corresponding physical EBs.

Definition 4.9 (EB Mapping Table)**.** *An EB mapping table MT contains a set of mapping entries, MT_E,*

$$MT_E \in MT. \tag{4.18}$$

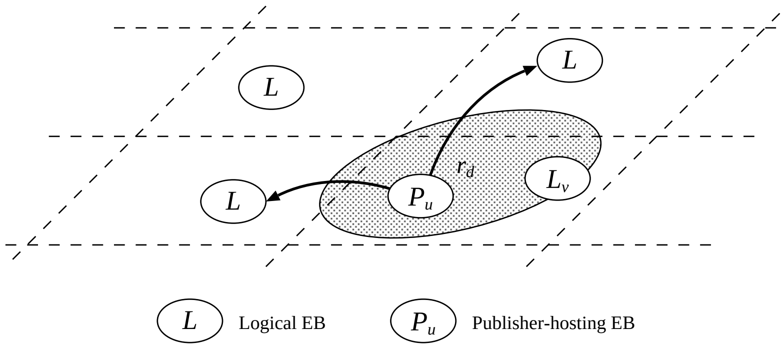


Figure 4.9: Advertisement Messages

A mapping entry MT_E is a tuple,

$$MT_E = (u, v) \ : \ map(u) = v, \tag{4.19}$$

where $u \in G_{LV}$ indicates a logical EB and $v \in G_{PV}$ indicates its corresponding real EB (from the set of brokers $\mathbb{B}$).

Entries in the EB mapping table are soft-state and need to be refreshed periodically by mapped EBs $v \in G_{PV} : v \in MT_E$.

4.3.5.3 Advertisement Messages

Apart from a local EB, an event publisher also has an associated *local logical EB*. These local logical EBs are leaf EBs, whose GSs cover the publisher-hosting (local) EBs' locations. If $u \in G_{PV}$ is a publisher-hosting EB (shown as P_u in Figure 4.9), then $v \in G_{LV} \ : \ c(v) = \emptyset \wedge p(u) \in s(v)$ is the local logical EB to u and all the publisher ECs that are connected to u (see L_v in Figure 4.9).

A publisher-hosting EB $u \in G_{PV}$ generates advertisement messages if and only if the event region of an advertisement r_d exceeds its local logical EB's GS, i.e. if $r_d \nsubseteq s(v)$, where $v \in G_{LV}$ is u's local logical EB. The generated advertisement messages are dispatched to all non-local leaf EBs, whose GSs overlap with the event region r_d, see Figure 4.9. The purpose of these messages is to inform addressed EBs about the event publishers that are *not* located within their GSs, but publish events that relate to their GSs. This set of related logical EBs can be expressed as $\{y \in G_{LV} | y \neq v, \ c(y) = \emptyset, \ s(y) \cap r_d \neq \emptyset\}$. Advertisement messages that are dispatched to each y reflect an overlapping advertisement region, $r_d \cap s(y)$. These messages register passive publish-subscribe links at the addressed logical EBs, which may be activated later, upon the realisation of a corresponding subscription.

More direct (and thus efficient) publish-subscribe links can be formed if advertisement states are stored at more logical EBs. This can be achieved by sending advertisement messages to nearby EBs. More precisely, advertisement messages may be sent to all non-local logical EBs whose GSs overlap with the advertisement region and location fall within a circle, that is centered at the publisher-hosting EB's location with a radius that reaches the non-local leaf EBs. This set of logical EBs, for a publisher-hosting EB $u \in G_{PV}$ and advertised event region r_d can be described by $\{y \in G_{LV} | s(y) \cap r_d \neq \emptyset.\ \exists z \in G_{LV}.\ |p(u) - l(y)| \leq |p(u) - l(z)|\}$, where z is a leaf EB and a descendant of y, i.e. $c(z) = \emptyset,\ s(z) \cap s(y) \cap r_d \neq \emptyset$.

Note that advertisement messages are directed to the location-based addresses of logical EBs, i.e. destination address is $\{l(y)\}$. The physical Pub/Sub layer will map these addresses to real nodes, as explained later in Section 4.3.6.1.

4.3.5.4 Subscription Messages

An EB, $u \in G_{PV}$, generates subscription messages when it realises subscriptions, $s = (t_s, r_s, \epsilon_s)$, from its connected ECs that are not covered by its subscription routing table entries, i.e. $r_s \nsubseteq cov(u)$. The subscription message contains the `eventTopic` value t_s, the `eventRegion` value $(r_s - cov(u))$, and the `epsilonFactor` value ϵ_s, and is dispatched to one or more EBs who can serve the subscription request. The values of other fields in the subscription message, and the set of destination EBs are determined according to the role of u. I first discuss the destination EBs (according to various roles of u), and then describe the actions that are taken upon receipt of such subscription message at the EBs.

Sending Subscription Messages. An EB that has generated a subscription message, can have one of the following roles.

Subscriber-hosting EB A subscriber-hosting EB generates subscription messages that reflect its local ECs' subscription requests. These subscription messages hold a `quadBit` field that is set to zero, and are dispatched to the *geographically nearest* logical EB, $v \in G_{LV}$, that can serve the subscription, i.e. $\exists v \in G_{LV} : r_s \subseteq s(v) \wedge \forall y \in G_{LV}, r_s \subseteq s(y).\ |p(u) - l(v)| \leq |p(u) - l(y)|$, where $u \in G_{PV}$ is the subscriber-hosting EB.

Forwarding (non-leaf) EB A subscription message may be generated by a forwarding EB who has registered a subscription entry, that is not covered by other entries in the subscription routing table (Section 4.3.5.2). In this case, u is already assigned the role of a logical EB, i.e. $\exists v \in G_{LV} : u = map(v)$. The `quadBit` field is set to one, and the subscription region is decomposed and forwarded to the set of child EBs, $c(v)$.

Forwarding leaf EB A leaf EB has no child EBs; thus, when u is a forwarding leaf EB, i.e. $\exists v \in G_{LV} : u = map(v), c(v) = \emptyset$, the subscription message's `epsilonFactor` and `quadBit` fields are set to zero, and the message is geographically broadcast to all EBs who are within v's GS, i.e. to $\{y \in G_{PV} | p(y) \in s(v)\}$. The subscription message is also sent to publisher-hosting EBs that have advertised corresponding event publishers, registered at u's advertisement routing table; this activates the passive publish-subscribe links that were noted earlier in Section 4.3.5.3.

Note that, like advertisement messages, subscription messages are mostly addressed to logical EBs.

Receiving Subscription Messages. Subscription messages, that are received by EBs, are also handled according to the role of the receiving EB, $u \in G_{PV}$, with respect to the message. The `destination` field of the subscription message indirectly highlights this role, and can be either a wild-card destination (ANY), a location address that corresponds to a real node (*physical EB address*), or a location address that reflects a logical EB (*logical EB address*). The EB u handles the subscription message, according to the `destination` field as follows.

ANY Subscription messages that are addressed to ANY have been geographically broadcast by a forwarding leaf EB. The purpose of the broadcast is to search for corresponding event publishers in the forwarding leaf EB's GS. The EB u, upon receiving such a message, examines its ECs for matching event publishers. If found, $\exists v \in C_P : r_v \cap r_s \neq \emptyset$ (where r_v is the advertisement region of v), then the subscription is registered at u's subscription routing table with a partial event region reflecting the overlap $r_v \cap r_s$. If guaranteed coverage fulfillment is requested, then a coverage fulfillment message is also generated (with matching event topic and overlapping event region) and dispatched to the leaf EB (indicated by the `source` field of the subscription message).

Physical EB address Subscription messages are addressed to physical EB addresses when a corresponding advertisement entry is seen at the sender's advertisement routing table. The addressed EB, u, ought to have a related event publisher, in which case it operates as described above.

Logical EB address The most common case is where the destination of a subscription message is the location-based address of a logical EB $v \in G_{LV}$. In this case the recipient, u, notices a mismatch between its own location-based address and the `destination` field of the subscription message. This indicates that a logical EB v has been mapped to u, i.e. $\exists v \in G_{LV} : u = map(v)$; u can examine the QT to identify v. A *localized subscription resolving algorithm* handles these subscription messages.

Localized Subscription Resolving Algorithm. The localized subscription resolving algorithm is called when a subscription message, with `destination` field relating to a logical EB $v \in G_{LV}$, is received at an EB $u \in G_{PV}$. This instance holds u responsible to the role of v, and to the subscription request, s, contained in the message. The addressed logical EB v, however, is most likely not the only EB that can serve s. A QT, with N GS levels, can offer at lease $2^N - 1$ different EB combinations that can serve any subscription.

The localized algorithm determines if and how much u should be involved in forwarding events to the subscriber, $q \in G_{PV}$. If it determines that the subscription should be partially or fully resolved, then the subscription is registered at u's subscription routing table and u becomes involved as a forwarding EB.

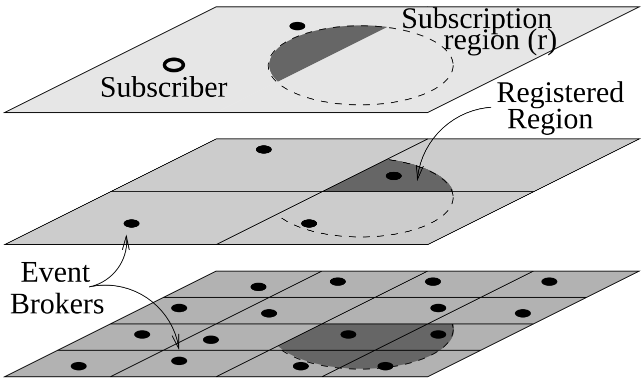


Figure 4.10: Resolved Subscription

The algorithm is iterated over the QT until the subscription is fully resolved. In Figure 4.10, a subscriber's subscription is forwarded to the root of the QT (shown at the top layer). The subscription region, $r \subseteq S$, is partially registered at the top level EB (shown as a shaded region), and partially decomposed and relayed to the children EBs. The process is iterated until the subscription coverage domain, r, is completely resolved over the QT. Immediate forwarding EBs are those which are responsible for the registered (shaded) regions. In this example, six EBs are selected (one at the top layer, one at the middle layer, and four at the bottom layer) as the immediate forwarding EBs to the subscriber-hosting EB.

Definition 4.10 (Localized Subscription Resolving Algorithm)**.** *A localized subscription resolving algorithm R is a function that maps an extended subscription tuple τ to a set of subscription resolved tuples Θ,*

$$R : \tau \to \Theta. \tag{4.20}$$

The tuple τ describes an addressed logical EB v, subscriber q, subscription event region r_q, and subscription epsilon ϵ_q, and Θ describes a set of subscription resolved tuples that each indicate a new target logical EB y, subscription event region r_y, and subscription epsilon ϵ_y,

$$R\left(v \in G_{LV}, q \in G_{PV}, r_q \subseteq S, \epsilon_q \in \mathbb{R}\right) \mapsto \{(y \in G_{LV}, r_y \subseteq S, \epsilon_y \in \mathbb{R})\}. \tag{4.21}$$

The algorithm R is localized*, such that new target logical EBs are selected without inter-broker collaboration, and it is* optimistic*, as it aims to form a publish-subscribe link (from u to q) where allowed for future path sharing.*

A publish-subscribe link is created (from u to q) if the resulting event forwarding path is in compliance with the subscription epsilon ϵ_q. The following describes how this compliance is evaluated, and the resulting Θ is produced.

The addressed logical EB v compares approximate event forwarding paths, for the options of *registering* or *relaying* the subscription request, with respect to each of its children's GSs, $s_i(v)$. u registers the subscription, if the ratio equals or falls below ϵ_q. Let $\{rg_i \in \mathbb{R} | i \in \{1, \cdots, 4\}\}$ and $\{rl_i \in \mathbb{R} | i \in \{1, \cdots, 4\}\}$ denote the sets of distances for the options of registering or relaying a

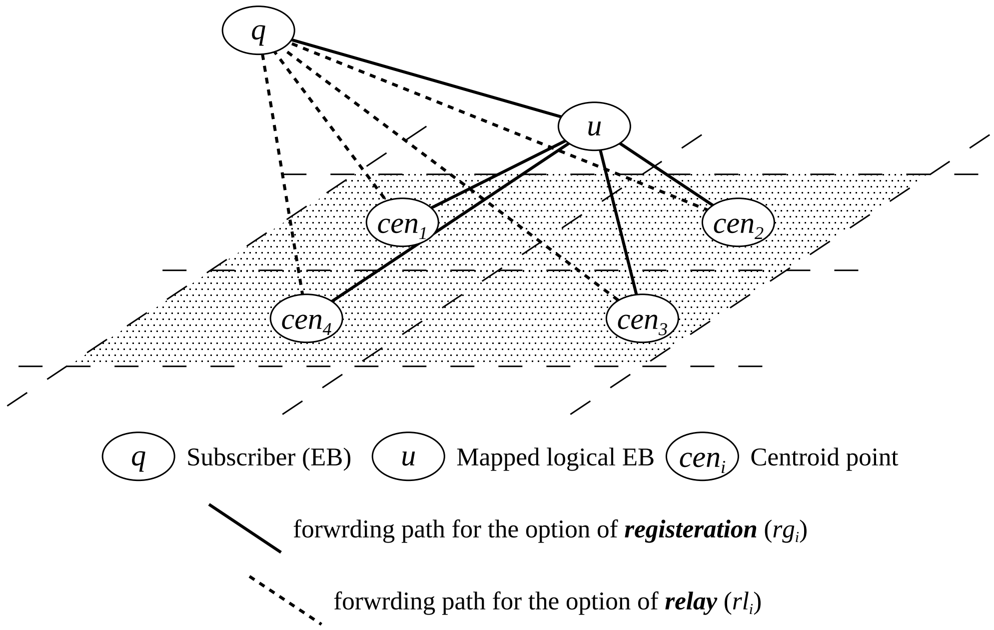


Figure 4.11: Register vs Relay distances

subscription request for each of the quadrants $s_i(v)$, see Figure 4.11. Three sets of coordinates are needed to approximate the event forwarding path lengths as shown in Figure 4.11.

Subscriber's location The subscriber's location, $p(q)$, is known from the `source` address that is included in the subscription message.

Addressed EB's location The addressed target logical EB v is mapped to a real EB, $u = map(v)$, whose location, $p(u)$, is known locally.

Publishers' locations If u's advertisement routing table holds relevant entries, then the location of those (publisher-hosting) EBs is used. In addition, four virtual publisher coordinates are computed that correspond to the publishers that reside in v's children GSs. These coordinates are defined as *centroid points*, $\{cen_i \in S | i \in \{1, \cdots, 4\}\}$, of overlapping subscription regions $\{r_{i\in\{1,\cdots,4\}} \subseteq S | r_i = s_i(v) \cap (r_q - cov(u))\}$.

Using these coordinates, the $\{rg_{i\in\{1,\cdots,4\}}\}$ and $\{rl_{i\in\{1,\cdots,4\}}\}$ distances may be computed as follows.

$$\{rl_i \in \mathbb{R} | rl_i = |p(q) - cen_i|\} \tag{4.22}$$

$$\{rg_i \in \mathbb{R}| \text{ if } r_i = \emptyset \text{ and best-effort } \epsilon \text{ compliance, then} \tag{4.23}$$

$$rg_i = |p(q) - p(u)|, \text{ otherwise } rg_i = |p(q) - p(u)| + |p(u) - cen_i|\}. \tag{4.24}$$

The first expression reflects the case where a subscription is relayed to the child EBs. This can potentially result in the formation of more direct (and shorter) event forwarding paths from the publishers to the subscriber q. The path is estimated as a straight line from the publishers to q (see the dashed lines in Figure 4.11). The second and third expressions capture the situation when the subscription is registered, and the events go through the addressed EB u to reach the subscriber q. This lengthens the event forwarding path such that events need to go through the EB u to reach the subscriber q, i.e. u intermediates the connection (see the solid lines in Figure 4.11).

For every $i \in \{1, \cdots, 4\}$, if $\frac{rg_i}{rl_i} \leq \epsilon$ then the resulting event forwarding path is sufficiently short and the subscription is registered, $\Theta = \Theta \cup \{(v, r_q \cap s_i(v), \epsilon_i) | \epsilon_i = \frac{\epsilon.|p(q)-cen_i|-|p(q)-p(u)|}{|p(u)-cen_i|}\}$. Otherwise, the subscription is relayed to the child EBs $c(u)$, $\Theta = \Theta \cup \{(c_i(v), r_q \cap s_i(v), \epsilon_q)\}$. If v is a leaf vertex on the QT (i.e. $c(v) = \emptyset$), then the subscription is involuntarily registered, $\Theta = \{(v, r_q, 0)\}$.

When guaranteed ϵ compliance is requested, shared paths are examined to ensure all related ϵ factors are satisfied. If the EB u registers the subscription (from subscriber q), then it must determine the permissible event forwarding path length (that is allowed by ϵ_q) from the publishers to itself. This path length is compared with the permissible path lengths that were previously granted by other subscribers in u's subscription routing table. If the permissible path length by ϵ_q is shorter than the previously granted path lengths by the registered subscriptions, then the covered subscription (whose forwarding path is shared) is renewed (un-subscribed and re-subscribed) with ϵ_q. This renewal reconstructs the shared path in a way that ϵ_q and hence all ϵ factors are satisfied. The permissible path length by ϵ_q (from the publishers to u) is computed as $\epsilon_q.|cen - p(q)| - |p(u) - p(q)|$, where cen is the centroid point of the covered region $r_q \cap cov(u)$.

Localized Unsubscription Resolving Algorithm. The localized unsubscription resolving algorithm R' handles unsubscription messages at the addressed logical EBs. It is like R in principle, but computes the Θ set differently. The epsilon factor ϵ_q is ignored and the unsubscription requests are resolved according to the entries in the addressed EB's subscription routing table. The algorithm unsubscribes the overlapping event region $r_q \cap cov^q(v)$ and relays the remainder, $r_q - cov^q(v)$, to the child EBs $c(v)$. It is formally expressed below.

$$R' \ (v \in G_{LV}, q \in G_{PV}, r_q \subseteq S, \epsilon_q \in \mathbb{R}) \mapsto \tag{4.25}$$

$$\{(v, r_q \cap cov^q(v), \epsilon_q)\} \cup \{(y \in c(v), (r_q - cov^q(v)) \cap s(y), \epsilon_q)\}. \tag{4.26}$$

4.3.5.5 Coverage Fulfillment Messages

When guaranteed coverage fulfillment is requested, coverage fulfillment messages are used to communicate the covered and uncovered subscription regions in QPS. Coverage fulfillment messages are initially produced by the publisher-hosting EBs, who register a subscription request (as discussed in the previous section). A coverage fulfillment message, generated by a publisher-hosting EB $u \in G_{PV}$ about a received subscription message (with subscription region r_v) from a subscriber $v \in G_{PV}$, contains the overlapping covered region $r_v \cap cov^v(u)$ and is sent to the subscriber, v. Coverage fulfillment messages, generated by the publisher-hosting EBs, reflect the *covered* subscription region; thus have the `fulfillBit` set to 1.

EBs that receive coverage fulfillment messages are divided into two groups: those that receive coverage fulfillment messages reflecting *covered* regions, and those that receive coverage fulfillment messages reflecting *uncovered* regions.

Receiving covered regions This group of EBs, $v \in G_{PV}$, have related advertisement entries in their advertisement routing tables, or are forwarding leaf EBs. They receive coverage fulfillment messages from the addressed publisher-hosting EBs, and aggregate these to obtain an accumulate covered region, R. This is compared against the registered subscription region for a subscriber $y \in G_{PV}$. If the accumulate region matches the registered region, $R = cov^y(v)$, then the subscription region is fulfilled and the operation is terminated. Otherwise, a new coverage fulfillment message is generated to reflect the *uncovered* region, $cov^y(v) - R$, and dispatched to the affected subscriber y, with the `fulfillBit` set to zero. The registered coverage region is also reduced to reflect the accumulate covered region, i.e. $cov^y(v) \leftarrow R$. EBs, $v \in G_{PV}$, usually set a timeout for examining subscription coverage after a registeration.

Receiving uncovered regions Coverage fulfillment messages that reflect uncovered regions are generated when event publishers fail or when there is not sufficient coverage for requested subscription regions. An EB $v \in G_{PV}$, who has received a coverage fulfillment message reflecting an uncovered region $r \subseteq S$, examines its subscription routing table and informs the affected subscribers about their unfulfilled subscriptions. The generated messages contain overlapping uncovered regions, i.e. $r \cap cov^y(v)$ where $y \in G_{PV}$ is a subscriber in v's subscription routing table. The `fulfillBit` is set to zero to indicate that the contained region is uncovered, and the message is directly dispatched to y. The registered coverage region for y is also reduced to reflect the covered subscription region, i.e. $cov^y(v) \leftarrow (cov^y(v) - r)$.

If a subscription region is not fully covered, then a coverage fulfillment message, reflecting this lack of coverage, arrives at the subscriber-hosting EB. The subscriber-hosting EB informs the event subscriber about the uncovered region r, using the `failed_coverage_fulfillment` callback function.

4.3.5.6 Publication Messages

Event publishers introduce events using the exported publish operation. The publisher-hosting EB wraps the event publication into an event publication message and dispatches it for dissemination over the EDT. The routing of publication messages is only controlled by the state in subscription routing tables. In contrast to the advertisement and subscription messages, where most target addresses related to logical EBs, event publications are addressed to real EBs. A Q_{bit} field controls the propagation of event publications along the EDT. I set $Q_{bit} = 1$ when the message is addressed to a forwarding EB, and $Q_{bit} = 0$ when it is addressed to a subscriber-hosting EB. This simple mechanism prevents cyclic event forwarding loops when forwarding EBs and subscriber-hosting EBs happen to be co-located on a single node, see Figure 4.12.

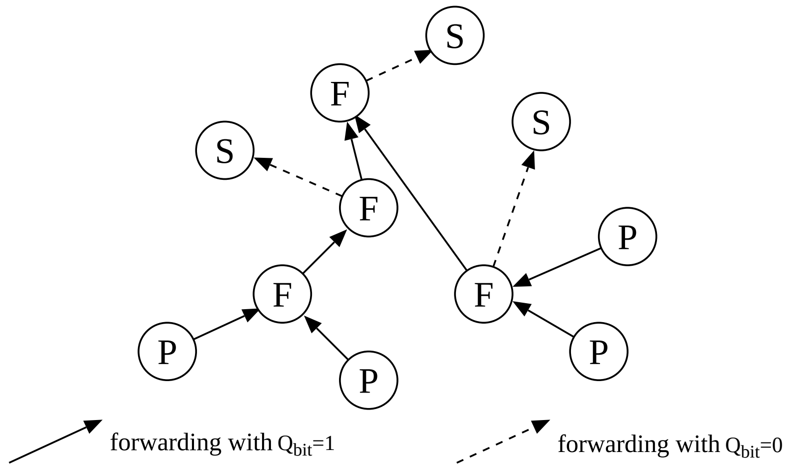


Figure 4.12: Event Publication Messages

More precisely, an event publication message (from a publisher-hosting EB $u \in G_{PV}$, reflecting an event e with location attribute value l_e) is initially dispatched, with $Q_{bit} = 1$, to all subscribers $v \in G_{PV}$, who have a matching subscription region, $l_e \in cov^v(u)$, and hold $(Q_{bit} = 0, \epsilon = 0)$ in u's subscription routing table - these entries are forwarding leaf EBs on the EDT.

A forwarding EB $y \in G_{PV}$ that receives an event publication, with $Q_{bit} = 1$, forwards the event to related subscribers, $\{z \in G_{PV} | l_e \in cov^z(y)\}$, with Q_{bit} settings that match their subscription entries (i.e. $Q_{bit} = 1$ for entries with $Q_{bit} = 1$ and $Q_{bit} = 0$ for entries with $Q_{bit} = 0$). An EB $y \in G_{PV}$ that receives an event publication, with $Q_{bit} = 0$, only forwards it to the local subscribers. Subscriber-hosting EBs deliver the events (to the locally connected event subscribers) using the `notify` callback function.

4.3.6 Resource-Awareness Model

The operation of a Pub/Sub protocol, in WSNs, can not be independent of nodal resources. Pub/Sub functionality has great impact on induced communication costs, which have been shown to greatly affect power consumption in WSNs (Section 2.3.1). The physical Pub/Sub layer, in QPS, supports a resource-awareness model that considers real-time nodal resources.

The resource-awareness model implements two services in QPS: *on-demand EB mapping* and *proactive hand-over*. When a logical EB $v \in G_{LV}$ is addressed (by its location address $l(v)$), the physical Pub/Sub layer performs a "search and map" operation that identifies a suitable EB $u \in G_{PV}$ for accepting the role of v, i.e. $map(v) = u$. Nodal resources, however, change over time; the proactive hand-over service acknowledges this by relieving overloaded or low-resourced EBs from their assigned roles, and initiating a fresh mapping for the logical EBs. The notion of *suitability*, at the physical Pub/Sub layer, describes the resourcefulness of an EB and its appropriateness for taking on the role of a logical EB.

Definition 4.11 (Suitability Function)**.** *A suitability function, suit, returns a real number, $o \in \mathbb{R}$, that reflects the suitability of an EB $u \in G_{PV}$ for taking on the role of a logical EB $v \in G_{LV}$,*

$$suit(u, v) \mapsto o \in \mathbb{R}. \tag{4.27}$$

A larger o value means that u is more suited to taking on the role of v. This function is implemented by the system designer, and is assumed to take account of nodal resources, as well as u's closeness to the location-based address of v, i.e. $o \propto \frac{1}{|p(u)-l(v)|}$.

4.3.6.1 On-demand Mapping

Logical EBs are mapped to real EBs on demand basis. References to logical EBs are realised when publisher- or subscriber-hosting EBs dispatch advertisement or subscription messages as discussed in Section 4.3.5. These messages are destined for the location-based addresses of logical EBs, $l(v \in G_{LV})$, which are unique and often do not relate to any real node addresses, i.e. $\forall v \in G_{LV}.\nexists u \in G_{PV} : l(v) = p(u)$.

The on-demand mapping service operates in two steps: *search* and *assignment*. The first step aims to identify a previously mapped EB or a suitable EB that can adopt the role of the addressed logical EB. If a previously mapped EB is not found, then the EB with the highest suitability value (found in the process) is selected for mapping. The latter step ends the search process and routes the message to the mapped EB.

Search. The search process exploits the multi-hop nature of the routing process in WSNs. The service also uses the `peek` callback function to see and modify any Pub/Sub messages that are intercepted by the routing protocol. A `candidateEB` field, that piggybacks the advertisement and subscription messages, contains the last most suitable EB $u \in G_{PV}$ that is found for taking the role of the addressed logical EB $v \in G_{LV}$. The suitability value of u, $o_u = suit(u, v)$, is reflected in the `suitabilityFactor` field.

Every EB $y \in G_{PV}$ that encounters the message, examines its own suitability value, $o_y = suit(y, v)$, and compares this against o_u (the `suitabilityFactor` field in the message). If higher, $o_y > o_u$, then y replaces the `candidateEB` and `suitabilityFactor` fields with y and o_y; otherwise, the message is intact. The EB y then returns the message to the routing protocol for continued routing towards the destination address. An encountered EB is excluded from the search if it is already assigned the role of another logical EB from the same QT (i.e. it excludes itself by skipping the comparison).

Since the destination (logical EB) address most likely does not relate to any real node, $l(v) \neq p(w \in G_{PV})$, most routing protocols perform an extended search operation to ensure the absence of voids or local maxima. This procedure further enhances the search results for a suitable EB, as the message passes through some nodes in the vicinity of the destination address.

Assignment. The assignment process follows or interrupts the search process. The search process is interrupted if a previously mapped EB $u \in G_{PV}$ for $v \in G_{LV}$ is found at an encountered EB's (y's) EB mapping table. The only exception to this is when the `handoverEB` field is set to u (explained later in Section 4.3.6.2). Alternatively, the search process continues until the message is routed and finally delivered to what is often the *geographically nearest* node, $y \in G_{PV}$, to the destination address, $l(v)$, i.e. $\forall w \in G_{PV} \ |p(y) - l(v)| \leq |p(w) - l(v)|$.

If an already mapped EB for v is not found, then the most suitable EB, u, indicated by the `candidateEB` field of the message is assigned the role, i.e. $u = map(v)$. This mapping is registered in y's EB mapping table, and the message is dispatched to u's address, $p(u)$.

4.3.6.2 Proactive Hand-Over

Forwarding EBs are subject to more communication and storage costs than other EBs. For example, these brokers maintain advertisement and subscription entries in their routing tables, receive, replicate, and forward events from publishers to subscribers. Prolonged operation of these EBs can lead to their early failure. Resource consumption must be balanced across the EBs, and in order to achieve this an active policy is required to relieve mapped EBs from their duties and utilize other available EBs in the network. The physical Pub/Sub layer implements a proactive hand-over service that achieves this functionality.

The hand-over service is independent of the Pub/Sub functionality, periodically compares the current suitability value of the EB against its original value (when it was appointed as the forwarding EB), and shifts the role to another EB if the compare function returns *true*. If the function returns *false*, then a heart-beat signal is sent to the logical EB's address to refresh the EB mapping tables along the path - a similar approach is used in GHT [RKY+02] to maintain DCSs points.

Definition 4.12 (Suitability Compare Function)**.** *A suitability compare function, compare, takes the original suitability value* o_{original} *(at the time of logical to physical EB mapping) and the present suitability value* o_{present}*, and determines if a hand-over operation should be performed,*

$$compare(o_{\text{original}}, o_{\text{present}}) \mapsto h \in \{true, false\}. \tag{4.28}$$

If the function returns true*, then a hand-over operation is deemed necessary, otherwise it is not.*

The hand-over service uses the discussed on-demand mapping service to achieve its functionality. It envelopes the assigned operations into original messages, and triggers a new mapping for the logical EB. The operation can be described in three steps: *role transfer*, *parallel subscribe/unsubscribe*, and *clean-up*.

Role transfer. An EB $u \in G_{PV}$ has sufficient information in its advertisement and subscription routing tables to re-generate the original messages intended for the logical EB $v \in G_{LV}$. These messages reflect the operations that u was assigned to perform in the role of v. By releasing these messages into the network, u triggers a fresh mapping of the logical EB v to some other EB. These messages differ from their originals, in that the `handoverEB` field is set to u. This signals any EB that encounters the messages to remove u from its EB mapping table. The role transfer is complete when a new mapped EB $y \in G_{PV}$ for logical EB v is found.

Parallel subscribe/unsubscribe. Advertisement and subscription messages, received at the newly mapped EB y are treated as explained in Section 4.3.5. While advertisement messages are simply inserted in y's advertisement routing table, the subscription messages trigger publish-subscribe link formations that connect y to the higher and lower EBs on the EDT. Subscription messages are generated to register y at lower EBs. In this phase, they are also accompanied by unsubscribe messages that remove u (the previously mapped EB) from the lower EBs' subscription routing tables. The unsubscribe messages are generated by y on behalf of u and sent with the subscription messages for parallel processing. A parallel (atomic) subscribe/unsubscribe operation is necessary for transactional hand-over, i.e. to ensure that no events are missed or duplicated during the hand-over process. The event service aims to transparently shift the responsibilities of u to the newly mapped EB y. This step is completed when the subscribe and unsubscribe messages are resolved over the QT.

Clean-up. The previously mapped EB u may erase advertisement entries, immediately after dispatching the re-generated advertisement and subscription messages. The subscription entries, however, may only be removed after the parallel subscribe/unsubscribe process. u may remove these entries after a timeout period (best-effort approach) or wait for a confirmation message from the newly appointed EB y.

4.3.7 Reliability Model

WSNs exhibit moderate network and nodal dynamics. A reliable Pub/Sub protocol should maintain correct functionality in the view of these dynamics, which can be categorized into: *network dynamics* and *component dynamics*. Network dynamics affect other architectural layers in QPS, while component dynamics directly affect the Pub/Sub functionality. These are discussed separately below.

4.3.7.1 Network Dynamics

Sensor systems are subject to high network topology changes. The primitiveness of devices not only means that nodes can fail, but also suggests that re-deployments are very likely. These introduce frequent node join and leave operations that are handled by the network layer. The network layer maintains unique addresses for nodes in the system at all times. In addition, the unreliable nature of wireless communications impacts link-layer connectivity, and can lead to disconnections or packet losses during transmissions. The location-based routing layer takes charge in repairing failed routes and retransmitting failed packets. The routing layer is assumed to maintain a reliable operation, where dispatched messages are not permanently lost, and all dispatched messages are eventually delivered. Combined reliability, at the networking and routing layers, ensures a reliable Pub/Sub operation in the face of these network dynamics.

QPS neither supports extended periods of network disconnectivity (network partitions) nor does it support frequent node mobility (MANETs). Irregular node mobility or shift in nodal positions, however, may be imitated by a sequence of component leave and join operations that are discussed in the next section. This work-around becomes costly when the rate of mobility increases in the network.

4.3.7.2 Component Dynamics

Node failures, join or leave operations also affect the components in QPS. The physical Pub/Sub layer ensures that reliable Pub/Sub functionality is achieved at the logical Pub/Sub layer despite moderate component dynamics. I examine component dynamics separately and discuss their impacts on the EDT.

EC Component Dynamics. Publisher-hosting EBs manage event publisher dynamics, and subscriber-hosting EBs manage the event subscriber's. In both instances, a join operation is covered by the standard Pub/Sub functionality: an event publisher calls the `advertisement` function, and an event subscriber calls the `subscription` function.

Their leave operation is also catered for by the `unadvertise` and `unsubscribe` functions, but uncalled leaves (e.g. EC failures) must be treated differently. The publisher-hosting EB unadvertises *on behalf of* the event publisher and the subscriber-hosting EB unsubscribes *on behalf of* the event subscriber, when ECs leave without calling the `unadvertise` or `unsubscribe` functions. Uncalled leaves may be detected through periodic interactions between the EBs and their local ECs.

EB Component Dynamics. Node failures, join or leave operations directly affect the EB dynamics. EB joins are simple: they require no operation because they operate on demand basis. EB failures and leaves, however, are complex and in fact indistinguishable in QPS. Thus, studying EB dynamics can be summarized into analysing EB failures. These failures are best discussed in relation to the EB roles: *publisher-hosting EB*, *subscriber-hosting EB*, and *forwarding EB*.

Publisher-hosting EB Publisher-hosting EB failures are assumed to entail the failure of connected event publishers as well. The failure removes publishers, along with their publish-subscribe links (stored at the publisher-hosting EBs' routing tables), from the EDT. The failure may be neglected if best-effort fulfillment policy is active, otherwise the failure must be detected by the immediate forwarding EBs on the EDT. For this, periodic heartbeat signals can be used from the publisher-hosting EBs to the immediate forwarding EBs. The failure may be intermittent, but if it persists then the forwarding EB removes the region (covered by the failed EB) and re-examines the subscription coverages as detailed in Section 4.3.5.5.

Subscriber-hosting EB If a subscriber-hosting EB fails, then I similarly assume that the corresponding event subscribers have also failed. This failure is detected when the routing protocol can not reach the failed EB, to deliver an event notification (with $Q_{bit} = 0$). Instead, the routing protocol delivers it to a non-destined EB. The recipient (EB) stores the notification and aims to forward it to the intended EB in the future, but if failure persists then it can dispatch an unsubscription message *on behalf* of the failed subscriber-hosting EB to the immediate forwarding EB.

Forwarding EB Forwarding EB failures greatly affect the EDT and compromise Pub/Sub functionality. Failures are noticed when Pub/Sub messages are delivered to non-destined EBs. The recipients are often the nearest nodes (to the failed EBs), where the routing protocol realises that no closer node to the destination address can be found. These recipients can operate in place of the failed EBs, if the routing tables of the forwarding EBs were replicated at the nearby nodes. QPS replicates the forwarding EBs' routing tables at nearby EBs, and synchronizes them on demand or periodic basis, whichever is less frequent. Nearby nodes can operate temporarily in the role of the forwarding EB, when the intended forwarding EB has intermittently failed; but if the failure persists, then the nearby EB can initiate a hand-over operation (discussed in Section 4.3.6.2) on behalf of the failed forwarding EB. The hand-over operation removes the failed EB from the EDT and replaces it with a new forwarding EB.

4.4 Evaluation

QPS can be evaluated with performance measurements using a real, deployed sensor system or with experiments in a simulator. Although an actual deployment results in a more realistic evaluation, it is more difficult to instrument since a large-scale distributed system has substantial resource requirements. Instead, I decided to set up experiments in a distributed systems simulator that support simulations with large number of nodes and QPS EBs. The goal of this evaluation is to quantify and assess the contributions of QPS against cross-layer data dissemination protocols that are widely used in sensor systems. An adaptation of GHT is also used for examining simple DHT-based data dissemination in sensor networks. In the next section I describe the metrics that were used to evaluate QPS against the other protocols.

4.4.1 Evaluation Metrics

The metrics used for evaluation were selected according to the stated claims and objective of the protocols. Attained path freedom and path sharing in Pub/Sub protocols were compared, and other metrics were introduced to examine the performance of the protocols. Three metrics were used to quantify the evaluations.

