SPS: A Middleware for Multi-User Sensor Systems

Salman Taherian and Jean Bacon
University of Cambridge
Computer Laboratory
JJ Thomson Avenue,
Cambridge, CB3 0FD, UK
{st344,jmb}@cl.cam.ac.uk

ABSTRACT

With the increased realisation of the benefits of studying environmental data, sensor networks are rapidly scaling in size, heterogeneity of data, and applications. In this paper, we present a State-based Publish/Subscribe (SPS) framework for sensor systems with many distributed and independent application clients. SPS provides a state-based information deduction model that is suited to many classes of sensor network applications. State Maintenance Components (SMCs) are introduced that are simple in operation, flexible in placement, and decomposable for distributed processing. Publish/Subscribe communication forms the core messaging component of the framework. SPS uses the decoupling feature of Pub/Sub and extends this across the SMCs to support a more flexible and dynamic system structure. Our evaluation, using real sensor data, shows that SPS is expressive in capturing conditions, and scalable in performance.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

Keywords

Sensor networks, middleware, states, publish/subscribe

1. INTRODUCTION

As benefits of studying environmental data become more apparent, applications of sensor networks evolve beyond tasks of data collection and passive monitoring. Future applications are expected to operate over larger and more heterogeneous sensor networks, with interests that involve detection of conditions, situations, and contexts. Large scale deployment of application-specific networks become less likely, and the concurrent operation of applications over a shared infrastructure is promoted by economical costs and availability of data. Applications, in these systems, vary in number, and often operate independently.

While related work has explored more efficient mechanisms of transporting data from sources to sinks (e.g. [5]), and developed suitable programming models for sensor devices (e.g. [6]), little has been done in the context of large-scale multi-user sensor systems. These systems are characterized by their sheer scale, ad hoc infrastructure, and multi-application setting. Application clients are often enclosed within the network, and dynamically emerge as a result of new findings, evolving needs, and feasibilities. These systems demand frameworks that allow for more flexible system structures.

In this work we support heterogeneous and distributed applications, that have diverse high-level interests. Our framework decouples applications from the sensor devices, and processes sensor data internally to capture conditions, situations, or contexts of interest, using the notion of state. SMCs are introduced that are simple in operation, but expressive in capturing conditions. They can be decomposed, replicated, and positioned flexibly in the system to reduce costs. SMCs extract knowledge from an Information Space (InfoS), that pre-processes data and provides a rich interface to the available knowledge in the system. The extracted knowledge is then examined against predicates for conditions of interest.

SPS supports sensors, actuators, and application clients through a unified Publish/Subscribe (Pub/Sub) interface, and uses the Pub/Sub communication paradigm for its network-wide messaging. The use of a Pub/Sub communication paradigm allows SPS to leverage from previous work and offers easy integration with existing Pub/Sub systems. SPS inherits its scalability, decentralization, and decoupling from the Pub/Sub component. As a result, sensors, actuators, applications, and SMCs can be transparently added, replaced, upgraded, moved, or removed (when redundant), without affecting other clients or the information processing mechanism.

We also realised that the stateless nature of Publish/Subscribe communication restricts its expressiveness for knowledge acquisition by the SMCs. The proposed InfoS model addresses this shortcoming, independently of the Pub/Sub, and enhances condition capturing through knowledge storage, and pre-processing for SMCs.

In the next section, we highlight a motivating case-study that illustrates the system characteristics. Section 3 provides an overview of our middleware, with an introduction to SPS's information-based elements, events and SMCs. Middleware component interactions and operations are detailed in section 4. Section 5 presents an application scenario that is simulated and evaluated in terms of processing overheads, storage requirements, and induced network-level communications. Concluding remarks are made in section 7.

MPAC 2007, November 26-30, 2007 Newport Beach, CA, USA

2. MOTIVATION

For the purpose of this study, we focused on a smart transportation system, composed of *heterogeneous sensor devices* on urban streets. The choice was motivated by the availability of real sensor data, and the scale reflects the size that we envisage for future sensor systems. We believe there are many other sensor systems that exhibit similar characteristics to those outlined below.

