UNIVERSITY OF MAURITIUS RESEARCH JOURNAL – Volume 15 – 2009
University of Mauritius, Réduit, Mauritius

Research Week 2008

MiPSCom: A Novel Content-Based Publish/Subscribe Communication Model for Wireless Sensor Networks

Kavi K. Khedo *
Faculty of Engineering,
University of Mauritius
Email: *k.khedo@uom.ac.mu*

R. K. Subramanian
University of Mauritius
Email: *rks@uom.ac.mu*

Paper Accepted on 11 August 2009

Abstract

Due to extreme resource constraints and lack of suitable programming abstractions, programming sensor networks remains a complex and tedious process. Dealing with sensor devices in large numbers, which are error prone and limited in terms of energy, memory and processing power, introduces a burden on application development. Well adapted to the loosely coupled nature of distributed interaction in large-scale applications, the publish/subscribe communication paradigm has recently received increasing attention in the domain of wireless sensor networks. In this paper we propose a well-defined content-based publish/subscribe service, MiPSCom, that allows the application designer to adapt the service by choosing appropriate communication protocol components for subscription and notification. A major design goal of the proposed communication model is to give the application designer a simple and flexible means to select protocol components and data attributes according to his needs, and to give him more fine-grained control over the publish/subscribe service through a number of extension components. The flexibility of MiPSCom to support different sensor node platforms, communication protocols and interaction patterns has been demonstrated experimentally. The experimental results show that our approach exports significant performance tradeoffs to the application in an easy-to-use fashion, and the communication model is general and flexible enough to support different interaction patterns and the execution time overhead is acceptable.

Keywords: Publish/Subscribe Paradigm, Sensor Networks, Communication Model, Subscription, Notification

**For correspondences and reprints*

1. INTRODUCTION

The continuous miniaturization of hardware components and the evolution of wireless communication technologies have stimulated the development and use of wireless sensor networks (WSNs) in the monitoring of physical environments. Consequently, the emerging field of sensor networks offers an unprecedented opportunity for a wide spectrum of applications (Levis et al. 2008). However, due to extreme resource constraints and lack of suitable programming abstractions, programming sensor networks becomes a complex and tedious process. Indeed, wireless sensor networks are ad-hoc self-organizing untethered networks of smart sensors characterized by severe energy resource constraints. Dealing with sensor devices in large numbers, which are error prone and limited in terms of energy, memory and processing power, introduces a burden on application development. Moreover, the distribution of nodes and their shared communication medium call for multi-hop routing algorithms and distributed coordination (Sun et al. 2008).

WSN applications need to continuously collect and integrate data generated from a large and physically dispersed contingent of sensor nodes. In this scenario, there are a large number of devices exchanging data, whilst some information sources and sinks may not be present in the network at the same time. Therefore, the request/response communication is not adequate to satisfy this requirement. Moreover, as energy is a scarce resource, unnecessary information requests should be avoided. In addition, the communication between applications in WSNs is essentially based on events, which suggests that the traditional request/response approach (synchronous) is not appropriate. In most applications, data transmission is triggered when either an event occurs or the sink node generates a query.

An appealing way to organize cooperation is to employ an event-driven style of interaction by exploiting publish/subscribe (Eugster et al. 2003): producers publish notifications, while consumers selectively subscribe to notifications, for example, using topic-based or content-based filters (Costa et al. 2005). Publish/subscribe, discussed in section 2, ideally fits the targeted setting because, usually, neither producers nor consumers do address their counterparts explicitly. This leads to a loosely and dynamic coupling of components allowing for more fault resistance than if an explicit addressing of individual nodes was used.

In this paper we propose MiPSCom, MiSense Content-based Publish/Subscribe Communication Model which is based on an enhanced published/subscribed scheme. We present the design, implementation and evaluation of a flexible communication model that provides a well-defined content-based publish/subscribe service and allows the application designer to adapt the service by making orthogonal choices about the communication components for subscription and notification delivery, the supported data attributes, and a set of service extension components. This allows the decoupling between the publish/subscribe core and the communication protocols. The communication model uses an attribute-based naming scheme augmented with metadata containing soft requirements for the publishers and run-time control information for the service extension components. It supports different addressing schemes and interaction patterns. Initial results show that our approach provides good performance in terms of high delivery and low overhead, and is resilient to changes in connectivity, therefore making it amenable to our target deployment scenario.

2. THE PUBLISH/SUBSCRIBE PARADIGM

The increasingly popular publish/subscribe paradigm allows processes to exchange information without explicit knowledge about any particular destination address where producers or consumers can be found. This is founded on the principle that producers simply make information available and consumers place a standing request for information by issuing subscriptions. The notification service is then responsible for making information flow from a producer (publisher) to one or more interested consumers (subscribers). A publish/subscribe notification service provides asynchronous communication, it naturally decouples producers and consumers, makes them anonymous to each other, and allows a dynamic number of publishers and subscribers. The loose coupling of producers and consumers is the prime advantage of publish/subscribe systems. This makes publish/subscribe a key technology for information dissemination in wireless sensor networks.