Path freedom Path freedom quantifies the freedom that a message has in taking a route from publishers to subscribers. This freedom is measured by the level of restriction that is imposed on event routing by the formed EDT. These restrictions are the intermediate forwarding EBs that are neither publisher-hosting nor subscriber-hosting, but events need to pass through them to support Pub/Sub functionality or path sharing. The *number of publish-subscribe links* in an EDT is a representative of these restrictions, which limit path freedom. Where the total numbers of links are similar, the number of links between publisher and subscriber pairs can be studied for more accurate quantification.

Path sharing Path sharing was motivated, due to its positive impact on communication savings and messaging performance. Quantifying path sharing on its own reveals very little about the overall efficiency and performance of the Pub/Sub protocol; thus I decided to measure this by examining the *overall messaging cost*. The overall messaging cost is inclusive of path sharing effects, and presents a more meaningful information about the efficiency and performance of the Pub/Sub protocol.

Maximum load (per node) Distribution of event dissemination loads across nodes is key to network survival in sensor systems. Pub/Sub protocols need to distribute this load to avoid early node failures. The load may be measured by the number of subscribers that each EB supports; and may be accurately quantified by the *number of publish-subscribe links* that are registered at the EB's subscription routing table. These links induce a messaging cost that is nearly proportional to the number of registered publish-subscribe links in the table.

4.4.2 Simulation Environment

I used the discrete event simulator JiST/SWANS [BHvR05] as my simulation testbed. SWANS, developed at Cornell university, is a scalable wireless network simulator built on top of the JiST platform. It implements a data structure, called hierarchical binning, for computation of signal propagation, which is more efficient than linear searches, used in ns-2 [NS2] and GloMoSim [ZBG98]. SWANS is organized as independent software components that can be composed to form complete wireless network or sensor network configurations. The following describes the software components that were used to set up the simulation environment.

Physical Topology. Every experiment had a unique physical topology that described the placement of nodes on to a two dimensional square grid environment. The nodes were randomly placed, using a uniform distribution function, and corresponded to a unique experiment number. A network connectivity test was used to discard physical topologies that led to network partitions. The connectivity tests were based on radio parameters that were adopted from the existing platforms, described next.

Radio Configuration. I adopted wireless parameters from the CC1000 radio [CC1], that is in use on the BTnode [BT] platform, on Mica motes [MIC], and many other platforms. Table 4.5 provides the details. These parameters affect the link-layer connectivity, packet transmissions and receptions in the simulation environment.

Link layer protocol. A contention-based MAC protocol was used for link-layer communication. CSMA/CA was implemented by a sequence of RTS-CTS-Data-ACK messages, as in the standardized IEEE 802.11 Distributed Coordination Function (DCF) [LAN97]. This design originated from the MACAW protocol [BDSZ94], and is widely used among the WSN MAC protocols (e.g. S-MAC [YHE02; YHE04] and PAMAS [SR98]) due to its simplicity, reliability, and robustness.

Network Layer. The network layer assigned location-based addresses to the nodes. Mirroring the grid environment, a 2-D location coordinate system was used and nodes were assigned

Parameter	Value	Unit
frequency	868	Mhz
bandwidth	38.4	kbit/s
transmit power	5	dBm
antenna gain	0	dB
sensitivity	-96	dBm
rcv. threshold	-84	dBm
interference limit	-96	dBm

Table 4.5: Wireless Radio Parameters

locations based on their placement within the grid environment. These location coordinates were fixed and independent of the network topology.

Location-based routing protocol. A GPSR-like routing protocol was implemented to provide location-based routing functionality as described in Section 4.3.2. The routing protocol used greedy forwarding combined with perimeter routing to reach destination nodes. The greedy forwarding policy is most commonly used in location-based routing protocols, and yields the shortest communication path between sources and destinations.

Event Dissemination (Pub/Sub) Protocols. QPS was compared against two event dissemination protocols: a cross-layer event dissemination protocol and a GHT-based event dissemination protocol. These protocols are described below.

Cross-layer Opportunistic-sharing Data Dissemination (CODD) I selected the SAFE [KSS+03] data dissemination protocol as my cross-layer Pub/Sub protocol. It operates like a Pub/Sub protocol, in that it distributes requests like event subscriptions, forms data dissemination paths like an EDT, and forwards data like event dissemination in Pub/Sub. It is designed for location-aware WSNs and supports opportunistic path sharing along the lowest latency links. Essentially, queries are dispatched into the network, which might reach the existing data dissemination paths or the publisher at multiple points. Acknowledgements are then forwarded back to the upstream node, containing information about the communication cost of the intercepted link. These links are analyzed by the subscriber to re-enforce the path that leads to the least communication cost (minimum number of messaging hops) for event delivery. SAFE [KSS+03], however, assumes that only a single source (publisher) corresponds to every subscription, and requires the employment of a geographical broadcast protocol like GEAR [YGE01] to support subscription regions. I implemented the combination as the CODD Pub/Sub protocol. The protocol also resembles the one-phase pull diffusion protocol [HSE03] (discussed in Section 2.3.1), but with the advantage of not needing negative re-enforcements. CODD may be considered equivalent to the combination of one-phase pull diffusion protocol, opportunistic path sharing, positive re-enforcements, and the GEAR protocol.

Parameter	Description	Value
i	number of experimental runs	30
g	simulation grid size	256 × 256
n	number of physical nodes in the grid	500
n_P	number of event publishers	n (one per node)
n_T	number of event topics	1
n_E	number of event publications	50
n_S	number of event subscribers	75
n_s	number of event subscriptions	n_S (one per subscriber)
r_s	event subscription region	(64,64)–(128,128)
t_s	time interval between event subscriptions	3000 ticks
n_B	number of QPS EBs	n (one per node)
N	number of QPS hierarchical GS levels	4
ϵ_s	QPS event subscription ϵ value	1.5
f_H	QPS and GHT hash function algorithms	ELF hash function

Table 4.6: Simulation parameters

GHT Dissemination (GHTD) GHTD is an event dissemination protocol, implemented over GHT. GHT was originally proposed for DCS points [REG+02] in sensor networks. These storage points are identified according to hash values, similar to how the logical Pub/Sub layer defines logical EBs in QPS. I implemented GHTD by means of using the DCS points as *rendezvous* points for interconnecting event publishers and event subscribers. Essentially, event advertisements and event subscriptions meet at the rendezvous point and set up an EDT that forwards events from publishers to the rendezvous node and then to the subscribers. Event subscriptions that overlap are joined at the rendezvous points.

4.4.3 Experimental Setup

All experiments that will be described in the next section were carried out in the discussed simulation environment. Each event dissemination protocol was studied in isolation, with identical physical topologies and Pub/Sub clients. Experiments examined the impact of different physical topologies and changing Pub/Sub demands, with reference to the highlighted evaluation metrics.

My experimental strategy was to keep parameters fixed, and vary one parameter at a time to study its impact on the overall EDT formation and Pub/Sub performance. Table 4.6 lists all the simulation parameters. Each experiment was repeated $i = 30$ times to obtain a converged arithmetic mean value and 95% confidence interval. Where confidence intervals were negligible, error bars are eliminated from the diagrams to avoid cluttering. Network sizes of 300 and 350 nodes are frequently used to evaluate performances in related work (e.g. in Directed Diffusion [IGE00; IGE+03]). I chose a slightly larger network size, $n = 500$ nodes, to represent a sensor network that can be shared by many applications – sensor networks supported by Directed Diffusion are application-specific and thus expected to be smaller. This size, however, was later varied from 150 to 1000 nodes to examine the effects of density on performance. A grid size of 256 × 256 ensured sufficient node density to prevent network disconnections.

Each experiment was concluded when $n_E = 50$ events were successfully delivered to $n_S = 75$ event subscribers in the network. These numbers resulted in converged arithmetic mean values for each experiment. The event subscription region, r_s, was defined away from the grid corners, the grid boundaries, and the center of the grid. This decision was made to eliminate unrepresentative performance measurements that could be due to the grid structure or boundaries. Finally, a non-zero time interval, t_s, between the event subscriptions simulated a real-world setting in which users expressed their interests at different times and yet their subscriptions could be resolved concurrently over the network. The parameters listed in Table 4.6 were kept fixed, unless stated otherwise in the experiments.

In order to compare equivalent functionality between the Pub/Sub protocols, event publishers were assumed to publish events about their locality (as commonly assumed in related work), i.e. $\forall e \in \mathbb{E}\ l_e = l_{e^p}$ where l_e is the location attribute of the event publication and l_{e^p} is the location of the event publisher. Furthermore, since this work focuses on the EDTs, which are constructed at a higher level than the physical network, the performance of the examined Pub/Sub protocols were compared under optimal conditions. These conditions were defined by no node failures, and sufficient resources at all nodes to participate in the EDT. This means that the physical Pub/Sub layer (in QPS) was not used.

I did not compare the performance of QPS with physical Pub/Sub layer functionality against CODD and GHTD, because neither SAFE [KSS+03], nor one-phase pull diffusion [HSE03], nor GHT [RKY+02] support resource-awareness. Nonetheless, since the operational semantics of the QPS physical Pub/Sub layer is similar to that of "home node" and "perimeter refresh protocol" in GHT, the performance evaluations of GHT can be taken as indicative of physical Pub/Sub layer's performance in QPS. Briefly, those evaluations indicate that the on-demand EB mapping service induces no additional communication costs than what is incurred by the routing protocol for delivering event subscriptions and event notifications, the communication cost of the hand-over service linearly grows with an increase in the frequency of suitability value comparisons until successive comparisons overlap, and that success rates[1] of about 94.7% can be achieved when nodes undergo cycling failures on the order of every 6 minutes.

Experiments began by the random placement of nodes in the simulation grid, and address assignments by the network layer. The routing protocol updated its tables using local (`HELLO`) messaging, and Pub/Sub functionality was only initiated after 10,000 simulation clock ticks. Subscribers invoked their event subscriptions independently and in turn, with t_s delay in-between them to emulate dynamic and distributed subscriptions. Events were generated randomly (with uniform distribution across time and space) and were published independently by the event publishers in the system. The next section describes the experiments and their observed results.

[1] Success rates indicate chances of events reaching replica routing tables when forwarding EBs have failed.

4.4.4 Experiments

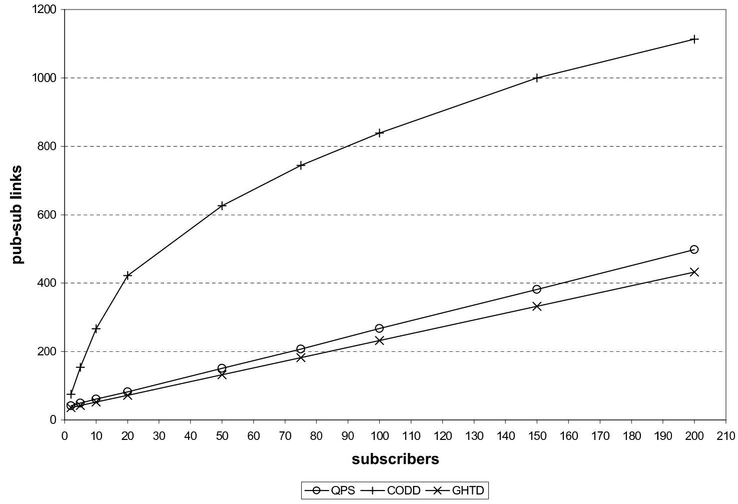
The experiments aimed to study the operation and performance of the Pub/Sub protocols under different conditions. Specifically, I was interested in factors that affected the path sharing and the EDT construction in a non-linear manner. Factors such as the number of event topics, number of event publications, size of the network (grid size), and the size of the subscription region affect the EDTs and the Pub/Sub performance in an independent or near linear manner. These factors were maintained fixed, and normalised when studying the evaluation metrics. The number of subscriptions, however, had a strong impact on EDTs and path sharing. I studied the Pub/Sub protocols under varying numbers of event subscriptions (from 2 to 200, accounting for almost half of the nodes in the network). I also realised that the number of nodes (hence the density of the network) has a strong impact on some Pub/Sub protocols, therefore Pub/Sub protocols were also examined under varying network density (from 150 nodes to 1000 nodes in the grid). Finally, the impact of varying the ϵ factor alone was studied in QPS.

4.4.4.1 Number of subscribers

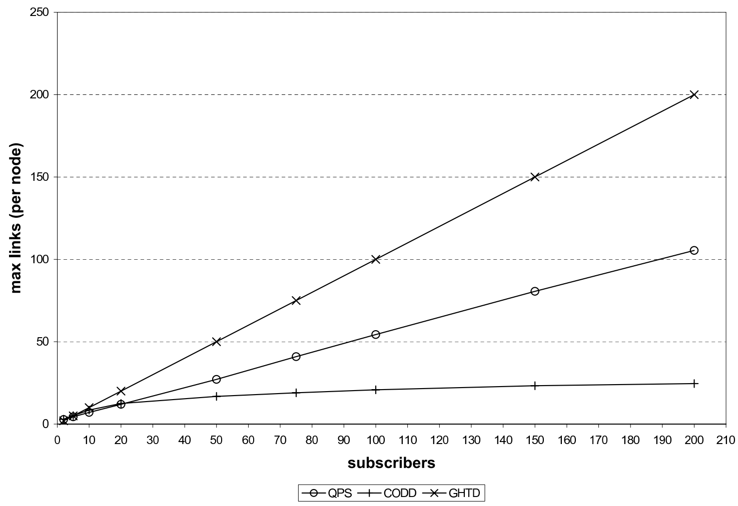
In my first set of experiments, I varied the number of event subscribers from 2 to 200 while keeping other parameters fixed (see Table 4.6). As the number of subscribers increased, the event forwarding paths could be shared among a larger number of event subscribers, therefore allowing for greater communication savings. These savings were partly balanced against losses that were inherent in the non-optimal nature of shared paths. An optimal static EDT requires knowledge about all subscribers and the network topology, neither of which are usually known in advance.

Experiments were repeated i times with different physical topologies, and statistics were gathered after successfully delivering n_E event publications from publishers to subscribers. The average number of publish-subscribe links that was observed in EDTs, constructed by each Pub/Sub protocol, is shown in Figure 4.13(a). These measurements also include links from subscriber-hosting EBs to their local event subscribers. We see that the number of publish-subscribe links in QPS and GHTD demonstrate a strong linear relationship with the number of subscribers. They resolve subscriptions at a set of designated nodes, which are independent of the subscribers. These nodes are the rendezvous nodes in GHTD and logical EBs in QPS. Following the resolution of the first subscription in QPS and GHTD, subsequent subscriptions often resulted in just two additional publish-subscribe links; one at the subscriber-hosting EB and the other at the rendezvous or logical EB. The number of publish-subscribe links in QPS always exceeded that of the GHTD by a small amount - the difference related to the operation of the localized subscription resolving algorithm, which decomposed and relayed subscriptions when the addressed logical EBs were not deemed suitable for participation on the EDT.

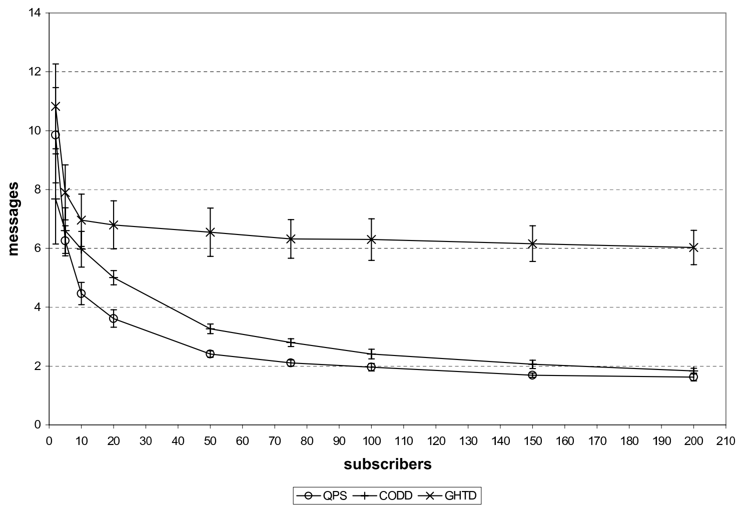
The publish-subscribe links constructed by the CODD protocol, however, showed a different trend to the previous two protocols. The initial value started higher than that of QPS's and GHTD's, due to subscription registration and link formation at every intermediate node. The number of links increased rapidly with an increase in the number of subscribers, demonstrating a



(a) Publish-subscribe links



(b) Maximum publish-subscribe links



(c) Dissemination complexity

Figure 4.13: Varying the number of subscribers (2. . . 200)

near linear relationship and hinting about few subscription intersections and joins in the network. But as the number of publish-subscribe links increased so did the chances of subscription joins - this was noticed by the reduced rate of increase in the number of publish-subscribe links when the number of subscribers exceeded 50. The large number of publish-subscribe links that were formed by CODD, compared to QPS and GHTD, significantly diminished path freedom.

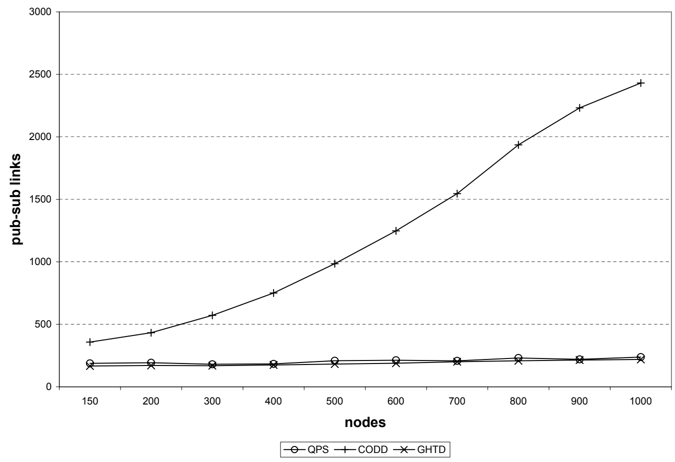
When examining dissemination complexity, the GHTD protocol demonstrated low performance with significant variability. Figure 4.13(c) shows the normalized costs (i.e. messaging cost per event publication per subscriber) that exclude messaging overheads induced by the lower communication layers (the routing, the network, and the link-layer protocols). All protocols demonstrated significant variability for lower numbers of subscribers. This is due to the random distribution of subscribers, which at higher numbers converge to a uniform distribution but at lower numbers show significant variability depending on the placement. In comparison, the QPS protocol performed remarkably better and with lesser variation than GHTD. This was largely due to the QPS localized subscription resolving algorithm, which eliminates inappropriate EBs from the EDT.

The CODD protocol constructed more direct paths but benefited little from path sharing. At the start, the small number of subscribers meant that there were fewer chances for path intersection and subsequent path sharing. Later, as the number of subscribers and subsequently the number of publish-subscribe links were increased, path intersection and path sharing became more frequent and the gap between the CODD and QPS performances were closed. One key area where the CODD protocol is a clear winner is load balancing. The opportunistic nature of the protocol results in a more uniform distribution of Pub/Sub load across the network. In contrast, the GHTD and QPS protocols impose high loads on few nodes (rendezvous nodes in GHTD, and logical EBs in QPS). Figure 4.13(b) shows this with a plot of the maximum publish-subscribe links that were registered at any one node in each of the Pub/Sub protocols. QPS still performs half as good as GHTD due to load distribution across a wider set of logical EBs. This diagram motivates the need for resource-awareness at the Pub/Sub, which has been developed as the physical Pub/Sub layer in QPS.

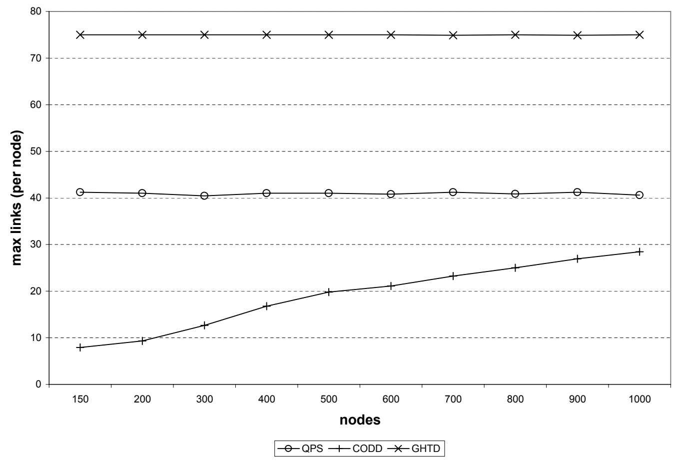
4.4.4.2 Number of nodes

Varying the number of nodes, thereby the network topology and density, impacts the routing protocol functionality and performance, but should have little impact on Pub/Sub functionality. Due to the strong correlation between the routing performance and the Pub/Sub performance, the impact of the node density on the routing performance is partially realised when examining the Pub/Sub dissemination complexity.

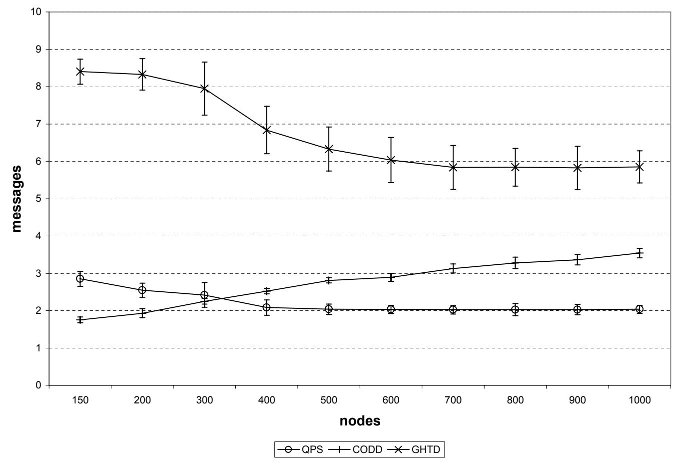
I varied the number of nodes from 150 to 1000 in another set of experiments, while keeping other parameters fixed as in Table 4.6. This affected the node density and the network topology, such that the average number of neighbors for any single node varied from 3 to 33 nodes. We can see in Figure 4.14(c) that increased density has had a positive impact on the event dissemination performance of QPS and GHTD. As the number of nodes increased, shorter



(a) Publish-subscribe links



(b) Maximum publish-subscribe links



(c) Dissemination complexity

Figure 4.14: Varying the number of nodes (150. . . 1000)

Parameter	Description	Value
i	number of experimental runs	10
g	simulation grid size	512 × 512
n	number of physical nodes in the grid	2000
n_S	number of event subscribers	100
r_s	event subscription region	(16,16)-(32,32)
ϵ_s	QPS event subscription ϵ value	$\{1.0, 1.2, \cdots, 2.8\}$

Table 4.7: Altered simulation parameters

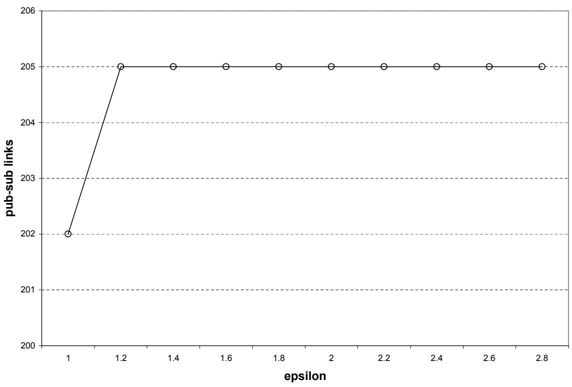
and straighter routing paths became available, and the theoretical estimations about the event forwarding paths in QPS became more realistic. The performance of CODD, however, decreased with an increase in the number of nodes in the network. A study of the publish-subscribe links indicated the reason.

With an increase in the network density, the number of publish-subscribe links rapidly increased in CODD, while other Pub/Sub protocols were unaffected. This susceptibility was because of CODD's reliance on the routing protocol to create shared paths. Opportunistic sharing requires two or more event forwarding paths to intersect prior to reaching the publishers' nodes. With lower density, the routing protocol was restricted in the set of forwarding nodes, and hence there was a great chance of intersection if subscribers were geographically close to one-another. As the network density increased, so did the number of nodes that could route events from publishers to subscribers. Thus, the event forwarding paths had lower chances of intersection as subscription messages could be routed on alternative paths from the subscribers to the publishers. This reduced path sharing in CODD, and increased the cost of event dissemination as shown in Figure 4.14(c). These findings are consistent with results presented in [IEGH02]. The opportunistic sharing scheme is evaluated against a greedy sharing mechanism in [IEGH02], and results indicated that the performance was roughly the same in low-density networks, but differed significantly (in favor of the greedy mechanism) at higher densities.

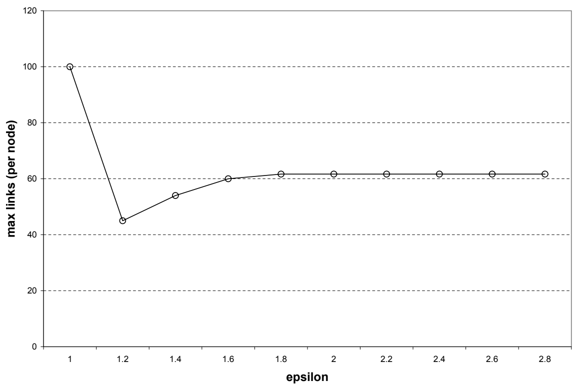
This set of experiments indicated that CODD performs better in sparse networks and QPS performs better in moderately dense to strongly dense WSNs. Path freedom is best achieved by GHTD and QPS, as before, and GHTD's maximum load (per node) is higher than QPS's as in the previous set of experiments. In these experiments, the increase in the maximum load in CODD indicated the many subscription messages that did not intersect prior to reaching the publishers' nodes.

4.4.4.3 Epsilon value

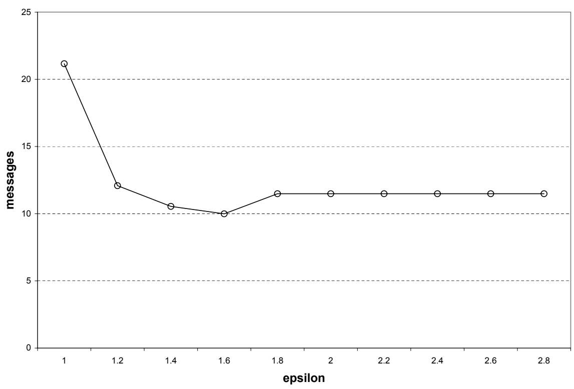
In the last set of experiments, I varied the subscription ϵ factor in QPS to observe its impact on the EDT and Pub/Sub performance. To observe this impact in great detail, I scaled the network size to $n = 2000$ nodes and enlargened the grid size to 512 × 512, maintaining a fixed



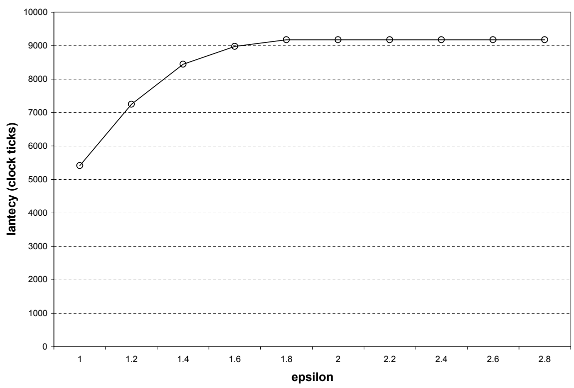
(a) Publish-subscribe links



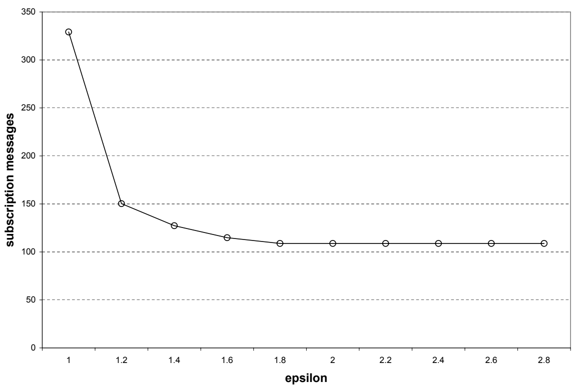
(b) Maximum publish-subscribe links



(c) Dissemination complexity

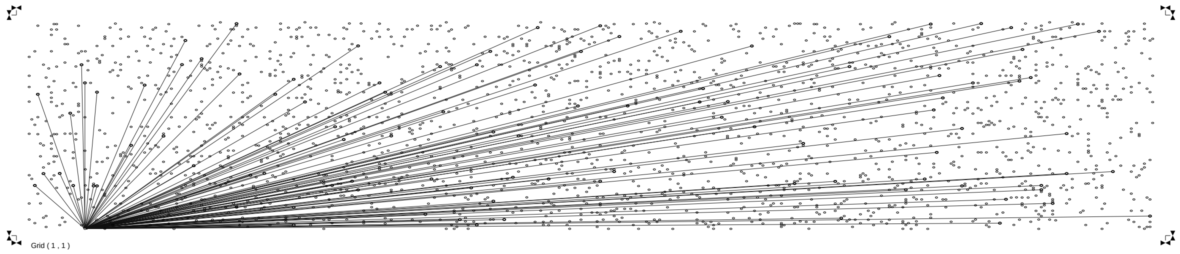


(d) Event delivery latency

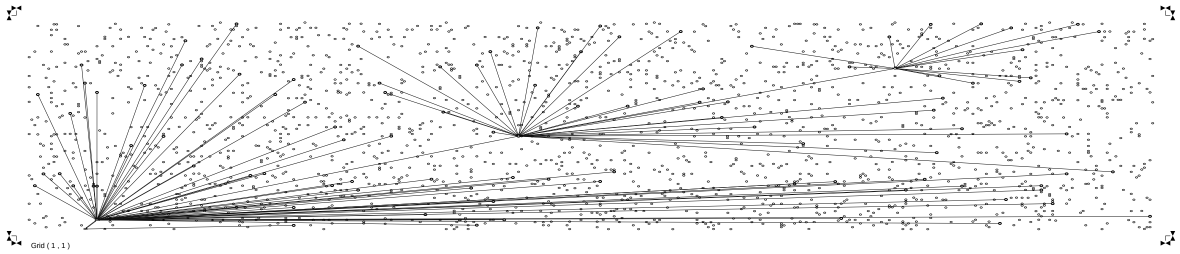


(e) Subscription complexity

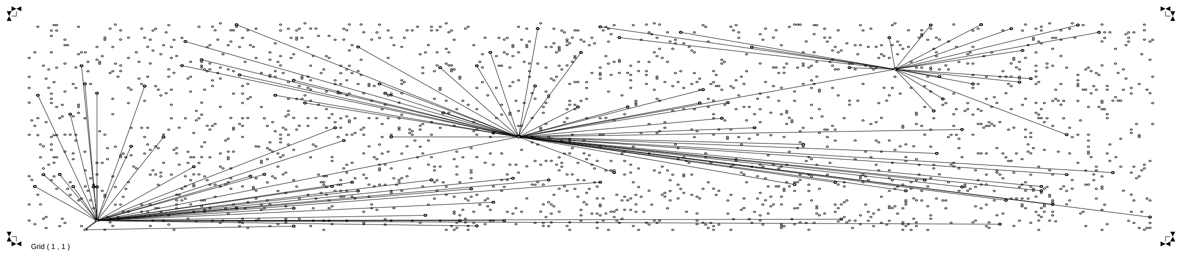
Figure 4.15: Varying the subscription ϵ value



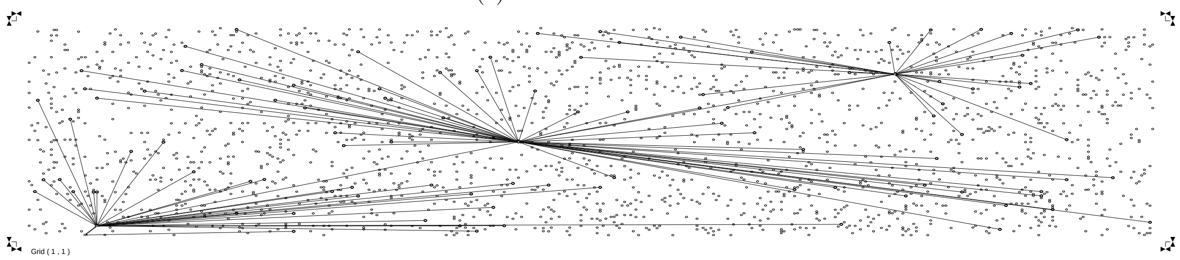
(a) EDT with $\epsilon = 1$



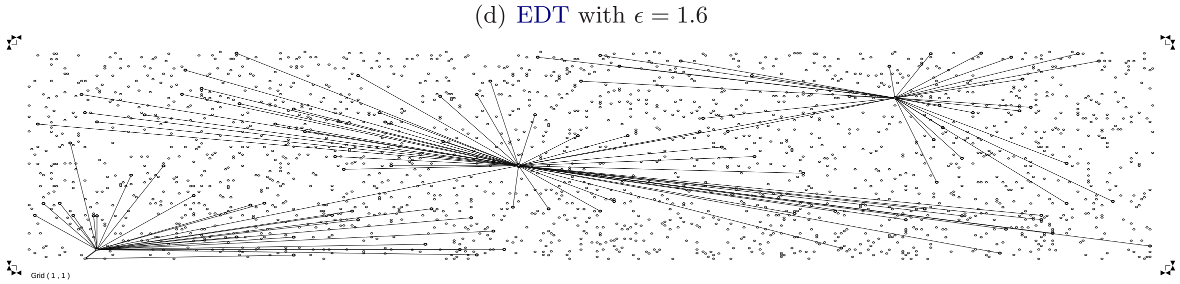
(b) EDT with $\epsilon = 1.2$



(c) EDT with $\epsilon = 1.4$



(d) EDT with $\epsilon = 1.6$



(e) EDT with $\epsilon = 1.8$

Figure 4.16: QPS EDTs with varying ϵ value

network density (see Table 4.7). This scaling allowed a wider range of path length ratios to be examined by the subscription ϵ factor than those that could be examined in a 256×256 grid size network. Thus, the performance trend could be observed more clearly even by small subscription ϵ factor increments. The performance of QPS under both network sizes ($n = 500$ and $n = 2000$ nodes) was studied in a separate set of experiments, and verified to be consistent. In this set of experiments, the subscription ϵ factor was varied from 1.0 to 2.8 (with 0.2 increments), and the event subscription complexity and event delivery latency were examined in addition to the previous metrics. Figure 4.15 shows the average results after i experiments, and Figure 4.16 shows a typical result (QPS EDT) for varying the subscription ϵ value from 1.0 to 1.8; the dots represent the nodes, the lines represent the publish-subscribe links, and the bottom-left corner of the diagrams represent the (0,0) location coordinate. Note that the original diagrams have been scaled down unevenly in x and y; distances have been distorted but (node) order is preserved.

The subscription complexity in Figure 4.15(e) was measured according to the number of subscription messages that were transmitted in the network until all subscriptions were resolved. The difference between this value and the dissemination complexity is that the subscription complexity is a *cumulative* number, while the dissemination complexity is a *normalised* one. As expected, the subscription complexity decreases (i.e. subscriptions are resolved quicker and with lesser messaging) as the subscription ϵ value increases in QPS.

As Figure 4.15(a) shows, the number of publish-subscribe links increased with an increase in the ϵ value, but only for the first increment; the total number (subsequently) stayed constant because QPS established the same number of links but to different logical EBs. Figure 4.16 shows how the number of forwarding EBs between publisher-hosting and subscriber-hosting EBs increased while the total number of publish-subscribe links stayed the same. Originally, all subscriber-hosting EBs were connected to the leaf logical EB that is responsible for r_s (Figure 4.15(b) confirms this with a high (per node) load shown for $\epsilon = 1$). As the ϵ value increased, these links were distributed among the logical EBs that were closest to the subscriber-hosting EBs. Experimental results indicate that the performance stayed constant for $\epsilon > 1.8$. This is because QPS algorithms did not allow event dissemination paths to become unreasonably long even if a high subscription ϵ value was specified (Section 4.3.5.4 indicates that subscription messages are dispatched to the nearest logical EB). The threshold ϵ value, beyond which the performance remains constant, depends on the size of the network, g, and the number of GS hierarchy levels, N.

Figure 4.15(c) shows that the highest ϵ value (and hence the highest level of path sharing) does not lead to the best performance in QPS. This is despite the large number of overlapping subscriptions that is present in this setup. This is because dissemination complexity depends on more than just the event dissemination path lengths. An even distribution of EDT load is also important, and high ϵ values in QPS often result in more load at the higher level logical EBs than the lower level logical EBs - recall that higher level logical EBs handle larger GSs.

The event delivery latency diagram (Figure 4.15(d)) was plotted according to the time interval between the time of event publications and the time of event deliveries. The global simulation clock was used to measure these intervals accurately, but the experimental setting was altered as follows. In order to minimize the impact of the link-layer and routing layer performances on the event delivery latency, and grasp a better view of QPS's effect on the event delivery latency, event dissemination was restricted to one subscriber at a time (each event publication was delivered to one event subscriber, in isolation, and measurements were averaged to obtain the final latency value). With increasing ϵ value and longer event dissemination paths, the event delivery latency increased as expected but only under controlled conditions. Uncontrolled measurements indicated that event delivery latency strongly depends on link-layer performance, and (with possible network congestion) is largely non-deterministic. This is consistent with the observation made in [SI07] that timely delivery requires the cooperation of many communication layers, especially the data link layer.