A smart transportation system consists of many information producing (sensor) devices, including but not limited to the inductive loop sensors, speed cameras, ANPRs, GPS devices, and traffic light signals. Its *ad hoc* feature means that the sensors and actuators are not deployed and interconnected statically, or at once, but are progressively deployed and changed over time. Redundant deployments and frequent failures further complicate the system's structure. At the application layer, the system is a *multi-user* sensor system, that serves many independent and heterogeneous application clients, concurrently. As a result, any proposed system solution should regard the system as an ad hoc and dynamic environment, and account for nondeterministic changes in users, applications, and the infrastructure.

Diverse application interests demand a generic data structure and information processing model, that can aid many applications with their rich data requirements. An examination of the user interests revealed that very few (if any) relate to the raw sensor data. Instead, user interests often relate to high-level information, such as *traffic congestion*, *journey times*, *nearest taxi ranks*, which could only be realised when data from multiple (possibly heterogeneous) sources are aggregated and evaluated in a specific manner. Despite the independence of the application clients, we also realised that many interests may be similar or overlap (e.g. two independent users may be interested in traffic congestion in the same area). The overlap of interest is most likely when familiar conditions, situations, or contexts are involved.

In this paper, we focus on traffic congestion phenomena, which we detect, internally, using our framework. Our source data is obtained from SCOOT[12]. We define traffic congestion as a condition signaled by the mutual occurrence of "high road occupancy" and "slow vehicle speeds" on the same road. The first condition is deduced from the inductive loop sensor data, and the latter is deduced from the speed camera readings. The condition lasts until the speed of the moving vehicles exceeds a given threshold value ($15MpH$). In our evaluation, we have augmented this condition by associating a notion of proximity to the user's interest – users subscribe to the traffic congestion conditions that are located within 2 road-junction distances of their present location. This introduces dynamic interests that vary in time, and depend on users' movements and locations. The proposed framework, detailed in the next section, detects such complex conditions using low-cost distributed processing.

3. State-based Publish/Subscribe

SPS is a State-based Publish/Subscribe framework that supports its clients, comprising sensors, actuators, applications, controllers, and etc., through a *uniform Pub/Sub interface*. Clients may hold one or more of the Pub/Sub roles. An information producing client is called a *publisher*, and a consuming one is called a *subscriber*. This communication paradigm is also extended across the internal information-

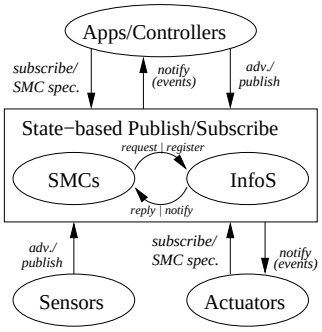

Figure 1: SPS Architecture

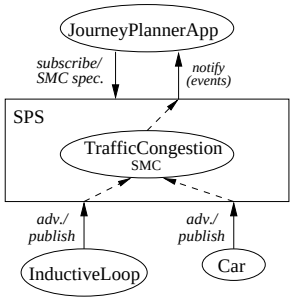

Figure 2: Case-study Overview

capturing elements, SMCs. For example, an SMC that detects "road traffic congestion" acts as a subscriber to the sensor (inductive loop and speed camera) data, and as a publisher to those interested in the traffic information. SMCs are user-defined expressions that capture momentary or lasting conditions using the notion of *state*.

SPS provides an expressive information processing model, while using a stateless Pub/Sub paradigm for messaging. More importantly, it supports a flexible and dynamic system structure through the decoupling of clients (by the Pub/Sub) and the decoupling of the data (by the InfoS).

3.1 Architecture

The SPS comprises a middleware layer that resides on all nodes. This is shown as a rectangular component in figure 1. Every node may also hold zero or more other components, depending on its resources, hardware accessories, and application roles. We have depicted interactions corresponding to the conventional Pub/Sub roles for the commonly known clients in figure 1 (sensors are publishers, actuators are subscribers, and applications/controllers may be both).