The reification of an event in a publish/subscribe system is a *notification*. It represents the data describing the observed happening. A notification is created by the observer of the event. The content of a notification usually is application-dependent and may just indicate the plain occurrence, but it can also carry additional information describing the circumstances of the event. On the transport level described here, notifications are forwarded by *messages*, which basically are containers for data on the network level. They carry data between the endpoints of the underlying communication mechanism.

The clients of an event-based system act as producers and/or consumers of notifications. *Producers* emit notifications whenever an event occurs. A producer does not necessarily have to publish every single event. In general, producers are components that are self-contained. Hence, the course of action taken after an event's occurrence is up to the internal computation done within the producer.

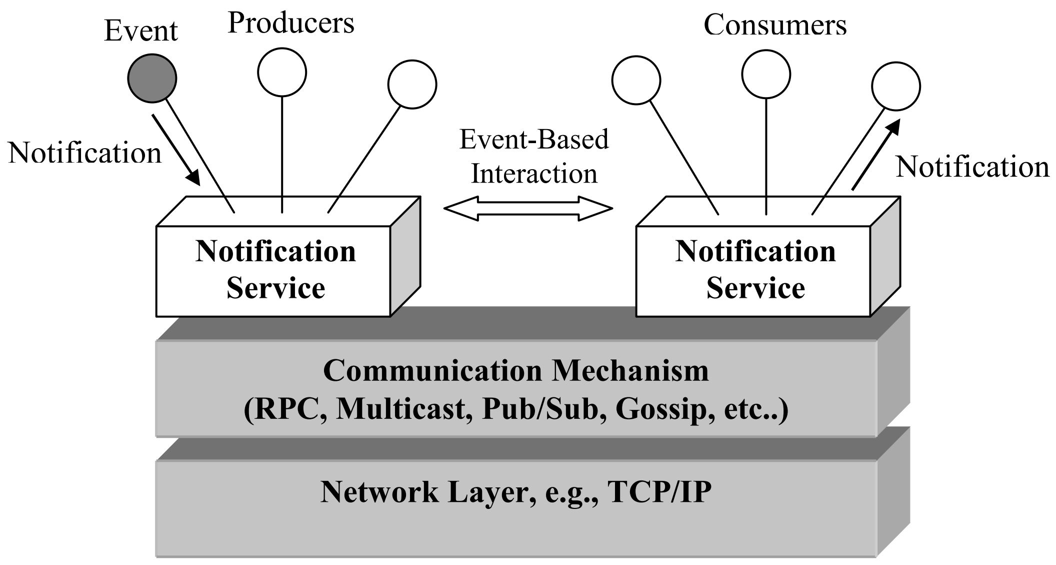

Figure 1: Event-Based Interaction

Whenever a notification is generated the producer “simply” publishes it into the notification service. The producer is not aware of the (potential) recipients of its notifications. This mode of *decoupling in space* is one of the major advantages of publish/subscribe systems. After publishing a notification the notification service is responsible for distributing notifications reliably to any subscriber that issued a matching subscription as shown in figure 1. On the other end of the communication relationship the *consumers* or *subscribers* are placed. They issue a standing request for certain notifications. Once they receive such notifications via the notification service, they react to them, accordingly. They, too, are oblivious to the issuer of the notification. Thus, interaction is inherently data-driven. Not knowing the actual communication peer, they issue a description of the data they want to receive. This description is called a *subscription*. Different classes of subscriptions are introduced in the next subsection. It must be noted that a component can act both, as consumer and also as producer of notifications. No exclusive separation of roles is assumed.

2.1 Subscriptions and Filters

A *subscription* describes and represents the interest for a certain set of notifications. Consumers register their interest by submitting subscriptions to the notification service, which evaluates the subscriptions on behalf of the consumers. The intended semantics is to filter out all unwanted information and only let information pass that exactly matches a subscription. Thus, subscriptions are commonly implemented as *filters* in the notification service. In particular, filters constitute an evaluation function that tests “incoming” notifications. Thereby, the function’s range is restricted to boolean values, i.e., either a notification matches a subscription (*true*) or not (*false*). In general, a subscription function may specify more constraints on message delivery than a pure filter function on a message’s content. It also might include meta-data. This can be exploited for the specification of additional data influencing the delivery decision.

2.1.1 Filter Models

Obviously, the expressiveness of a subscription is dependent on the specification language used. In distributed notification services, essentially five *filter models* are distinguished: channels, subjects, types, content-based, and concept-based.

Channels. *Channels* are the simplest form of subscribing to sets of notifications. In the channel-based model (OMG 2000), a producer has to select a named channel into which a notification is then published. For selecting certain notifications the client wants to receive, it only can select a channel. Any information published on this channel is delivered to the client; independent of the *concrete* interest of the client.

Subject-based addressing. *Subject-based addressing* uses string matching for notification selection (Oki et al. 1993). Every notification is part of a hierarchy of subjects. I.e., every notification is annotated with a *character string*, describing the position relative to the hierarchy this data item belongs to.

Type-based selection. *Type-based selection* uses similar path expressions and sub-type inclusion tests to select notifications (Eugster et al. 2001). With multiple inheritance, the subject tree is extended to type lattices that allows for different rooted paths to the same node. Often, type checking is complemented with content-based filters to improve selectivity.