4.5 Related Work

I discuss related work under three headings that group it by design, application setting, or implementation.

Cross-layer data dissemination protocols. I have already discussed how QPS compares to the cross-layer data dissemination protocols. In this section, I extend my comparison by *operational analysis*, *Pub/Sub abstraction*, and *functional design*. QPS is a complete and self-contained Pub/Sub protocol. This contrasts with some data dissemination protocols, such as Directed Diffusion [IGE00; IGE+03], that leave some design decisions to the system developer. While these protocols offer great flexibility (by involving the system developer), they risk correctness by relying on the human factor to make appropriate design decisions. QPS avoids this risk, yet delivers flexibility through a layered architecture. System developers may select different implementations for lower functional layers, or develop their own (routing and networking) protocols for added flexibility.

Another interesting distinction between QPS and the data dissemination protocols is the lack of positive or negative re-enforcements in QPS and its localized EDT construction; end-users (ECs) do not need to acknowledge or re-enforce paths for EDT construction. In contrast, most data dissemination protocols (such as Directed Diffusion [IGE00; IGE+03] and SAFE [KSS+03]) need end-user re-enforcements to select appropriate nodes for the EDT.

Finally, the Pub/Sub abstraction, that is provided by QPS and cross-layer data dissemination protocols, can be compared with respect to *location* and *time decoupling* [EFGK03] as follows. Where location-awareness is utilized in cross-layer data dissemination protocols (e.g. Directed Diffusion combined with GEAR [YGE01]), the location of the clients (sensors and sinks) is constrained as opposed to the location of the data. This has two consequences (see also Section 2.3.1.1): (a) location decoupling between ECs is lost, and (b) only data that corresponds to the location of the publisher is supported. QPS, however, only constrains the location attribute of the event (data); thus it does not have the above disadvantages. With respect to time decoupling, when a pull-based (Pub/Sub-like) data dissemination protocol [HSE03] is operating, time decoupling (analogous to real-time coverage in QPS) is not guaranteed; this is because EDTs (in pull-based data dissemination protocols) do not maintain ECs' dynamics, but are periodically re-constructed to exclude failed subscribers and include newly joined publishers.

Wireless ad-hoc network Pub/Sub protocols. As discussed in Section 2.1, wireless ad hoc networks are different from WSNs. Pub/Sub protocols that are designed for the former, often perform poorly in the latter - largely due to the lack of energy and load considerations. WSN applications also have different characteristics from those motivated in ad hoc environments. For example, in [HGM03], a single (*root*) node is assumed to be sufficient for publishing events in an ad hoc network; this is in clear contradiction to the WSN setting where the majority of nodes are publishers. The work-around solution (unicast all events to the root node and then disseminate them from there onwards) is equally inappropriate. Researchers have thus begun to migrate solutions from the wireless ad hoc network setting to the WSN setting. Examples include [CPR05] that is based on [CCP05], and [HCRW04] that is based on [CCRW04]. These efforts, however, do not exploit the notion of location that is frequently present in large-scale WSNs. QPS outperforms many of these protocols, either in operational settings or in operational performance, due to the use of location-awareness.

Although QPS uses the notion of location, the subscription language supports constraints over the location of the data and not the ECs. QPS can be considered as a content-based Pub/Sub protocol that makes assertions about the first two attributes of the events, and supports a location-based coverage relationship to achieve shared event dissemination paths. This is different from some related work, such as STEAM [MC02] and [CCdC05], that introduce location-awareness about the ECs. In STEAM (that is designed for MANETs), ECs are assumed to interact once they are in close proximity; *proximity filters* are introduced that define a certain geographical area (surrounding the *publisher*) in which events are valid. The work in [CCdC05] generalizes location-awareness with respect to both types of ECs (the event publishers and the event subscribers). They trade-off location decoupling against added functionality and performance in their motivating application environments.

Rendezvous-based Approaches. Many rendezvous-based Pub/Sub protocols have been proposed that build EDTs over Peer-to-peer (P2P) systems. The original idea was introduced in [WQA+02] and since then several topic-based (Scribe [CDKR02] and Bayeux [ZZJ+01]) and content-based (Hermes [PB02], Meghdoot [GSAA04] and [BBM+05; TBF+03; TAJ04]) protocols have been proposed that implement Pub/Sub functionality over some structured overlays. The main challenge addressed in these is the mapping of multi-dimensional, multi-typed content-based subscriptions to uni-dimensional or bi-dimensional numerical-only address spaces of structured overlays. Their application environment is also restricted by the underlying P2P systems, which are mostly designed for wired (Internet-like) infrastructures.

Rendezvous-based routing protocols (designed for WSNs) avoid the use of structured overlays, which may induce much maintenance and control overheads. Examples of rendezvous-based routing protocols that facilitate the intersection of data (events) and queries (subscriptions) at some (rendezvous) points in the network are Rumour routing [BE02], TTDD [YLC+02], and [LHZ04]. Recently, a low maintenance DHT (called GHT [RKY+02]) was proposed for DCS [REG+02] points in WSNs. The same scheme has been used in DIMENSIONS [GEH03], DIMS [LKGH03], and DIFS [GER+03], to implement more complex DCS solutions. QPS is similar to this work as it uses a geographical hash function, but addresses a different problem - namely, Pub/Sub functionality in WSNs. I am not aware of any event dissemination protocols (to date) that have used these schemes, but examined a comparable GHT-based approach as part of my evaluations. QPS gains performance by reducing the randomness that is inherent in DHT solutions, and constructing the EDT according to an approximate location-based metric space.

4.6 Summary

In this chapter, I presented QPS, a distributed and scalable Pub/Sub protocol, for location-aware WSNs. QPS provides a topic- and location-based Pub/Sub abstraction, and provides a complete time and location decoupling between its ECs. It operates at two layers: a network-independent logical layer that provides Pub/Sub functionality, and a network-dependent physical layer that provides resource-awareness and fault-tolerance. At the Pub/Sub layer, QPS builds logical EB trees[1], and constructs an EDT to interconnect the publishers and the subscribers. The physical layer maps this EDT to real nodes with EB functionality.

QPS's independent operation from the routing layer allows different flavors of location-based routing protocols to operate transparently in the system. I showed that supporting this while also attaining scalability (through shared event dissemination paths) in the existing work, is difficult. I exploited location-awareness to offer a trade-off at a logical layer. This trade-off can be manipulated by subscriber-specified ϵ factors. Performance evaluations, within a simulation environment, indicated that QPS has limited gain in small and sparse WSNs but offers consistently better performance than its counterparts in large-scale and dense WSNs. My literature survey revealed that there are several reasons for preferring dense WSN deployments. For example, [CDGS04] argues that many proposed localization schemes (for WSNs) require high node densities to offer acceptable performances, [SSS05] shows that higher network densities can ensure sensor network coverage and connectivity even when nodes are highly unreliable and the transmission power is small, and [STGS02] shows that high network densities allow for more energy savings. Thus QPS is expected to operate as a scalable Pub/Sub protocol in a wide range of WSN deployments.

[1]These trees resemble Quad-Trees from which QPS acquired its name.

Chapter 5

State-based Publish/Subscribe

With the increased realisation of the benefits of studying environmental data, sensor networks are rapidly increasing in size, heterogeneity of data, and applications. Future applications are expected to operate over larger and more heterogeneous sensor networks, with interests that involve detection of conditions, situations, and contexts. I expect large scale deployment of application-specific networks to become less likely, and the concurrent operation of applications over a shared infrastructure to become predominant and more economical. Smart environments are an example of these systems, where environments are equipped with wired/wireless sensor devices of various types and platforms, and the system serves many diverse and dynamic applications.

These systems demand frameworks that abstract the low-level data, the infrastructural details, and capture high-level knowledge or conditions for their applications and users. Knowledge about time and location can significantly enhance the meaning of observed data, and aid in accurate capture of high-level conditions with appropriate temporal and spatial interrelationships. For example, high temperature readings at two distinct locations, in a forest, may indicate separate (multiple) fire incidents or a widespread fire; but repeated high readings at a single temperature sensor indicates only the continuation of a single fire incident.

In this chapter, I present the State-based Publish/Subscribe (SPS) framework [TB07b; TB08] for sensor systems with many distributed and independent application clients. SPS decouples applications from the sensor devices, and processes sensor data internally to capture conditions, situations, or contexts of interest. SPS provides a state-based information deduction model that suits many classes of sensor network applications with interests in temporal and spatial conditions. State Maintenance Components (SMCs) are introduced that capture conditions over events, received from a Pub/Sub system. These SMCs store information about the captured conditions, and perform context-based data processing for increased efficacy and efficiency. Considering the heterogeneity of resources in sensor systems, these components were designed to be predictable in resource usage, flexible in network placement, and decomposable for distributed processing. They operate independently, but may collaborate through Pub/Sub to attain higher level knowledge. The Publish/Subscribe communication forms the core messaging component of the framework, and the decoupling feature of Pub/Sub is used and extended across the SMCs to support a more flexible and dynamic system structure. My performance evaluation, using real sensor data in the context of a smart transportation system, shows that SPS is expressive in capturing conditions, and scalable in performance.

This chapter begins with an insight into some of the emerging large-scale sensor systems. High-level application interests are discussed, and an overview of system requirements are outlined. Section 5.2 details the component-based architecture of SPS. SPS's flexible architecture supports even the most resource-constrained sensor devices, but can also exploit the resourceful nodes for resource-intensive computations and messaging. These architectural components are further detailed in Section 5.3. Data processing is governed by SMCs that envelope condition definitions and maintain the capturing condition's context. The semantics of SMCs are explored by an example in Section 5.4. A data model describes the purpose and structure of inter-component messages that drive data processing in SPS (Section 5.5), and the detection model (Section 5.6) explains how high-level conditions are captured in SPS. Condition detection may be performed in-network, or even distributed across many nodes. The advantages and semantics of SMC decomposition and distribution are sketched in Section 5.7. With every distribution comes the concerns of non-deterministic network behavior and node failures; SPS addresses these through a reliability model described in Section 5.8. Finally, the expressiveness and performance of the proposed framework is analysed in Section 5.9. A discussion of the related work is presented in Section 5.10, and a summary of contributions is provided in Section 5.11.

5.1 Application Scenarios

Following recent technological advances in sensor networks and ad hoc systems, large-scale ad hoc networks are envisaged that span large geographical areas and contain large volumes of data, input by many primitive sensor hardwares. This data benefits an equally large number of consumers that wish to process it and/or extract information from it. These *multi-user* systems are further motivated by the economical incentives that emerge when a large number of information consumers (users) are involved in a system.

Unlike conventional sensor networks, which are deployed for specific applications and often pre-programmed for specific tasks [IGE+03], these systems are dynamic in the number of information consumers and the nature of their interests. Applications, sensors, and protocols are no longer restricted by the constraints of hardware platforms. Instead, hardware platforms (such as UCB motes [HC02a] [HSW+00b], uAMPS [UAM], PC104 [PC1], GNOMES [WFF03] etc.) are carefully selected to support the required sensing, computation and performance.

These new classes of sensor systems are expected to support applications beyond the tasks of data collection and passive monitoring. Support for actuation, application messaging, capture of conditions and contexts inspire a new range of pervasive applications that can operate in an open distributed environment. I examine two application scenarios below that demonstrate the scale, characteristics, and potential applications of these systems.

Monitoring Underground Water. Underground water supplies are one of the oldest man-made infrastructures beneath the ground. This aging infrastructure is subject to deterioration,

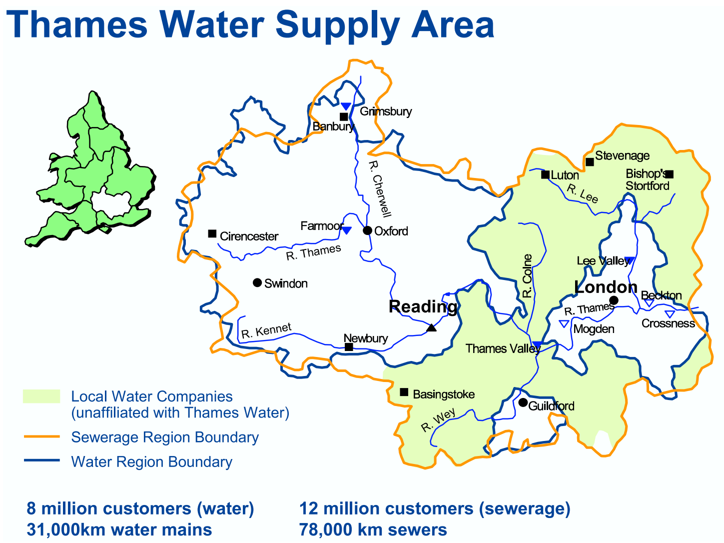


Figure 5.1: Thames water supply coverage (taken from [Rob06])

and is greatly influenced by nearby activities, such as constructions, excavations, and tunneling. Excess water leakage and pipe bursts are frequent, and often unanticipated. These pose safety and operational reliability issues, they can also impact neighboring infrastructures, such as telecommunication cables. Considering the scale of these infrastructures and constraints posed by the environment, physical inspections are difficult and in many cases costly to achieve. The emerging WSNs, however, can offer viable and cheap monitoring and condition assessment tools for predicting performance, detecting damage and using the resources efficiently.

Sensors can be deployed to monitor various contexts such as water pressure, chemical-levels, air and water flows in the pipes and acoustic signals throughout the underground water infrastructure. Each node in the network integrates specific sensing capabilities with communication, data processing and power supply. These nodes can form a network that could potentially scale into thousands of kilometers. For example Thames Water supply covers 31,000km of water mains and 78,000km of sewers in the greater London area (see Figure 5.1). Civil engineers have already begun to investigate how large numbers of sensors can be integrated into large-scale engineering systems [Rob06; SNMb; LWW08; SNMT07]. Another work [AS06] has motivated the shift from existing centralized wired architectures to decentralized wireless solutions for monitoring underground environments. In summary, [AS06] argues that five factors contribute to this shift: *concealment*, *ease of deployment*, *timeliness of data*, *reliability*, and *coverage density*. A wireless connection will avoid aboveground obstructions that can limit application areas. Absence of wiring eases the deployment and enhancement of these networks. Nodes can also relay data to network sinks as opposed to storing data locally at dataloggers which become single points of failure. Finally, the authors argue that removal of dataloggers allows for even more network coverage than when clusters are formed around the dataloggers.

Applications and interests that arise from these deployments can be diverse in nature. The deployment and maintenance costs often encourage the involvement of many application domains that can share and re-use the deployed system. For example, the water-supply company has immediate interest in the collected data. Similarly, telecommunication services can benefit from information such as water-leakage and pipe bursts that affect their assets, and building construction units can enhance their performance if they have real-time knowledge about the underlying environments. Application interests are rarely related to raw data; elements of data filtering, aggregation, and fusion can always yield more meaningful results. In fact, in-network data processing is beneficial as it increases the network lifetime by reducing data communications. Water leakage and pipe bursts are often inferred as a result of such data processing, and rarely have dedicated sensors for detection.

Characteristics that are observed from the above description are the large network scale, dynamic network topology (i.e. many independent and distributed application clients), and high-level information interests.

Smart Transportation System. My second application touches upon the same characteristics, though in a completely different application domain. Transportation is the ability to move people and goods from one location to another. Being able to move from one point to another is important, and sometimes the key to survival. In order to maintain a functioning economy, people must be able to circulate between the various points that are important to them, and do so with ease. Today, fossil fuel availability and road, rail, marine, and air-based transportation networks have drastically increased the distances one can travel and increased the overall number of trips made.

Maintaining ease of transport with increasing numbers of trips however is difficult. Road congestion in the UK costs in the order of £20bn per annum [Rel03]. In the same report [Rel03], 85% of senior business people are said to believe that investment decisions are influenced by the quality of transport. These business people believe that investment in monitoring, distribution and processing of traffic information will cause a substantial and significant increase in transport efficiency. This would not only improve business efficiency but would also have a profound effect on pollution control and social cohesion. Existing transportation information systems are largely vertically integrated and single themed. A generic framework can offer better abstraction (hide low-level sensor data), and support many applications through a unified higher level interface.

WSNs can play a key part in this, and researchers have already begun developing wireless sensors [Kna00] for easy low cost deployments. A smart transportation system consists of many information producing (sensor) devices, including inductive loop sensors, speed cameras, ANPRs, GPS devices, and traffic light signals, on urban streets. It is an ad hoc infrastructure, meaning that the sensors and actuators are not deployed and interconnected statically, or at once, but are progressively deployed and changed over time. At the application layer, the system is a multi-user sensor system, which serves many independent and heterogeneous application clients concurrently.

Diverse application interests demand a generic data structure and information processing model, that can aid many applications with their rich data interests. An examination of the user interests revealed [BBE+08] that very few (if any) relate to raw sensor data. Instead, user interests often relate to high-level information, such as congestion detection and projection, car-park status, bus arrival times, and free taxi locations, which could only be realised when data from multiple (possibly heterogeneous) sources are aggregated and fused in a specific manner. Despite the independence of the application clients, interests are frequently observed to be similar or overlap (e.g. two independent users may be interested in traffic congestion in the same area). In this work, I focus on traffic congestion phenomena, which I detect using data obtained from SCOOT [SCO].

5.1.1 Requirements

The highlighted applications have a number of characteristics in common that call for a generic framework. They exhibit the *large scale* and *heterogeneity* (of sensing hardware and device platform) that extend the system coverage and enable data capture from large geographical spaces. The scale of these networks suggests *topological dynamicity* and *reliability* concerns that cannot be overlooked. Additionally, the scale and maintenance costs restrict the frequency of sensor node inspections and replacements. Sensor nodes deployed in low-profile areas are expected to have several years of lifetime, before any replacement or maintenance services are needed.

Wireless communications are often employed due to their ease of deployment, low cost hardware, high robustness, and flexibility of network configuration, but a pure wireless network infrastructure should not be assumed. Wireless communications provide expensive and unreliable communications that demand fault-tolerant protocols for reliable operation.

Application clients, in the above systems, are *distributed* and often *enclosed* within the sensor network. There may be a *large number* of application clients, operating *concurrently* and *independently* at any one time. They express *dynamic interests*, which may vary in number and nature throughout the system lifetime. Application client interests rarely relate to raw sensor data, and in many instances raw sensor data may be discarded in favor of more *meaningful* and *compact high-level information*. An application's *prior knowledge* about the environment plays a key role in defining its interests, which could lead to overlapping interests among independent application clients.

A generic framework can abstract the highlighted concerns about the sensor networks, and provide a unified interface for supporting the described applications. This framework must encompass four features: *abstraction*, *scalability*, *expressiveness*, and *openness*.

Abstraction Infrastructural details, consisting of sensing/actuation devices, network properties, and topological configurations, are heterogeneous and can change over time. These details describe the very low-level dynamics of the network, and are often of little interest to the users. Users can instead benefit from higher level abstractions, such as data-centric abstractions, that closely match their interests. Abstraction, in sensor systems, shifts the application/sensor network interaction from a fragile address-based communication to more robust data-centric communication.

Scalability It is clear that managing a large-scale network with many independent or collaborative devices and users is difficult. Managing a sensor network is more difficult as it embeds a large volume of data that is produced by many networked devices (notably the sensors). A framework should support these devices and users without sacrificing efficiency or reliability. In most sensor networks, using wireless (radio-based) communications, scalability implies support for efficient communications and managing client dynamics with low overheads.

Expressiveness Sensors often produce data that is basic, redundant and highly correlated because they observe a common external entity (the *environment*). This primitive data needs to undergo processing to yield interesting information. Aggregation and fusion are preliminary steps in this regard, and can be extended to support lasting conditions, situations, or contexts. With high-level conditions, support for temporal and spatial interrelationships becomes important, and context-based data processing can be used to increase effectiveness.

Openness Openness in sensor systems allows users, devices, and applications to dynamically emerge (join) or disappear (leave) the system without centralized coordination or central management. It also supports the equal treatment of users and devices such that they can select one or more roles flexibly to suit their applications. For example, a user may operate as a consumer and receive sensor messages (data) in one application, and simultaneously operate as a producer and send messages (commands) to actuators in another. Inter-device and inter-user interactions are also supported in an open system.

5.2 Architecture

SPS is an event-based framework with a layered architecture. It uses Pub/Sub to loosely couple the end-points (applications and sensors), and to attain abstraction and scalability in the view of network topology changes. The architecture of a system that utilizes SPS is sketched in Figure 5.2. SPS enhances Pub/Sub with network-wide services, and builds two functional layers at the (larger) clients layer. These two support data processing in SPS.

Although the SPS implementation extends into the clients layer, SPS clients, comprising sensors, actuators, applications, controllers, etc., are supported at the lower layer's (Pub/Sub's) interface. A unified Pub/Sub interface abstracts the sensor network and supports all clients that reside at the top layer, including the SMC manager and the Information Space (InfoS). These functional layers are further described below.

Network Layer The network layer provides reliable message delivery from sources to destinations. Its implementation is external to the framework and embodies characteristics of the sensor network and communication media used in-between the networked nodes. Unicast and local (1-hop) broadcast services are the only communication services that are required from this layer.

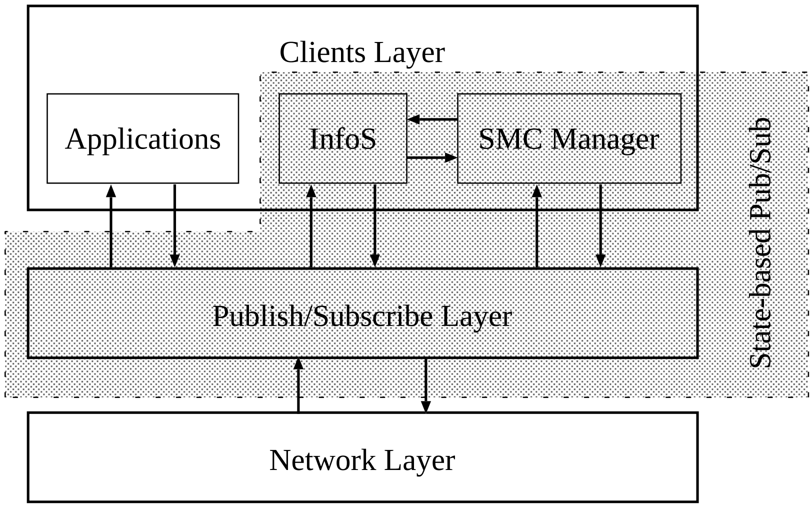


Figure 5.2: SPS architecture

Pub/Sub Layer The Pub/Sub layer implements a lightweight many-to-many messaging service. It provides a topic- and location-based abstraction, and exports a unified interface for its event clients. The Pub/Sub paradigm adapts SPS to topological changes, such that node joins and leaves and corresponding data changes are transparently reflected in the system. This layer provides core functionality and distributed messaging for SPS. In addition, the use of Pub/Sub in the SPS architecture allows SPS to build on previous work and offer easy integration with existing Pub/Sub systems.

Clients Layer The highest level is the clients level, comprising sensors, actuators, users, embedded processors, etc., that reside on individual nodes and use SPS for their data processing and messaging requirements. Clients may interact directly at the local (nodal) level, but network-wide interactions are solely via the Pub/Sub layer. SPS provides the *InfoS* and the *SMC Manager* that are isolated from other clients and support distinct functionalities as described below.

Information Space The InfoS is an event repository that receives events from the Publish/Subscribe layer and presents it to the *local* (co-located) SMC manager for processing. It maintains a globally consistent view of the low-level data, and presents data changes to the SMC manager component for condition detection. InfoS supports primitive data processing, consisting of selection and aggregation operations, and shares data (events) among SMCs, which are maintained by the SMC manager component.

SMC Manager The SMC manager houses SMCs which envelope condition definitions and maintain the capturing condition's context. The SMC manager fuses different types of data that are received from the InfoS and examines them according to SMCs' condition definitions which constrain condition initiations and terminations. Data,

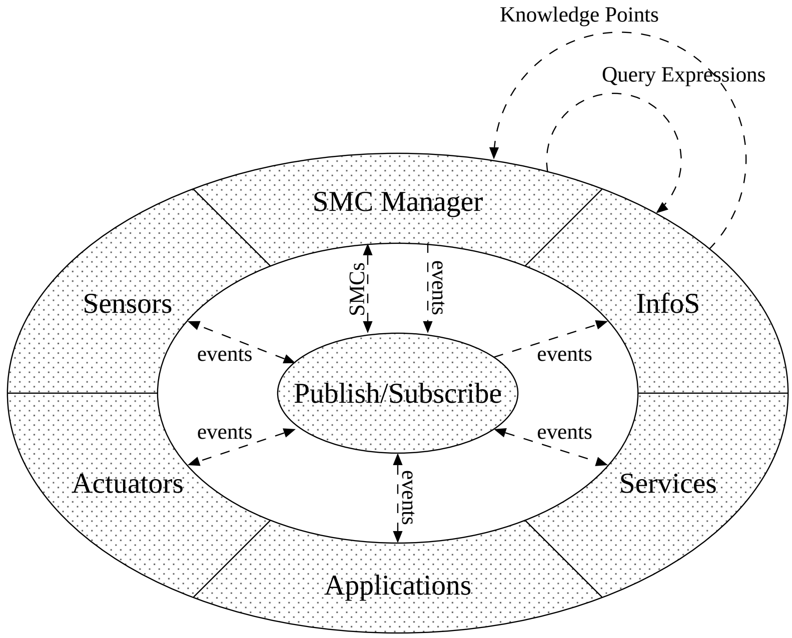


Figure 5.3: SPS components

which corresponds to a condition initiation or termination, is then mapped to higher level data type and sent to interested clients through the Pub/Sub layer. The expressiveness of SPS is indicated by the high-level conditions that can be captured at this layer.

The above functional layers are implemented as components. The component model supports the scalability and robustness properties of SPS.

5.3 Component Model

SPS functional layers are implemented as components. This helps to decentralize the operation of the framework, and increase robustness. Components at each node are instances of classes that communicate with each other using method invocations. All nodes in the network are assumed to provide SPS functionality, nonetheless at different levels. A lightweight Pub/Sub component centers the implementation of SPS and operates on all nodes (see Figure 5.3). This component, representing minimal SPS functionality, supports zero or more client components that consist of sensors, actuators, applications, services, InfoSs, and SMC managers. These clients adopt different Pub/Sub roles depending on their function. For example, an application client who wishes to consume sensor data becomes an *event subscriber*. Conversely, an application client that wishes to command an actuator becomes an *event publisher*.

Clients that reside on different nodes are loosely coupled by the Pub/Sub. Consider Figure 5.4, where two sensors, denoted by S, and three application clients, denoted by A, are

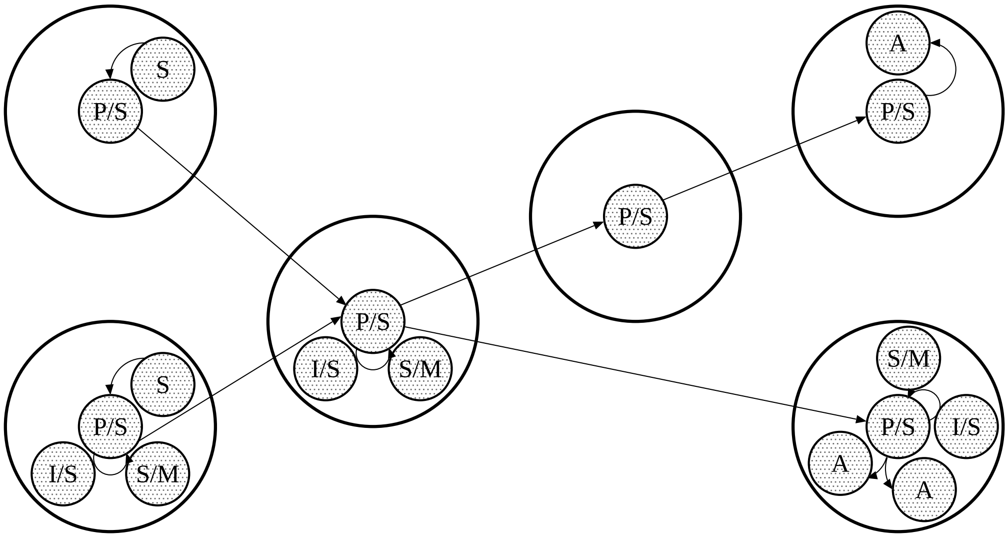


Figure 5.4: Network view of SPS components

supported by SPS in a network of six nodes. In this example, three nodes provide full SPS functionality (they implement all functional layers of SPS, the Pub/Sub component is denoted by P/S, the InfoS component is denoted by I/S, and the SMC manager component is denoted by S/M) and three nodes provide minimal SPS functionality - perhaps due to resource shortages. Sensor data may be transferred from sensors to nodes that house SMC manager components for processing; and results (reflecting captured conditions) may then be transferred to the application clients as shown by arrows in the diagram. The loose coupling of Pub/Sub clients means that sensors, applications, InfoSs, and SMC managers can be transparently added, replaced, upgraded, moved, or removed (when redundant), without affecting the described data processing. The SMC manager and InfoS components are *tightly coupled*, meaning that if an SMC manager component is present at a node then an InfoS component is also present and vice-versa. Figure 5.4 shows a view of the SPS components deployed in the network. The component-based view (Figure 5.5) shows a Directed Acyclic Graph (DAG) that indicates the flow of information through different types of SPS component. InfoS components (internal event subscribers) are excluded from the information flow diagrams, as they are tightly coupled with the SMCs managers that house internal event publishers. Every SMC transforms some named data that is provided by its *downstream publishers* into some higher level (named) data for its *upstream subscribers*. The flow of information,within SPS is always acyclic - enforced by partial ordering of data names. The functional components of SPS (Pub/Sub, InfoS, and SMC manager components) are further described below.

5.3.1 Pub/Sub Component

The Pub/Sub component is the only network-aware component of SPS that directly interacts with its peers to disseminate events across the network. It also exports the only interface that is available to the SPS clients, thus it must support the specification of high-level conditions,

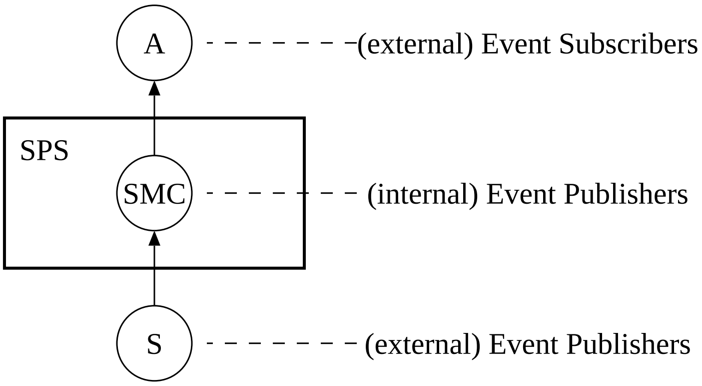


Figure 5.5: Component view of information flow

Returns	API Call	Parameters
`void`	`advertise`	`(Publisher pub, EventTopic topic, EventRegion region)`
`void`	`publish`	`(Publisher pub, Event event)`
`void`	`subscribe`	`(Subscriber sub, EventTopic topic, EventRegion region, Callback callback)`
`void`	`insertSMC`	`(SMC smc)`

Table 5.1: SPS Pub/Sub component's API

via the SMCs, by the SPS clients as shown in Table 5.1. This table shows the interface that is exported to all SPS clients.

The Pub/Sub component bears a number of additional responsibilities in SPS, mainly due to its network-awareness. Firstly, it can more effectively identify resourceful nodes and distribute data processing components as detailed later in Section 5.7.1. Secondly, it can protect SMCs (by storing passive replicas at alternative nodes) against moderate levels of node failure in the network (discussed in Section 5.8). Finally, it stores the most recent (latest timestamped) event, that is published by every event publisher in the system, to support dynamic subscriptions and information consistency features that are explained in Section 5.8.1.

In my prototype SPS implementation, I chose QPS [TB07a], a location-based Pub/Sub protocol for WSNs, as the Pub/Sub component. Although this choice restricted my prototype application to WSNs, alternative Pub/Sub implementations may be used to support other (wired or hybrid) network infrastructures.

5.3.2 SMC Manager Component

SMC manager components implement the information processing engine of SPS. They interact with their local InfoS components to obtain input data, and aggregate, join, and evaluate it

Returns	API Call	Parameters
`void`	`insertSMC`	`(SMC smc)`
`void`	`input_data_notification`	`(SMCName topic, QueryExpression qe, KnowledgePoints kps)`

Table 5.2: SMC manager component's API

to detect high-level condition occurrences or terminations. These computations are directed by some internal components, called SMCs. The SMC manager manages SMCs in terms of storage, replacement (when a newer component becomes available), evaluation (when newer data becomes available, Section 5.6), and decomposition (when distributed detection is desired, Section 5.7.2). The interface that is exported for the local Pub/Sub and InfoS components is shown in Figure 5.2. The first method is used by the Pub/Sub component, and the second one by the local InfoS.

State Maintenance Components. SMCs are the basic information processing elements of SPS. They are instances that capture conditions or situations of interest through the notion of *state*, and perform *context-based* data processing with two predicates (similar to that of State Filters in Chapter 3); they capture (or sample) lasting conditions at two distinct points, the moment of condition initiation, and the moment of condition termination. Event aggregation, parameterization, fusion, and interrelationships are supported through expressive operators that have predictable resource usage. They store state, thereby have *memory* to capture lasting and complex conditions.

Captured conditions are enveloped as events that can be published using the Pub/Sub components; thus SMCs are viewed as event publishers to their local Pub/Sub components. The context-based data processing feature of SMCs guarantees these events to be *unique* with respect to their observing conditions (under failure-free operations), and eliminates the possibility of duplicates at the corresponding end-points. Details of the processing and condition detection model are described later, in Section 5.6.

Definition 5.1 (State Maintenance Component (SMC))**.** *An SMC m consists of a tuple,*

$$m = (N_m, Q_m, G^n_m, G^x_m, s_m, e_m), \tag{5.1}$$

where the combination of N_m, Q_m, G^n_m, G^x_m is a user-defined expression which describes the condition of interest, the s_m is a status bit, and the e_m is the last published SMC event. The user-defined expression consists of an SMC name N_m, SMC QEs (definition 5.4) Q_m, and two conditional mapping statements G^n_m and G^x_m. A conditional mapping statement, G, consists of a transition predicate P and a set of attribute computation expressions A, $G = (P, A)$.

The SMC name is composed of a topic name, $n_m \in D_P$, and an internal signature value i_m. The topic name, n_m, describes the type of condition that is captured by the SMC, and the

Returns	API Call	Parameters
void	notify	(Event event)
void	register_input_data_selection	(SMCName topic, QueryExpression qe)

Table 5.3: InfoS component's API

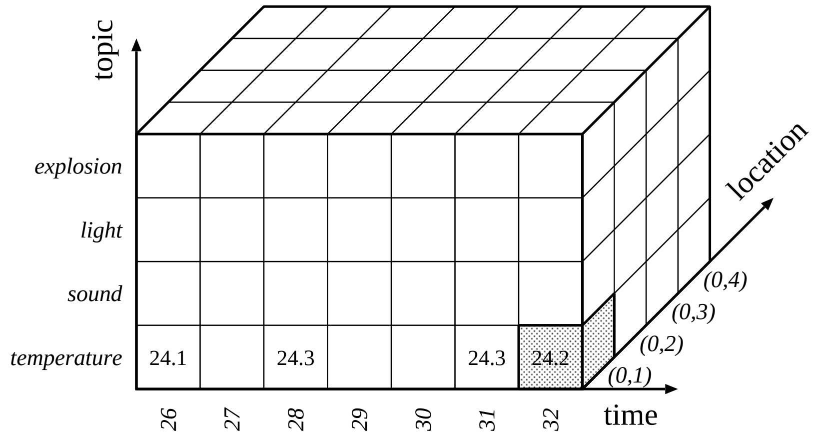


Figure 5.6: InfoS cube representation (excludes the status attribute)

signature value, i_m, distinguishes between two SMCs with the same topic name. These signature values are either generated by the Pub/Sub component (when it replicates and distributes the SMCs, Section 5.7.1) or assigned by the SMC manager component (when it spawns temporary SMCs to monitor concurrent conditions, Section 5.6.6). The Q_m highlights the input data that is relevant for condition evaluation, and indicates the condition interrelationships properties (discussed later in Sections 5.4.1 and 5.5.3). The conditional mapping statements examine the input data according to the SMC's context, examining for condition initiation if the condition is not yet active, and condition termination if it is already active. A condition detection results in the generation of an SMC event that is constructed using the attribute computation expressions, A.

5.3.3 InfoS Component

The InfoS component is a data repository that serves the local SMC manager component. The SMC manager presents the InfoS with QEs that indicate SMCs' input data, and the InfoS component in turn subscribes to the Pub/Sub component to receive the input data from all publishers. The InfoS interface is shown in Table 5.3.