The solid arrows indicate interactions that are bound to local nodes. Clients strictly communicate with their local SPS component. The communication is via a rich Pub/Sub interface that is detailed later in §4.1. There are also interactions between SMCs and InfoS components, that are internal to the SPS and constitute the high-level information deduction core of the framework. InfoS components contain the knowledge that is available to SPS, and SMCs capture high-level information from this available knowledge.

Figure 2 shows the architectural diagram corresponding to the traffic congestion detection example. There are nodes with the depicted sensor components that connect to the SPS and publish events. A *TrafficCongestion* SMC, defined by the *JourneyPlannerApp* client, detects traffic congestion conditions from the available knowledge. The dashed arrows reflect information flow that could be across the network (i.e. the *TrafficCongestion* SMC may reside on a node remote to the sensor or application clients). Information flow, in SPS, is by means of event notifications, formally described below.

3.2 Event Notifications

Event notifications are asynchronous messages, comprising a set of attribute/value pairs that describe information, either introduced to or captured within SPS. Some attributes are static, others are topic-related. The static attributes are as follows.

Name labels the information contained in the event. It has two fields, the topic (major) name and a minor name.

Time is the event notification's timestamp at the publisher. Also viewed as the event occurrence's timestamp.

Location is the geographical location of the event's occurrence. This may be different from the geographical location of the event's publisher.

Class describes the temporal significance of the information contained in the event notification. It may be momentary (*atomic*), signal a condition initiation (*ingress*), or signify the termination of a condition (*egress*).

The topic-related attributes may hold arbitrary names. However, for the sake of discussion, we label these as *value1*, *value2*, and ...; each holding a numerical/textual value that is understood relative to the event name.

3.3 State Maintenance Component (SMC)

SMCs are the information processing elements of SPS. They capture conditions or contexts of interest through the notion of state. They may capture lasting or momentary conditions, and are expressed as follows.

- **SMC <name>: {<data-elements>}**
  [<entrance predicate> | <SMC attr. computations>
  ◊ <exit predicate> | <SMC attr. computations>]

SMC names also hold two fields, a major and a minor. The major is the significant part of the name, and labels the information that the SMC captures. The minor name is used to distinguish between SMCs with a common major name. These relate to the discussed event notification names.

Predicates are boolean expressions that examine some user-defined conditions over the attained knowledge in SPS. The knowledge is queried by Data Elements (DEs). The DEs resolve into Knowledge Points (KPs), when examined against the local InfoS. Predicates, subsequently, evaluate the KPs for condition occurrence (initiation) or termination. The *TrafficCongestion* SMC is specified below.

- **SMC TrafficCongestion: {A=InductiveLoop.agg[30s,avg] ; B=Car.agg[1m,avg] ; C=local.local.local}**
  [(A.value1>2.5) && **(B.value1<7)** && **(A.location==B.location) | value1=B.value1 ◊ (B.value1>15)** && **(C.location==B.location) | value1=B.value1]**

The three DEs (**A**, **B**, and **C**) extract information that corresponds to the "average road occupancy over the last 30 seconds", "average vehicle speed in the last minute", and the "previously published SMC event", respectively. The entrance predicate is composed of three boolean expressions that ensure *high road occupancy*, *slow moving speed*, and *occurrence of the previous two phenomena on the same road*, respectively. The exit predicate detects the end of the traffic congestion condition, and is evaluated after the entrance predicate is satisfied. It is composed of two boolean expressions that examine the *speed of the moving vehicles* and ensure that the *examining speeds relate to the captured traffic congestion*. The first topic-related SMC attribute (*value1*) is set to the speed of the moving vehicles on the road, as we feel this measure best indicates the degree of congestion.