Content-based filtering. *Content-based* filtering is the most general scheme of notification selection (Muhl, 2001). Where other approaches use distinct addressing schemes for notification selection (e.g., *strings* for subject specification), content-based addressing uses the complete content of a message as possible selection criteria. Boolean expressions evaluate the whole content of notifications, where the data model of the notifications and the expressiveness of the predicates determine the filter selectivity.

Concept-based filtering. *Concept-based* filtering (Cilia et al. 2003) is another general scheme of notification selection and an extension to content-based filtering. In this approach, filtering has to be done on a level where semantic translations have to be performed in order to identify matching filter/notification pairs. Semantic translations usually employ meta-data and are based on ontologies. Hence, concept-based filtering introduces much flexibility on the one hand, but limits its applicability to domains where well-defined ontologies exist.

2.2 Event Notification Service

Because publish/subscribe is intended to decouple producers and consumers of information a mediator between the participants is needed. An event notification service, or notification service for short, can implement this role. In event-based systems, the notification service alone is responsible for message delivery from publishers to subscribers.

	Description
publish(N)	Publishes event observations into the event system.
subscribe(Sub)	Subscribes to certain information.
unsubscribe(Sub)	Unsubscribes to certain information.
notify(N)	Notifies a client about the arrival of a notification *N* matching a previously issued subscription *Sub*

Figure 2: The publish/subscribe interface of an event notification service

The notification service offers a simple, yet sufficient, publish/subscribe interface for clients. Only the *publish*, *subscribe*, *unsubscribe*, and *notify* calls are needed (Figure 2). Messages get into the notification service by a *publish* call of an attached client and publisher. The notification service then tests the newly arrived notification against all subscriptions which are currently active in the system. Active subscriptions are issued by some consumers, stating their interest by issuing a standing request, using the *subscribe* call. The notification service then adds a new subscription to the set of active subscriptions. Whenever the test of a notification against an active subscription is positive, the notification eventually is delivered to a subscriber. Delivery is done by calling the *notify* call of a registered client.

3. RELATED WORKS

The problem of providing an effective abstraction representing the sensor network services has been the focus of several prior works. The proposed solutions have ranged from database inspired approaches, tuple space approaches to event based approaches and service discovery based approaches. Below is a discussion of each of the approaches.

3.1 Database-Inspired Approaches

The database-inspired approach allows a simple, declarative style of querying at the application level. Examples of solutions that adopt this approach are COUGAR (Bonnet *et al.* 2001) and TinyDB (Madden *et al.* 2005). Some of the work in this area has been on pure sensor database systems, which essentially provide a distributed database solution appropriate for resource-constrained sensor networks, focusing on efficient query routing and processing. The COUGAR and TinyDB sensor database systems are designed for use by relatively simple data collection applications, such as environmental monitoring applications. The main forms of data processing they support within the network are selection and aggregation based on arithmetic functions such as summation and averaging. To some extent, both are concerned with power conservation, providing query processing strategies that aim to conserve resources. A key limitation of sensor database systems is the assumption that sensor nodes are largely homogeneous. Sensor nodes must agree in advance on the data types/relations that will be used at every node.

3.2 Tuple Space Approaches

The database approaches described in the previous section provide a form of "shared memory" model, in which queries can be submitted to the sensor network as if the data was stored in a centralised repository. A similar approach, but with a different query paradigm, is provided by the TinyLIME middleware (Curino *et al.* 2005). TinyLIME is based on the tuple space shared memory model made popular by Linda (Gelernter 1985). TinyLIME is designed for environments in which clients typically only need to query data from local sensors. It does not provide multi-hop propagation of data through the sensor network - the only way clients can obtain data from a remote location is by obtaining it from other clients in that location. The design of TinyLIME assumes that sensor nodes are sparsely distributed, while clients move around, accessing local resources.

3.3 Event-Based Approaches

Advocates of event-based and publish/subscribe middleware have long argued that they are appropriate in systems in which mobility and failures are common, as they support strong decoupling of event producers and subscribers. Yoneki and Bacon (2005, p.366) have produced a reasonably sophisticated set of event operators for describing event patterns in sensor networks. The main distinction between the event description language and the subscription languages used in previous publish/subscribe systems is that it supports not only standard operators, including conjunction, disjunction, negation, concatenation and iteration, but also spatial and temporal restrictions. A crucial limitation of this solution is the complexity that is necessarily involved in implementing it. Some work on handling uncertainty of events in sensor networks has been done by Li *et al.* (2004, p.351) in their work on DSWare (Data Service Middleware). They introduce the notion of confidence when looking at event correlations. For a compound event made up of several sub-events, they propose using a confidence function to determine the likelihood of the compound event, according to how many of the sub-events have occurred. In contrast, the Mires middleware (Souto *et al.* 2006) is a more pragmatic publish/subscribe solution that has been designed and implemented to run on TinyOS (Berkeley, 2006). TinyOS provides built-in support for event handling and a message-oriented communication paradigm (Active Messages), both of which, Souto *et al.* argue, provide a strong basis for implementing a publish/subscribe middleware. Mires provides an architecture that allows: sensor nodes to advertise the types of sensor data they can provide; client applications to select from the advertised services; and sensor nodes to publish their data to clients in accordance with their subscriptions.