The InfoS component *stores* data, because high-level conditions often depend on more than just the latest received data. Historic data may be needed to aggregate, fuse, or examine data for condition detection. It also *stabilizes* (buffers) data to ensure a consistent view of the data across the distributed InfoS components. Unless the Pub/Sub component offers *total* *ordered event delivery*, it is imperative to establish consistency for distributed processing (InfoS consistency is discussed later in Section 5.8.1). Finally, it *updates* data because newly received information could affect the validity or correctness of the past data.

The implementation resembles an MDX [MDX] cube, which stores data indexed according to multiple dimensions. Data in the InfoS is stored as tuples of attributed values, referred to as Knowledge Points (KPs) (described in the next section). These KPs contain some generic attributes (used for indexing), some unconditional attributes that reflect topic-related information, and a validity bit that indicates their temporal continuity. There are four generic attributes, *topic*, *time*, *location*, and *status*, that are used for indexing. They describe the type, temporal, spatial, and continuity properties of the stored data at each unit of the cube. Each unit can only store a single KP, i.e. the combination of the four generic attribute values in the InfoS can at most correspond to a single KP. Figure 5.6 shows an InfoS cube with some data, indexed by four topic values, seven time values, and four location values. Topic values are data names, often represented in textual format, that are introduced by users or applications into the system. Time and location values are point-based values with a vector-based representation that may be n dimensional ($n \in \mathbb{N}$). Status values, which are not shown in Figure 5.6, consist of three values that are defined internally.

InfoS supports two modes of referencing data in its cube structure: *absolute referencing* and *relative referencing*. Correspondingly, two domains of values are defined for each dimension of the cube: an *absolute domain* (D) and a *relative domain* (D^r). Relative values are distinguished from absolute values by an r superscript (e.g. 2^r is a relative value). The relative domains are only meaningful with respect to the dimensions of time and location, and are defined with respect to two pieces of information: an *InfoS time* t_I and a *host location* l_H. The InfoS maintains the InfoS time t_I and the host location l_H, which are discussed later in Section 5.8.1.2; it is sufficient to note that only data with timestamps (time values) less than or equal to the InfoS time is considered stable and processed in the SPS, and that the 'host location' indicates the location value of the node that houses the corresponding SMC manager and InfoS component at t_I. Each dimension of the InfoS cube is further described below.

Topic The absolute topic domain, D_P, reflects the set of topic values that are known in the system (e.g. $D_P \supseteq \{temperature, sound, light, explosion\}$ for Figure 5.6). Values are partially ordered according to their inter-relationship on the information flow diagram. For example, if the *explosion* data is obtained from fusing the *temperature*, *light*, and *sound* data, then $(explosion > temperature) \wedge (explosion > sound) \wedge (explosion > light)$. This ordering may be maintained centrally or perhaps embedded in the values (e.g. topic values can be appended with Bloom filter expressions [Mit01] that represent their sub-topic values). The relative domain D_P^r is identical to the absolute domain, $D_P^r = \{p^r | \forall p \in D_P, p^r = p\}$.

Time The absolute time domain, D_T, reflects a set of time values that are represented in a one dimensional vector space $\mathbb{R}^1$. For convenience, I illustrate these as real numbers $\mathbb{R}$ (e.g. $D_T \supseteq \{26, 27, 28, 29, 30, 31, 32\}$). The set D_P is *totally ordered*, because $\mathbb{R}$ itself is totally ordered. The relative domain D_T^r also defines a set of totally ordered values, each of which is the (vector) difference from a given value (the InfoS time $t_I \in D_T$), $D_T^r = \{t^r | \forall t \in D_T, t^r = t - t_I\}$. Since the InfoS time $t_I \in D_T$ continuously changes (Section 5.8.1.2), the relative time values may not be permanently mapped to absolute values; instead, they are mapped on demand for referencing absolute data in the InfoS cube.

Location The absolute location domain, D_L, reflects a set of location values that may be represented in a one, two, or three dimensional vector space $\mathbb{R}^n$ (where $n \in \{1, 2, 3\}$)[1]. Using the lexicographical order structure, the set D_L is *totally ordered* given that $\mathbb{R}$ itself is totally ordered. The relative domain D_L^r also defines a set of totally ordered values, each of which is the (vector) difference from a given value (the host location $l_H \in D_L$), $D_L^r = \{l^r | \forall l \in D_L, l^r = l - l_H\}$. Relative location values may be permanently mapped to absolute location values if and only if the host location l_H is constant (i.e. the SMC manager host is stationary).

Status The absolute status domain, D_S, is an unordered set of values, $D_S = \{atomic, ingress, egress\}$. The relative domain D_S^r is identical to the absolute domain, $D_S^r = \{s^r | \forall s \in D_S, s^r = s\}$.

While the absolute domain D allows data to be *globally* referenced, the time and location relative domains, D_T^r and D_S^r, allow *contextual* referencing where data is identified according to the temporal and spatial context of the node that hosts the InfoS. For simplicity, I use one dimensional location values in my examples, though two or three dimensional values may be used similarly. With the highlighted details, an InfoS may be considered as a relation, over which data may be aggregated and queried using relational operators. SPS defines custom aggregation and selection operators, that preserve the relational schema. These are defined as part of the Query Expressions, described in Section 5.5.3.

5.4 Condition Specification

Application interests rarely match the granularity and primitiveness of data that is realised in a sensor system. Filtering, aggregation, and fusion provide basic data processing tools that can transform low level data into higher level information or customized data in the system. This high-level data is referred to as a *condition* in SPS. Conditions often relate to real-world situations or contexts (e.g. a condition of "temperature over 60 °C" may relate to a "fire" situation). The expressiveness of SPS aids the *accurate* and *complete* detection of situations, whereby false-positives (erroneous detections) and false-negatives (missed detections) can be eliminated.

[1]The two dimensional vector space $\mathbb{R}^2$ (representing a 2-D geographical space) is perhaps the most useful and researched format among the existing localization schemes.

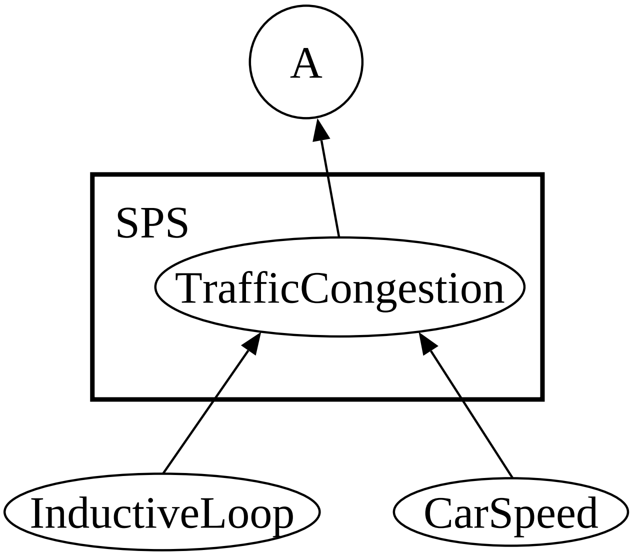


Figure 5.7: Traffic congestion information flow

Conditions in SPS are defined using SMCs. These components contain condition definitions, and monitor them using the notion of state (as described in Section 5.3.2). In order to capture a condition, the condition must be explicitly expressed using one or more SMCs. If the condition relates to a situation, then a mature understanding of the situation is required to use the expressiveness of SPS and eliminate false-positives and false-negatives in the detection. In SPS, the terms “condition” and “situation” are interchangeable and only reflect the user’s perception of his/her interest. In this section I demonstrate the specification of a condition, using an SMC. I first describe a situation of interest, outline my understanding about the situation, and then illustrate its specification in an SMC.

Traffic Congestion. Re-visiting my motivating application scenarios (Section 5.1) and the “smart transportation system” in particular, I decided to capture *traffic congestion* situations within SPS. For the purpose of this study I confined myself to the inductive loop sensor data and speed measurements taken by speed cameras, and described the traffic congestion situation as the mutual occurrence of “high road occupancy” and “slow vehicle speeds” on the same road. The *high road occupancy* situation was described by this condition: the ($30s$) average reading of the sensor data must be above the $2.5occ$[1] threshold value. Similarly, the *slow vehicle speeds* situation was described by this condition: the ($1min$) average speed measurements must be below $7MpH$. The traffic congestion was said to last until the ($1min$ average) speed of the moving vehicles, on the congested road, exceeded a given threshold value ($15MpH$). The corresponding information flow and SMC structure that capture this condition are shown in Figure 5.7 and Table 5.4, respectively. In the next few sections I explain how this SMC structure is derived from the above interest.

[1] A sensor occupancy unit, which indicates that an inductive loop sensor is covered by a large metal object (e.g. a vehicle) for a quarter-second interval.

n	Index	Value	Annotations
1	N	"TrafficCongestion"	SMC (topic) name
2	Q	$A := ((closest, InductiveLoop, \text{null}, \text{null}),$	select *InductiveLoop* event topic
3		$(aggregate, \text{null}, avg, (-30^r, 0^r)),$	aggregate values over last 30s
4		$(multiple : separate, \text{null}, \text{null}, D_L),$	distinct conditions (per location)
5		$(closest, atomic, \text{null}, \text{null}));$	select atomic events
6		$B := ((closest, CarSpeed, \text{null}, \text{null}),$	select *CarSpeed* event topic
7		$(aggregate, \text{null}, avg, (-60^r, 0^r)),$	aggregate values over last 60s
8		$(multiple : separate, \text{null}, \text{null}, D_L),$	distinct conditions (per location)
9		$(closest, atomic, \text{null}, \text{null}))$	select atomic events
10	P^n	$(A.value > 2.5)$ && $(B.value < 7)$ && $(A.location == B.location)$	condition entrance predicate
11	A^n	$location := A.location;\ level := B.value;$	ingress event attribute computations
12	P^x	$(B.value > 15)$ && $(B.location == last.location)$	condition exit predicate
13	A^x	$location := last.location;\ level := B.value;$	egress event attribute computations

Table 5.4: *TrafficCongestion* SMC structure (definition 5.1)

5.4.1 SMC QEs (Q)

Query Expressions (QEs) (whose structure are described later in Section 5.5.3) provide a systematic way of describing SMC's input data. They define multi-dimensional queries, which select and/or aggregate data from each dimension of the InfoS cube independently. In Table 5.4, two QEs are used to extract two types of information from the InfoS cube. QE A (lines 2–5) extracts data about road occupancy, and QE B (lines 6–9) extracts data about vehicle speed.

Three custom operators (*closest*, *aggregate*, and *multiple*) are designed, each of which can be used as part of a query to extract data with respect to one dimension of the InfoS cube. Assuming InfoS contains data that corresponds to two topic values (*InductiveLoop* and *CarSpeed*), several location values (reflecting readings about different roads), several time values (reflecting readings at different times), and *value* attributes that reflect the measured values, then the custom operators can be described (in the context of my example) as follows. Figure 5.10 shows an example of such InfoS at the top-left corner.

closest This operator (also called the nearest-index operator) selects the data that has an equal or closest index value to a given index (on the corresponding InfoS cube dimension). In the case of topic or status dimensions, this operator can only select the exactly indexed data from the InfoS. Whereas for the time and location dimensions, nearest indexed data can also be selected. For example, knowledge about road occupancy can be retrieved by using this operator with an *InductiveLoop* parameter for the topic dimension, as in Table 5.4 line 2. For temporal and spatial data, *closest* can be defined as absolute or relative to the current time and location. For example, the *most recent data* may be selected by using this operator with the 0^r parameter for the time dimension, and the *geographically nearest data* to the local node may be found by using the 0^r parameter for the location dimension. Data that holds the closest absolute time and location values to the t_I and l_H values are selected from the InfoS.

aggregate This operator allows a group of tuples to be aggregated into a single tuple across an InfoS dimension. For example, the inductive loop sensor data may be aggregated over time (as required by my condition definition) to give average road occupancy (Table 5.4 lines 3). The same operator is used to aggregate data on vehicle speed in line 7. This aggregation is performed independently from other dimensions of the InfoS cube, therefore data about each topic, location (road), and status is aggregated separately over time (i.e. measured values and time values are aggregated into one measured value and one time value for every unique combination of topic, location, and status). Figure 5.10 illustrates this in two steps: *group by Topic, Location, Status* and *aggregate over Time*.

multiple This operator, when applied to a generic attribute, selects all tuples whose value for that attribute is contained in the group argument passed to the operator. The group argument, can specify a set or a range of values (e.g. $\{temperature, light, sound\}$ or $(10, 25)$). This operator is useful when data from multiple index values need to be examined separately. For example, lines 4 and 8 (in Table 5.4) use this operator with the location dimension to select data corresponding to different locations and examine them separately.

Because multiple instances of data (corresponding to different index values) are selected and examined separately, multiple results that satisfy the user's interest may be found. The semantics of these results must be defined by the user. For example, they could relate to a unique condition, in which case one condition is detected, reported, and monitored in SPS. Alternatively, they could relate to multiple conditions that have occurred concurrently, in this case, each condition must be detected, reported and monitored separately. SPS supports *condition interrelationships* by allowing the user to specify one of four sub-operators $O^{multiple} = \{one, all, any, separate\}$ that indicate how multiple instances of data (after examination) need to be constrained or handled to detect conditions reliably. Note that although these sub-operators are specified as part of the QEs, they can only be evaluated after the SMC transition predicate (see next section) has been evaluated.

one Asserts that the end result should correspond to *one and only one* index value, from the group of index values that were provided as a parameter to the *multiple* operator. In the context of my example, the use of this sub-operator implies that the traffic congestion condition is only detected when congestion occurs at just one location (road) at a time.

all Asserts that *all* index values (given as a parameter to the *multiple* operator) should appear in the satisfying data set. The use of this sub-operator implies that traffic congestion is only detected when all roads are congested.

any Asserts that *any one* index value, from the set of given index values, is sufficient to appear in the end result. This sub-operator is always satisfied when an end result is present. It it useful for when multiple end results relate to the same condition (i.e. re-iterate one condition). In my example, the use of this sub-operator implies that the traffic congestion condition is a global phenomenon, which is detected when congestion occurs at one or more location values (roads).

separate Asserts that *every unique* index value, from the set of given indices, corresponds to a separate and unique condition. When used in my example, it implies that concurrent traffic congestion conditions can occur, each of which corresponds to a unique location value (road) that is observed in the satisfying data set. This sub-operator best describes my condition of interest, and has been used in Table 5.4 (lines 4 and 8). Note that once concurrent traffic congestion conditions have been detected, they are reported and monitored separately until termination.

5.4.2 SMC Transition Predicates ($P^{n/x}$)

SMC predicates are boolean expressions that guard the initiation and termination of lasting conditions. For an SMC m, the entrance predicate, P^n_m, defines a constraint that signals the occurrence (or presence) of a condition, and the exit predicate, P^x_m, defines an alternative constraint that indicates its termination (or absence). The condition is said to *hold* for a continuous duration that starts from the time of entrance predicate satisfaction to the time that the exit predicate is satisfied. Momentary conditions can be captured if P^x_m is explicitly set to *true*.

Data tuples that are received from the InfoS as a result of resolving the SMC QEs (detailed in the previous section) are examined individually or cross-examined, within the predicates, to constrain the condition initiation or termination. In these expressions, attribute names are used as operands, and mathematical, comparative, and logical operators (see Section 3.3.1.1) are used to examine relationships of interest between attribute values. Absolute values and zero relative values (i.e. 0^r) can also be used in these expressions. A special keyword ("*last*") is used to access the last SMC event, e_m, that is stored within the SMC. *last* usually refers to the SMC event that is published by the opposite transition predicate, i.e. refers to the condition initiation event when condition termination is being examined, and refers to the condition termination event when condition initiation is being examined.

The SMC predicates, P^n and P^x, are specified in lines 10 and 12 of the Table 5.4. The entrance predicate (line 10) constrains the input knowledge as follows.

- The level of road occupancy should be above $2.5occ$.
- The speed of vehicles should be under $7MpH$.
- Knowledge about road occupancy and vehicle speed should relate to the same location (road).

Similarly, the exit predicate (line 12) constrains the knowledge as follows.

i	n_i	v_i	Annotations
1	name	$n \in D_P, i$	event topic name and signature value
2	time	$t \in D_T$	point-based event (occurrence) timestamp
3	location	$l \in D_L$	point-based location of event's occurrence
4	status	$p \in D_S$	temporal significance of event's information

Table 5.5: Fixed Event Attributes

- The speed of vehicles should exceed $15MpH$.
- Knowledge about vehicle speeds should relate to the location where congestion was previously detected (at the entrance predicate).

5.4.3 Condition Attributes ($A^{n/x}$)

Following a condition detection, an SMC event, e_m, is generated that reflects the captured condition. The topic-related attributes of e_m may borrow information from knowledge instances that have satisfied the predicate to deliver additional information about the condition initiation or termination. These attributes may be assigned using the SMC's attribute computation expressions, A^n and A^x. For example, the *TrafficCongestion* SMC may include the average speed of the vehicles (on the congested road) as an indication of the *level* of congestion, see Table 5.4 lines 11 and 13. The user has some flexibility in assigning time and location attribute values, but these must fall within particular ranges that are defined later in Section 5.6.7.

5.5 Data Model

The data model describes the data structures that are realised in SPS. Since SPS' functionality is implemented by components, this model is best described by studying the inter-component messages (see Figure 5.3). These messages are detailed in the following sections, followed by some data structures that are realised solely within the SMC manager component. Inter-EB messaging, event client advertisements and subscriptions are excluded from these discussions as they solely relate to the Pub/Sub component implementation (see Chapter 4 for a candidate implementation).

5.5.1 Events

The primary data that is communicated between the Pub/Sub component and the Pub/Sub event clients (SMC managers, InfoS components, and SPS clients) is *events*. Events are asynchronous messages that describe an information or situation that is realised in the system. It may be as primitive as a temperature value, or as comprehensive as signaling a nearby traffic congestion condition.

Definition 5.2 (Event Notification)**.** *An event notification e belongs to an event space $\mathbb{E}$, and consists of a tuple of attributes,*

$$e \in \mathbb{E}. \;\; e = (a_1, \cdots, a_n). \tag{5.2}$$

Each attribute, a_i, is a name-value pair*, (n_i, v_i), with name n_i and value v_i. Attribute names are locally unique, i.e. $i \neq j \Rightarrow n_i \neq n_j$.*

The first four attribute names of an event are fixed and predefined in SPS, and the latter $(n-4)$ attributes relate to the first attribute value v_1. These attributes are explained in Table 5.5. The first attribute labels the information that is contained in the event notification. It also provides a context in which the last $n-4$ attributes (referred to as the *topic-related attributes*) are understood. These names correspond to the SMC names, discussed earlier. The event's time of occurrence is set by the event's publisher, and reflects the time of observation (i.e. the time at which the conveyed information is probably most accurate). Similarly, a point-based location describes the location where the event's information was measured. This location depends on the source of the event's information, and may be different from the location of the event's publisher.

When events transport high-level information, captured by SMCs, the status attribute indicates whether the event signals the initiation of a condition (*ingress*) or its termination (*egress*). This attribute describes the temporal continuity of the captured condition with respect to the event's timestamp. If set to *ingress*, the event signals the start of a lasting condition that begins from v_2 (the event's timestamp). If set to *egress*, the event signals the termination of a condition which lasted until v_2. Otherwise, the attribute value is set to *atomic* to indicate that the event's information is momentarily valid, at v_2. The remaining (topic-related) attributes are defined by the event publisher, and hold values that are assumed to be either numerical or textual.

5.5.2 Knowledge Points

The InfoS component feeds the SMC manager with the extracted input data for SMC evaluations. This data corresponds to some QEs, and consists of one or more KPs. KPs have the same relational schema as data stored in the InfoS, and extend the semantics of event notifications by a validity parameter, which indicates the validity of the data at the InfoS time, $t = t_I$. This validity is determined by examining the status attribute value of an event, as described earlier in Section 5.5.1.

Definition 5.3 (Knowledge Point (KP))**.** *A KP k is a data structure that extends the event notification data structure with a validity attribute,*

$$k = (validity, e) = ((valid, v), a_1, a_2, \cdots, a_n). \;\; e = (a_1, \cdots, a_n) \in \mathbb{E}. \tag{5.3}$$

The validity attribute value, v, is a boolean parameter, $v \in \{true, false\}$, that indicates the temporal validity of e's information at the InfoS time, $t_I \in D_T$.

The validity attribute is initially set to *true*, but later changed to *false* as more recent knowledge becomes available at the InfoS. Once changed to *false*, it can no longer be changed back to *true*.

5.5.3 Query Expressions

QEs are one of the structures that are communicated between the SMC manager components and the InfoS components. They contain expressions and operators that extract data from the InfoS for SMC condition detection. Essentially they describe a query to the InfoS that yields the input data for SMC evaluations. Note that the relational schema of the InfoS data is preserved.

Definition 5.4 (Query Expression (QE))**.** *A QE q is a tuple of selection parameters that hold a one-to-one relationship with the dimensions of the InfoS cube,*

$$q = (s_1, s_2, s_3, s_4), \tag{5.4}$$

where s_1, s_2, s_3, s_4 relate to the dimensions of topic, time, location, and status, respectively. Each selection parameter s_i describes a tuple that has an operation o_i and a set of arguments, comprising a value v_i, an aggregation functions list f_i, and a group of values g_i,

$$s_i = (o_i \in O, v_i \in D \cup D^r, f_i, g_i \subseteq D \cup D^r), \tag{5.5}$$

where $O = \{closest, aggregate, multiple\}$ is the set of possible operations, D and D^r are the domains of absolute and relative attribute values, and f_i is a set of aggregation functions for each data attribute as follows.

$$f_i = \{(f, a) | f \in F,\ a \in e \in \mathbb{E}\}. \tag{5.6}$$

The set of supported aggregation functions is $F = \{max, min, sum, avg\}$.

If the attribute values (in D and D^r) are totally ordered, then g can be described by two values $r_l, r_h \in D \cup D^r$ which denote the lower and higher bound values of a range,

$$g = \{v \in D \cup D^r | r_l \leq v \leq r_h\} \tag{5.7}$$

This is most useful when a one dimensional vector space $\mathbb{R}^1$ is used. For ease of presentation and discussions, I use the above syntax frequently, and specify the aggregation functions list f as a single common aggregation function to all data attributes, i.e. $f = x \in F$ implies $f = \{(x, a)\}$ for all $a \in e \in \mathbb{E}$.

When a QE q is applied to the InfoS, its selection parameters are transformed into queries and evaluated against the stored data to output the SMC's input data. Let $A = \{n_1, \cdots, n_k\}$ represent the set of all event attribute names, and $A^F = \{n_1, \cdots, n_4\}$ and $A^T = \{n_5, \cdots, n_k\}$ represent the set of fixed and topic-related event attribute names, i.e. $A = A^F \cup A^T$ and $A^F \cap A^T = \emptyset$. If one considers the relational operators in Table 5.6, then the QE selection parameters can be translated into relational queries as shown in Table 5.7. The processed attributes set,

Relational operator	Arity	Symbol	Annotations
select	unary	$\sigma_p R$	p is a predicate
project	unary	$\pi_s R$	s is a set of attributes
rename	unary	$\rho_x R$	x is a relation name
aggregation	unary	${}_g\gamma_e R$	g is a set of attributes (for grouping), e is a list of aggregation expressions
cartesian product	binary	$R_1 \times R_2$	
union	binary	$R_1 \cup R_2$	
set-difference	binary	$R_1 - R_2$	
set-intersection	binary	$R_1 \cap R_2$	

Table 5.6: Relational algebra operators

Attr.	Selection param.	Input	Output relation
$a \in A^F$	$(closest, v, f, g)$	X	$\bigcup({}_{\{x\|x \in A^F - A^P, x \neq a\}}\gamma_{min(\|a-v\|),\{y\|y \in A^T \cup A^P\},valid} X)$
$a \in A^F$	$(aggregate, v, f, g)$	X	$\bigcup({}_{\{x\|x \in A^F - A^P, x \neq a\}}\gamma_{f(\{y\|y \in A^T \cup A^P\}),a,valid} [\sigma_{a \in g} X])$
$a \in A^F$	$(multiple, v, f, g)$	X	$\sigma_{a \in g} X$

Table 5.7: QE selection operation translations

$A^P \subseteq A^F$, is detailed later in Section 5.6.4, but for now can be considered to be empty, $A^P = \emptyset$. Different operators result in different data storage (at the InfoS) and processing (at the SMC manager) complexities that are outlined in Table 5.8. QE selection operators are more comprehensively described, using set notation, in Appendix B.1.

5.5.4 SMC Manager Data Structures

While examining SMCs for condition detections, the SMC manager introduces two data structures that are solely seen within the SMC manager component. These data structures are defined as follows.

Definition 5.5 (Satisfying Knowledge (SK))**.** *An SK s is a set of attributed tuples, whose attribute values satisfy an SMC predicate p,*

$$s = \{(a_1, \cdots, a_k) | p(v_1, \cdots, v_k) = true\}, \tag{5.8}$$

where v_i is the value associated with the attribute a_i.

Definition 5.6 (Detected Knowledge (DK))**.** *A DK c has the same data structure as an SK s,*

$$c = \{(a_1, \cdots, a_k) | p(v_1, \cdots, v_k) = true\}, \tag{5.9}$$

Op. (o_i)	Expressiveness	Storage Complexity	Processing Comp.
closest	nearest-index constraint	$O(1)$	$O(1)$
aggregate	data aggregation	$O(1), f_i \in \{max, min, sum\}$	$O(1)$
		$O(n), f_i \in \{avg\}$	$O(1)$
multiple	condition interrelationships	$O(n)$	$O(n)$

Table 5.8: QE selection operators

but with the difference that the set members (attributed tuples in c) correspond to a single unique *condition. A DK c always has a corresponding SK s_c, of which it is a subset,*

$$c \subseteq s_c. \tag{5.10}$$

5.6 Detection Model

The detection model governs the capture of user-specified conditions in SPS. The outcome is a set of SMC events that signal condition detections; an *ingress* SMC event signals the initiation of a condition, an *egress* SMC event signals its termination, and an *atomic* SMC event signals the detection of a momentarily valid condition. These classifications are based on the *status* attribute values of the published SMC events.

SPS supports an *assertive* detection model, where conditions are detected using certain knowledge. This means inactive conditions need to be explicitly declared using positive knowledge, i.e. absence of data does not contribute to any knowledge such as 'negation' or 'non-existence'. This assertiveness prevents false-positive detections that may occur in unreliable or disconnected networks.

In order to give an overview of SPS's operations, Figure 5.8 shows a process timeline diagram for typical component messaging in SPS. The diagram covers different phases of SPS operation, from the clients' start-up to the high-level event delivery to the application clients. I briefly describe the *setup phase*, and then focus on the *condition detection phase* that is frequently repeated in the system. Before discussing the formal semantics, the following section describes the condition detection process with reference to an example introduced earlier in Section 5.4.

5.6.1 Example: traffic congestion detection

The condition detection process can be describes in five steps: *knowledge update*, *knowledge selection*, *knowledge examination*, *knowledge encapsulation*, and *knowledge transformation*. I have illustrated these in Figure 5.9 using the traffic congestion condition example introduced earlier. The figure depicts data processing at a single node, which houses the *TrafficCongestion* SMC (Table 5.4) and detects traffic congestion conditions.

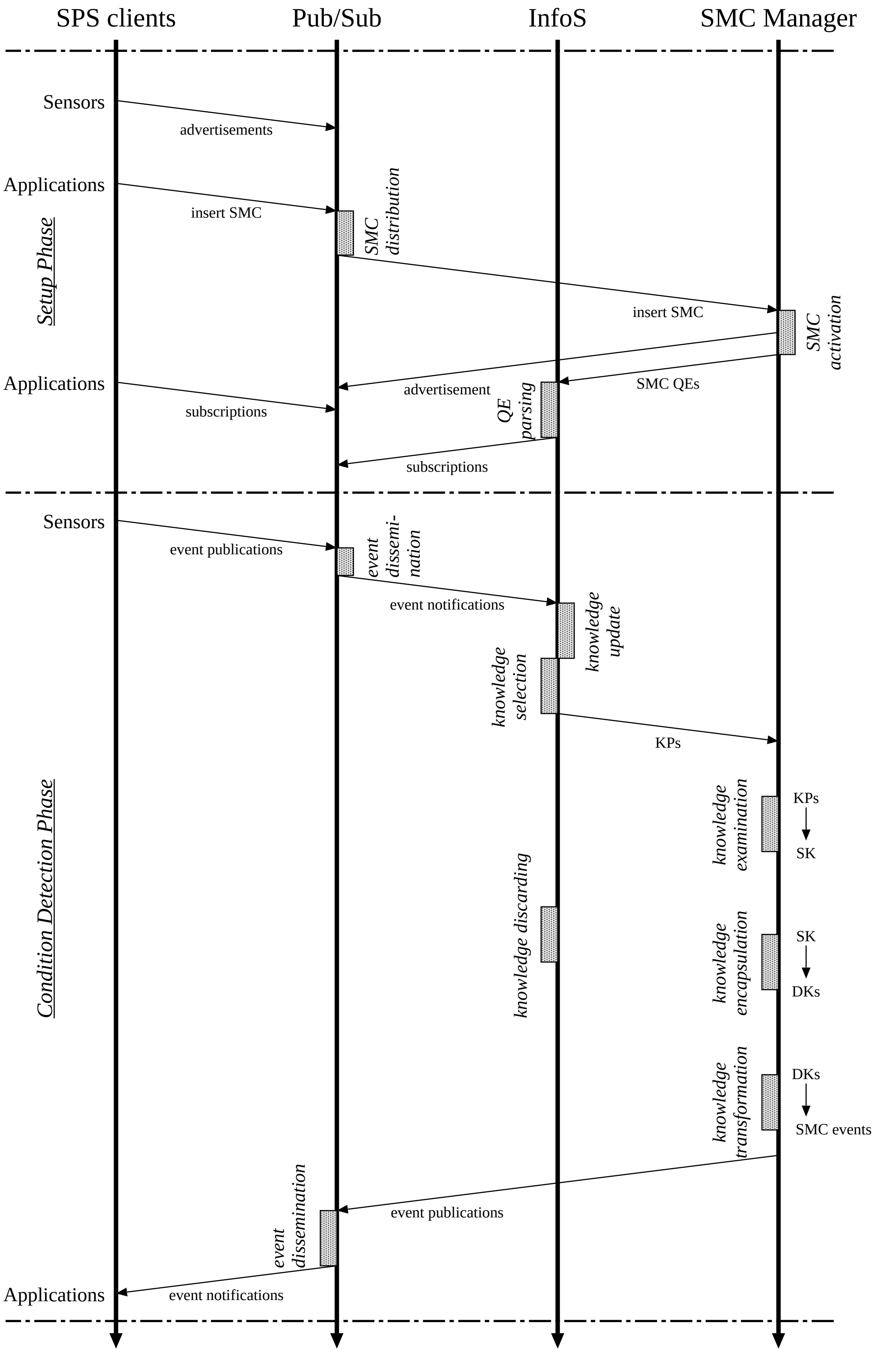


Figure 5.8: SPS process timelines

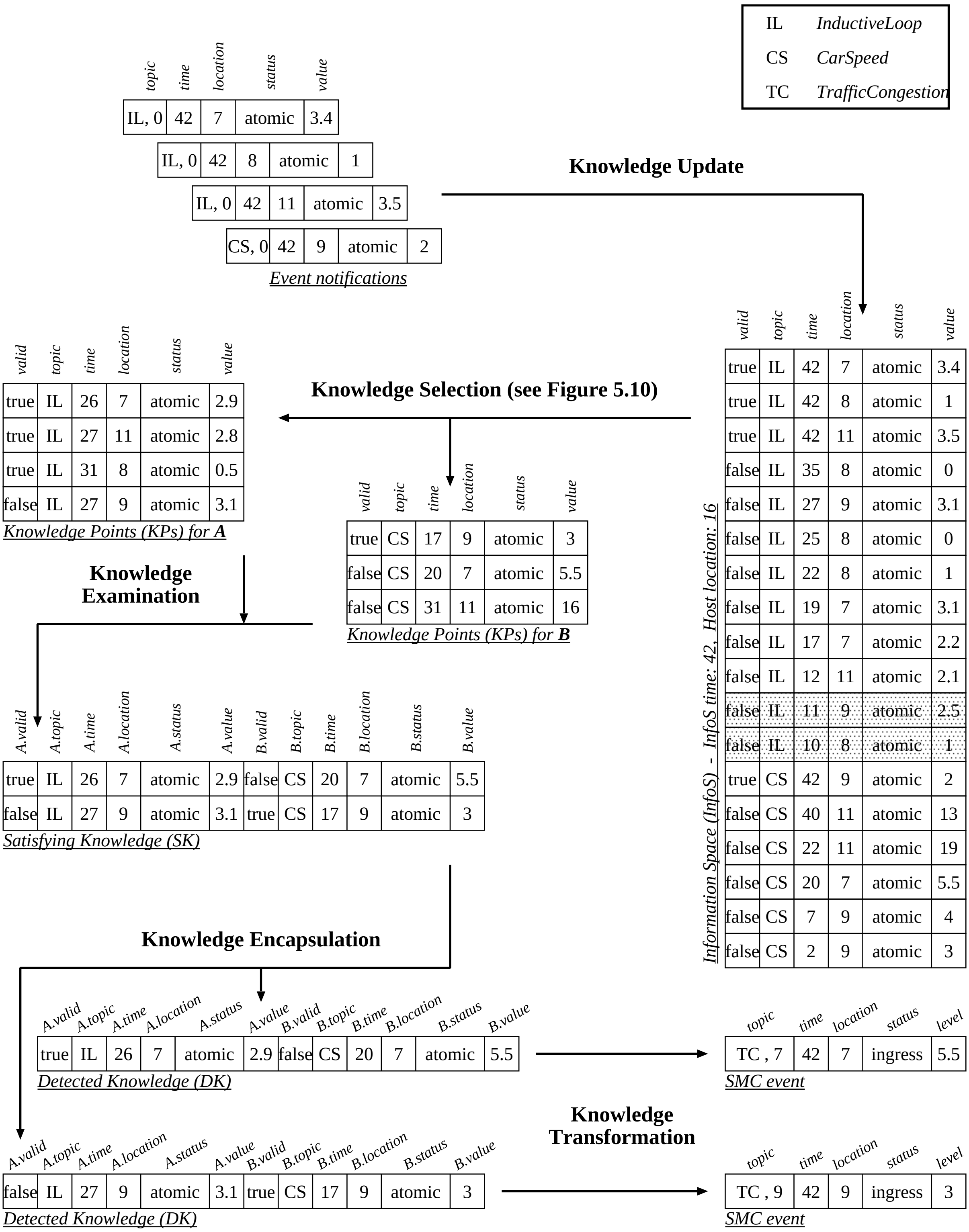


Figure 5.9: Traffic congestion detection

Knowledge Update As event notifications are received from the local Pub/Sub component, the InfoS is updated with new knowledge contained in these events.

Knowledge Selection New knowledge at the InfoS component triggers the selection of KPs which are examined for new traffic congestion conditions. The result is two tables, one of which corresponds to data reflecting road occupancy, and the other corresponds to vehicle speeds. These relate to the A and B QEs from the *TrafficCongestion* SMC (Table 5.4), respectively, and are delivered to the SMC manager for examination.

Knowledge Examination In this step, the KPs for A and B are combined into a set of larger tuples (called *KP-combinations*) and examined according to the SMC predicate. Tuples, labeled as SK, are KP-combinations, which have satisfied the SMC predicate.

Knowledge Encapsulation This step further constrains the SK tuples, and groups them into subsets (DKs) that represent unique conditions. In Figure 5.9, the SK is partitioned according to location values to reflect distinct traffic congestion conditions at different locations (i.e. one at location value 7, and the other at location value 9).

Knowledge Transformation Finally, DKs are transformed into SMC events according to the SMC attribute computation expressions, and published for delivery to the event subscribers.

5.6.2 Setup Phase

The setup phase begins by external components initiating interaction with the Pub/Sub components. Pub/Sub components form event dissemination paths that direct data from producers to consumers. Applications, however, need to express their conditions of interest as SMCs before subscribing to them. These SMCs are replicated (if necessary) and distributed within the network by Pub/Sub components as described later in Section 5.7.1. When given to SMC manager components, these SMCs are activated. The SMC manager component replaces existing SMCs with new SMCs if their names match. SMC activation entails an advertisement to the Pub/Sub component, dispatch of the SMC QEs to the InfoS component, and the initiation of the condition detection process that is described in the following sections. See Figure 5.8 for a summary of the described inter-component messaging.

In order to detect high-level conditions, sufficient knowledge about the environment must exist in the system. This is either introduced by external event publishers (e.g. sensor clients) or internal event publishers (SMCs). SMC QEs describe what knowledge is required for condition evaluation, and the InfoS components describe these in the form of event subscriptions to the Pub/Sub component to receive data from all publishers.