Each SMC holds a *status-bit* that indicates its status (active or inactive). This is manipulated to reflect the state transitions. When the appropriate predicate is satisfied, the status-bit is toggled, and the SMC attributes are computed. A set of attributes (matching that of the event notification's static attributes) is also computed internally following every state transition (see §4.2). The resulting values may be enveloped into an event notification, that is referred to as an *SMC event*. SMC events can transport the captured knowledge to others, such as the *JourneyPlannerApp* clients.

4. SPS INTERNALS

Figure 3 shows the internals of the SPS middleware: the *Pub/Sub*, *InfoS manager*, and *SMC manager* components.

SPS, as a resource-aware middleware framework, classifies nodes into three classes: *micro nodes*, *macro nodes*, and *power nodes*. Micro nodes possess the least resources, perhaps only to perform sensing or actuation in their local environments. The SPS framework is basic for these nodes, comprising just the Pub/Sub component. Nodes with some storage and computational resources (macro nodes) hold all three components. These nodes can house InfoS and SMCs that deduce higher level information. Finally, power nodes possess sufficient computational resources to decompose SMCs (discussed later in §4.2). The Pub/Sub is the only network-aware component of the architecture, that interacts with its peers to disseminate events across the network. The SMC manager publishes SMC events, and the InfoS manager subscribes to event topics of interest. These components are detailed below.

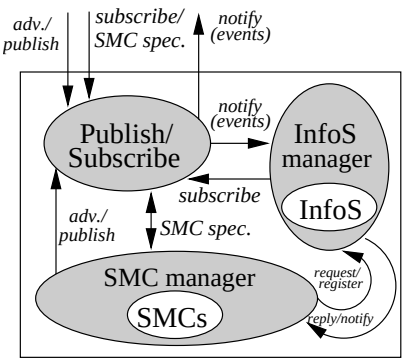

Figure 3: SPS internals

4.1 Publish/Subscribe

Our work uses Pub/Sub[4] to leverage from previous work and achieve scalability and decoupling of event clients. This decoupling and the unified Pub/Sub interface allows for a dynamic system structure, in which SPS clients and SMCs may transparently change, switch roles, or adopt multiple roles in respect of different event clients in the system.

We have chosen Quad-PubSub[13], a location-based Publish/Subscribe for wireless sensor networks, as our Pub/Sub component. The location-awareness of Quad-PubSub enhances the value of data, and reflects the following.

- Events have associated locations (location-stamps for events' occurrences).
- Advertisements specify bounding regions for publishable events' locations.
- Subscriptions may filter events by topic and location (region of interest).

In addition to the conventional *Advertise*, *Subscribe*, and *Publish* operations, the Quad-PubSub, in SPS, is augmented with the *SMC specification* and *placement* services. Given an SMC definition, the protocol checks the registered publishers and discards the SMC if an equivalent SMC[1] is registered. Otherwise, a suitable host is found and the SMC definition is dispatched to its SMC manager. The SMC name and DEs are used to place the SMC close to its source data, or on nodes which already host similar SMCs. This placement policy aims to reduce messaging costs.

[1]Two SMCs are equivalent, if their names and DEs match.

4.2 SMC Manager

The SMC manager stores and evaluates SMCs. It acquires input information from the local InfoS component, and evaluates SMCs for condition initiations and/or terminations. SMCs with the explicit "true" exit predicate capture momentary conditions and produce *atomic SMC events*, otherwise they capture lasting conditions and produce *ingress* and *egress SMC events* accordingly.

SMC specifications, that are received from the Pub/Sub component, may be registered, deferred, or decomposed (only at the power nodes). SMC registration involves the allocation of limited storage space for the SMC and the advertisement of publishable SMC events to the Pub/Sub component[2]. Deferral is done when the SMC manager lacks sufficient resources to house a new SMC, or when the SMC specification is decomposed into simpler SMCs. SMCs can be decomposed along the predicates and/or the DEs. For example, the fore-mentioned *TrafficCongestion* SMC may be decomposed, along the predicates, as follows.