3.4 Service Discovery Based Approaches

The MiLAN middleware (Heinzelman *et al.* 2004) is builds on existing networking and service discovery protocols, using a plug-in mechanism to incorporate arbitrary protocols. Applications specify their sensing requirements to the middleware through a standard API, in terms of graphs describing sensor quality of service (QoS) and state-based variable requirements. Variables are the means used by applications to describe the types of sensor data they require. The use of a state-based variable graph means that applications can specify which subset of the variables is required in each application state (and with what QoS). One shortcoming is that MiLAN relies on existing service discovery protocols, most of which are not suitable for use in resource-poor sensor networks. This includes the two service discovery protocols mentioned by Heinzelman *et al.*, SDP and SLP. MiLAN appears to target a different class of sensor network (i.e., one that is richer in resources and closer to traditional heterogeneous distributed systems) than the previously described solutions.

4. MIPSCOM ARCHITECTURAL MODEL

In this section we discussed the core decomposition of MiPSCom, MiSense Content-based Publish/Subscribe Communication Model, the proposed communication which is based on an enhanced published/subscribed scheme shown in figure 3 below.

Enhanced Publish/Subscribe API	

Subscriber:	Subscribe([C] [M])
	Unsubscribe()
	Notify([A] [M])

Publisher:	Publish([A] [M] , push)
	Listener([C] [M])

Matching:	Matching([C] , [A])

Figure 3: The enhanced publish/subscribe API that is provided by communication model. A square bracket represents a set of constraints (C), metadata (M) or attributevalue pairs (A).

Figure 4 shows the decomposition of the communication model. The Publish/Subscribe service is distributed and the figure represents an instance of the model on one sensor node. A publish/subscribe application is divided into a variable number of Publisher and Subscriber components. A Publisher component can listen for subscriptions, collect data and publish notifications and Subscriber components can issue subscriptions and receive matching notifications. The Broker component provides the publish/subscribe service to the application, it manages the subscription table and it can apply the matching algorithm to filter out notifications that do not match a registered subscription.

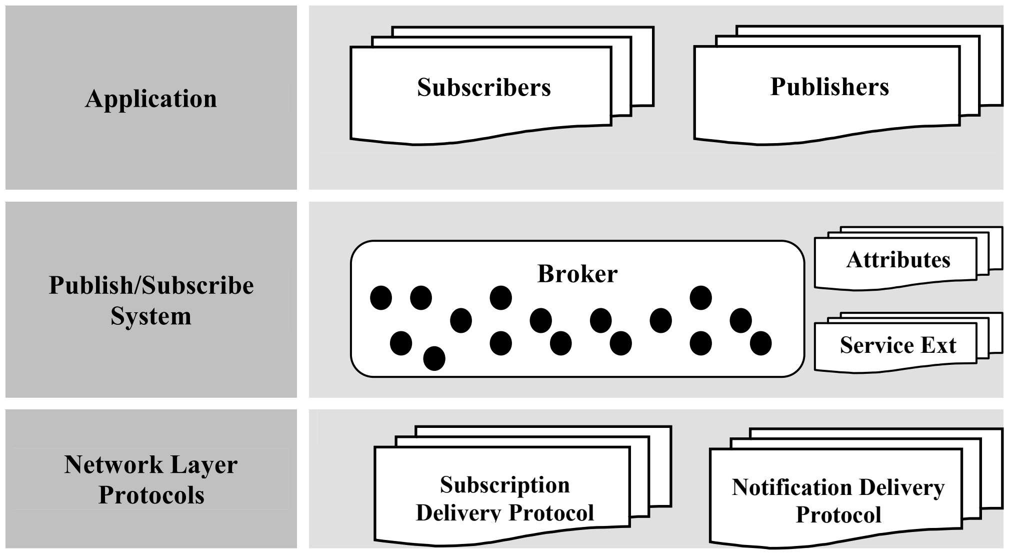

Figure 4: The MiPSCom Architecture

The data ("events") that subscribers can subscribe to and publishers can publish are encapsulated in Attribute components. In addition to a data collection interface, an Attribute component must provide a matching interface that compares two of its data items based on an attribute-specific operator. The motivation is twofold: First, an Attribute component represents functionality that Publisher components should be able to reuse and access independent of the specific attribute properties (data type, metric, etc.). Secondly, matching operators are usually attribute dependent: for example, when sensor readings are affected by hardware-related jitter, the operator "=" should not be interpreted as the exact equality of two values. To increase modularity and keep the core matching algorithm decoupled, this information should be provided by the particular Attribute component. Within the network, all attributes and operators are represented by integral identifiers. Attribute identifiers are globally unique, while operator identifiers are unique within the scope of a particular attribute. The AttributeCollector component structures access to the attributes: it maps a request based on the attribute/operator identifier to an actual Attribute component that is registered at compile time (but could even be added at runtime by dynamic over the air code updates).

In MiPSCom, the proposed communication model, the publisher publishes its interface (Listener), including the events it will notify. A subscriber *registers* interest in events indicating, where appropriate, constraints on the event parameters. The publisher *notifies* the subscriber of event occurrences that match the subscriber's registration. The broker service acts as a mediator between the publisher and the subscriber decoupling the subscriber and the publisher in space, flow and time, undertaking event filtering and event storage and, at the same time, providing services such as message buffering and message forwarding to disconnected subscribers. In MiPSCom subscribers register their interest in events by typically calling a *Subscribe()* operation on the event service without knowing the publishers of these events. A symmetric operation *Unsubscribe()* terminates a subscription. To generate an event, a publisher calls a *Notify()* operation on the event service. The event service directs the call to all relevant subscribers so that *every subscriber* receives a notification for every event conforming to its registration.