Parsing QEs to Event Subscriptions. The InfoS transforms every QE, $q = (s_1, s_2, s_3, s_4)$ (definition 5.4), into a set of location-based event subscriptions $\{s\}$. The number of subscriptions, $|\{s\}|$, exceeds one if the QE topic selection parameter, s_1, has a *multiple* selection operator, i.e.

if $o_1 = multiple$[1]. The set of event subscriptions $\{s\}$, derived from a QE q, may be defined as follows for the selected (QPS) Pub/Sub protocol.

$$\{s\} = \{(t_s, r_s, \epsilon_s)|\ t_s \in (\{v_1\} \cup g_1).\ r_s = \{v_3\} \cup g_3\}. \tag{5.11}$$

If the Pub/Sub component does not support relative-valued subscriptions, then the QE selection parameter values need to be mapped onto absolute values if they belong to the relative domain. Additionally, if a QE selection operator (for time or location) is *closest*, then the corresponding selection parameter value, v, is extended into a group of values $\{x|x \in D_{T,L} : |x - v| \leq \epsilon\}$ to cover the nearby knowledge[2]. In practice, a large ϵ is used initially and reduced following observations. The ϵ value has an upper bound value that is defined by the system designer.

5.6.3 Knowledge Update

The InfoS receives events (from the Pub/Sub component) and transforms them into KPs, i.e. extends them with validity attributes. These attributes are initially set to *true*, but later updated as more recent knowledge becomes available. The following two rules hold about the KP validity attributes.

- A KP, k, whose status is *atomic*, holds a *true* validity only when its time value matches the InfoS time (see Figure 5.9), $(v_2^k = t_I) \wedge (v_4^k = atomic) \Rightarrow v^k = true$.
- The validity of a KP, k, whose status is *ingress* or *egress*, is *true* only if no later timestamped KP exists with the same name (topic name and signature value[3]) and different status value in the InfoS, i.e. $\nexists j \in \texttt{InfoS} : (v_2^j > v_2^k) \wedge (v_1^j = v_1^k) \wedge (v_4^j \neq v_4^k) \Rightarrow v^k = true$.

Every KP occupies a single unit, in the InfoS cube, whose coordinates are identified by the KP fixed attribute values. Conflicts may arise, when two or more KPs hold identical fixed attribute values. This occurs only when the granularity of event publications is finer than the granularity of time and location domains, D_T and D_L; thus event publishers are forced to publish events with identical fixed attribute values. An immediate remedy to this conflict is to increase the granularity of domain values. Otherwise a conflict-resolution policy must be applied to maintain the singleness of KPs at each InfoS cube unit. SPS uses an aggregation policy that combines conflicting KPs into single KPs.

If we assume $K = \{k_1, k_2, \cdots, k_n\}$ represents a set of conflicting KPs (i.e. $\forall i \in \{1, 2, 3, 4\}$. $k_1.v_i = k_2.v_i = \cdots = k_n.v_i$, where $k_a.v_b$ is the b^{th} attribute value of $k_a \in K$), then the k^{agg} KP, with attribute values shown below, is the aggregate representation of K. I assume every $k \in K$ is of the form $((valid, v), a_1, \cdots, a_m)$, where each a_i is a name-value pair (n_i, v_i) as in definition 5.2.

[1] I assume that the Pub/Sub component supports one event topic name per event subscription.

[2] The parameter value v is extended into a closed line if $D \subseteq \mathbb{R}^1$, into a closed disc if $D \subseteq \mathbb{R}^3$, or into a sphere if $D \subseteq \mathbb{R}^3$. In all cases v is at the center, and ϵ indicates the distance to the closed boundary.

[3] Signature values are used to pair SMC events, when concurrent conditions are detected.

$$k^{agg} = ((valid, v), a_1, \cdots, a_m) \tag{5.12}$$

$$k^{agg}.v = k_1.v \quad \forall i \in \{1, \cdots, 4\}\ k^{agg}.v_i = k_1.v_i \quad \forall j \in \{5, \cdots, m\}\ k^{agg}.v_j = f(k_1.v_j, \cdots, k_n.v_j) \tag{5.13}$$

The conflict-resolving aggregation function, f, is a user-defined *order-insensitive* function that is globally defined. The set of conflicting KPs, K, may then be discarded in favor of the aggregated KP k^{agg}.

5.6.4 Knowledge Selection

This process extracts knowledge, according to SMC QEs, and forwards it to the SMC manager. The extracted data is structured as tables of KPs, where each table relates to a single QE (see Figure 5.9). These KPs are not limited to the newly received event notifications, but may include historic data that is re-used in SMC evaluations. The InfoS maintains the set of historic data, since it may be needed for present and future SMC evaluations.

Selection is triggered when a change or update has occurred in the InfoS. This change may affect the resulting KPs that are extracted for a QE, in which case knowledge selection and examination must be repeated to re-evaluate the condition of interest. Of course, every change does not affect every QE and its associated table of KPs; hence, it is computationally efficient to identify relevant QEs and only re-evaluate those against the InfoS. The following is a list of changes and the corresponding QEs that are affected by the change.

1. **Event Reception** The receipt of an event affects a set of QEs, $\{q\}$, whose selection parameter values (excluding the *status* attribute), $\{v_1^q, g_1^q, v_2^q, g_2^q, v_3^q, g_3^q\}$, cover the event's fixed attribute values $\{v_1^e, v_2^e, v_3^e, v_4^e\}$,

$$\{q | (v_1^e \in \{v_1^q\} \cup g_1^q) \wedge (v_2^e \in \{v_2^q\} \cup g_2^q) \wedge (v_3^e \in \{v_3^q\} \cup g_3^q)\}. \tag{5.14}$$

2. **InfoS Timeline Advancement** As the InfoS timeline (discussed in Section 5.8.1) advances, the relative time values map to different absolute values and therefore query different data in the InfoS. The affected QEs are all those who use relative values in their time selection parameter,

$$\{q | (v_2^q \in D_T^r) \vee (g_2^q \subseteq D_T^r)\}, \tag{5.15}$$

where the notation is adopted from the previous change. As we shall see in Section 5.8, this change often coincides with the latter change.

3. **Host Relocation** Similarly, a change of host's location affects those QEs which use relative location values,

$$\{q | (v_3^q \in D_L^r) \vee (g_3^q \subseteq D_L^r)\}, \tag{5.16}$$

where the notation is adopted from the first change.

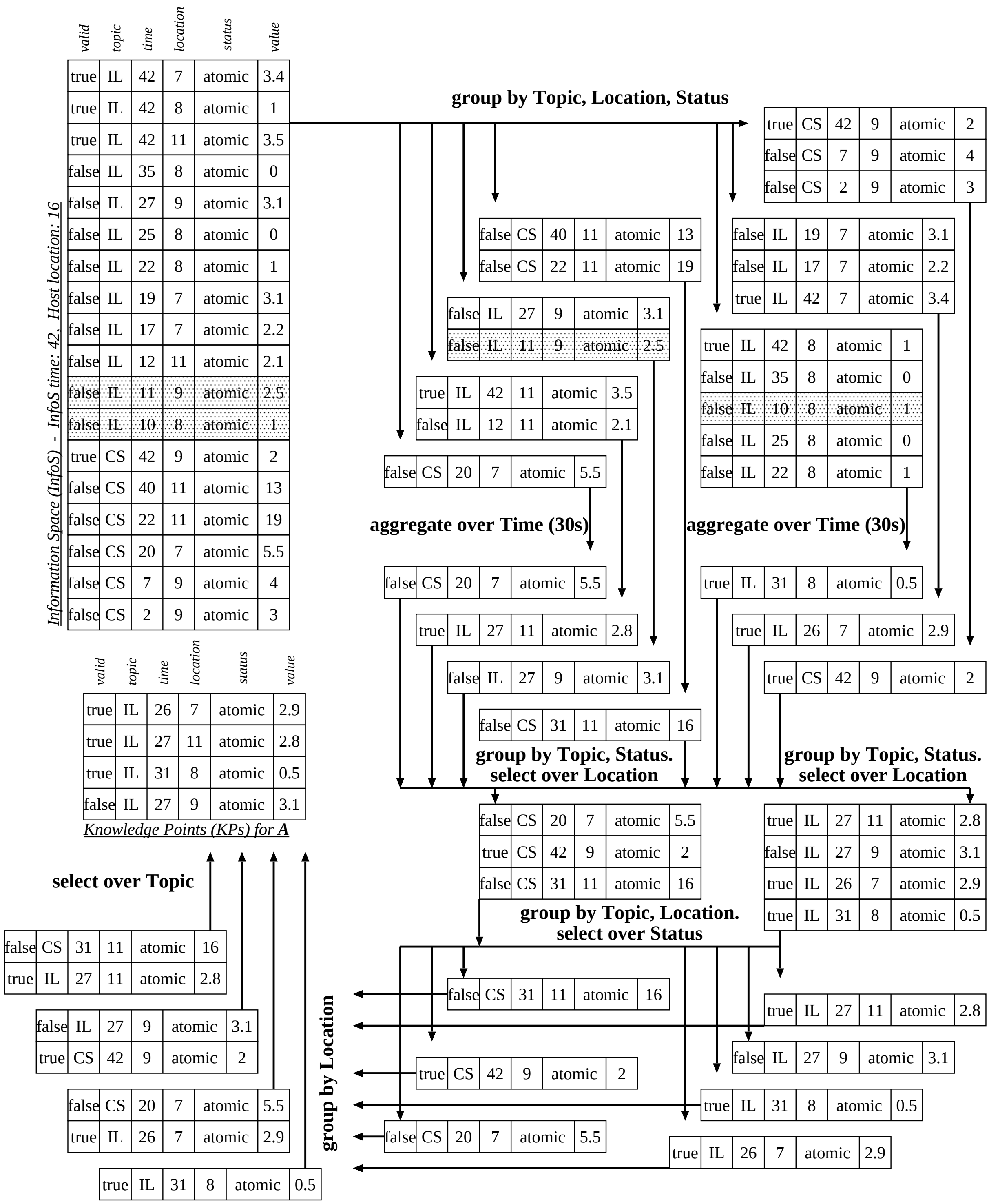


Figure 5.10: Knowledge (KPs) selection

Each QE, q, has four selection parameters that are translated into relational queries and evaluated (in order) against the InfoS. The *closest* and *aggregate* selection operators are order-sensitive, whilst the *multiple* operator is order-insensitive. SPS imposes a fixed ordering over the evaluation of attributed queries against the InfoS. If we label the relation that results from evaluating a selection parameter (corresponding to a fixed attribute $a \in A_F$) over an input relation X as $R_a(X)$ (Section 5.5.3), then the extracted table of KPs, K_q, for QE q is as follows.

$$K_q = R_{topic}\left(R_{status}\left(R_{location}\left(R_{time}(\texttt{InfoS})\right)\right)\right). \tag{5.17}$$

In this expression, data is initially queried (processed) according to the time attribute, then location, then status, and finally topic. At each evaluation phase, a *processed attributes set*, $A^P \subseteq A^F$, indicates which selection parameters have been evaluated. Attributes are placed into the set if and only if they have been processed and have contained a *closest* or *aggregate* selection operator in their selection parameters. The processed attributes set A^P affects the grouping that is applied prior to data selection. Figure 5.10 shows a detailed account of this process for extracting KPs for the A QE in the *TrafficCongestion* SMC (Table 5.4). In Figure 5.10, InfoS tuples are initially grouped by topic, location, and status, and averaged by time and value (the figure shows the grouping and the aggregation process in separate steps). The results are then grouped by topic and status, and selected according to the location attribute. In my example, tuples corresponding to all location values are selected because the group argument D_L contains all the possible location values. Subsequently, results are grouped by topic and location and selected according to status (here only tuples with the *atomic* value are selected). Finally, results are grouped by location and the set of tuples corresponding to the specified topic value (IL) is selected. Note that throughout this process tuples were always grouped by location (when processing with respect to other attributes). This is because the user had specified a *multiple* selection operator, which meant tuples corresponding to different location values should be kept separate (the result table, in Figure 5.10, has tuples corresponding to 7, 8, 9, and 11 location values that were present in the InfoS). The result table (relation) from this process, K_q, is forwarded to the SMC manager for condition evaluation.

5.6.5 Knowledge Examination

Tables of KPs, received from the InfoS, are now used by the SMC manager to examine SMC predicates for condition detection. SPS performs *context-based* data processing - knowledge about the current state of the condition is used to prevent redundant detections. Depending on the SMC's (m's) status bit, s_m, the received knowledge (KPs) is examined against the entrance or exit predicate to yield new information. This context-based data processing saves significant messaging and computation, when correlated and redundant data (e.g. sensor data) are used as input for condition detection (see SFs in Chapter 3).

If we label the QEs of an SMC, m, as $Q_m = \{q_A, q_B, q_C, \cdots\}$, and their corresponding tables of KPs as $K_m = \{K_A, K_B, K_C, \cdots\}$, then the SK, s, can be determined as follows.

$$s = \sigma_P((\rho_{\{A.topic,A.time,A.location,\cdots\}}K_A) \times (\rho_{\{B.topic,B.time,B.location,\cdots\}}K_B) \times \cdots), \tag{5.18}$$

where P is set to P_m^n if $s_m = 0$ and P_m^x otherwise (i.e. examine for condition initiation if the condition is inactive and for condition termination if it is already active). Figure 5.9 shows the result when the entrance predicate (line 10 from Table 5.4) is examined over the table of KPs extracted from the InfoS.

The cartesian product of the relations $(K_A, K_B, K_C, \cdots)$, and the examination of all KP-combinations may be computationally expensive. In Section 5.7.2, I discuss the decomposition of QEs, which enables the distribution of this processing load across many networked devices.

5.6.6 Knowledge Encapsulation

The knowledge encapsulation process groups the tuples (KP-combinations) that are in the SK s (from the previous step) to yield more meaningful and condition-specific sets of knowledge. The cardinality of s relates to the cardinality of input KPs $(K_A, K_B, K_C, \cdots)$, such that if $|s| > 1$ then $\exists x \in \{A, B, \cdots\} : |K_x| > 1$. For $|K_x| > 1$ to hold, the corresponding QE, q_x, must hold at least one *multiple* operator among its attributed selection parameters. We assume this operator, $o \in O^{multiple}$, relates to the $a \in A^F$ attribute for the purpose of the following discussion. In Figure 5.9, the size of the SK is two ($|s| = 2$), and there is a *multiple* selection operator in the A and B QEs of the *TrafficCongestion* SMC (shown in Table 5.4).

The outcome of the knowledge encapsulation process is a set of zero or more DKs (definition 5.6), $\{c\}$, where each set member signals a unique condition detection. The size of the set (the maximum number of concurrent condition detections per evaluation) is bound by $|\pi_a K_x|$, i.e. $|\{c\}| \le |\pi_a K_x|$. Let's rewrite the sub-operator assertions using relational algebra notations.

multiple:one $|\pi_{x.a}c| = 1$. Asserts that only one unique a value, from K_x, should appear in a DK c.

multiple:all $|\pi_{x.a}c| = |\pi_a K_x|$. Asserts that all a values, from K_x, should appear in a DK c.

multiple:any $|\pi_{x.a}c| \ge 1 \equiv |c| \ge 1$. Asserts that any one satisfied a value, from K_x, may be taken as a representative in a DK c.

multiple:separate $|\pi_{x.a}s| \ge 1 \equiv |s| \ge 1$. Similarly asserts that any one satisfied a value is sufficient for condition detection, but with the difference that every unique a value, from K_x in s, can signal a unique and distinct condition.

A DK c is a subset of the SK s, such that all assertions, by the *multiple* sub-operators in q_x, are satisfied within c. In order to determine all DKs $\{c\}$ from an SK s, the following steps are taken.

1. The SK s is divided into DKs according to the *multiple:separate* sub-operator, i.e. $\{c|c = \sigma_{(x.m=i\in(\pi_m K_x))}s\}$, where $m \in A^F$ is the attribute whose selection parameter, in q_x, contains the *multiple:separate* sub-operator. In my example (Figure 5.9), the SK is split according to unique location values. The result is two DKs, each of which corresponds to a different location value.

2. Every DK, produced in the previous step, is examined by the *multiple:one* and *multiple:all* sub-operator assertions, and discarded from the set if $|\pi_{x.n}c| \neq 1$ or $|\pi_{x.o}c| \neq |\pi_o K_x|$, where $n, o \in A^F$ are attributes whose selection parameters contain the *multiple:one* and *multiple:all* sub-operators, respectively.

3. If the *multiple:separate* sub-operator was involved in any of the selection parameters, then $|\{c\}|$ temporary SMCs are spawned[1], each of which is assigned a unique DK $c \in \{c\}$. The unique m attribute value of c determines the temporary SMC's (u's) name signature value, i.e. $i_u \leftarrow \pi_{x.m}c$. In Figure 5.9, one SMC is assigned the unique location value of 7, and the other is assigned the unique location value of 9. These values are seen in the generated SMC events as name signature values.

If $|\{c\}| > 1$, then multiple concurrent conditions are detected, each of which is monitored by a temporary spawned SMC. These temporary SMCs are distinguished by different signature values, and last until their corresponding conditions are terminated. The uniqueness of m attribute value is preserved throughout time, and enforced by unique SMC names, i.e. if u, v are two SMCs, then $(u \neq v) \wedge (n_u = n_v) \Rightarrow i_u \neq i_v$, where n and i are topic and signature values of an SMC name, respectively.

5.6.7 Knowledge Transformation

Following the knowledge encapsulation process, every SMC, u, is at most assigned a single DK c. These DKs contain knowledge (KP-combinations) that have contributed to a unique condition detection. Every SMC transforms its assigned DK into an event notification (called an SMC event) that is published in the system (see Figure 5.9).

The topic and status attribute values (of the SMC event) are strictly set by the SMC manager. They are set according to the SMC name and the satisfied predicate in the knowledge examination process - the *status* is set to *ingress* if P^n_u is satisfied and $P^x_u \neq true$, to *atomic* if P^n_u is satisfied and $P^x_u = true$, and to *egress* if P^x_u is satisfied. The SMC manager has default assignments for the remaining fixed event attributes and topic-related attributes of the SMC event. These may be overridden by the SMC's attribute computation expressions, A^n_u or A^x_u, see Table 5.9.

[1] This is implemented by temporarily extending the SMC's (u's) name and data structures (N_u, s_u and e_u) into array structures of length $|\{c\}| + 1$.

i	n_i	Default v_i	Override Permissables
1	name	N_u	$\emptyset$
2	time	$max(t_I, last.time + 1)$	$t \in D_T : (t > last.time) \wedge (t \geq t_I)$
3	location	host location (l_H)	$l \in D_L$
4	status	$p \in D_S$	$\emptyset$
5-n	topic-related	topic-related attribute values of a most recent KP in c	$v \in D$

Table 5.9: SMC Event Attribute Assignments

The condition detection process is completed when the SMC event is published, the SMC status bit, s_u, is appropriately toggled[1], and the last capture event, e_u, is overwritten by the newly published SMC event.

5.6.8 Knowledge Discarding

The knowledge discarding policy removes knowledge (KPs) from the InfoS as it becomes outdated or irrelevant. The exact semantics of outdated or irrelevant knowledge is defined by SMC QEs, which highlight what knowledge is related to the condition detections. For a set of SMC QEs, $Q = \{q_1, q_2, \cdots, q_n\}$, where each $q_i = (s^i_1, s^i_2, s^i_3, s^i_4)$ (definition 5.4), the set of KPs that may be permanently discarded from the InfoS are the set of tuples that persist in the following query result.

$$\sigma_{(\exists i \in \{1,\cdots,4\}:\texttt{InfoS}.a_i \notin \{V_i \cup G_i\})}\texttt{InfoS}, \tag{5.19}$$

where $\texttt{InfoS}.a_i$ is the i^{th} attribute of the InfoS ($a_i \in A^F$) and $\forall j \in \{1, \cdots, 4\},\ V_j = \bigcup_{i\in\{1,\cdots,n\}}\{v^i_j\}$ and $G_j = \bigcup_{i\in\{1,\cdots,n\}} g^i_j$. This query gives all stored KPs who have at least one attribute value that falls outside the range of all known QEs interests. Note that the group $\{x|x \in D_{T,L} : |x-v| \leq \epsilon\}$ is also used here when values correspond to the *closest* selection operator in time or location selection parameters.

For many attributes and values, the resulting KPs (from the above query) are persistent and may be discarded upon initial observation. Exceptions to the above are relative time and location values. For relative time values, the InfoS timeline is known to be monotonically increasing (Section 5.8.1). Therefore the KPs that may be discarded can be defined by the following query. In Figures 5.9 and 5.10, these tuples are shaded and excluded from data evaluations at the knowledge selection phase.

$$\sigma_{(\texttt{InfoS}.a_2 < min(V_2 \cup G_2))}\texttt{InfoS} \tag{5.20}$$

[1]The status bit, s_u, is not toggled (remains unset) if the SMC event's *status* attribute value is *atomic*.

Unless the InfoS host is stationary or its movement patterns are known in advance, InfoS knowledge (KPs) that only fall outside the QE relative location values cannot be discarded as they may become within range some time in the future.

5.7 Distributed Detection

Distribution is the key to load balancing, communication savings, and robustness. In SPS, I support distribution in two ways: decentralized placement of SMCs and distributed processing of SMCs. The former allows components to be spread and positioned on resourceful devices, while the latter aims at the decomposition and distribution of each individual SMC in the SPS. I discuss these under *distribution policy* and *distributed processing*, respectively.

5.7.1 Distribution Policy

SPS, as a Pub/Sub-centric framework, benefits from features and properties that come with Pub/Sub. The loose-coupling of event clients in Pub/Sub provides location transparency, where the location of event publishers and event subscribers does not affect the data-centric communication (see location decoupling in Section 2.3.1.1). Location decoupling allows SMCs (as event publishers) to be located anywhere in the network and relocated dynamically without affecting the corresponding event subscribers.

Since the Pub/Sub component is the only network-aware component of the SPS, it is held responsible for positioning SMCs within the network. The placement affects the downstream messaging cost (the cost of event delivery from the downstream publishers to the SMC's host) as well as the upstream messaging cost (the cost of delivering SMC events from the SMC's host to the upstream subscribers). The Pub/Sub component can locate the downstream publishers and the upstream subscribers, and position SMCs strategically to reduce communication costs. Assuming the downstream (input) event rate is higher than the upstream (output) event rate, the Pub/Sub component should aim to position SMCs closer to their downstream publishers than their upstream subscribers.

If QPS is used as the Pub/Sub component, SMCs may be placed on the logical Event Brokers (EBs) that are mapped to resourceful nodes. The logical Pub/Sub layer (in QPS) offers many ($\frac{4^N-1}{3}$, where N is the number of hierarchical GS levels) EBs for SMC placement. The nearest logical EB to the downstream publisher may be selected, and mapped to a resourceful EB in the network for initial SMC placement.

The only exceptions to the above are the SMCs which hold Query Expressions with relative location values in their selection parameters, i.e. $\{v_3\}, g_3 \subseteq D^r$ where v_3, g_3 belong to location selection parameter s_3 of a QE (definition 5.4). Since knowledge selection in these SMCs is relative to their hosts' locations, their positioning flexibility is restricted. They are replicated (with different signature values) and positioned *statically* at or closest to nodes that host their downstream publishers. These SMCs may be re-located if and only if the relative location values can be permanently mapped to absolute values following the initial placement. This permanent mapping is only possible if the SMC's host is stationary. The outlined policy is a unique case were the *data-centric abstraction* is by-passed and a *client-centric abstraction* is used for SMC placement. This allows for a number of services that are useful but not attainable with a data-centric abstraction. For example, users can associated heart-beat SMCs with individual publishers (as used later in Section 5.8.1.2) or count the number of publishers that are operating in the system.

SMC Relocation. The above scheme concerns the initial placement of SMCs. SMCs, if not statically positioned or localized at their downstream publishers' hosts, should be mobilized and shifted according to their run-time downstream and upstream messaging costs. This process, of course, needs to be transactional to ensure no events, states, or conditions are lost, and has all the traditional challenges of process migration [Zay87]. Existing methods, such as [BB03] and DFuse [KWA+03], can be used to incrementally shift SMCs to optimal locations within the network. These methods need to be integrated into the Pub/Sub component, and the SMC manager component needs to periodically dispatch its SMCs (to the Pub/Sub component) for relocation. This is beyond the scope of this work.

5.7.2 Distributed Processing

SPS supports the decomposition of SMC to distribute the SMC processing load across many network nodes. Complex SMCs are decomposed into simpler SMCs, which may be evaluated independently on different nodes.

This process may also reduce the overall processing and communication costs if information sharing and/or more effective SMC placements become possible. The decomposition of complex SMCs into simpler SMCs lengthens the information processing chain, hence increases the chance of information sharing among multiple independent event subscribers. Communication costs may also be reduced, if the decomposed SMCs can be placed closer to their downstream publishers in the network. SMC decomposition may be with respect to the SMC predicates or SMC QEs. These are discussed separately below.

5.7.2.1 Predicate Decomposition

An SMC predicate is a boolean expression, which may be decomposed using boolean algebra. These decompositions, however, are only effective if disjoint operands are produced (i.e. an SMC is decomposed into many SMCs, that hold mutually exclusive QEs). Every SMC then either examines a distinct part of the overall condition or joins the partial results to examine the overall condition. For example, the *TrafficCongestion* SMC (shown earlier in Table 5.4) may be decomposed as in Table 5.10. Note how the *validity* attribute is used in the new *TrafficCongestion* SMC to conveniently examine the status of the *IL_High* and *Car_Slow* lasting conditions in the predicates.

Intermediate *IL_High* and *Car_Slow* SMCs are introduced, which capture the pre-requisite conditions independently (see Figure 5.11). This decomposition decouples the *TrafficCongestion* SMC from the primitive *InductiveLoop* and *Car* event notifications that may be high rate and

Table 5.10: Decomposed TrafficCongestion SMC

(a) TrafficCongestion SMC

In.	Value
N	"TrafficCongestion"
Q	$A := ((closest, IL\_High, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(closest, ingress, \text{null}, \text{null}));$ $B := ((closest, CarSpeed\_Slow, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(multiple : any, \text{null}, \text{null}, D_S))$
P^n	$A.valid \ \&\& \ B.valid \ \&\& \ (A.location == B.location)$
A^n	$location := A.location; \ level := B.value;$
P^x	$!B.valid \ \&\& \ (B.location == last.location)$
A^x	$location := last.location; \ level := B.value;$

(b) IL_High SMC

In.	Value
N	"IL_High"
Q	$A := ((closest, InductiveLoop, \text{null}, \text{null}),$ $(aggregate, \text{null}, avg, (-30^r, 0^r)),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value > 2.5$
A^n	$location := A.location;$
P^x	$(A.value < 2.5) \ \&\& \ (A.location == last.location)$
A^x	

(c) CarSpeed_Slow SMC

In.	Value
N	"CarSpeed_Slow"
Q	$A := ((closest, CarSpeed, \text{null}, \text{null}),$ $(aggregate, \text{null}, avg, (-60^r, 0^r)),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value < 7$
A^n	$value := A.value;$
P^x	$(A.value > 15) \ \&\& \ (A.location == last.location)$
A^x	$value := A.value;$

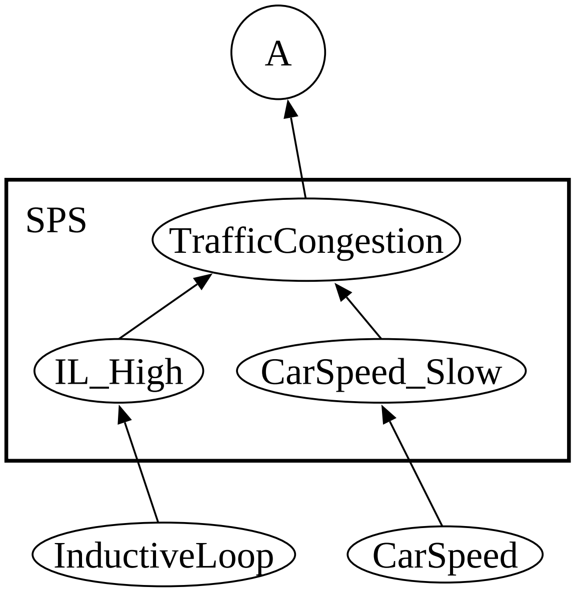


Figure 5.11: Decomposed traffic congestion information flow

expensive to process. Instead, these primitive event notifications are pre-processed and transformed into higher level knowledge (by the introduced SMCs) prior to undergoing examination for traffic congestion detections. These SMCs may capture meaningful conditions, that can also be shared (used) for other high-level conditions.

SMC predicate decomposition is not automated in SPS, as it requires careful consideration when complex predicates such as those examining condition interval relationships are involved. Galton and Augusto [GA02] discuss the complexities that arise when interval relationships are decomposed under the point-based time model.

5.7.2.2 QE Decomposition

Predicate decomposition leads to the separation of QEs, but each individual QE may also be decomposed. This decomposition distributes the SK search-space over a number of SMCs, such that every SMC searches a disjoint portion of the KP-combinations (Section 5.6.5).

QE decomposition splits the table of KPs that result from the knowledge selection process (Section 5.6.4) across multiple SMCs. This separation is achieved by decomposing the given group of values, G, in a QE's selection parameter, into a number of smaller groups, $\{g_1, g_2, \cdots \mid \bigcup_i g_i = G,\ i \neq j \Rightarrow g_i \cap g_j = \emptyset\}$. These smaller groups form parts of new QEs and thereby SMCs that capture conditions over a partition of the data. There are several issues that must be considered when decomposing QEs:

- It is only applicable to QEs that hold a *multiple* selection operation in one or more of their selection parameters.

- Segments of the SMC predicates, P^n and P^x, that involve the decomposing QE are evaluated as part of the new SMCs. These segments may be extended to the entire predicates, but must exclude references to the last SMC event.

- Context-based data processing over partitioned data is error-prone, i.e. the context of no individual decomposed SMC can be associated with the context of the overall condition. Hence, decomposed SMCs capture conditions that are momentarily valid and generate atomic SMC events. To achieve this, selected segments of the SMC predicates are examined as part of two separate SMCs, one for the P^n predicate segment and the other for the P^x predicate segment.

- SMC events that are published by the decomposed SMCs reflect conditions over partial data. These must be joined to examine the condition of the sub-operator over complete data. A join SMC must be specified that takes the decomposed SMC events (as input) and generates suitable SMC events that reflect the overall condition (as output). The type of join function depends on the involved sub-operator:

  multiple:one requires a logical *XOR* operation (over the decomposed SMC events) to ensure the uniqueness of the satisfied attribute value over the complete data.

  multiple:all requires a logical *AND* operation to ensure all attribute values (in the complete data) are satisfied across all decomposed SMCs.

  multiple:any requires a logical *OR* operation to yield only a single result (SMC event) from the decomposed SMCs.

  multiple:separate neither requires a join function nor a join SMC. The independence of the satisfied attribute values (indicated by this sub-operator) means that conditions can be captured independently. The two previous decomposition policies are also irrelevant for this sub-operator. Instead, the decomposed SMCs hold the same topic (name) value as the original SMC, but differ in signature (name) values.

  These join function correspondences are verified in Appendix B.2.

- The decomposition of the *multiple:separate* sub-operator is prioritized over the decomposition of the other *multiple* sub-operators.

Table 5.11 shows an example of this decomposition, for a *Fire* SMC that captures lasting fire conditions (indicated by high temperature readings across all sensors) in a predefined area. Table 5.11(a) shown the un-decomposed SMC, Tables 5.11(b) and 5.11(c) show the decomposed SMCs for the entrance predicate, Tables 5.11(d) and 5.11(e) show the decomposed SMCs for the exit predicate, and Table 5.11(f) shows the join SMC, which combines and processes the partial results from the decomposed SMCs.

Localization. Localization (or localized processing) is where an SMC and its downstream publishers are co-located (i.e. have the same host). In this case, published data is examined locally by the corresponding SMC, and messaging is confined to the local node.

Table 5.11: Fire SMC (singular and decomposed)

(a) Singular Fire SMC

In.	Value
N	"Fire"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: all, \text{null}, \text{null}, (-10, +10)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value > 50$
P^x	$A.value < 30$

(b) Decomposed (entrance,range1) Fire SMC

In.	Value
N	"Fire_EntRange1"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: all, \text{null}, \text{null}, (-10, 0)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value > 50$
P^x	$true$

(c) Decomposed (entrance,range2) Fire SMC

In.	Value
N	"Fire_EntRange2"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: all, \text{null}, \text{null}, (1, +10)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value > 50$
P^x	$true$

(d) Decomposed (exit,range1) Fire SMC

In.	Value
N	"Fire_ExtRange1"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: all, \text{null}, \text{null}, (-10, 0)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value < 30$
P^x	$true$

(e) Decomposed (exit,range2) Fire SMC

In.	Value
N	"Fire_ExtRange2"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: all, \text{null}, \text{null}, (1, +10)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.value < 30$
P^x	$true$

(f) Join SMC

In.	Value
N	"Fire"
Q	$A := ((closest, Fire\_EntRange1, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: any, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}));$ $B := ((closest, Fire\_EntRange2, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: any, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}));$ $C := ((closest, Fire\_ExtRange1, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: any, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}));$ $D := ((closest, Fire\_ExtRange2, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple: any, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}));$
P^n	$A.valid$ && $B.valid$
P^x	$C.valid$ && $D.valid$

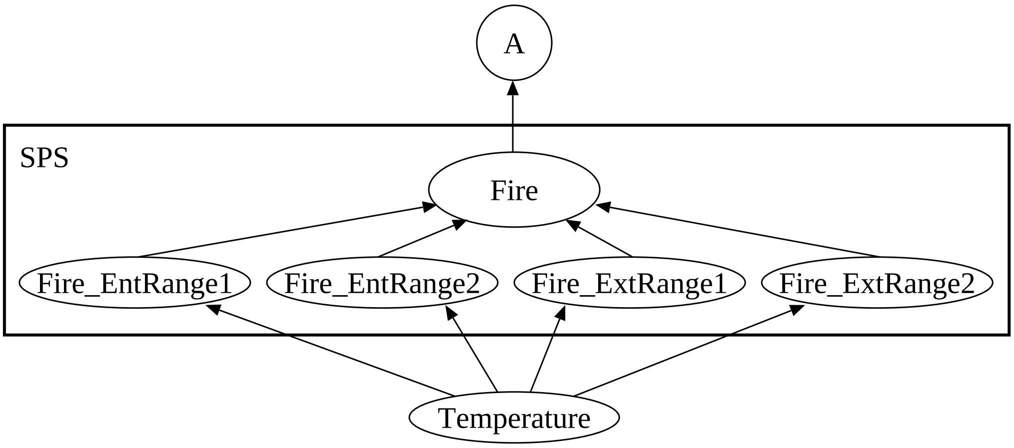


Figure 5.12: Decomposed fire detection information flow

Non-deterministic factors	Network layer	Pub/Sub layer	Clients (SPS) layer
Node failure	Topology maintenance	EDT maintenance	Component robustness
Clock drift	Time synchronization	×	×
Location drift	Localization	×	×
Network delay	×	×	InfoS consistency
Packet loss	Reliable delivery	×	×

Table 5.12: Non-deterministic factors and their treatments

Localization may be achieved by decomposing the location selection parameter of (SMC) QEs. If the downstream publishers' advertisement regions, $A = \{a\}$, are geographically disjoint (i.e. $i \neq j \Rightarrow a_i \cap a_j = \emptyset$, where i, j are two event publishers), then the group of location values (the region) of a QE, G, may be decomposed into a set of smaller groups (regions) $G_d = \{g\}$, such that $\forall g \in G_d \; \exists a \in A \; : \; g \subseteq a$. Then, the decomposed SMCs may be positioned on nodes that host their corresponding publishers for localized processing.

5.8 Reliability Model

SPS features a largely deterministic information processing model, which helps independent users to collaboratively build hierarchical levels of knowledge in the system. A high-level condition that is captured by one SMC may be used by another and so on until end-point subscribers are reached. The SPS reliability model maintains this deterministic operation in the view of non-deterministic factors, such as failures and delays. These non-deterministic factors largely originate from the environment and affect nodal and network behaviors. Table 5.12 shows a

SPS component	Failure resolution	
	low (moderate) failure	high (mass) failure
External publishers	deployed redundancy	unsubscribe
External subscribers	unsubscribe	unsubscribe
SMCs	replicated recovery	unsubscribe

Table 5.13: Component failure resolutions

list of the most influential factors in SPS, and outlines treatments that are provided by each architectural layer in each case.

As shown in the table, SPS does not treat every non-deterministic factor, but only focuses on those that are overlooked or have cross-layer impact. Thanks to the SPS's layered architecture, many concerns that are met at the lower layers do not need further attention at the upper layers. Factors that are treated by the SPS (excluding the Pub/Sub component) are further discussed below.

Network delay (interference) Network connectivity is a complex and environmental-dependent phenomenon. Interference, congestion, and different messaging path lengths contribute to non-deterministic network delays when messages are routed from sources to destinations. This delay results in *unordered event delivery*, where distributed event subscribers receive notifications out of order. In turn, unordered event delivery means that the state of distributed InfoS components is inconsistent, and thus could lead to non-deterministic or conflicting condition detection across the network. SPS provides a best-effort InfoS consistency mechanism that combats this effect. Users can enhance this consistency to a *guaranteed* consistency model on demand.