- **SMC TrafficCongestion: {A=IL_High ; B=Car_Slow ; C=local.local.local ; D=Car_Slow.egress}** [**A** && **B** && **(A.location==B.location)** | **value1=B.value1** ♢ **D** && **(C.location==D.location)** | **value1=D.value1**]
- **SMC IL_High: {A=InductiveLoop.agg[30s,avg] ; B=local.local.local}** [**(A.value1>2.5)** ♢ **(A.value1<=1.5)** && **(B.location==A.location)**]
- **SMC Car_Slow: {A=Car.agg[1m,avg] ; B=local.local.local}** [**(A.value1<7)** | **value1=A.value1** ♢ **(A.value1>15)** && **(B.location==A.location)** | **value1=A.value1**]

Intermediate *IL_High* and *Car_Slow* SMCs are introduced, which capture the pre-requisite conditions independently. This decomposition decouples the *TrafficCongestion* SMC from the primitive *InductiveLoop* and *Car* event notifications, that may be high rate and expensive in processing.

SMC evaluations may be periodic or trigger-based. The SMC manager can periodically dispatch the DEs to the InfoS component for KP acquisitions (*request/reply*), or (in the case of available storage space) register DEs at the InfoS component for KP notifications (*register/notify*). KPs conform to the semantics of the event notifications, and are often retrieved in sets (e.g. the **Car.agg[1m,avg]** DE results in a set of KPs, that each correspond to a last-minute average speed reading at a distinct speed camera sensor). SMC predicates are examined until one or more satisfying KP-combinations are found. Evaluation policies are governed by the DE parameters, details of which are omitted due to space limitations.

When an SMC predicate is satisfied the status-bit is toggled, and the SMC attributes are computed or adopted from the most recent KP, as the topic-related attributes of the SMC event. The static attributes are assigned according to the SMC name, satisfied predicate, and the time and location of the most recent KP in the satisfying combination.

4.3 Information Space (InfoS) Manager

InfoS is a knowledge container, that is used by the local SMCs to deduce high-level information. The SMC manager expresses knowledge of interest through DEs, and the InfoS manager, in turn, subscribes to event notifications to receive this knowledge globally. The InfoS offers an expressive, yet simple, knowledge querying mechanism (via DEs), that is otherwise unattainable in stateless Pub/Sub systems. In this design, events are stored on demand, and knowledge is shared among the SMCs.

The InfoS serves three purposes. Firstly, it pairs the related ingress and egress events, and offers rich, condition-based information to the SMCs. As such the *TrafficCongestion* SMC can evaluate conditions as simply as evaluating boolean **A**(=IL_High) and **B**(=Car_Slow) DEs. Secondly, it abstracts correlated data through aggregations. This results in individual aggregated data that eases computations and provides data transparency for the SMC manager. For example, when examining the "average speed of the cars in the last minute", the SMC manager is no longer concerned with the aggregation of an indeterminate number of *Car* event notifications, but is simply supplied with a single aggregated value (KP) which abstracts the fine-grained speed readings in the last minute. This abstraction could further be investigated to supply approximations or modeled data when fine-grained data is unavailable or erroneous. Thirdly, knowledge storage and consistency concerns are decoupled from the SMC manager. This provides a clean separation of roles for the InfoS and SMC managers.

In the current implementation of SPS, the InfoS is modeled as a multi-dimensional indexed structure. The dimensions match the static attributes of the event notifications, and points in this space contain knowledge in the form of tuples that hold topic-related attribute/value pairs. DEs query the InfoS by means of selecting and processing knowledge along each dimension of the InfoS. For example, the **Car.agg[1m,avg]** DE selects the 'Car' value on the name dimension, selects and averages values on the last unit (i.e. [-1,0]) of the time dimension, selects the entire range for the location (default), and the 'ingress' value for the class dimension (default). The knowledge confined in the defined space resolves to a set of KPs, that reflect the last-minute's average speed readings, for each knowledge-contained location in the entire system. SMC decomposition along the DEs (mentioned in §4.2) allows this SMC to be decomposed into SMCs that represent smaller regions.