The key elements in the proposed communication model are the notification service and the buffer where the messages are queued before they are passed to subscribers. The notification service takes responsibility to inform the subscribers when a new message arrives. In this way, it allows the asynchronous communication as producers and consumers are fully decoupled. This loose coupling is the prime advantage of this kind of communication in the context of ad-hoc and pervasive environments such as wireless sensor networks.

5. MIPSCOM NAMING SCHEME

To represent subscription and notification content, our model adopts the attribute-based naming scheme presented in (Carzaniga and Wolf, 2003): a subscriber expresses its interest in data through a conjunction of constraints over attribute values. Disjunctive constraints need to be expressed as separate subscriptions. A constraint is a (attribute, operator, value) tuple and represents a filter on attribute data, for example (Temperature, >=, 30). Publishers publish data in form of notifications containing (attribute, value) tuples, for example (Temperature, 32). A notification matches a subscription if every constraint in the subscription is satisfied by a (attribute, value) tuple in the notification.

If a subscription consisted only of constraints over attribute values a subscriber would not be able to explicitly influence the properties of the communication or sensing process like, for example, the sampling rate. Such control properties are conceptually different from the data constraints and can usually not be matched by corresponding (attribute, value) tuples in the notification. We extended the basic naming scheme by allowing subscribers to include metadata in subscriptions. Metadata is either exchanged between publisher/subscriber components or plays a key role in controlling service extensions. It represents control information with soft semantics and is excluded from the matching process.

Metadata is represented by one or more (attribute, value) pairs, for example (SamplingRate, 10). Conceptually, it represents a notification that the subscriber attaches to the subscription. Metadata is specified per subscription and multiple active subscriptions may have different values for the same metadata attribute. Since metadata is non-binding a publisher may apply local optimization techniques: for example, in order to reduce sampling overhead the publisher may decide to combine two subscriptions that address the same attribute by sampling only once with an average sampling rate when the rates are similar, or using the maximal sampling rate when not.

The modified naming scheme is supported by two extensions of the basic publish/subscribe service: a “listener" service and a “matching" service. The “listener" service can be used to inform the application about newly arrived subscriptions, which it then can inspect to decide whether to start or stop publishing notifications. The “matching" service may be used by the publisher to check whether a set of attributes disqualifies it from matching a registered subscription. If, for example, the first collected attribute violates a constraint, collecting further data is pointless. When used, these primitives may result in a tighter coupling between publishers and subscribers than in the traditional model, but they have the potential to increase the efficiency of the data collection process, resulting in overall application performance gain.

6. MIPSCOM FLEXIBLE COMMUNICATION MECHANISM

The classical content-based publish/subscribe systems have tightly integrated filtering, routing and forwarding mechanisms (Muhl et al. 2002, Intanagonwiwat et al. 2003, Hall et al. 2004) resulting in more optimized, but less flexible solutions. Our model departs from this tradition and decouples the communication mechanisms from the publish/subscribe core. The core broker component has clean interfaces towards the external protocol components, thus trading some of the optimization potential for increased flexibility in selecting the subscription and notification protocols.

By exposing the choice of the protocols to the application designer, our model allows the adaptation of the publish/subscribe service to the specific needs of the application. The type of the communication protocols as well as their energy consumption are likely to have a huge impact on the overall performance, and the application designer should be aware of these implications (Heidemann et al. 2003) to make an optimal selection for the particular application. In contrast to the integrated solutions that rely on a pure content-based routing and forwarding mechanisms, the flexibility of our model raises the challenge of interfacing with communication protocols that support different dissemination patterns like broadcast, multicast, convergecast, point-to-point, etc., using various addressing models like address-free, id-centric or geographic addressing.

To support this wide range of communication mechanisms we rely on three architectural features. First, the core of the model is agnostic to the underlying addressing model, and all information relevant for operation of the service is encapsulated in the form of metadata, subscription filters or notification data. Secondly, the interfaces towards the subscription and notification delivery components are kept address-free. Finally, all the addressing information for the communication protocols is provided/consumed by their respective components or wrappers, while the model provides hooks that facilitate its encapsulation and tunneling when so required. To illustrate this process, we examine the handling of the address information on the subscription and notification path separately.

7. MIPSCOM EVALUATION

In this section the results of the evaluation of the proposed approach is described. For the evaluation, it will be inappropriate to compare MiPSCom with other monolithic publish/subscribe frameworks because the overall performance of the communication models is dominated by the underlying protocols and not the architectural features, there is currently no TinyOS implementation of a monolithic publish/subscribe framework that would facilitate direct comparison, and even if such an implementation was available, the comparison results would be vulnerable to differences in the invested optimization effort. Instead, the evaluation scenarios in this section are focused on demonstrating the flexibility and versatility of the design. To this end, we have developed a reference implementation of the MiPCom architecture model using the TinyOS (Berkeley, 2006) execution environment. The development of the reference implementation and its evaluation will allow us to demonstrate that the general design can be implemented under the specific constraints of a target domain.