Node failures Node failures are often unforeseen and abrupt in sensor systems. SPS components may be lost when nodes happen to fail. More importantly, user interests, contexts, and most recent SMC events may be lost if SMCs are lost. In order to prevent this, SMC structures are replicated at nearby nodes and activated when primary SMCs happen to fail. This strategy is detailed in Section 5.8.2.

5.8.1 InfoS Consistency

Since SMCs deduce high-level information from InfoS data, it is important to maintain a consistent (and preferably correct) view of the world across the InfoS components. This consistency reflects a unified view of the world, and makes the distribution of knowledge transparent to the SMCs. Inconsistencies may emerge as a result of unordered event delivery by the Pub/Sub component (the sole introducer of knowledge into the InfoS), and dynamic event subscriptions that are initiated by InfoS components.

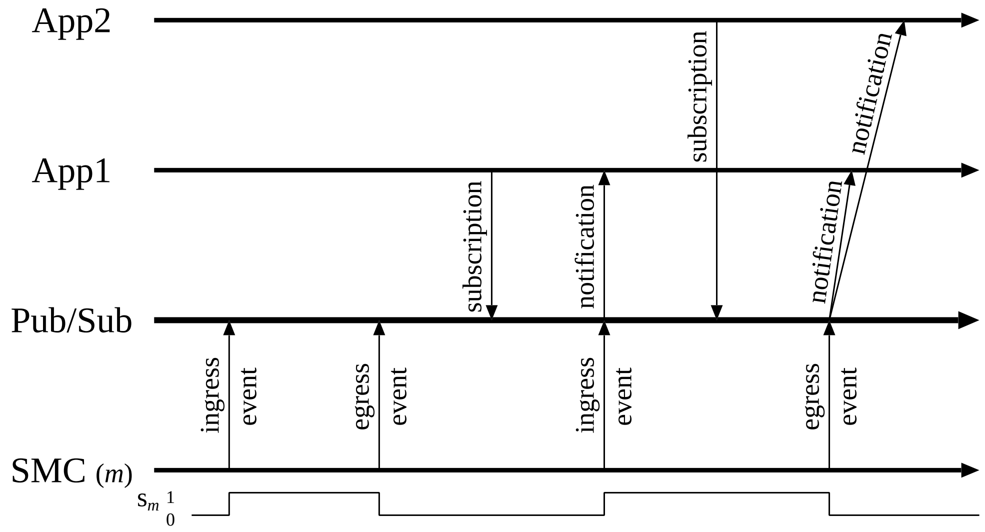


Figure 5.13: Conflict of information (due to dynamic subscriptions)

5.8.1.1 Initial Consistency

When an SMC is activated (at the SMC manager component), the SMC QEs are dispatched to the local InfoS component. The InfoS component makes relevant event subscriptions, and receives events from the Pub/Sub component. The Pub/Sub component delivers events that are published from the moment when event subscriptions are resolved to the moment when an unsubscribe operation occurs. This time period bounds the knowledge that is available for SMC examination. This bounded knowledge may affect initial SMC evaluations, as well as QEs that select knowledge based on absolute time values.

Furthermore, the use of event pairs to capture lasting conditions may lead to conflicting knowledge at the subscriber's side when dynamic subscriptions are involved. Event subscribers (e.g. App1) who have subscribed prior to condition occurrences receive event notifications appropriately, but those who have subscribed (e.g. App2) during a condition's active interval miss the ingress event and only receive the condition's egress event (see Figure 5.13). This leads to conflicting information at the subscribers, where some (App1) know about a condition's initiation and some others (App2) don't (though they do realise the missed ingress event after receiving the egress event).

The above two cases argue for initial consistency, where knowledge (that is published in the past) may need to be stored and delivered to the subscribers who arrive after the event publications. This is achieved by *persistent event storage* at the Pub/Sub component. In order to minimise storage costs, I provide a lightweight weak consistency mechanism as follows.

- The Pub/Sub protocol stores the most recent (largest timestamped) event notification that is published by each event publisher in SPS. This bears little cost in the case of SMCs, as their last events, e, are already stored within their data structures - the local Pub/Sub component may simply hold references to these last SMC events. The storage of most recent event publications by the external publishers, however, may incur additional costs[1].

- The Pub/Sub protocol, upon encountering a new event subscription, delivers the last set of events that are published by the corresponding event publishers to the new subscriber. This prevents the conflict of information that may arise due to dynamic event subscriptions, and provides *weak* InfoS consistency following SMC activations. InfoS consistency improves as the InfoS time advances and more recent knowledge becomes available.

5.8.1.2 Run-time Consistency

Event dissemination in sensor networks is subject to network delay. If unmanaged by the Pub/Sub protocol, this could result in unordered event delivery to event subscribers. SPS provides a lightweight best-effort consistency mechanism for maintaining a consistent view of knowledge across distributed InfoS components. Application clients may improve this consistency to a guaranteed consistency model, by using SMCs, when higher reliability is desired.

The best-effort mechanism consists of an input buffer and an InfoS time, that compensates for the unordered event delivery by providing best-effort ordered delivery at the subscriber-side. Received events, from the Pub/Sub component, are queued and ordered according to their timestamps prior to insertion into the InfoS cube structure. A monotonically increasing InfoS time is introduced that traverses the time values in a discrete manner, $t_I \in D_T$. At every InfoS time, the cube structure represents a view of the world that is composed of all events timestamped equal or less than t_I, $\{e|e \in \mathbb{E},\ v_2 \le t_I\}$.

The time interval between the system clock[2] (mapped to the time domain $t_G \in D_T$) and the InfoS time t_I is called the *stabilization interval*, $t_s = t_G - t_I$. Events are stabilized within this interval, meaning that they traverse the network and reach all their corresponding event subscribers. Events, that have a timestamp greater than the InfoS time, $e \in \mathbb{E} : v_2 > t_I$, are considered *unstable* and remain in the input buffer. Because of SPS's hierarchical information processing model, information may propagate through multiple SMCs before reaching another SMC. Thus multiple InfoS times, $T_I = \{t_I^1, t_I^2, \cdots\}$, are introduced that correspond to different levels of information processing and are separated by the stabilization interval t_s,

$$T_I = \{t_I^i \in D_T | t_I^i = t_I^{(i-1)} - t_s\}, \tag{5.21}$$

where the non-member element t_I^0 denotes the mapped system clock t_G, $t_I^0 = t_G$. Note that with this setup, the InfoS also needs to maintain a set of host location values, $L_H = \{l_H^i | l_H^i = l_H \text{ at } t_I^i\}$.

The stabilization interval, t_s, aims to find an upper bound for the event delivery latency in the network. Network latency, however, is a variable quantity and subject to network dynamics

[1]The network layer often stores these events to ensure reliable delivery, in which case references can be used again (in a cross-layer implementation) to minimize storage costs.

[2]I assume the operation of internal and external time synchronization protocols, see Sections 2.1.3 and 2.4.2, that bound the system clock variation from the global time.

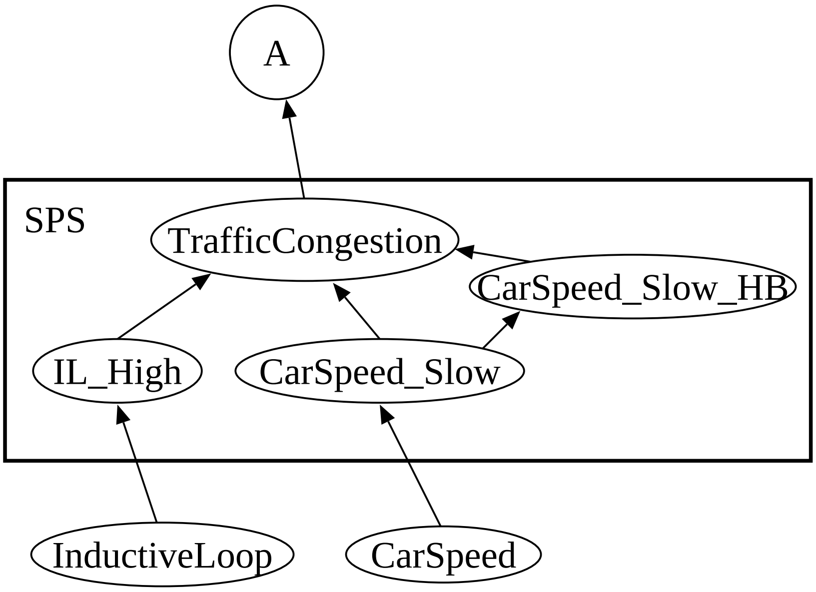


Figure 5.14: Reliable traffic congestion detection

such as node or link failures. InfoS components continuously monitor the event delivery latency and adjust the stabilization interval within their connected network[1]. This run-time consistency mechanism reduces the chances of events arriving late (i.e. having a timestamp earlier than the InfoS time), but is still subject to failure when network latency changes abruptly or network disconnections occur[2].

Guaranteed Consistency. SPS does not provide guaranteed InfoS consistency, due to its costs and complexity. Instead, the expressiveness of SMCs can be used to emulate guaranteed InfoS consistency. This provides a degree of flexibility and user control, where the costs of guaranteed InfoS consistency are justified by the user's decision. The approach involves *HeartBeat* SMCs that monitor the downstream publishers (at their local nodes) and generate periodic updates (heart-beat signals) that contain their most recent event publications. At the consumer-side, the heart-beat signals may be used to ensure reliable detection.

Consider the two introduced SMCs in Table 5.14. The *CarSlow_HB* SMC monitors the local *Car_Slow* SMC and generates periodic heart-beats that contain the last *Car_Slow* SMC event. The *TrafficCongestion_Reliable* SMC validates its condition detection against a recent heart-beat signal, and revokes the detection if continuous heart-beat signals cease to exist. In this setup, high network latency or network disconnections result in absence of heart-beat signals which prevent the *TrafficCongestion_Reliable* SMC from capturing false-positives. Correct detection resumes when SPS's best-effort InfoS consistency mechanism has resolved the inconsistencies and/or network disconnections have been repaired.

This approach has two useful features when network disconnections occur.

[1]This is preferably achieved in conjunction with the internal time synchronization process.
[2]Late events are inserted into the InfoS cube and may result in some inconsistencies.

Table 5.14: Reliable TrafficCongestion SMC

(a) CarSpeed_Slow_HB SMC

In.	Value
N	"CarSpeed_Slow_HB"
Q	$A := ((closest, CarSpeed\_Slow, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : any, \text{null}, \text{null}, D_S))$
P^n	$(last.valid == 0) \mid\mid (last.time < 0^r - 10)$
A^n	$location := A.location;$
P^x	$true$
A^x	

(b) TrafficCongestion_Reliable SMC

In.	Value
N	"TrafficCongestion_Reliable"
Q	$A := ((closest, IL\_High, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(closest, ingress, \text{null}, \text{null}));$ $B := ((closest, CarSpeed\_Slow, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(multiple : any, \text{null}, \text{null}, D_S));$ $C := ((closest, CarSpeed\_Slow\_HB, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : any, \text{null}, \text{null}, D_L),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$A.valid \;\&\&\; B.valid \;\&\&\; (A.location == B.location == C.location) \;\&\&\; (C.time \geq (0^r - 10))$
A^n	$location := A.location;\; level := B.value;$
P^x	$(!B.valid \;\&\&\; (B.location == last.location)) \mid\mid ((C.time < (0^r - 10)) \;\&\&\; (C.location == last.location))$
A^x	$location := last.location;\; level := B.value;$

- Affected SMCs are halted; their conditions are not detected because heart-beat signals are no longer received.

- The InfoS time does not stop; therefore conditions that can receive their input data are continually examined and detected by SMCs. I refer to this as *disconnected operations* in SPS, where despite network disconnections, unaffected conditions are still examined and captured in the framework.

5.8.2 SMC Replication

SMC distribution avoids a single point of failure, but increases the chance of partial failure where a node that houses some SMCs is more likely to fail and affect some condition detection. These failures are often permanent, as sensor nodes often receive no maintenance. SPS adopts a *replicated storage* policy to protect condition definitions (at SMCs) against failure. The Pub/Sub protocol stores SMC replicas at nodes that are close (topologically near) to the selected SMC hosts. These replicas are passive, and are only activated when failures have been detected.

The Pub/Sub component can learn about node failures to maintain its EDT and/or activate the stored SMC replicas. When a failure is detected, replicas of the SMCs that were on the failed node are retrieved from storage and dispatched to suitable SMC managers as discussed in Section 5.7.1. These activations initiate the SMC and condition detections as described in Section 5.6.

Loss of SMCs' context (data structures). To save communication, SPS neither replicates condition detections (with SMC replicas), nor synchronizes SMC data structures. Instead, SMC replicas (when activated) initiate from an undetected state ($s = 0$ and $e =$ null, where s and e are the SMC status bit and the last SMC event), and re-capture the conditions if they are still active at the time of their activation.

Two factors aid this re-capture: (a) the initial InfoS consistency mechanism (Section 5.8.1.1) provides some knowledge about the past, (b) future data (event publications) in sensor systems are expected to repeat the observed knowledge about the environment (e.g. if traffic congestion persists on a road, then subsequent car and inductive loop event publications are expected to re-iterate this information and result in the re-capture of the traffic congestion condition). Application clients can introduce (active) SMC replicas (with different names), if the SMC data structures are valuable and non-recoverable. This leads to replicated monitoring and condition detection, providing higher robustness at the expense of increased communication and computation costs. An example of an SMC whose data structure is non-recoverable is one that maintains the count of people in a building - it is impossible to recover this data structure without revisiting the entire historic data. The monotonically increasing nature of the InfoS time disallows roll-backs.

5.9 Evaluation

In this section I evaluate the proposed SPS framework. This evaluation takes note of the expressiveness of the framework, as well as its performance in the context of a realistic application scenario. I initially discuss the expressiveness of SMCs and then describe a prototype implementation of SPS, an evaluation application scenario, and discuss the performance results of SPS in operation. The goal of this evaluation is to demonstrate the usability and efficiency of SPS for a wide range of sensor system applications described as "smart environments".

5.9.1 Expressiveness

The expressiveness of SMCs is determined by the range and type of conditions that they can capture. Sections 5.4 and 5.6 elaborated on these conditions, their specification and detection in SPS, but this section provides an overview of SMC's expressive features - their strengths and weaknesses.

An SMC detects a condition in three steps: knowledge selection, knowledge examination, and knowledge encapsulation. The knowledge examination (middle) phase provides the most expressiveness, enabling individual and cross-examination of KP attribute values with a range of mathematical, comparative, and logical operators (within the SMC predicates). Dual SMC predicates, and access to the last SMC event further enhance this expressiveness, providing *context-based data processing* and support for *memory-based condition detection*. This expressiveness is partially limited by the fact that the examination of each KP-combination is independent of any other. This independence is partially controlled by the knowledge encapsulation phase, discussed below.

The knowledge selection (first) phase is where input data (KPs) for condition detection is retrieved from the InfoS. Although the semantics of knowledge selection is limited to a few predefined attributes (topic, time, location, and status), the expressiveness of knowledge selection (about these attributes) is high: the three selection operations aid knowledge confinement in different ways (Section 5.4.1), and the introduced absolute and relative data domains enhance the *contextual* selection of data. Shortly, I will describe two SMCs that exploit this contextual awareness to provide useful services.

Finally, the knowledge encapsulation (last) phase asserts some user-defined conditions over the SK (set of independent KP-combinations that have satisfied the SMC predicate). These assertions enhance expressiveness, but are likewise limited to a set of fixed attributes. The introduced *multiple* sub-operators (*separate*, *one*, *all*, and *any*) support partitioning, singleness, entireness, and randomness for the fixed event attribute values in the SK, with the latter sub-operator often used as a dummy operator for no assertions. Combination of these sub-operators (in different QEs) can be used to achieve more complex assertions over the SK.

SPS's extended support for topic, time, location, and status attributes makes it suitable for detecting conditions that are typed, temporal, or spatial in the environment. I have already demonstrated SMC's expressiveness by examples, including the *TrafficCongestion* SMC and the

Table 5.15: Filtering SMCs

(a) Temperature_0.5Hz SMC

In.	Value
N	"Temperature_0.5Hz"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : any, \text{null}, \text{null}, (-10, +10)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$last.time \leq 0^r - 2$
A^n	$value = A.value$
P^x	$true$
A^x	

(b) Temperature_10%Change SMC

In.	Value
N	"Temperature_10%Change"
Q	$A := ((closest, Temperature, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(multiple : any, \text{null}, \text{null}, (-10, +10)),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$\lvert A.value - last.value \rvert \geq 0.10 * last.value$
A^n	$value = A.value$
P^x	$true$
A^x	

CarSpeed_Slow_HB SMC (for reliable condition detection). Below, I outline another that exploits contextual awareness.

Controlling the rate of events. Although Pub/Sub subscribers have no control over the rate of event publication, the need for this control is evident. Consider a temperature sensor that publishes temperature readings (events) every second. While this granularity is suited to some applications (e.g. fire breakout monitoring), it may be too fine-grained for others (e.g. daily temperature logging). Subscriber-asserted control over the rate of event publication is useful, particularly when communication is a scarce resource.

In SPS, a subscriber may define an SMC that filters events according to a custom specification. Table 5.15 shows two SMCs that limit the rate of events, that are delivered to their subscribers, either by time or by value. *Temperature_0.5Hz* SMC publishes the most recent *Temperature* event every $2s$, maintaining a fixed $0.5Hz$ event publication rate. *Temperature_10%Change* SMC passes an event when the *value* attribute has changed by more than 10% (relative to the previously passed event's *value*).

5.9.2 Simulation Environment

The proposed framework, SPS, was implemented on JiST/SWANS [BHvR05] to leverage from the already developed Pub/Sub protocol, QPS. Additional enhancements were made to QPS to use it as the Pub/Sub component, and limitations were realised that were due to QPS's constraints. I discuss the prototype implementation of each SPS component below.

Pub/Sub component. QPS EBs were used as Pub/Sub components in SPS. These were extended to suit the SPS framework as follows.

- QPS's API was extended to support SMC insertion and delivery to SMC manager components.
- The SMC distribution policy (Section 5.7) was implemented in QPS.

- Support for initial InfoS consistency, as discussed in Section 5.8.1.1, was implemented by storing the most recent events at the publisher-hosting EBs, and delivering them whenever a new related subscription was realised. An event retrieval request passed the subscription coverage points and reached all the relevant publisher-hosting EBs.

The use of QPS also led to some limitations, that are listed below.

- Mobile networks could not be supported (QPS was restricted to static networks). I supported mobile subscribers, for my experimental setup, using *proxies* [CCW03] that maintained subscribers location and redirected events towards them.

- Subscription to a group of event topics was not possible; QPS supported a single event topic per event subscription.

- QPS only supported subscriptions to absolute values; hence subscription to relative location values were not possible unless mapped to absolute values (by the InfoS component) as described in Section 5.3.3. A naive form of subscription to an 0^r (relative) location value was supported by resolving the subscription at the local node. This minor support was necessary for my experimentation, described in the next section.

InfoS component. The InfoS was implemented as a simple MDX cube; knowledge was stored in a multi-dimensional indexed table. The table eased attribute-based grouping of knowledge for selection parameter evaluation - note that the *closest* and *aggregate* selection operations involve grouping operations. KPs were stored in non-compressed format, and relational queries were resolved using non-optimized instructions to gain insight into SPS's basic performance. The following policies were also adopted to reduce operational memory overhead.

- QEs were passed by reference to the InfoS component; thus the InfoS maintained a table of references to the original QEs maintained at the SMCs.

- A simple hashtable was used to speed up the pairing of ingress and egress events within the InfoS cube. The SMC names were used as *keys* to the hashtable to retrieve the location and time index of the corresponding event pair.

- A sub-component, QE Analyzer, was implemented to translate QEs with relative values or ranges into QEs with absolute values and ranges. This sub-component also indicated whether the translated relative range is momentarily or permanently valid in SPS. It was used by the InfoS as well as the Pub/Sub component (for SMC distribution analysis). The SMC manager also used this to receive mapped absolute values for the 0^r relative values.

SMC manager component. The SMC manager component maintained SMCs as instances of a class, and evaluated them according to the SPS detection model; BeanShell [BS] was used to examine the SMC predicates. SMC decomposition by predicates was assumed to be performed externally and SMC decomposition by QEs was supported to allow the automatic distribution and detection of conditions in the network.

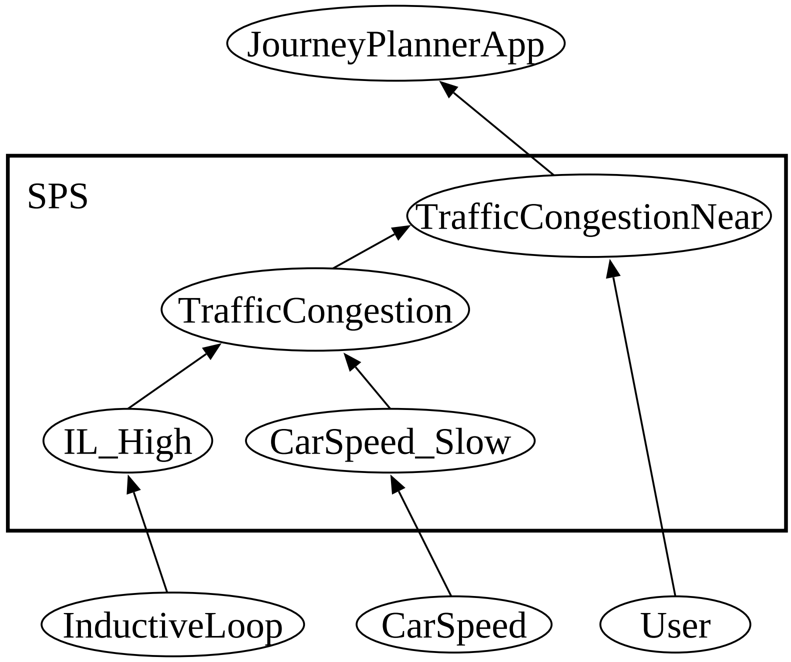


Figure 5.15: Application scenario overview

5.9.3 Experiment: Journey Planner Application

An application scenario was implemented and tested (using real-data from SCOOT [SCO]) to observe the SPS performance. The application was an extension of the traffic congestion example, used throughout this chapter. The SPS clients comprised 620 (external) event publishers and 500 (external) event subscribers. Every external event publisher belonged to one of three types, described below.

Inductive loop sensors These clients monitored individual roads and provided periodic reports on road-segment occupancies; published periodic *InductiveLoop (IL)* event notifications at 1Hz. Raw SCOOT data was used to represent these event publications.

Speed measurement sensors These clients reported on the speed of individual vehicles, passing by in a single direction, on each road. They published irregular, but potentially high-volume, *CarSpeed* event notifications. The data was inferred from a secondary stream of raw SCOOT data.

Location sensors These clients reported on the location of individual application clients, that were simulated in the experiment. They published periodic *User* events at 1Hz. The data was generated by the simulation engine.

Apart from the location sensors, whose data were simulated, all other sensors exhibited temporary failures by means of missed data (event notifications). The external event subscribers were of a single type, *Journey Planner Applications*. These clients were co-located with their (mobile) users, and subscribed to real-time traffic information to aid their users in planning their journeys. The application clients subscribed to the *TrafficCongestionNear* event topic, whose corresponding SMC (event publisher) is illustrated in Table 5.16. This SMC filtered traffic congestion reports according to the present location of the user[1]. The SMC passed real-time reports that were situated within the 2 road-junction distance of the user's location. The manually decomposed version of the *TrafficCongestion* SMC (shown in Table 5.10) was introduced to transform low-level data into high-level traffic congestion reports.

The overall information flow diagram is shown in Figure 5.15, and the experimental parameters are shown in Table 5.17. To examine SPS's highest performance, a total of 1000 nodes were used to allow complete SMC decomposition and distribution. A larger number would not affect data processing, but only increase the cost of event routing and latency at the network layer. A grid size of 256 $\times$ 256 ensured that every node had at least 5 neighboring nodes, thereby prevented network disconnections. I selected real data from two distinct days, one relating to a weekday and the other relating to a weekend, in the hope of detecting more traffic congestion on the weekday than the weekend. The two days were Saturday 1^{st} of *July*, 2006 for Exp1 and Thursday 6^{th} of *April*, 2006 for Exp2. Sensor data was examined from early morning ($1AM$) to late evening ($9PM$) when traffic congestion was expected to decease. This expectation was later confirmed when captured traffic congestion conditions were studied. To verify SPS's operation and correctness, the set of high-level events that were delivered to the SPS's external event subscribers were compared against an alternative centralized implementation. The SPS operation and its performance results are discussed below. My performance evaluation takes note of three vital resources: *processing*, *storage*, and *communications*; and explores how efficiently these resources were utilized to achieve the desired functionality.

5.9.3.1 Operational setup

The operational setup of SPS is defined by the decomposition and distribution of SMCs, and the resolution of resulting event subscriptions that form the information processing chain shown in Figure 5.15. Table 5.18 shows the statistics about the SMCs and the event subscriptions.

The *TrafficCongestionNear* SMC was replicated and positioned on nodes that hosted the *User* event publishers (i.e. the mobile user nodes) in accordance with the SPS distribution policy (Section 5.7.1). They could not be relocated, and totalled 500 SMCs as indicated in the table. The *TrafficCongestion* SMC was decomposed along its QEs to distribute the detection of traffic congestion conditions across multiple nodes. The range of location selection parameters of A and B QEs were decomposed into 16 segments each, resulting in a total of 256 decomposed *TrafficCongestion* SMCs. The majority of these were ineffective because the $A.location == B.location$ condition (imposed by the SMC entrance predicate) meant that only decomposed SMCs with matching location ranges could detect the condition. Thus, only 16 *TrafficCongestion* SMCs were observed to capture traffic congestion conditions in the system.

[1] I decided to use location values from the *User* events for a more natural application setting, though the 0^r relative location value could equally be used, i.e. the entrance predicate could be written as $|A.location - 0^r| \leq 2$.

Index	Value
N	"TrafficCongestionNear"
Q	$A := ((closest, TrafficCongestion, \text{null}, \text{null}),$ $(multiple : any, \text{null}, \text{null}, (0^r, 0^r)),$ $(multiple : separate, \text{null}, \text{null}, D_L),$ $(multiple : any, \text{null}, \text{null}, D_S));$ $B := ((closest, User, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(closest, 0^r, \text{null}, \text{null}),$ $(closest, atomic, \text{null}, \text{null}))$
P^n	$\lvert A.location - B.location \rvert \leq 2$
A^n	$location := A.location$
P^x	$true$
A^x	

Table 5.16: TrafficCongestionNear SMC

The *IL_High* and *CarSpeed_Slow* SMCs were also decomposed along QEs, but with the difference that each of these only relied on a single event topic for its input data; therefore localization was achieved. These decomposed SMCs matched their corresponding number of publishers, and totalled 60 each. SMC decomposition helped to balance the overall storage and processing load across the network.

5.9.3.2 Processing

The processing complexity of SPS relates to the cost of SMC evaluations. SMCs are evaluated whenever a new table of KPs are received from the InfoS. In these experiments, this only occurred when a new event was received at the InfoS. Table 5.19 shows that SMC decomposition lowered the number of events that were received at any one InfoS, thus reducing the frequency of SMC evaluations at each node. For example, although the *IL* event publications total 4.32e+6 events, the maximum number of *IL* events (received at any InfoS) totals just 72000 events as a result of SMC decomposition and localization. Similarly, the maximum number of events that was received at any one InfoS, co-located with a *TrafficCongestion* SMC, was lowered from 1616 (Exp1) and 3588 (Exp2) events to 236 and 701 events, respectively. These figures are 14.6% and 19.5% of the 1616 and 3588 numbers, which are the sums of *IL_High* and *CarSpeed_Slow* event publications.

The processing complexity of all SMC predicates was n, except for the *TrafficCongestion* SMC predicates. This means that an incoming event (in most cases) triggered only a single KP-combination examination at the SMC manager. The maximum processing complexity, observed for the *TrafficCongestion* SMC, was $10n$ (Exp1) and $14n$ (Exp2), indicating that at worst-case an SMC manager component examined 10 (Exp1) and 14 (Exp2) KP-combinations that involved

Parameter	Value
Simulation parameters	
simulation grid size	256 × 256
number of nodes	1000
number of roads	60
number of experiments	2
duration of experiments	20 hours
Topological parameters	
number of nodes housing SPS components	1000
number of Journey Planner Application clients	500
number of Inductive loop sensors	60
number of Speed measurement sensors	60
number of Location sensors	500
Input parameters	*Exp1 / Exp2*
number of SPS client subscriptions	500
number of SMC insertions	4
initial event stabilization interval period	3000 ticks
number of *IL* event publications	4.32e+6
number of *CarSpeed* event publications	330736 / 494682
number of *User* event publications	3.6e+7

Table 5.17: Experiment parameters

a newly received event notification. This compares to $74n$ (Exp1) and $90n$ (Exp2) processing complexity that would have been realised had the *TrafficCongestion* SMC not been decomposed. This processing complexity relates to the receipt of *IL_High* event notifications, that were examined against all pairs of recent *CarSpeed_Slow* KPs according to the *TrafficCongestion* SMC.

5.9.3.3 Storage

SMCs and knowledge are the two main elements that require storage in SPS. Table 5.20 shows that SMC distribution has resulted in a maximum of one SMC allocation per node in the network. The maximum number of *observed* SMCs (per node) reflects the maximum number of temporarily spawned SMCs, which reflected the concurrent capture of traffic congestion conditions at any one SMC. Table 5.20 also shows that a total of 876 SMCs served all (500) mobile users. From these 876 SMCs, 376 SMCs were shared and collaboratively deduced the traffic congestion information for the entire system. This sharing was achieved by the Pub/Sub component which interconnected independent event subscribers (with overlapping interests) to the same set of event publishers (SMCs), thereby avoiding duplicate data storage and processing in the system.

Statistics	Exp1 & Exp2
SMC decomposition & distribution	
number of SMC insertions	4
number of SMCs decomposed along predicates	0
number of SMCs decomposed along QEs	3
total number of decomposed SMCs	876
SMC counts	
number of *TrafficCongestionNear* SMCs	500
number of *TrafficCongestion* SMCs	256
number of *CarSpeed_Slow* SMCs	60
number of *IL_High* SMCs	60
Event subscribers	
total number of event subscribers	1376
external (SPS client) event subscribers	500 (36%)
internal (InfoS) event subscribers	876 (64%)
Event subscriptions	
total number of event subscriptions	2132
subscriptions by SPS clients	500 (23%)
subscriptions by InfoSs	1632 (77%)

Table 5.18: SPS operational performance

The highest number of KPs, stored at any one InfoS, related to the knowledge stored for the *Car_Slow* SMCs. The highest numbers of *Car* KPs, stored for deducing the aggregation information, were 37 (Exp1) and 30 (Exp2). These figures exclude any compressions or functional optimizations that could further reduce this storage. Similarly, the highest numbers of KPs stored for the *TrafficCongestion* SMCs were 14 and 21 KPs for Exp1 and Exp2, respectively. This indicates that the aforementioned 236 and 701 input events (in Table 5.19) continuously updated and overrode 14 and 21 storage units at the corresponding InfoS components. The same analogy holds for InfoS components co-located with other types of SMCs, i.e. 72000 input events only updated 30 storage units at the InfoS component co-located with an *IL_High* SMC, 11548 (Exp1) and 17940 (Exp2) input events updated 37 (Exp1) and 30 (Exp2) storage units within InfoS components co-located with two *CarSpeed_Slow* SMCs, and 72168 (Exp1) and 72296 (Exp2) input events updated just 2 storage units within InfoS components co-located with the *TrafficCongestionNear* SMCs. Finally, a maximum of just one event per node, and a total of 620 events were stored at the Pub/Sub components to support initial InfoS consistency.

Statistics	Exp1	Exp2
Processing complexity		
maximum number of events, received at an InfoS,		
for the *IL_High* SMC	72000	72000
for the *CarSpeed_Slow* SMC	11548	17940
for the *TrafficCongestion* SMC	236	701
for the *TrafficCongestionNear* SMC	72168	72296
maximum number of predicate evaluations		
per *IL_High* SMC evaluation	1	1
per *CarSpeed_Slow* SMC evaluation	1	1
per *TrafficCongestion* SMC evaluation	10	14
per *TrafficCongestionNear* SMC evaluation	1	1
Condition detections		
number of *IL_High* event publications	842	2540
number of *CarSpeed_Slow* event publications	774	1048
number of *TrafficCongestion* event publications	168	296
number of *TrafficCongestionNear* event publications	4032	7104

Table 5.19: SPS computational performance

Statistics	Exp1	Exp2
SMC storage		
number of decomposed and distributed SMCs	876	876
maximum number of SMC placement per node	1	1
maximum number of observed SMCs per node	4	6
Knowledge (KP) storage		
maximum number of KPs, maintained at an InfoS		
co-located with an *IL_High* SMC	30	30
co-located with a *CarSpeed_Slow* SMC	37	30
co-located with a *TrafficCongestion* SMC	14	21
co-located with a *TrafficCongestionNear* SMC	2	2
Event storage		
total number of events, stored at Pub/Sub components	620	620
maximum number of events, stored at one Pub/Sub component	1	1

Table 5.20: SPS storage performance

Statistics	Exp1	Exp2
Event subscriptions		
maximum number of subscriptions per InfoS	2	2
Events		
total number of event publications	40656552	40825670
categorized by subscriber		
external (SPS client) event publications	40650736	40814682
internal (SMC) event publications	5816	10988
categorized by communication cost		
events, disseminated in the network	1784	3884
(events delivered in the network)	(109856)	(205408)
events, delivered locally	40654768	40821786
to SPS (subscriber) clients	4032	7104
to InfoS components	40650736	40814682

Table 5.21: SPS communication performance

5.9.3.4 Communication

Communication costs are often measured by the total energy used to deliver events from the publishers to the subscribers. This largely depends on the network structure and the performance of the adopted Pub/Sub protocol. Nonetheless, because the distribution of SMCs impacts the formation of Pub/Sub links, I have measured this cost by examining the "number of event notifications that were disseminated" and "the number of events that were delivered in the network".

Table 5.21 shows that out of the 40656552 (Exp1) and 40825670 (Exp2) event notifications that were published in the system, only 1784 (Exp1) and 3884 (Exp2) events were disseminated in the network. These figures account for 0.0044% and 0.0095% of the total event publications in Exp1 and Exp2, respectively. Two factors contributed to these small percentages. Firstly, the decomposition and localization of *IL_High* and *CarSpeed_Slow* SMCs as well as the partial localization of the *TrafficCongestionNear* SMC led to localized processing of a substantial portion of these events in the system. Secondly, the context-based data processing feature of SMCs meant that only a small number of transitive and highly informative events (5816 events in Exp1, and 10988 events in Exp2) were published by SMCs.

The number of events that were delivered in the network is substantially higher than the number of disseminated events because some events (e.g. the *TrafficCongestion* events) were forwarded to a large number of event subscribers (e.g. 500 InfoS components that were co-located with the *TrafficCongestionNear* SMCs).

5.10 Related Work

In this section I provide an overview of related work. I discuss similarities and differences between SPS and three classes of research:

State-based Approaches. SPS is not the first framework to use the notion of state for sensor systems. Others [KR05; LCRZ03; LCL+04; ABE+04; SB07] have also provided the expressiveness of states to sensor network applications. But they are mainly based on the principles of FSMs or enhanced state hierarchy and concurrency models, such as Statecharts [Har87], and describe the internal state of a program in sensor networks. They are predominantly "state-oriented programming models", in which one or more user applications can be modelled and programmed over sensor devices (Section 2.2). Target tracking is a popular application domain among this work and in some cases has dominated their design; for example Envirotrack [ABE+04] facilitates the coding of tracking applications where tracking objects follow external environmental entities that are detected by application states.

Other work uses the notion of state to reflect knowledge about the real-world. Examples include [TB07c] where lasting conditions are captured over correlated events, and [RM04b] where high-level information is deduced from primitive state events. In [RM04b], primitive state events are sent to a centralized server, where expressive state predicates are evaluated. The proposed high-level predicates resemble interval arithmetic [All83; WR94] where temporal relationships between primitive sensor states are examined. This work is analogous to interval composite event detection (explored in [Ksh05]) in the CE frameworks. My work uses a similar notion of state to represent high-level conditions and contexts, but focuses on an open distributed environment were detectors (SMCs) are distributed and processed independently. In addition, contextual awareness, and temporal and spatial event processing capabilities are absent in [RM04b].

Composite Event Frameworks. CE frameworks (Section 2.4.2) extract high-level information through patterns of event occurrences. These patterns are encapsulated as individual CEs, which may subsequently serve as events to other CEs. Event parameterization, which implies constraints over event attribute values, hinders the sharing of CEs, and is often performed pre- or post-CE detection. In contrast, SPS supports event parameterization as part of its condition detection process.