Delayed processing, with roll-back capabilities, is used for knowledge consistency. SMCs are evaluated with respect to delayed timelines that are transparent to them. The delay maximizes the receipt of related events (knowledge), that is used for the selection and processing of knowledge for SMCs.

5. EVALUATION

The proposed framework has been implemented on Jist/Swans[3]. The following sections describe the application settings, and the performance evaluation of SPS.

5.1 Application: Journey Planner

The discussed journey planner application was used to evaluate the correctness of SPS. The condition of interest was complicated, as follows. The *JourneyPlannerApp* clients subscribed to the *TrafficCongestionNear* event topic, whose SMC captures nearby traffic congestion (situated within the 2 road-junction distance of the user's present location. The corresponding SMC is expressed as follows.

- **SMC TrafficCongestionNear: {A=TrafficCongestion ; B=User.local.local}** [**abs(A.location - B.location) <= 2** | **value1=A.value1** ♢ **(true)**]

[2]SMC advertisements help to discover and share SMCs.

The **A** DE extracts traffic congestion information, and the **B** extracts the locally published user information. The *TrafficCongestion* SMC is as before (see §3.3). Details of sensor clients are given below.

- Inductive Loop sensors, with periodic reports on road-segment occupancies (continuous *InductiveLoop (IL)* event notifications at 1Hz).
- Speed Measurements[3], reporting on the speed of the passing cars on the road (discrete, but potentially high-volume, *Car (C)* event notifications).
- Location sensors, indicating the current location of the simulated users (continuous *User* events at 1Hz).

A grid-like road network was simulated and two-dimensional road coordinates were used for the location attribute of the events. We examined sixty roads, each equipped with a single inductive loop and speed measuring sensor. In addition to these (120) sensor devices, 300 network nodes were introduced to ensure wireless network connectivity in the simulation environment. Five hundred mobile users were simulated, who benefitted from the journey planner application, local to their nodes.

A high-level view of the system, following SMC decomposition (along predicates), is shown in figure 4. We processed twenty (20) hours of real data, relating to two distinct days, 1^{st} *July*, 2006, $1AM$–$9PM$ for Sim1, and 6^{th} *April*, 2006, $1AM$–$9PM$ for Sim2. Table 1 outlines the simulation results (including SMC counts after the geographical (DE) decompositions).

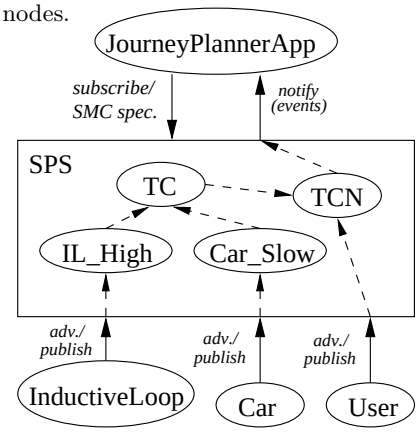

Figure 4: Sys. Overview

5.2 Processing

The processing complexity of SPS relates to the cost of SMC evaluation. SMCs are evaluated when their corresponding KPs are updated. Hence, received events, at InfoS, can trigger SMC evaluations and induce processing costs. Table 1 shows that the SPS has lowered the number of received events through SMC decompositions along the geographical space (DEs). The max. received events, for the *TrafficCongestion* SMC evaluations, is lowered from 1616 (Sim1) and 3580 (Sim2) to 236 and 701 events, respectively. These figures are 14.6% and 19.5% of the 1616 and 3580, the sum of *InductiveLoop_High* and *Car_Slow* events.

The processing complexity of all SMC predicates was $O(n)$, except for the *TrafficCongestion* SMC predicates. This means that an incoming event at most triggered a single KP-combination examination. The maximum observed processing complexities, for the *TrafficCongestion* SMC predicates, were $6n$ (Sim1) and $8n$ (Sim2). This means that, at worst-case, SPS was observed to evaluate 6 KP-combinations (Sim1) and 8 KP-combinations (Sim2), when one event was received at the *TrafficCongestion* SMC's local InfoS.