Starting with a simple data collection application scenario we present experimental results which show that the choice of dissemination protocols can exhibit considerable performance tradeoffs. To demonstrate the tradeoffs that MiPSCom exposes to the application designer through protocol selection we contrast two subscription delivery protocols: a plain flooding protocol (every node that hears a subscription broadcasts it to all its neighbours once) and an epidemic broadcast protocol. The latter lets nodes continuously broadcast status information about the subscriptions they have received. Whenever a node hears an older subscription than its own, it broadcasts an update to its neighbours. In contrast to the flooding protocol, which ends its operation after a short time, the epidemic dissemination protocol remains active.

We created a simple application with one subscriber and the rest of the nodes used as publishers. In our first measurement we disseminated the subscription via plain flooding. In the second, we used the epidemic dissemination protocol. The modification is done by changing a single line of the application configuration. For notification delivery in both measurements we use the Collection Tree Protocol (CTP) performing best-effort, multihop delivery of notifications to the sink of the tree (subscriber). Both measurements lasted 90 minutes and were made with 85 publisher nodes and one subscriber (used as base station, bridging to/from a PC). At time t_0 a subscription was injected asking for notifications to be published with a rate of one notification per minute by each publisher. After 30 minutes, at time t_1, one third of the publisher nodes (randomly chosen) were shut down and 30 minutes later, at time t_2, they were powered up again. Nodes that were shut down lost all state including subscription table entries.

Figure 5 shows the percentage of active publishers over time. We define active publisher as a node that has registered a subscription and published at least one notification. At time t_1 the number of active publishers decreases by about 30% due to our active power management. The difference between the protocols becomes visible at time t_2 when these nodes are powered up again: the epidemic

Dissemination protocol quickly manages to spread the subscription to the recovered nodes, while the flooding protocol cannot (the subscription was injected only once at time t_0).

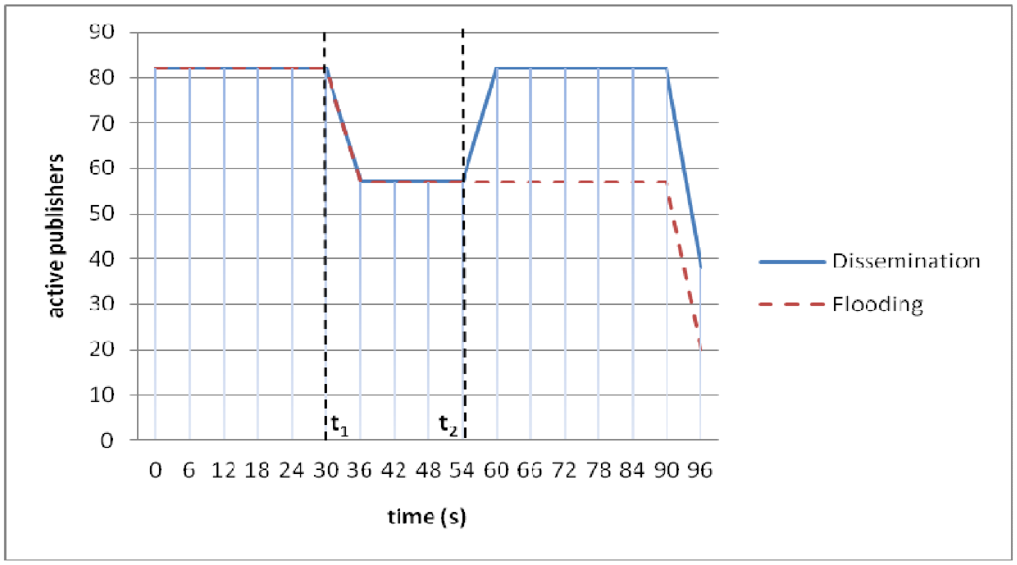

Figure 5: Number of active publisher nodes.

Figure 6 shows the changes in notification throughput perceived by the subscriber. We define notification throughput as the number of distinct notifications that arrive at the subscriber in a fixed time window of one minute. The curves almost match the number of active publishers and indicate a very good delivery ratio of CTP.