With regards to expressiveness and usability, previous work [TB07c; RM04b; KBM04] has shown that some real-world conditions may be difficult to express by event patterns. In [RM04b], the authors show that in order to detect the presence of multiple people in the same room with a CE framework all possible event sequences that lead to this must be specified - in fact, the number of sequences grows exponentially as the number of individuals involved in the evaluation increases. In addition, CE frameworks were not originally designed for application environments where publishers (like sensors) observe a *shared external entity* (such as the environment); they emerged from active DBMSs where events were certain, unique, and independent (see Section 2.4.2). Event occurrences in sensor systems, however, may indicate much the same information as others close is time and space, and my analysis within this chapter and Chapter 3 have demonstrated how states can be used to filter events that convey redundant information about conditions to the user.

SMCs maintain states and data structures that aid context-based data processing (for increased efficiency) and memory-based condition detection (for increased expressiveness). Of course some CE frameworks (specifically those implemented using FSA) also use states internally to maintain partial data structures, but in SPS these states are made available (externally) to the users to aid the capture of more expressive and lasting conditions. On the downside, SPS performs worse than CE frameworks, when detection of event occurrence patterns is of interest. This is because SPS uses a join operation for composing knowledge, which is more expensive than some detection models that are developed for CE frameworks.

Focusing on the event consumption policies, events are never consumed in SPS; instead, they are discarded when their contained knowledge falls outside the scope of SMCs' interests. This eases detector (SMC) recovery, as SPS only needs to recover the lost events. But within CE frameworks, one needs to also worry about which events were consumed prior to detector failure. Finally, SPS defines a unified model for temporal and spatial selection of events, condition detection, and condition interrelationships. Condition interrelationships can lead to the detection of multiple concurrent conditions, which a single SMC can monitor without the need for pre-existing replicated detectors.

Databases & Stream Processing. With a Database-oriented view [GHH+02] on sensor networks, Database-related frameworks [MFHH03; BGS00; MFHH02; SKA03] have been developed, that support application-level SQL queries over resource-constrained sensor devices (Section 2.4.1.1). These efforts are best suited for application environments where little is known about the environment, and data collection is of major interest. Data are represented as rows in a two-dimensional relational table and manipulated as in traditional DBMSs. In contrast, I chose an MDX-like cube structure which stores data in cells and benefits from multiple symmetrical dimensions. MDX dimensions allow for higher expressiveness as data is indexed according to multiple attributes, and their symmetry was used to achieve a uniform model for SMCs' contextual, temporal, and spatial data manipulation capabilities in SPS.

Tight resource considerations, in the discussed DBMSs, have often restricted the expressiveness and the range of operators that are available to the user, though more recent efforts have begun to address the need for added expressiveness (e.g. as in REED [AML05]). This class of work is not suitable for open distributed environments as adaptation and reconfiguration issues have been largely overlooked by emphasis on query optimizations and evaluations. In addition, the distinguished role of centralized gateways limit the scalability and openness that can be achieved in these systems.

Data-stream processing systems support continuous queries over streams of data. They are also based on the principles of DBMSs, but focus on environments where data comes at a higher rate, such as from stock markets or sensor networks with gateways. This data cannot be stored or processed as in traditional DBMSs. These systems are largely centralized and attempt to achieve the expressiveness of DBMSs over passing data streams. This is difficult as many DBMS operators have been designed for persistent data (Section 2.4.1.2). SPS is distinguished from data-stream processing systems by its operational environment, as well as state-based features that incorporate context and memory for data processing. The use of relational algebra in SPS, however, allows the framework to benefit from a large body of work that surrounds the implementation and optimization of DSMSs. For example, [WDR06] (a paper on SASE) discusses how predicates can be examined as part of a join operation to optimize performance in situations like the knowledge examination phase in the SPS condition detection model, and [ZDNS98] discusses how multiple dimensional queries (like those defined by QEs in SPS) can efficiently be examined against an MDX cube (similar to the InfoS component). However, there is a trade-off between the performance gained by these optimizations and the complexity that arises from implementing and executing these on certain node platforms.

Primitive sensing and limited bandwidth in sensor networks often restrict the rate of data that is realised from one or a few sensor devices. Distributed frameworks, like SPS, exploit this by pushing computations into the network and processing raw data before they turn into large data streams. When applications do not know what processing to perform a priori, or when all sensor data needs to be archived for later analysis, stream-processing systems can be used (at sensor network gateways) to handle the incoming high rate data-streams. Stream-processing systems could also be used when hard real-time guarantees are required - hierarchical information processing in SPS induces delays that may be eliminated if all data is processed centrally (at the gateway). Overall, I envisage SPS and stream-processing systems to complement one-another, such that distributed frameworks like SPS operate within the network and stream-processing systems operate at gateways or at the client side.

5.11 Summary

In this chapter, I described SPS [TB07b; TB08], a State-based Publish/Subscribe framework that is designed for open distributed sensor systems. SPS builds on the Pub/Sub communication paradigm to support a flexible and dynamic system structure; all components (applications, sensors, actuators, and even internal SMCs and InfoS components) are served as event clients through a unified Pub/Sub interface. In SPS, the network infrastructure is separated from the data processing components by a Pub/Sub layer. Thus, SPS can operate in wired, wireless, and hybrid networked environments as chosen by the system designers[1].

Central to the design of SPS are SMCs that capture high-level user-specified conditions or situations through internal data processing. These components process data according to the run-time context of the condition being observed; therefore save significant communication and processing resource. Their expressiveness allows data to be selected according to time, location, and context, and processed (aggregated, fused, examined, and/or partitioned) according to user-specified expressions. They are often composed together to perform hierarchical information processing; the results of which are re-usable data that is also meaningful to applications.

[1]It is interesting to note that some work, like [SM05], argue for hybrid infrastructures as opposed to pure wireless infrastructures that are commonly perceived by the sensor network researchers.

SMCs are also flexible and scalable at the component level; they have an independent and decomposable operational semantics. This allows SPS to distribute, share, and (where possible) localize SMCs for efficient and effective data processing. My performance evaluation, in the context of a smart transportation system, examined these benefits with respect to three system resources: processing, communication, and storage, and demonstrated that SPS with SMCs provides an expressive and scalable solution for large-scale heterogeneous sensor systems.

Chapter 6

Conclusions

With recent technological advancements and increased realisation of the benefits of studying environmental data, a wide range of sensor networks are emerging that can serve many diverse applications. At the center of these systems lies the *data*, which often constitutes the sole means of interaction between sensor networks and applications. Facilitating data-centric interaction is a complex challenge that has been explored in various work, including this dissertation. To meet this challenge, I developed a data manipulation framework called SPS for large-scale and dynamic sensor networks.

The wide design space of sensor networks and their diverse set of applications have enabled researchers to explore different problems, often corresponding to different sensor network design points. I focused on the problem of developing a decentralized information processing tool for sensor networks that contain *many devices* and serve *many applications*. This design point motivated the use of Pub/Sub, which is a data-centric many-to-many communication paradigm. I also realised that these applications, particularly in the context of smart environments, are increasingly interested in the detection and capture of *high-level conditions (situations)*. Influenced by the expressiveness of states (used in FSMs and some sensor network programming models), I opted for a state-centric information processing model in which user interests are reflected as transitions in binary states (*null state* and *detection state*). Therefore SPS is a composition of Pub/Sub messaging and state-centric data processing, which together offer four key features: abstraction, openness, scalability, and expressiveness.

In the design of SPS, many sensor network challenges were considered and addressed by distinct operational components. These challenges, which originated from three key attributes: data, scale, and resource, reflected an inter-related set of issues that arise in many sensor systems, including smart environments. Given that they may be observed in other sensor networks (corresponding to different design points), I developed modular components that can be re-used with other systems and/or employed standalone. These components and their contributions are summarized below.

State Filter (SF) SFs were developed for Resource-constrained Sensor Networks; these networks are motivated by their low manufacturing costs and small device sizes. Since operational complexity must be extremely low for these devices, they demand tailored solutions and components. SFs exploit the correlation and redundancy of data in sensor networks and perform simple filtering (similar to that in content-based filters) to reduce the messaging overhead and improve the expressiveness. The key idea at the center of the design of these components is context-based data processing.

Quad-PubSub (QPS) QPS is a standard Pub/Sub protocol for location-aware Wireless Sensor Networks. It implements a topic- and location-based subscription model and satisfies three design goals: abstraction, openness, and scalability. Influenced by the need to support more flexible routing policies, such as low latency routing or resource-aware routing, it separates itself from data routing and instead uses a location-based overlay (in the form of Quad-Trees) to construct Event Dissemination Trees with shared paths. QPS provides complete time and location decoupling and allows Event Dissemination Trees to be maintained through a standard Pub/Sub interface.

State Maintenance Component (SMC) Shifting from tight resource constraints (in Resource-constrained Sensor Networks) to heterogeneous resources (in large-scale sensor networks), I enhanced my SFs with data fusion and temporal/spatial data manipulation capabilities for more expressive condition detection. Inspired by the event selection and consumption policies in Composite Event frameworks, which originated from active DBMSs, I facilitated similar features for sensor networks with different attributes, namely event content, event time, and event location. In order to increase scalability, I preserved the simplicity of SFs in terms of their independent operational semantics and extended them with decomposability for distribution. Their detection model is described using relational algebra, which implies that related efforts in the context of DBMSs and Data Stream Management Systems can be used to implement and/or optimize the framework on different sensor network platforms. These optimizations are constrained by the available resources on the selected sensor network platforms.

Information Space (InfoS) InfoS is the data storage component of SPS. It is closely related to, but separate from, SMCs as it addresses a different set of problems. These problems are mainly related to non-deterministic network behavior and local contextual (time and space) information. With simple data processing, InfoS offers rich data reflecting aggregated, continuous, and/or contextual data to SMCs for processing. Since the design of InfoS resembles an MDX cube, its implementation can benefit from the large class of optimizations that have been developed for relational Database tables and MDX. These optimizations may also be constrained due to the limited available nodal resources.

6.1 Further Work

There is a wide range of potential research avenues in which a sensor network middleware framework, such as SPS, can be extended. These extensions should increase the usability of the framework across a wider sensor network design space. In this section, I describe five research areas that could improve SPS.

Real-world Sensor Noise Analysis. In this work, I assumed that sensor noise can be treated internally in a simple fashion, either by attribute-based computation in SFs or data aggregation functions in SMCs, or externally manipulated (by an application-defined component) in an accurate manner. Internal noise treatment is preferred as it benefits from middleware's features and optimizations, but any such integrated method needs to be justified by wide usage and efficient implementation. Only real-world sensor network deployments can suggest how appropriate and useful the integrated noise manipulation methods are in capturing high-level conditions for applications and users. Although few data aggregation functions have been implemented in SMCs, the formal semantics of these components allow support for extensible data aggreation functions that can be defined using relational algebra.

Support for Feedback. A user's knowledge of an environment improves as he/she monitors and receives data about it. Conditions (or situations) of interest can then be re-specified with greater accuracy and detected more reliably in the system. SPS allows SMCs to be replaced when a newer SMC (with an identical name to an existing SMC) is introduced to the system. An alternative solution is to relax the condition specification requirements and continually refine a probabilistic specification through learning and user feedback. The latter option is also useful when a user cannot express his/her condition of interest explicitly. Support for learning and feedback requires the existing SPS predicate language (Section 3.3.1.1), which was influenced by conditional statements in common programming languages, to be re-visited and augmented with probabilistic expressions that can be continually refined through user feedback. At the messaging service a lightweight transactional model is also needed to support feedback from the subscribers to the publishers. This transactional model ensures a consistent knowledge about the observed conditions at the event publishers. Support for transactions (in Pub/Sub systems) has recently received some attention [VPGB07] from researchers.

Support for Debugging. As we shift from centralized solutions to decentralized solutions for increased scalability and fault-tolerance, debugging applications and protocols becomes increasingly difficult. To enable debugging of large-scale distributed systems, novel solutions must be developed that allow a combination of one or more components (and their operations) to be studied in isolation. This capability enables system administrators to examine the distributed system at micro (i.e. individual component) and/or macro (group of components) levels, depending on the error or type of malfunction.

Mobility Support. In sensor networks, individual devices are often static. QPS (Chapter 4) was developed assuming this. Some applications, however, use mobile devices and pose challenges to system researchers [UVA06]. To support mobility, data routing and data dissemination algorithms must be adaptable to changes in the location of clients and data. Preliminary work [SH04] has investigated the maintenance of data in the face of mobility and can be further investigated to maintain QPS data structures in mobile Wireless Sensor Networks. However, maintaining a dynamic and scalable Event Dissemination Tree requires more research. In addition, because the cost of maintaining optimal operation should be compared to the cost of non-optimal operation, the notion of run-time SMC relocation changes significantly in mobile environments.

Security. Sensor networks are often unattended and operate in remote areas. Devices may be accessible by remote parties and are susceptible to a variety of attacks including node capture, physical tampering, and denial of service [PSW04; DX08]. Compromise generally occurs when an attacker finds a node and then directly connects to it via a wired connection of some sort. Security concerns and trust issues [CGS+03] are difficult to manage due to the limited capabilities of devices in terms of computation, communication, and energy. SPS and QPS need significant enhancements if they are to be used in un-secured and/or un-trusted open environments. The semantics of data sharing must be controlled [BMY03; BEP+03] and some collaborative behaviors (such as the open manipulation of the Event Dissemination Tree by event clients in QPS) should be restricted.

Appendix A

Replica SFs Theorem

Theorem. *For any $n \geq 1$ number of SF replicas (with the same status bit values) applied sequentially to an event e, the outcome is the same as applying just a single SF to e,*

$$\forall n \in \mathbb{N}. n \geq 1 \;\; F^n(e) = F^1(e), \tag{A.1}$$

where F denotes an SF, and $F^i(e)$ denotes the application of i number of sequential SFs to e, i.e. $F^i(e) = F^1(F^{i-1}(e))$.

Proof. This proof is by induction. For every natural number $n \geq 1$ the statement $P(n)$ must be true,

$$P(n) : F^n(e) = F^1(e). \tag{A.2}$$

The base case, where $n = 1$, is trivial. $P(1)$ holds by substitution.

$$P(1) : F^1(e) = F^1(e). \tag{A.3}$$

For the induction case, I show that $P(n)$ implies $P(n + 1)$.

$$F^{n+1}(e) = F^1(F^n(e)) \;\text{ by definition.} \tag{A.4}$$

$P(n)$ holds, therefore

$$F^n(e) = F^1(e) \Rightarrow F^{n+1}(e) = F^1(F^1(e)). \tag{A.5}$$

F is an SF, therefore $F = \{P^n, P^x, b\}$ (definition 3.4) and

$$F^1(e) = \begin{cases} e & \text{if } (b = 0 \;\wedge\; P^n(e) = \{true\}) \;\;\vee\;\; (b = 1 \;\wedge\; P^x(e) = \{true\}) \\ \emptyset & \text{otherwise} \end{cases} \tag{A.6}$$

Let $F^1(F^1(e)) \equiv F_a^1(F_b^1(e))$, where $F_a = F_b = F$ (i.e. $F_a(x) = F_b(x) = F(x)$). I show that $F_a^1(F_b^1(e)) = F_b^1(e)$ for every possible outcome of $F_b^1(e)$.

If $F_b^1(e) = e$, then $F_a^1(F_b^1(e)) = F_a^1(e) = e$; therefore $F_a^1(F_b^1(e)) = F_b^1(e)$ holds for $F_b^1(e) = e$.

If $F_b^1(e) = \emptyset$, then $F_a^1(F_b^1(e)) = F_a^1(\emptyset)$.

$$P^n(\emptyset) = P^x(\emptyset) = \{false\} \text{ (by SF predicate definition)} \Rightarrow F(\emptyset) = \emptyset \Rightarrow F_a^1(\emptyset) = \emptyset \tag{A.7}$$

Therefore, $F_a^1(F_b^1(e)) = F_b^1(e)$ also holds for $F_b^1(e) = \emptyset$.

I have shown that $F_a^1(F_b^1(e)) = F_b^1(e)$ holds for all possible outcomes of $F_b^1(e)$; therefore $F^1(F^1(e)) = F^1(e)$ and $P(n+1)$ holds. $\square$

Appendix B

SPS QEs

B.1 QE selection operators

In this section, I describe the output of QE selection parameters (for different selection operators) using set notations. The selection parameters are examined with respect to a candidate attribute $a \in A^F$, where A^F is the set of fixed attribute names $A^F = \{name, time, location, status\}$. I first define the input and output relations X and Y, and then describe how Y is attained in the case of each selection parameter (containing a different selection operator).

B.1.1 Input relation (X)

Let's assume the input relation, X, as a set of tuples,

$$X = \{x\}, \tag{B.1}$$

where each tuple x conforms to a KP data structure (definition 5.3) as follows.

$$x = \{(valid, vl), (n_1, v_1), (n_2, v_2), \cdots, (n_m, v_m)\}. \tag{B.2}$$

All names are unique (i.e. $i \neq j \Rightarrow n_i \neq n_j$). The first four names (n_1, n_2, n_3, and n_4) hold a one-to-one mapping with the A^F set members, and the remaining $m-4$ names are unrestricted.

B.1.2 Output relation (Y)

Similarly, Y consists of a set of tuples,

$$Y = \{y\}, \tag{B.3}$$

where each tuple y consists of the same set of pairs as in $x \in X$.

$$y = \{(valid, vl), (n_1, v_1), (n_2, v_2), \cdots, (n_m, v_m)\}. \tag{B.4}$$

All names match the set of names used in the input relation,

$$\forall (n_i, v_i) \in x \in X, (n_j, v_j) \in y \in Y.\ i = j \Rightarrow n_i = n_j. \tag{B.5}$$

The values, however, may differ.

B.1.3 Nearest-index operator (*closest*)

The output relation, Y, for a selection parameter $(closest, v, f, g)$ over X is defined as follows.

$$Y = \{x \in X | \exists (n_i, v_i) \in x : (n_i = a) \wedge N(i, v, x)\}, \tag{B.6}$$

where N is a *nearest-value* assertion function,

$$N : \mathbb{N} \times \mathbb{R} \times \{\mathbb{R}^2\} \to \mathbb{B}. \tag{B.7}$$

$$N(i, v, x) \mapsto \begin{cases} v = v_i \in x & \text{for} \quad i \in \{1, 4\} \\ \exists Z \in Z(i) : \forall (n_j, v_j) \in z, \forall z \in Z.\ i = j \Rightarrow & \\ \quad LEAST\, v_i \in x.\ |v - v_i| \leq |v - v_j| & \text{for} \quad i \in \{2, 3\}, \end{cases} \tag{B.8}$$

where $LEAST$ is the *least* operator (defined shortly) and $Z(i)$ is a mutually exclusive set of groups that partition X according to $A^F - \{a\}$ attribute values.

The $LEAST$ operator is defined as follows.

$$(LEAST\, x.\ P\, x) = (THE\, x.\ P\, x \wedge (\forall y.\ P\, y \to x \leq y)), \tag{B.9}$$

where the THE operator is the *definite description* - it denotes the x such that $P(x)$ is true, provided there exists a unique such x; otherwise, it returns an arbitrary value of the expected type.

The $Z(i)$ function is defined as follows.

$$Z(i) = \{Z_i\}, \tag{B.10}$$

$$Z_i = \{x \in X | \forall l \in \{1, \cdots, 4\}, \forall z \in Z_i.\ (n_l^x, v_l^x) \in x, (n_l^z, v_l^z) \in z, l \neq i \Rightarrow v_l^x = v_l^z\} \tag{B.11}$$

B.1.4 Aggregation operator (*aggregate*)

The output relation, Y, for a selection parameter $(aggregate, v, f, g)$ over X is defined as follows.

$$Y = \{y | y = r(Z \in Z(a))\}, \tag{B.12}$$

where r is an aggregation function and Z is a distinct partition from the set of X partitions as follows.

$$Z(a) = \{Z_a\}, \tag{B.13}$$

$$Z_a = \{x \in X | \forall l \in \{1, \cdots, 4\}, \forall z \in Z_a, \forall (n_l^x, v_l^x) \in x, \forall (n_l^z, v_l^z) \in z. \tag{B.14}$$

$$(n_l^x \neq a \Rightarrow v_l^x = v_l^z) \wedge (n_l^x = a \Rightarrow v_l^x \in g)\} \tag{B.15}$$

The aggregation function r aggregates the set of input tuples Z into a single tuple y as follows.

$$r(Z) = \{(valid, vl), (n_i, v_i) | \forall i \in \{1, \cdots, 4\}\ v_i = h(i, Z), \forall i \in \{5, \cdots, m\}\ v_i = f(i, Z)\}, \tag{B.16}$$

where h and f are index and value aggregation functions, respectively. The function f is an aggregate function, $f \in F = \{max, min, sum, avg\}$, and computed as follows.

$$f(j, Z) \mapsto \begin{cases} v_j \in (n_j, v_j) \in z \in Z : \forall v'_j \in (n'_j, v'_j) \in z' \in Z.\ v_j \geq v'_j & \text{for} \quad f = max \\ v_j \in (n_j, v_j) \in z \in Z : \forall v'_j \in (n'_j, v'_j) \in z' \in Z.\ v_j \leq v'_j & \text{for} \quad f = min \\ \Sigma v_j : (n_j, v_j) \in z \in Z & \text{for} \quad f = sum \\ \frac{\Sigma v_j}{|Z|} : (n_j, v_j) \in z \in Z & \text{for} \quad f = avg \end{cases} \tag{B.17}$$

The function h assigns the fixed attribute values as follows.

$$h(j, Z) \mapsto \begin{cases} LEAST\, v_j \in (n_j, v_j) \in z \in Z : f(5, Z) = v_5 \in (n_5, v_5) \in z & \text{for} \quad f \in \{max, min\} \\ \frac{\Sigma v_j}{|Z|} : v_j \in (n_j, v_j) \in z \in Z & \text{for} \quad f \in \{sum, avg\} \end{cases} \tag{B.18}$$

The validity attribute value vl is also determined as follows.

$$vl = \begin{cases} vl \in (valid, vl) \in z \in Z : f(5, Z) = v_5 \in (n_5, v_5) \in z & \text{for} \quad f \in \{max, min\} \\ true \text{ if } \exists vl \in (valid, vl) \in z \in Z : vl = true, \text{ otherwise } false & \text{for} \quad f \in \{sum, avg\} \end{cases} \tag{B.19}$$

B.1.5 Range operator (*multiple*)

The output relation, Y, for a selection parameter $(multiple, v, f, g)$ over X is defined as follows.

$$Y = \{x \in X | \exists (n_i, v_i) \in x : (n_i = a) \wedge (v_i \in g)\} \tag{B.20}$$

B.2 Joining decomposed SMCs

In this section, I show what logical operations are required to join SMC events that emerge from decomposed SMCs. Let s be the main SMC that is decomposed into s_1, s_2 SMCs with respect to an QE, B, that has the $o \in O^{multiple}$ sub-operator in its $a \in A^F$ attributed selection parameter. I chose a decomposition size of two (i.e. s_1 and s_2 SMCs) for simplicity and clarity; the following arguments equally apply for a larger number of decompositions.

Let us also assume that s is an SMC that captures momentary conditions with $P^n = p$ and just two QEs, A and B - the latter of which is decomposed. The corresponding KPs from these QEs are labeled as K_A and K_B, respectively. Of course, s_1 and s_2 examine disjoint portions of the InfoS; therefore, we label their resultant K_B KPs as K_1 and K_2 (K_A is the same). The decomposed group attributes (of B QE in s_1 and s_2) imply the following.

$$(K_B = K_1 \cup K_2) \wedge (K_1 \cap K_2 = \emptyset) \tag{B.21}$$

Let us also label the SKs, that result from the evaluation of s, s_1, s_2 SMCs, as S, S_1, S_2, respectively; and further assume that their DKs are equivalent to their SKs. The latter assumption holds if no $multiple : separate$ sub-operator exists in B's selection parameters (in s) or if it has already been decomposed across each attribute value as required by the QE decomposition policy (Section 5.7.2.2). This assumption is dropped for when $o = multiple : separate$.

With the above assumption, SMC events mirror their SMCs' SKs when the condition of the o sub-operator is satisfied over SKs. Thus, the relationship between *satisfied* S, S_1, and S_2 SKs can be studied to determine how SMC events (from s_1 and s_2) should be joined to attain results equivalent to s SMC events. The following, however, holds about S, S_1, and S_2.

$$S = S_1 \cup S_2 \tag{B.22}$$

$$S_1 \cap S_2 = \emptyset \tag{B.23}$$

Proof. The $S = \sigma_p(K_A \times K_B)$ holds by definition (Section 5.6.5). Therefore

$$\begin{aligned} S = \sigma_p(K_A \times K_B) &= \sigma_p(K_A \times (K_1 \cup K_2)) && \text{(B.24)} \\ &= \sigma_p((K_A \times K_1) \cup (K_A \times K_2)) \;\; \text{(as B.21)} && \text{(B.25)} \\ &= (\sigma_p(K_A \times K_1)) \cup (\sigma_p(K_A \times K_2)) && \text{(B.26)} \\ &= S_1 \cup S_2 && \text{(B.27)} \end{aligned}$$

$$\begin{aligned} K_1 \cap K_2 = \emptyset &\Rightarrow (K_A \times K_1) \cap (K_A \times K_2) = \emptyset && \text{(B.28)} \\ &\Rightarrow S_1 \cap S_2 = \emptyset && \text{(B.29)} \end{aligned}$$

□

B.2.1 *multiple* : *one* sub-operator

Given $o = multiple : one$, SMC events (from s) are realised when $|\pi_a S| = 1$.

$$\begin{aligned} |\pi_a S| &= |\pi_a(S_1 \cup S_2)| && \text{(B.30)} \\ &= |\pi_a S_1| + |\pi_a S_2| \;\; \text{(as B.23)} && \text{(B.31)} \\ &= \begin{cases} |\pi_a S_1| = 0 \;\; \wedge \;\; |\pi_a S_2| = 1 \\ |\pi_a S_1| = 1 \;\; \wedge \;\; |\pi_a S_2| = 0 \end{cases} && \text{(B.32)} \end{aligned}$$

Therefore, SMC events (from s) are realised when SMC events from either (but not both) s_1 or s_2 are realised. Hence, one can eliminate s in favor of s_1 and s_2 decomposed SMCs if the resulting SMC events are joined at a joining SMC by an *XOR* logical operator.

B.2.2 *multiple* : *any* sub-operator

Given $o = multiple : any$, SMC events (from s) are realised when $|S| \geq 1$.

$$\begin{aligned} |S| &= |S_1 \cup S_2| && \text{(B.33)} \\ &= |S_1| + |S_2| \;\; \text{(as B.23)} && \text{(B.34)} \\ &= \begin{cases} |S_1| \geq 1 \\ |S_2| \geq 1 \end{cases} && \text{(B.35)} \end{aligned}$$

Therefore, SMC events (from s) are realised when SMC events from either s_1 or s_2 are realised. Hence, one can eliminate s in favor of s_1 and s_2 decomposed SMCs if the resulting SMC events are joined at a joining SMC by an *OR* logical operator.

B.2.3 *multiple* : *all* sub-operator

Given $o = multiple : all$, SMC events (from s) are realised when $|\pi_a S| = |\pi_a K_B|$.

$$|\pi_a S| = |\pi_a K_B| \tag{B.36}$$

$$|\pi_a S_1| + |\pi_a S_2| = |\pi_a K_1| + |\pi_a K_2| \tag{B.37}$$

$$= \begin{cases} (|\pi_a S_1| = |\pi_a K_1|) & \wedge \quad (|\pi_a S_2| = |\pi_a K_2|) \\ (|\pi_a S_1| \neq |\pi_a K_1|) & \wedge \quad (|\pi_a S_2| \neq |\pi_a K_2|) \end{cases} \tag{B.38}$$

The latter option can't hold true, because if $(|\pi_a S_1| \neq |\pi_a K_1|) \wedge (|\pi_a S_2| \neq |\pi_a K_2|)$ then S_1 and S_2 don't satisfy the condition of the o sub-operator and SMC events (from s_1 or s_2) are not generated. Hence, only the former option, $(|\pi_a S_1| = |\pi_a K_1|) \wedge (|\pi_a S_2| = |\pi_a K_2|)$, can be true. In this case, SMC events from all decomposed SMCs (s_1 and s_2) should be realised. This assertion can be tested by an *AND* logical operator in a joining SMC.

B.2.4 *multiple* : *separate* sub-operator

Given $o = multiple : separate$, SMC events (from s) are realised when $|S| \geq 1$. The generated events are defined by the DKs, which are determined as follows.

$$\{c | c = \sigma_{(a=i\in(\pi_a K_B))} S\} \tag{B.39}$$

Let us represent this set by a relation called C, $C = \{c\}$.

$$C = \sigma_{(a=i\in(\pi_a K_B))} S \tag{B.40}$$

$$= \sigma_{(a=i\in(\pi_a (K_1 \cup K_2)))} (S_1 \cup S_2) \tag{B.41}$$

$$= \sigma_{(a=i\in(\pi_a K_1 \cup \pi_a K_2))} S_1 \cup \sigma_{(a=i\in(\pi_a K_1 \cup \pi_a K_2))} S_2 \tag{B.42}$$

Since $S_1 = \sigma_p(K_A \times K_1)$, $S_2 = \sigma_p(K_A \times K_2)$, and B.21, the following holds.

$$\sigma_{(a=i\in(\pi_a K_2))} S_1 = \emptyset \tag{B.43}$$

$$\sigma_{(a=i\in(\pi_a K_1))} S_2 = \emptyset \tag{B.44}$$

Therefore

$$C = \sigma_{(a=i\in(\pi_a K_1))} S_1 \cup \sigma_{(a=i\in(\pi_a K_2))} S_2 \tag{B.45}$$

$$C = C_1 \cup C_2, \tag{B.46}$$

where $C_1 = \sigma_{(a=i\in(\pi_a K_1))}S_1$ and $C_2 = \sigma_{(a=i\in(\pi_a K_2))}S_2$. Observe that C_1 and C_2 are DKs of S_1 and S_2 (from s_1 and s_2) should they happen to satisfy the o sub-operator condition. Therefore, the set of SMC events (from s) is equivalent to the union of SMC events from s_1 and s_2. From B.23 and B.39, it follows that $C_1 \cap C_2 = \emptyset$; therefore, the decomposed SMCs (s_1 and s_2) can publish events that match the event topic name of s SMC without the need for a joining SMC.

Bibliography

[AAB+05] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 277–289, Asilomar, CA, USA, January 2005. 55, 56

[ABE+04] T. Abdelzaher, B. Blum, D. Evans, J. George, S. George, L. Gu, T. He, C. Huang, P. Nagaraddi, S. Son, P. Sorokin, J. Stankovic, and A. Wood. EnviroTrack: Towards an environmental computing paradigm for distributed sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 582 – 589, Tokyo, Japan, March 2004. 44, 197

[ABW02] A. Arasu, S. Babu, and J. Widom. An abstract semantics and concrete language for continuous queries over streams and relations. Technical Report 2002-57, Stanford University, November 2002. 55

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006. 57

[AE04] Asaf Adi and Opher Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2):177–203, 2004. 59, 60, 61, 62

[AKK04] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, 11(6):6–28, 2004. 45

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. 197

[AML05] Daniel J. Abadi, Samuel Madden, and Wolfgang Lindner. REED: robust, efficient filtering and event detection in sensor networks. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 769–780. VLDB Endowment, 2005. 54, 198

[And96] C. André. Representation and analysis of reactive behaviors: A synchronous approach. In *Proceedings of the IMACS Multiconference on Computational Engineering in Systems Applications (CESA)*, pages 19–29, Lille, France, July 1996. IEEE Computer Society. 43

[AS06] Ian F. Akyildiz and Erich P. Stuntebeck. Wireless underground sensor networks: Research challenges. *Ad Hoc Networks*, 4(6):669–686, 2006. 143

[ASSC02] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40:102–114, 2002. 41, 42

[AY03] Kemal Akkaya and Mohamed Younis. An energy-aware qos routing protocol for wireless sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 710–715, Washington, DC, USA, 2003. IEEE Computer Society. 46

[AY05] Kemal Akkaya and Mohamed F. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, 2005. 45

[Bag05] A. Baggio. Wireless sensor networks in precision agriculture. In *Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks (REALWSN)*. ACM, 2005. Poster Session. 90

[BB03] Boris Jan Bonfils and Philippe Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 47–62, 2003. 175

[BBB04] Jenna Burrell, Tim Brooke, and Richard Beckwith. Vineyard computing: Sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, 2004. 41

[BBB+06] Can Basaran, Sebnem Baydere, Giancarlo Bongiovanni, Adam Dunkels, M. Onur Ergin, Laura Marie Feeney, Isa Hacioglu, Vlado Handziski, Andreas Kopke, Maria Lijding, Gaia Maselli, Nirvana Meratnia, Chiara Petrioli, Silvia Santini, Lodewijk van Hoesel, Thiemo Voigt, and Andrea Zanella. Research integration: Platform survey - critical evaluation of platforms commonly used in embedded wisents research. Technical Report EW-T21/D01-SICS-001-01, Embedded WiSeNts, 2006. 38

[BBC] Brainy buildings conserve energy. Website. `http://www.coe.berkeley.edu/labnotes/0701brainybuildings.html`. 41

[BBC+01] Ljubica Blazevic, Levente Buttyan, Srdjan Capkun, Silvia Giordano, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Self organization in mobile ad hoc networks: the approach of terminodes. *IEEE Communications Magazine*, 39(6):166–174, 2001. 89

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 1–16, New York, NY, USA, 2002. ACM. 56, 57

[BBE+08] Jean Bacon, Alastair Beresford, David Evans, David Ingram, Niki Trigoni, Alexandre Guitton, and Antonios Skordylis. Time: An open platform for capturing, processing and delivering transport-related data. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, pages 687–691, Las Vegas, NV, USA, January 2008. Session on Sensor Networks in Intelligent Transportation Systems. 145

[BBM+05] R. Baldoni, R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In C. Marchetti, editor, *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 437–446, 2005. 139

[BBMD03] Brian Babcock, Shivnath Babu, Rajeev Motwani, and Mayur Datar. Chain: operator scheduling for memory minimization in data stream systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 253–264, New York, NY, USA, 2003. ACM. 57

[BCSW98] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 76–84, New York, NY, USA, 1998. ACM. 89

[BDG+07] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1100–1102, New York, NY, USA, 2007. ACM. 61

[BDSZ94] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: a media access protocol for wireless lan's. In *Proceedings of the ACM International Conference on Data Communication (SIGCOMM)*, pages 212–225, New York, NY, USA, 1994. ACM. 126

[BE02] David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of the Workshop on Sensor Networks and Applications (WSNA)*, page 22, Atlanta, GA, USA, September 2002. 46, 139

[BEP+03] András Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean Bacon, and Ken Moody. Role-based access control for publish/subscribe middleware architectures. In *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, San Diego, CA, USA, 2003. ACM. 204

[BGS00] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications [see also IEEE Wireless Communications]*, 7:10–15, 2000. 46, 52, 54, 55, 57, 85, 198

[BGS01] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 3–14, London, UK, 2001. Springer-Verlag. 52

[BHE00] Nirupama Bulusu, John Heidemann, and Deborah Estrin. GPS-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000. 88

[BHS03] Athanassios Boulis, Chih-Chieh Han, and Mani B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. In *Proceedings of the The International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 187–200, New York, NY, USA, 2003. ACM. 43

[BHvR05] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. Scalable wireless ad hoc network simulation. *Handbook on Theoretical and Algorithmic Aspect of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks*, pages 297–311, 2005. 126, 188

[BKZD04] M. Beigl, A. Krohn, T. Zimmer, and C. Decker. Typical sensors needed in ubiquitous and pervasive computing. In *Proceedings of the International Conference on Networked Sensing Systems (INSS)*, pages 153–158, June 2004. 35

[BMY03] Jean Bacon, Ken Moody, and Walt Yao. Access control and trust in the use of widely distributed services. *Software - Practice and Experience*, 33(4):375–394, April 2003. 204

[BP00] P. Bahl and V.N. Padmanabhan. RADAR: an in-building rf-based user location and tracking system. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 775–784, 2000. 88

[BS] BeanShell. Website.
http://www.beanshell.org. 189

[BT] BTnodes. Website.
http://www.btnodes.ethz.ch. 82, 126

[BV06] R. Baldoni and A. Virgillito. Distributed event routing in publish/ subscribe communication systems: a survey. Technical Report MIDLAB 1/2006, Dipartimento di Informatica e Sistemistica, University di Roma la Sapienza, 2006. 48

[CBB+03] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Çetintemel, Ying Xing, and Stanley B. Zdonik. Scalable distributed stream processing. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, page 23, 2003. 56

[CC1] CC1000 single chip very low power RF transceiver. Website.
http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf. 82, 126

[CcC+02] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams: a new class of data management applications. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 215–226. VLDB Endowment, 2002. 56, 57

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. TelegraphCQ: continuous dataflow processing. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 668–668, New York, NY, USA, 2003. ACM. 55, 56, 57