[3]inferred from a secondary stream of raw SCOOT data

Statistics *Annotations*	Sim1 *1/7/06*	Sim2 *6/4/06*
Number of SMCs and Clients		
Client – User	500	500
Client – InductiveLoop (IL)	60	60
Client – Car	60	60
SMC – IL_High	60	60
SMC – Car_Slow	60	60
SMC – TrafficCongestion (TC)	16	16
SMC – TrafficCongestionNear(TCN)	500	500
Client – JourneyPlannerApp (JPA)	500	500
Client Subscriptions *journey planner applications*	500	500
Resolved Subscriptions *(including InfoS Subscriptions)*	1652	1652
Max Subscriptions per node *InfoS subscriptions for TrafficCongestion and TrafficCongestionNear SMCs*	2	2
Decomposed SMC types *IL_High, Car_Slow, TC and TCN*	4	4
Decomposed and Distributed SMCs	636	636
Max SMC allocation per node	1	1
Max Events stored per InfoS *Car events*	37	30
Max Events stored per InfoS for the *TC* SMC evaluation *paired IL_High, Car_Slow, and TrafficCongestion events*	14	22
Max received input Events for the TrafficCongestion SMC	236	701
Max Predicate Evaluations per received Event Notification *TrafficCongestion predicate evaluations*	24	60
Event Publications by Clients *IL, Car, and User Events*	5121454	5276714
Total Event Notifications (ENs) *published Events (including SMC ENs)*	5127270	5287694
Events Disseminated (ED) *Events Disseminated within the Network*	1784	3876
Events Delivered to Subscribers *TCN Events to JPA Clients*	4032	7104
Number of ENs for each Event Topic		
User	3.6e+7	3.6e+7
Car	800954	956214
InductiveLoop	4.32e+6	4.32e+6
Car_Slow	774	1042
IL_High	842	2538
TrafficCongestion	168	296
TrafficCongestionNear	4032	7104

Table 1: Simulation Results

5.3 Storage

SMCs and events are the two main elements that require storage in SPS. Table 1 shows that SMC distribution has resulted in a maximum of one SMC allocation per node in the network. It also shows that a total of 636 SMCs served the 500 mobile users. From these 636 SMCs, 136 SMCs were shared and collaboratively deduced traffic congestion information for the entire system. This sharing was achieved by the Pub/Sub component which discarded duplicate SMCs, and interconnected independent subscribers with overlapping interests to the same SMCs (publishers). Hence, overlap of interests was used to save storage space, and avoid duplicate processing costs.

The highest number of events stored at any InfoS related to the knowledge stored for the *Car_Slow* SMCs. The highest numbers of *Car* events stored for deducing the aggregation information were 37 (Sim1) and 30 (Sim2). These figures exclude any compressions or functional optimizations that can further reduce this storage requirement. Similarly,

the highest numbers of events stored for the *TrafficCongestion* SMCs were 11 and 16 events for Sim1 and Sim2, respectively. This indicates that the aforementioned 236 and 701 input events (in section 5.2) were continuously updating and overriding 11 and 16 storage points in the InfoS.

5.4 Communications

Communication costs are often measured by the total energy used to deliver events across the network. This largely depends on the network structure and the performance of the adopted Pub/Sub protocol. Nevertheless, since the distribution of SMCs impacts the formation of Pub/Sub links, we have measured this cost by examining the "number of event notifications that were disseminated in the network".

Table 1 shows that out of the 5121454 (Sim1) and 5276714 (Sim2) event notifications published in the system, only 2348 and 5229 events were disseminated in the network. This means that a substantial portion of the published events (99.95% for Sim1, and 99.9% for Sim2) were processed locally. *InductiveLoop_High*, *Car_Slow* and *TrafficCongestion-Near* are three SMCs which localized the processing of high-rate *InductiveLoop*, *Car* and *User* event notifications, respectively. These localizations were achieved when the SMCs were fully decomposed over the network space (SMC decompositions along the DEs).