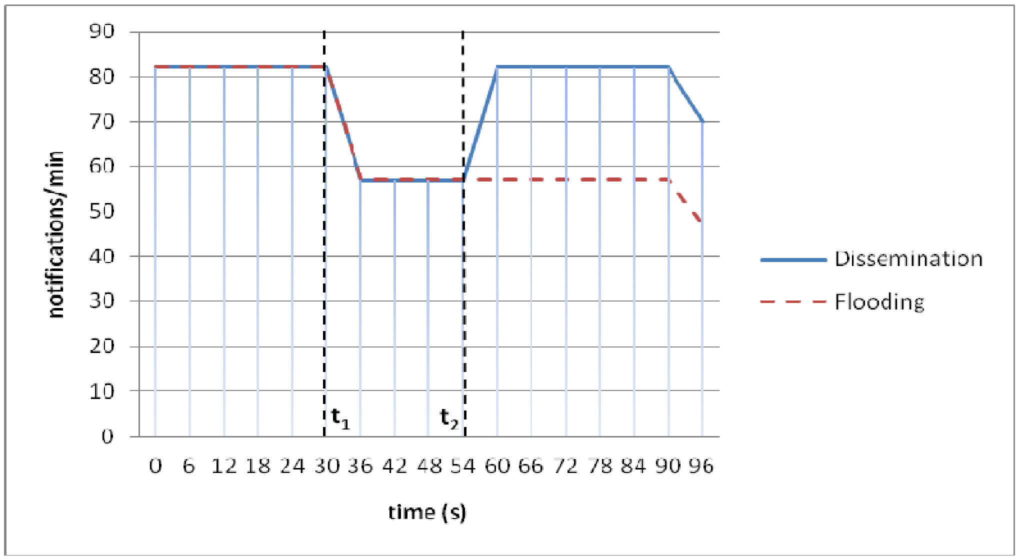

Figure 6: Notification Throughput

We let all nodes periodically output status information about the number of different messages they had sent over the wireless channel. This information allowed us to derive the traffic for subscription delivery as depicted in Figure 7. The figure visualizes the tradeoff between the protocols: the flooding protocol generates one message for each node in the network at the time the subscription is injected. The Dissemination protocol generates more messages, but is able to update the rebooted publishers at time t_2. Finally, our setup allowed us to determine the number of notification messages sent in the network by all nodes over a time window of one minute – on average 3 messages were sent per notification, however our setup did not allow us to differentiate between retransmission and forwarded messages.

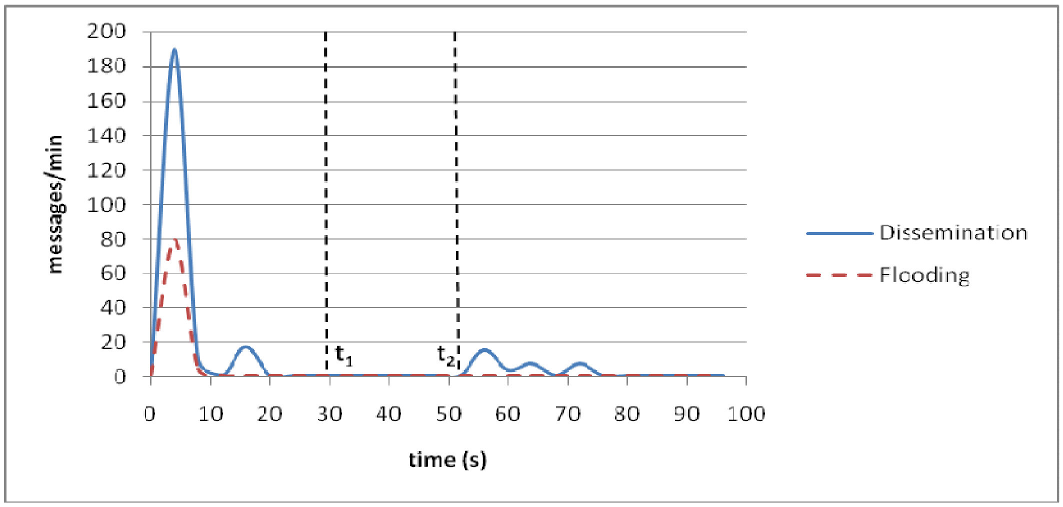

Figure 7: Subscription protocol traffic.

Previous work (Heidemann et al. 2003) has shown that the interaction pattern between publishers and subscribers (“pull" vs. “push") can significantly affect application performance and should be carefully aligned with the ratio of publishers to subscribers. We created an application that included two Publisher components, one for periodic temperature data collection and one for generating fire alarm messages. We wanted the fire alarm event to quickly propagate to all rooms of the office building, but periodic measurements to be collected only by a single subscriber.

We therefore selected a single node to disseminate a subscription which notifications from the first Publisher component had to match (locally, based on the “pull" model). Fire alarms, however, were “pushed": whenever the second Publisher component detected a fire alarm regardless of any registered subscription, it immediately distributed the notification to all nodes in the network. The first Publisher component was “wiring" the subscription delivery protocol to the core and using CTP for notification delivery. The second Publisher component “wired" the flooding protocol for notification delivery.

Figure 8 shows a trace of the communication rates collected over 20 minutes on 85 nodes. It represents the total number of packets sent by all nodes for a fixed time window of one minute. One subscription for periodic data collection is issued at the start of the measurement using the TinyOS Dissemination protocol, 10 minutes later we simulate a fire alarm, by sending a serial packet to one of the publisher nodes (randomly chosen). This node then started a flood of notification messages. The increase in traffic is visible by a small spike, however it is almost masked by the high level of CTP "pull" traffic.