[CCdC05] G. Cugola, G. Cugola, and J.E.M. de Cote. On introducing location awareness in publish-subscribe middleware. In J.E.M. de Cote, editor, *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, pages 377–382, 2005. 139

[CCMT04] M. Conti, J. Crowcroft, G. Maselli, and G. Turi. *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, chapter A modular cross-layer architecture for ad hoc networks. CRC, 2004. 91

[CCP05] P. Costa, P. Costa, and G.P. Picco. Semi-probabilistic content-based publish-subscribe. In G.P. Picco, editor, *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 575–585, 2005. 139

[CCRW04] A. Carzaniga, A. Carzaniga, M.J. Rutherford, and A.L. Wolf. A routing scheme for content-based networking. In M.J. Rutherford, editor, *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 918–928, 2004. 139

[CCW03] Mauro Caporuscio, Antonio Carzaniga, and Alexander L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. Technical Report CU-CS-944-03, Department of Computer Science, University of Colorado, January 2003. 189

[CDGS04] K.K. Chintalapudi, A. Dhariwal, R. Govindan, and G. Sukhatme. Ad-hoc localization using ranging and sectoring. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 4, pages 2662–2672, March 2004. 140

[CDKR02] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002. 139

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for internet databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 379–390, New York, NY, USA, 2000. ACM. 55, 56

[CEH+01] Alberto Cerpa, Jeremy Elson, Michael Hamilton, Jerry Zhao, Deborah Estrin, and Lewis Girod. Habitat monitoring: application driver for wireless communications technology. In *Proceedings of the ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean (SIGCOMM-LA)*, pages 20–41, New York, NY, USA, 2001. ACM. 41, 64

[CGH+02] E. Callaway, P. Gorday, L. Hester, J.A. Gutierrez, M. Naeve, B. Heile, and V. Bahl. Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks. *IEEE Communications Magazine*, 40:70–77, 2002. 41

[CGS+03] V. Cahill, E. Gray, J.-M. Seigneur, C.D. Jensen, Yong Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Di Marzo Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielson. Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2(3):52–61, July-Sept. 2003. 204

[CHZ02] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3), 2002. 46

[CJSS03] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 647–651, New York, NY, USA, 2003. ACM. 56

[CKAK94] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 606–617, San Francisco, CA, USA, 1994. Morgan Kaufmann. 60

[CKM+03] S. Chandrasekaran, S. Krishnamurthy, S. Madden, A. Deshpande, M. J. Franklin, J. M. Hellerstein, and M. Shah. Windows explained, windows expressed. Website, 2003. http://www.cs.berkeley.edu/~sirish/research/streaquel.pdf. 57

[CM94] S. Chakravarthy and D. Mishra. Snoop: an expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26, 1994. 59, 60, 62, 85

[Cod71] E. F. Codd. A database sublanguage founded on the relational calculus. In E. F. Codd and A. L. Dean, editors, *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 35–68. ACM, 1971. 62

[Cou02] Simon Courtenage. Specifying and detecting composite events in content-based publish/subscribe systems. In *Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 602–610, Washington, DC, USA, 2002. IEEE Computer Society. 59, 60

[CPR05] Paolo Costa, Gian Pietro Picco, and Silvana Rossetto. Publish-subscribe on sensor networks: a semi-probabilistic approach. In G.P. Picco, editor, *Proceedings of the International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 10–19, 2005. 46, 66, 139

[DA02] S.S. Doumit and D.P. Agrawal. Self-organizing and energy-efficient network of sensors. In *Proceedings of the Military Communications Conference (MILCOM)*, volume 2, pages 1245–1250, 2002. 41

[Dan97] Peter H. Dana. Global positioning system (GPS) time dissemination for real-time applications. *Real-Time Systems*, 12(1):9–40, 1997. 88

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauhari. The HiPAC project: combining active databases and timing constraints. *ACM SIGMOD Record*, 17(1):51–70, 1988. 60

[DGH+06] Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker M. White. Towards expressive publish/subscribe systems. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 627–644, 2006. 61, 62

[DGP+07] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A general purpose event monitoring system. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 412–422, 2007. 61

[DX08] Xiaojiang Du and Yang Xiao. *A Survey on Sensor Network Security*, pages 403–421. Springer-Verlag, 2008. 204

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003. 48, 138

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 263–270, Seattle, WA USA, 1999. 41

[FJK+05] Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy, Frederick Reiss, Shariq Rizvi, Eugene Wu 0002, Owen Cooper, Anil Edakkunni, and Wei Hong. Design considerations for high fan-in systems: The HiFi approach. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 290–304, 2005. 29, 56

[FJLM05] E. Fidler, Hans-Arno Jacobsen, Guoli Li, and Serge Mankovski. The PADRES distributed publish/subscribe system. In *Proceedings of the Feature Interactions in Telecommunications Systems (FIW)*, pages 12–30, 2005. 59, 61, 62

[FMG02] Ludger Fiege, Gero Mühl, and Felix C. Gärtner. A modular approach to build structured event-based systems. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 385–392, New York, NY, USA, 2002. ACM. 59, 60

[GA02] Antony Galton and Juan Carlos Augusto. Two approaches to event definition. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, pages 547–556, London, UK, 2002. Springer-Verlag. 177

[GC+07] Daniel Gyllstrom, Eugene Wu 0002, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. SASE: Complex event processing over streams (demo). In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 407–411, 2007. 61

[GD93] Stella Gatziu and Klaus R. Dittrich. Events in an active object-oriented database system. In *Proceedings of the International Workshop on Rules in Database Systems (RIDS)*, pages 23–39, 1993. 59, 60

[GD94] Stella Gatziu and Klaus R. Dittrich. Detecting composite events in active database systems using petri nets. In *Proceedings of the International Workshop on Research Issues in Data Engineering (RIDE-ADS)*, pages 2–9, 1994. 60

[GEH03] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? *ACM SIGCOMM Computer Communication Review*, 33(1):143–148, 2003. 50, 139

[GER+03] B. Greenstein, D. Estrin, R.Govindan, S. Ratnasamy, and S.Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of the International Workshop on Sensor Network Protocols and Applications (SNPA)*, pages 163–173, Anchorage, AK, USA, May 2003. 50, 140

[GGG05] 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)*, pages 126–140, 2005. 44

[GGHZ04] Jie Gao, Leonidas J. Guibas, John Hershberger, and Li Zhang. Fractionally cascaded information in a sensor network. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 311–319, New York, NY, USA, 2004. ACM. 50

[GHH+02] Ramesh Govindan, Joseph M. Hellerstein, Wei Hong, Sam Madden, Michael Franklin, and Scott Shenker. The sensor network as a database. Technical Report CS-02-771, Information Sciences Institute, University of Southern California, September 2002. 52, 198

[GHIGGHPD07] Carlos F. Garcia-Hernandez, Pablo H. Ibarguengoytia-Gonzalez, Joaquin Garcia-Hernandez, and Jesus A. Perez-Diaz. Wireless sensor networks and applications: a survey. *International Journal of Computer Science and Network Security*, 7(3):264–273, March 2007. 19, 41

[GJ96] Narain H. Gehani and H. V. Jagadish. Active database facilities in ode. In *Active Database Systems: Triggers and Rules For Advanced Database Processing*, pages 207–232. Morgan Kaufmann, 1996. 60

[GKMS01] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. QuickSAND: Quick summary and analysis of network data. Technical Report 2001-43, DIMACS, December 2001. 55

[GLvB+03] 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 Conference on Programming Language Design and Implementation (PLDI)*, pages 1–11, New York, NY, USA, 2003. ACM. 43

[GNC+01] J.A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile. IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks. *IEEE Network*, 15:12–19, 2001. 41

[GO03] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, 2003. 57

[GR94] Daniel D. Gajski and Loganath Ramachandran. Introduction to high-level synthesis. *IEEE Design & Test*, 11(4):44–54, 1994. 43

[GRI] The Grid project homepage. Website. `http://www.pdos.lcs.mit.edu/grid`. 89

[GSAA04] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 254–273, New York, NY, USA, 2004. Springer-Verlag. 139

[GT96] Andreas Geppert and Dimitrios Tombros. Event-based distributed workflow execution with EVE. Technical report, University of Zurich, 1996. 60, 62

[GTS06] O. Ghica, G. Trajcevski, , and P. Scheuermann. Alternating routes in sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, Nashville, TN, USA, 2006. Extended Abstract (Work In Progress Session). 92

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987. 43, 197

[HB01] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, August 2001. This article is also excerpted in "IT Roadmap to a Geospatial Future," a 2003 report from the Computer Science and Telecommunications Board of the National Research Council. 88

[HC02a] J. Hill and D. Culler. A wireless embedded sensor architecture for system-level optimization. Technical report, University of California at Berkeley, 2002. 142

[HC02b] Jason L. Hill and David E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, 2002. 54, 59

[HCB00] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society. 46

[HCRW04] Cyrus P. Hall, Antonio Carzaniga, Jeff Rose, and Alexander L. Wolf. A content-based networking protocol for sensor networks. Technical Report CU-CS-979-04, Department of Computer Science, University of Colorado, August 2004. 46, 66, 139

[Hef07] Mohamed Hefeeda. Forest fire modeling and early detection using wireless sensor networks. Technical Report CMPT2007-08, Simon Fraser University, Canada, 2007. 90

[HFH+05] M. Hirafuji, T. Fukatsu, Hu HaoMing, T. Kiura, T. Watanabe, and S. Ninomiya. A wireless sensor network with field-monitoring servers and metbroker in paddy fields. In *World Rice Research Conference*. Rice is life: scientific perspectives for the 21st century, 2005. 90

[HGM03] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, volume 2574 of *Lecture Notes in Computer Science*, pages 122–140, London, UK, 2003. Springer-Verlag. 139

[HHB+03] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 81–95, New York, NY, USA, 2003. ACM. 88

[HHKV01] Pai-Hsiang Hsiao, Adon Hwang, H. T. Kung, and Dario Vlah. Load balancing routing for wireless access networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 986–995, 2001. 92

[HKB99] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 174–185, New York, NY, USA, 1999. ACM. 46

[HL86] Ting-Chao Hou and Victor Li. Transmission range control in multihop packet radio networks. *IEEE Transactions on Communications [legacy, pre - 1988]*, 34:38–44, 1986. 89

[HMCP04] Wendi Beth Heinzelman, Amy L. Murphy, Hervaldo S. Carvalho, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, 2004. 41

[HRR99] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999. 55

[HSE03] John Heidemann, Fabio Silva, and Deborah Estrin. Matching data dissemination algorithms to application requirements. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 218–229, New York, NY, USA, 2003. ACM. 47, 48, 127, 129, 138

[HSLA03] 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 International Conference on Distributed Computing Systems (ICDCS)*, pages 46–55, Washington, DC, USA, 2003. IEEE Computer Society. 46

[HSW+00a] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, 2000. 43

[HSW+00b] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, 2000. 142

[HV02] A. Hinze and A. Voisard. A flexible parameter-dependent algebra for event notification services. Technical Report TR-B-02-10, Freie Universität Berlin, 2002. 60, 61, 62

[HWLC97] B. Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins. *Global Positioning System: Theory and Practice*. Springer-Verlag, 4th edition, May 1997. 88

[IEGH02] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 457–458, Washington, DC, USA, 2002. IEEE Computer Society. 93, 134

[IGE00] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 56–67, New York, NY, USA, 2000. ACM. 46, 47, 48, 82, 87, 91, 92, 128, 138

[IGE+03] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003. 47, 82, 87, 91, 92, 128, 138, 142

[IMK04] Mohammad Ilyas, Imad Mahgoub, and Laurie Kelly. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC, Boca Raton, FL, USA, 2004. 36

[JAF+05] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. Virtual devices: An extensible architecture for bridging the physical-digital divide. Technical Report UCB/CSD-05-1375, EECS Department, University of California at Berkeley, March 2005. 29, 56

[JS03] Yuhui Jin and Rob Strom. Relational subscription middleware for internet-scale publish-subscribe. In *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, pages 1–8, New York, NY, USA, 2003. ACM. 56

[KBM04] E. Katsiri, J. Bacon, and A. Mycroft. An extended publish/subscribe protocol for transparent subscriptions to distributed abstract state in sensor driven systems using abstract events. In *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, New York, NY, USA, 2004. ACM. 61, 197

[KEW02] Bhaskar Krishnamachari, Deborah Estrin, and Stephen B. Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 575–578. IEEE Computer Society, 2002. 45

[KFG+93] Hermann Kopetz, Gerhard Fohler, Günter Grünsteidl, Heinz Kantz, Gustav Pospischil, Peter Puschner, Johannes Reisinger, Ralf Schlatterbeck, Werner Schütz, Alexander Vrchoticky, and Ralph Zainlinger. Real-time system development: The programming model of MARS. In *Proceedings of the IEEE International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 190–199, April 1993. 59

[KK00] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, August 2000. 89

[Kna00] Ara N. Knaian. A wireless sensor network for smart roadbeds and intelligent transportation systems. Technical Report (MEng thesis), Massachusetts Institute of Technology, 2000. 144

[KP05] Joel Koshy and Raju Pandey. VMSTAR: synthesizing scalable runtime environments for sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 243–254, New York, NY, USA, 2005. ACM. 43

[KR05] Oliver Kasten and Kay Römer. Beyond event handlers: Programming wireless sensors with attributed state machines. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 45–52, Los Angeles, CA, USA, April 2005. 43, 197

[Kra05] Mark Kranz. SENSID: Situation detection for sensor networks. Technical Report (MEng thesis), University of Western Australia, 2005. 59

[KRKI04] Rajgopal Kannan, Lydia Ray, Ramaraju Kalidindi, and S. Sitharama Iyengar. Max-min length-energy-constrained routing in wireless sensor networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, pages 234–249, 2004. 92

[KS86] R Kowalski and M Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986. 71

[Ksh05] Ajay D. Kshemkalyani. Predicate detection using event streams in ubiquitous environments. In *Proceedings of the IFIP International Conference on Embedded And Ubiquitous Computing (EUC)*, pages 807–816, 2005. 197

[KSS+03] S. Kim, S.H. Son, J.A. Stankovic, S. Li, and Y. Choi. SAFE: A data dissemination protocol for periodic updates in sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, page 228, Providence, RI, USA, March 2003. 46, 127, 129, 138

[KV00] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000. 89

[KVJ05] A V U Phani Kumar, Adi Mallikarjuna Reddy V, and D. Janakiram. Distributed collaboration for event detection in wireless sensor networks. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC)*, pages 1–8, New York, NY, USA, 2005. ACM. 59, 61

[KW03] H. Karl and A. Willig. A short survey of wireless sensor networks. Technical Report TKN-03-018, Telecommunication Networks Group, Technische Universitat Berlin, 2003. 42

[KWA+03] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: a framework for distributed data fusion. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 114–125, New York, NY, USA, 2003. ACM. 175

[LAN97] LAN MAN Standards Committee of the IEEE Computer Society. *IEEE Std 802.11-1997 Information Technology- telecommunications And Information exchange Between Systems-Local And Metropolitan Area Networks-specific Requirements-part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications*, 1997. 126

[LC02] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 85–95, New York, NY, USA, 2002. ACM. 43

[LCH+07] G. Li, A. Cheung, Sh. Hou, S. Hu, V. Muthusamy, R. Sherafat, A. Wun, H.-A. Jacobsen, and S. Manovski. Historic data access in publish/subscribe. In *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, pages 80–84, New York, NY, USA, 2007. ACM. 61

[LCL+04] Juan Liu, Maurice Chu, Jie Liu, Jim Reich, and Feng Zhao. Distributed state representation for tracking problems in sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 234–242, New York, NY, USA, 2004. ACM. 197

[LCRZ03] J. Liu, M. Chu, J. Reich, and F. Zhao. State-centric programming for sensor-actuator network systems. *IEEE Pervasive Computing*, 2(4):50–62, 2003. 44, 197

[LGA96] Daniel F. Lieuwen, Narain H. Gehani, and Robert M. Arlein. The ode active database: Trigger semantics and implementation. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 412–420, Washington, DC, USA, 1996. IEEE Computer Society. 59, 60

[LGC05] Philip Levis, David Gay, and David Culler. Active sensor networks. In *Proceedings of the USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, pages 343–356, Berkeley, CA, USA, 2005. USENIX Association. 43

[LHZ04] Xin Liu, Qingfeng Huang, and Ying Zhang. Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 122–133, New York, NY, USA, 2004. ACM. 46, 139

[LJ05] Guoli Li and Hans-Arno Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 249–269, 2005. 61

[LKGH03] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 63–75, Los Angeles, CA, USA, November 2003. ACM. 50, 140

[LL07] Mo Li and Yunhao Liu. Underground structure monitoring with wireless sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 69–78, New York, NY, USA, 2007. ACM. 41

[LLCB99] C. Liebig, C. Liebig, M. Cilia, and A. Buchmann. Event composition in time-dependent distributed systems. In M. Cilia, editor, *Proceedings of the IECIS International Conference on Cooperative Information Systems (CoopIS)*, pages 70–78, 1999. 59, 60

[LM03] Ting Liu and Margaret Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the ACM SIGPLAN Principles and Practice of Paralle Computing (PPoPP)*, pages 107–118, New York, NY, USA, 2003. ACM. 43

[LP03] Ying Liu and Beth Plale. Survey of publish subscribe event systems. Technical Report TR574, Department of Computer Science (CSCI), Indiana University, May 2003. 48

[LPT99] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999. 56

[LR02] S. Lindsey and C.S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. In *IEEE Aerospace Conference*, volume 3, pages 1125–1130, 2002. 46

[LR03] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Computer Networks*, 43(4):499–518, 2003. 88

[LS03] Alberto Lerner and Dennis Shasha. AQuery: query language for ordered data, optimization techniques, and experiments. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 345–356. VLDB Endowment, 2003. 57

[LSS03] Shuoqi Li, Sang Hyuk Son, and John A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 502–517, 2003. 59, 60, 61, 82

[LWW08] Min Lin, Yan Wu, and I. Wassell. Wireless sensor network: Water distribution monitoring system. In *Proceedings of the IEEE Radio and Wireless Symposium (RWS)*, pages 775–778, 2008. 143

[LWWZ04] Yan-Nei Law, Haixun Wang, Haixun Wang, and Carlo Zaniolo. Query languages and data models for database sequences and data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 492–503. VLDB Endowment, 2004. 56, 57

[MÖ1] Gero Mühl. Generic constraints for content-based publish/subscribe. In *Proceedings of the IECIS International Conference on Cooperative Information Systems (CoopIS)*, pages 211–225, London, UK, 2001. Springer-Verlag. 59, 60

[MA01] Arati Manjeshwar and Dharma P. Agrawal. TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, page 30189a, Washington, DC, USA, 2001. IEEE Computer Society. 46

[MC02] R. Meier and V. Cahill. STEAM: event-based middleware for wireless ad hoc networks. In *Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 639–644, July 2002. 139

[MDX] Multidimensional expressions (MDX) reference. Website. `http://msdn2.microsoft.com/en-gb/library/ms145506.aspx`. 153

[ME01] D. Moreto and Markus Endler. Evaluating composite events using shared trees. *IEE Proceedings - Software*, 148(1):1–10, 2001. 59, 60

[MF02] Samuel Madden and Michael J Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 555–566, Washington, DC, USA, 2002. IEEE Computer Society. 55, 57

[MFHH02] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002. 53, 198

[MFHH03] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 491–502, New York, NY, USA, 2003. ACM. 52, 53, 54, 85, 198

[MHS] Mini hardware survey. Website. http://www.cse.unsw.edu.au/~sensar/hardware/hardware_survey.html. 38

[MIC] MICA/MICA2 sensor node. Website. http://www.xbow.com. 82, 126

[Mil89] D. L. Mills. Internet time synchronization: The network time protocol, 1989. 59

[Mit01] Michael Mitzenmacher. Compressed bloom filters. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, IEEE/ACM Trans. on Networking, pages 144–150, 2001. 153

[MP06] Luca Mottola and Gian Pietro Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 150–168, 2006. 44

[MSS97] Masoud Mansouri-Samani and Morris Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997. 60

[MWA+03] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Singh Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, approximation, and resource management in a data stream management system. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, page 22, 2003. 55, 56, 57

[MWH01] Martin Mauve, J. Widmer, and Hannes Hartenstein. A survey on position-based routing in mobile ad-hoc networks. *IEEE Network Magazine*, 15(6):30–39, November 2001. 89

[NAW05] Ryan Newton, Arvind, and Matt Welsh. Building up to macroprogramming: an intermediate language for sensor networks. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 37–44, Piscataway, NJ, USA, 2005. IEEE Computer Society. 44

[NLF07] Eduardo F. Nakamura, Antonio A. F. Loureiro, and Alejandro C. Frery. Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Computing Surveys*, 39(3):9, 2007. 51

[NMW07] Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 489–498, New York, NY, USA, 2007. ACM. 44

[NN03a] D. Niculescu and Badri Nath. Ad hoc positioning system (APS) using AOA. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 3, pages 1734–1743, 2003. 88

[NN03b] Dragos Niculescu and Badri Nath. DV based positioning in ad hoc networks. *Telecommunication Systems*, 22(1-4):267–280, 2003. 88

[NS2] The network simulator NS-2. Website. `http://www.isi.edu/nsnam/ns/`. 126

[NW04] Ryan Newton and Matt Welsh. Region streams: functional macroprogramming for sensor networks. In *Proceedings of the International Workshop on Data Management for Sensor Networks (DMSN)*, pages 78–87, New York, NY, USA, 2004. ACM. 44

[PB02] Peter R. Pietzuch and Jean Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society. 139

[PC1] Getting started with the PC-104 testbed. Website. `http://www.isi.edu/scadds/pc104testbed/guideline.html`. 142

[PD99] Norman W. Paton and Oscar Díaz. Active database systems. *ACM Computing Surveys*, 31(1):63–103, 1999. 58

[PI03] Neal Patwari and Alfred O. Hero III. Using proximity and quantized RSS for sensor localization in wireless networks. In *Proceedings of the Workshop on Sensor Networks and Applications (WSNA)*, pages 20–29, New York, NY, USA, 2003. ACM. 88

[PK00] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM (CACM)*, 43(5):51–58, 2000. 45, 78

[PSB03] Peter R. Pietzuch, Brian Shand, and Jean Bacon. A framework for event composition in distributed systems. In M. Endler and D. Schmidt, editors, *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 62–82, Rio de Janeiro, Brazil, June 2003. Springer-Verlag. 59, 60

[PSB04] Peter R. Pietzuch, Brian Shand, and Jean Bacon. Composite event detection as a generic middleware extension. *IEEE Network*, 18(1):44–55, February 2004. 58, 60

[PSW04] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Communications of the ACM (CACM)*, 47(6):53–57, 2004. 204

[RAF+01] J. Rabaey, E. Arens, C. Federspiel, A. Gadgil, D. Messerschmitt, W. Nazaroff, K. Pister, S. Oren, and P. Varaiya. Smart energy distribution and consumption: Information technology as an enabling force. Technical Report CITRIS White Paper, University of California at Berkeley, May 2001. 41

[REG+02] Sylvia Ratnasamy, Deborah Estrin, Ramesh Govindan, Brad Karp, Scott Shenker, Li Yin, and Fang Yu. Data-centric storage in sensornets. In *Proceedings of the Workshop on Sensor Networks and Applications (WSNA)*, page 78, Atlanta, GA, USA, September 2002. 50, 128, 139

[Rel03] CBI Press Release. Government failure on transport is tarnishing uk as a place to do business, 2003. 144

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the ACM International Conference on Data Communication (SIGCOMM)*, pages 161–172, New York, NY, USA, 2001. ACM. 50

[RKY+02] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. GHT: a geographic hash table for data-centric storage. In *Proceedings of the Workshop on Sensor Networks and Applications (WSNA)*, pages 78–87. ACM, 2002. 50, 93, 122, 129, 139

[RM99] V. Rodoplu and T.H. Meng. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in communications (JSAC)*, 17(8):1333 – 1344, August 1999. 46

[RM04a] K. Romer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications [see also IEEE Personal Communications]*, 11:54–61, 2004. 36, 37, 40, 41

[RM04b] Kay Römer and Friedemann Mattern. Event-based systems for detecting real-world states with sensor networks: A critical analysis. In *DEST International Workshop on Signal Processing for Sensor Networks*, pages 389–395, Melbourne, Australia, December 2004. 197

[Rob06] Simon Robertson. Sensors at thames water. In *Cambridge-MIT Institute Workshop on Wireless Sensor Networks*. CMI, 2006. 17, 143

[SA85] Richard Snodgrass and Ilsoo Ahn. A taxonomy of time databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 236–246, New York, NY, USA, 1985. ACM. 55

[SB07] Sebastian Schuster and Uwe Brinkschulte. Model-driven development of ubiquitous applications for sensor-actuator-networks with abstract state machines. In *Proceedings of the IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS)*, pages 527–536, 2007. 197

[SBC+98] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 1998. Extended Abstract. 56

[SBM+00] David C. Steere, Antonio Baptista, Dylan McNamee, Calton Pu, and Jonathan Walpole. Research challenges in environmental observation and forecasting systems. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 292–299, New York, NY, USA, 2000. ACM. 41

[Sch96] Scarlet Schwiderski. *Monitoring the behaviour of distributed systems*. PhD thesis, Cambridge University Computer Laboratory, 1996. 59

[SCO] SCOOT. Website. `http://www.scoot-utc.com`. 145, 190

[SDBL07] Rob Strom, Chitra Dorai, Gerry Buttner, and Ying Li. SMILE: distributed middleware for event stream processing. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 553–554, New York, NY, USA, 2007. ACM. 56

[SFCB04] Jan Steffan, Ludger Fiege, Mariano Cilia, and Alejandro Buchmann. Scoping in wireless sensor networks: a position paper. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC)*, pages 167–171, New York, NY, USA, 2004. ACM. 75

[SG08] Ryo Sugihara and Rajesh K. Gupta. Programming models for sensor networks: A survey. *ACM Transactions on Senor Networks*, 4(2):1–29, 2008. 42, 43, 44, 45

[SGAP00] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications [see also IEEE Wireless Communications]*, 7(5):16–27, 2000. 46

[SGV+04] Eduardo Souto, Germano Guimares, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, and Carlos Ferraz. A message-oriented middleware for sensor networks. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC)*, pages 127–134, New York, NY, USA, 2004. ACM. 29, 46, 49

[SH98] Mark Sullivan and Andrew Heybey. Tribeca: a system for managing large databases of network traffic. In *Proceedings of the USENIX Annual Technical Conference (ATEC)*, page 2, Berkeley, CA, USA, 1998. USENIX Association. 55, 56

[SH04] Karim Seada and Ahmed Helmy. Rendezvous regions: A scalable architecture for service location and data-centric storage in large-scale wireless networks. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, page 218, 2004. 50, 203

[Sha97] Murray Shanahan. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT, Cambridge, MA, USA, 1997. 71

[SHS01] Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 166–179, New York, NY, USA, 2001. ACM. 88

[SI07] S. Srivathsan and S. S. Iyengar. Minimizing latency in wireless sensor networks: a survey. In *Proceedings of the IASTED International Conference: Advances in Computer Science and Technolog (ACST)*, pages 159–164, Anaheim, CA, USA, 2007. ACTA. 138

[SJS00] Chavalit Srisathapornphat, Chaiporn Jaikaeo, and Chien-Chung Shen. Sensor information networking architecture. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, page 23, Washington, DC, USA, 2000. IEEE Computer Society. 52

[SK00] Lakshminarayanan Subramanian and Randy H. Katz. An architecture for building self-configurable systems. In *Proceedings of the ACM International*

Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pages 63–73, Piscataway, NJ, USA, 2000. IEEE Computer Society. 46

[SKA03] N. Sadagopan, B. Krishnamachari, and A.Helmy. The ACQUIRE mechanism for efficient querying in sensor networks. In *Proceedings of the International Workshop on Sensor Network Protocols and Applications (SNPA)*, pages 149–155, Anchorage, AK, USA, May 2003. 46, 52, 54, 198

[SM05] Gaurav Sharma and Ravi Mazumdar. Hybrid sensor networks: a small world. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 366–377, New York, NY, USA, 2005. ACM. 199

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM International Conference on Data Communication (SIGCOMM)*, pages 149–160, New York, NY, USA, 2001. ACM. 50

[SMP01] Mani Srivastava, Richard Muntz, and Miodrag Potkonjak. Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving environments. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 132–138, New York, NY, USA, 2001. ACM. 41

[SNMa] Sensor network museum. Website. `http://www.btnode.ethz.ch/Projects/SensorNetworkMuseum`. 38

[SNMb] Sensor networks for monitoring water supply systems. Website. `http://db.csail.mit.edu/dcnui/index.htm`. 143

[SNMT07] Ivan Stoianov, Lama Nachman, Sam Madden, and Timur Tokmouline. PIPENET: a wireless sensor network for pipeline monitoring. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 264–273, New York, NY, USA, 2007. ACM. 143

[SP04] E. Shi and A. Perrig. Designing secure sensor networks. *IEEE Wireless Communications [see also IEEE Personal Communications]*, 11(6):38–43, 2004. 41

[SPAM91] Ulf Schreier, Hamid Pirahesh, Rakesh Agrawal, and C. Mohan. Alert: An architecture for transforming a passive DBMS into an active DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 469–478, San Francisco, CA, USA, 1991. Morgan Kaufmann. 60

[SR98] Suresh Singh and C. S. Raghavendra. PAMAS: power aware multi-access protocol with signalling for ad hoc networks. *ACM SIGCOMM Computer Communication Review*, 28(3):5–26, 1998. 126

[SR02] Rahul C. Shah and Jan M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, volume 1, pages 350–355, Orlando, FL, USA, March 2002. 46, 87, 91, 92

[SS01] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *Proceedings of the Military Communications Conference (MILCOM)*, volume 1, pages 357–361, 2001. 46

[SSJ01] Chien-Chung Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications [see also IEEE Wireless Communications]*, 8:52–59, 2001. 41, 52, 54

[SSS05] Sanjay Shakkottai, Rayadurgam Srikant, and Ness B. Shroff. Unreliable sensor grids: coverage, connectivity and diameter. *Ad Hoc Networks*, 3(6):702–716, 2005. 140

[STGS02] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani Srivastava. Topology management for sensor networks: exploiting latency and density. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 135–145, New York, NY, USA, 2002. ACM. 140

[Str04] R. Strom. Fault-tolerance in the SMILE stateful publish-subscribe system. In *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, Edinburgh, Scotland, UK, May 2004. 56

[SZHK04] Karim Seada, Marco Zuniga, Ahmed Helmy, and Bhaskar Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 108–121, New York, NY, USA, 2004. ACM. 92

[TAJ04] David Tam, Reza Azimi, and Hans-Arno Jacobsen. *Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables*, pages 138–152. Springer-Verlag, 2004. 139

[TB07a] Salman Taherian and Jean Bacon. A publish/subscribe protocol for resource-awareness in wireless sensor networks. In J. Aspnes, C. Scheideler, A. Arora, and S. Madden, editors, *Proceedings of the International Workshop on Localized Algorithms and Protocols for Wireless Sensor Networks (LOCALGOS)*, volume 4549 of *Lecture Notes in Computer Science (LNCS)*, pages 27–38, Santa Fe, NM, USA, June 2007. IEEE Computer Society, Springer-Verlag. 87, 150

[TB07b] Salman Taherian and Jean Bacon. SPS: A middleware for multi-user sensor systems. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC)*, pages 19–24, New York, NY, USA, November 2007. ACM. 141, 199

[TB07c] Salman Taherian and Jean Bacon. State-filters for enhanced filtering in sensor-based publish/subscribe systems. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 346–350, May 2007. 63, 85, 197

[TB08] Salman Taherian and Jean Bacon. Capturing high-level conditions, using a publish/subscribe middleware, in sensor systems. In *Proceedings of the IET International Conference on Intelligent Environments (IE)*. IET, July 2008. To Appear. 141, 199

[TBF+03] Wesley W. Terpstra, Stefan Behnel, Ludger Fiege, Andreas Zeidler, and Alejandro P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the International Workshop on Distributed Event-based Systems (DEBS)*, pages 1–8, New York, NY, USA, 2003. ACM. 139

[TGN+92] Douglas Terry, David Goldberg, David Nichols, David Nichols, David Nichols, and Brian Oki. Continuous queries over append-only databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 321–330, New York, NY, USA, 1992. ACM. 52, 56

[TGS06] Goce Trajcevski, Oliviu Ghica, and Peter Scheuermann. CAR: Controlled adjustment of routes and sensor networks lifetime. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, page 23, Washington, DC, USA, 2006. IEEE Computer Society. 92

[TK84] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications [legacy, pre - 1988]*, 32:246–257, 1984. 89

[TMSF03] Peter A. Tucker, David Maier, Tim Sheard, and Leonidas Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on on Knowledge and Data Engineering*, 15(3):555–568, 2003. 57

[TOB04] S. Taherian, D. O'Keeffe, and J. Bacon. Event dissemination in mobile wireless sensor networks. In *Proceedings of the International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 573–575, 2004. 46

[UAM] uAMPS home. Website.
`http://mtlweb.mit.edu/researchgroups/icsystems/uamps`. 142

[UMWG04] Andreas Ulbrich, Gero Mühl, Torben Weis, and Kurt Geihs. Programming abstractions for content-based publish/subscribe in object-oriented languages. In *Confederated International Conferences CoopIS, DOA, and ODBASE*, volume 3291 of *Lecture Notes in Computer Science (LNCS)*, pages 1538–1557, 2004. 59

[UVA06] Zartash Afzal Uzmi, Thiemo Voigt, and Muneeb Ali. Mobility management in sensor networks. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 131–140, San Francisco, CA, USA, 2006. 203

[VPGB07] Luis Vargas, Lauri I. W. Pesonen, Ehud Gudes, and Jean Bacon. Transactions in content-based publish/subscribe middleware. In *Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW)*, page 68, Washington, DC, USA, 2007. IEEE Computer Society. 203

[WC94] Jennifer Widom and Stefano Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, San Francisco, CA, USA, 1994. 58

[WCS+07] Tim Wark, Peter Corke, Pavan Sikka, Lasse Klingbeil, Ying Guo, Chris Crossman, Phil Valencia, Dave Swain, and Greg Bishop-Hurley. Transforming agriculture through pervasive wireless sensor networks. *IEEE Pervasive Computing*, 6(2):50–57, 2007. 90

[WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 407–418, New York, NY, USA, 2006. ACM. 61, 199

[WFF03] Erik Welsh, Walt Fish, and J. Patrick Frantz. GNOMES: a testbed for low power heterogeneous wireless sensor networks. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 836–839, 2003. 142

[WM04] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *Proceedings of the USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, pages 29–42, Berkeley, CA, USA, 2004. USENIX Association. 44

[WQA+02] Y. M. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang. Subscription partitioning and routing in content-based publish/subscribe networks. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, October 2002. Brief Announcement. 139

[WR94] Thomas Wahl and Kurt Rothermel. Representing time in multimedia systems. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 538–543, 1994. 197

[WSBC04] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a neighborhood abstraction for sensor networks. In *Proceedings of the The International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 99–110, New York, NY, USA, 2004. ACM. 44

[XHE01] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 70–84, New York, NY, USA, 2001. ACM. 46, 89

[YG03] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 21–32, 2003. 52

[YGE01] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, Computer Science Department, University of California at Los Angeles, May 2001. 46, 89, 127, 138

[YHE02] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE Computer Society. 126

[YHE04] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 2004. to appear. 126

[YLC+02] Fan Ye, Haiyun Luo, Jerry Cheng, Songwu Lu, and Lixia Zhang. A two-tier data dissemination model for large-scale wireless sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 148–159, New York, NY, USA, 2002. ACM. 46, 139

[YRBL06] Yang Yu, Loren J. Rittle, Vartika Bhandari, and Jason B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 139–152, New York, NY, USA, 2006. ACM. 43

[YWM05] Liyang Yu, Neng Wang, and Xiaoqiao Meng. Real-time forest fire detection with wireless sensor networks. In *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, pages 1214–1217. IEEE Computer Society, 2005. 90

[YYA02] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 129–136, Washington, DC, USA, 2002. IEEE Computer Society. 46

[YZLZ05] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. GRAdient broadcast: a robust data delivery protocol for large scale sensor networks. *Wireless Networks*, 11(3):285–298, 2005. 46

[Zay87] E. Zayas. Attacking the process migration bottleneck. *ACM SIGOPS Operating Systems Review*, 21(5):13–24, 1987. 175

[ZBG98] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, pages 154–161, 1998. 126

[ZDNS98] Yihong Zhao, Prasad M. Deshpande, Jeffrey F. Naughton, and Amit Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. *ACM SIGMOD Record*, 27(2):271–282, 1998. 199

[ZS02] Yunyue Zhu and Dennis Shasha. StatStream: statistical monitoring of thousands of data streams in real time. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 358–369. VLDB Endowment, 2002. 55, 56

[ZSR02] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, pages 68–77, March 2002. 41

[ZU99] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 392–399, Washington, DC, USA, 1999. IEEE Computer Society. 62

[ZZJ+01] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 11–20, New York, NY, USA, 2001. ACM. 139