6. RELATED WORK

SPS is not the first framework to use the notion of state for sensor systems. Several [11, 6, 8, 1] have been designed before, offering the expressiveness of states to the sensor network applications. These are mainly based on the principles of Finite State Machines (FSMs), 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. Other works use the notion of state to reflect knowledge about the real-world. Examples include [14] where lasting conditions are captured over correlated events, and [11] where high-level information is deduced from primitive state events. In [11], primitive state events are drawn to a centralized server, where expressive state predicates are evaluated. In this design, high-level states are tightly coupled with low-level states. Our work uses a similar notion of state, but offers SMCs that are simpler in operation, and decomposable for distributed processing. Furthermore, SMCs are decoupled from each other, allowing for easy integration and replacement of SMCs.

Composite Event (CE) frameworks [10, 9, 2, 7], are also related, as they often use Pub/Sub for messaging. They extract high-level information through patterns of event occurrences, that when satisfied are enveloped as individual composite events. Event occurrences, in sensor networks, may indicate much the same information as others occurring in nearby space and time. Previous work [14, 11] has argued for the suitability and expressiveness of states against event patterns for sensor networks. SPS explicitly decouples event occurrences from SMCs, using an InfoS model, which also provides primitive aggregations over similar events. Conditions, in SPS, are evaluated using the knowledge that is contained within the event notifications. This allows for more expressive specification of conditions, and enables a condition-based events storage policy.

7. CONCLUSION

In this paper, we proposed State-based Publish/Subscribe (SPS), for sensor systems with many distributed and independent application clients. SPS processes data internally and leverages from the Pub/Sub communication paradigm. It supports a flexible and dynamic system structure through the decoupling of clients (by the Pub/Sub component) and the decoupling of data (by the InfoS component). Localized processing and information sharing, achieved by SMC decomposition and the Pub/Sub component, were also shown to greatly reduce costs, demonstrating SPS's scalability.

8. REFERENCES

[1] 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 *Proc. of 24th International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, Japan, Mar. 2004.

[2] A. Adi and O. Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2):177–203, 2004.

[3] R. Barr, Z. J. Haas, and R. 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.

[4] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[5] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

[6] O. Kasten and K. Römer. Beyond event handlers: Programming wireless sensors with attributed state machines. In *The Fourth International Conference on Information Processing in Sensor Networks (IPSN)*, pages 45–52, Los Angeles, USA, Apr. 2005.

[7] S. Li, S. H. Son, and J. A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In F. Zhao and L. J. Guibas, editors, *IPSN*, volume 2634 of *Lecture Notes in Computer Science*, pages 502–517. Springer, 2003.

[8] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-centric programming for sensor-actuator network systems. *IEEE Pervasive Computing*, 02(4):50–62, Oct-Dec 2003.

[9] D. Moreto and M. Endler. Evaluating composite events using shared trees. *IEE Proceedings - Software*, 148(1):1–10, 2001.

[10] P. R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *IEEE Network*, 18(1):44–55, 2004.

[11] K. Römer and F. Mattern. Event-based systems for detecting real-world states with sensor networks: A critical analysis. In *DEST Workshop on Signal Processing in Sensor Networks at ISSNIP*, pages 389–395, Melbourne, Australia, Dec. 2004.

[12] SCOOT. http://www.scoot-utc.com.

[13] S. Taherian and J. Bacon. A publish/subscribe protocol for resource-awareness in wireless sensor networks. In *Proceedings of the International Workshop on Localized Algorithms and Protocols for Wireless Sensor Networks (LOCALGOS'07)*, pages 27–38, Santa Fe, USA, June 2007.

[14] S. Taherian and J. Bacon. State-filters for enhanced filtering in sensor-based publish/subscribe systems. In *Proceedings of the International Workshop on Data Intensive Sensor Networks (DISN'07)*, Mannheim, Germany, May 2007.