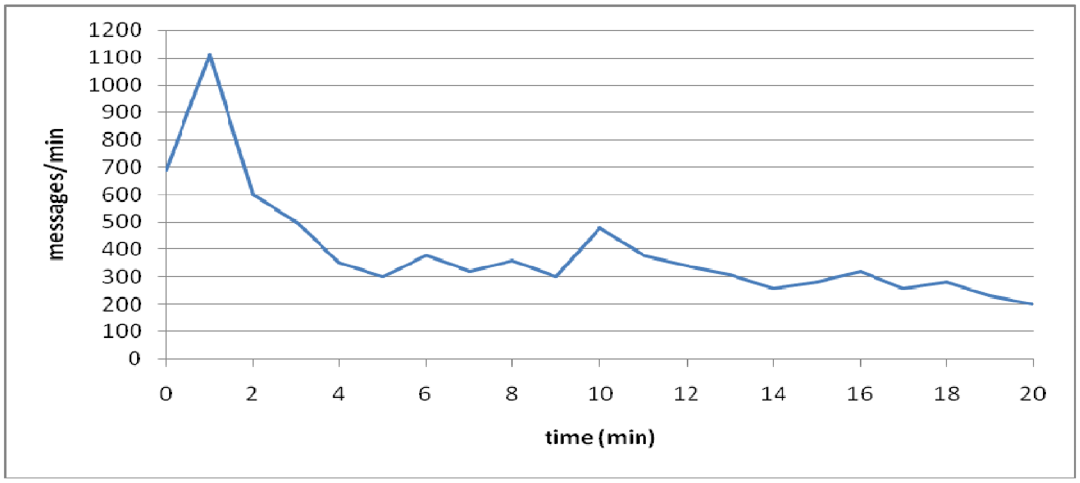

Figure 8: Push and Pull Interaction

The above results show that our approach exports significant performance tradeoffs to the application in an easy-to-use fashion, and the communication model is general and flexible enough to support different interaction patterns and the execution time overhead is acceptable.

8. LIMITATIONS AND FUTURE WORKS

The overhead introduced by our publish/subscribe system in terms of energy spent has been monitored for the scenario described in figure 8. It is observed, from figure 9, that the energy overhead is high in the first 2 minutes of the process and then there is a constant and low energy overhead for the remaining time. The energy overhead is attributed to the additional metadata and control information exchanged. The benefits of increased flexibility and more fine-grained control over the publish/subscribe service through a number of extension components outweigh the energy overhead introduced by the system.

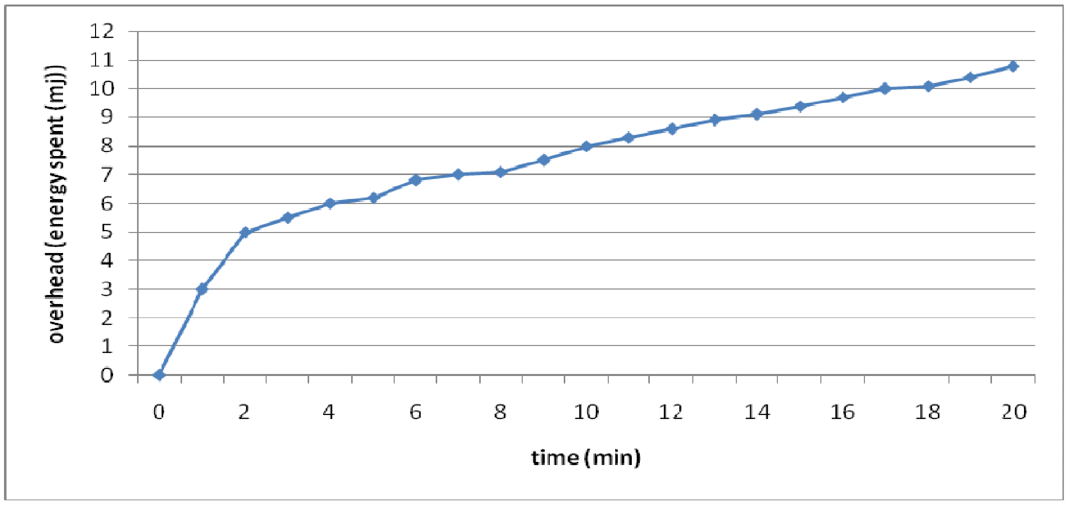

Figure 9: Energy overhead of MiPSCom

The MiPSCom communication model has many points of possible optimizations and further performance evaluations are required. We would like to implement a real-world working system as proof-of-concept application. Specifically, we will develop a testbed that can monitor the environment, for example, with respect to chemical pollutants – a possible application is the monitoring of garbage dump sites; this application will demonstrate our communication model and algorithms in different contexts. The availability of a real implementation will give us the opportunity to detect open problems, which do not appear in artificial simulated scenarios. By further real world studies we will be able to get further insights in the performance of MiPSCom. This will enable a more direct assessment of MiPSCom performance in real-world applications. We would also like to investigate the covering and advertisements optimizations further.

9. CONCLUSION

Publish/subscribe is a widespread communication paradigm for asynchronous messaging that naturally fits the decoupled nature of wireless sensor networks systems, allowing simple and effective development of distributed sensor network applications. A major design goal of the presented content-based publish/subscribe communication model is to separate out those service sub-tasks which are expected to have large impact on the resource usage. This decomposition strives to give an application designer a simple and flexible means to select protocol components and data attributes according to his needs, and to give him more fine-grained control over the publish/subscribe service through the concept of extension components.

The flexibility of MiPSCom to support different sensor node platforms, communication protocols and interaction patterns has been demonstrated experimentally. MiPSCom can be augmented in order to give the application designers additional control knobs for trading-off different performance objectives.

Our experiences with MiPSCom suggest that by careful component decomposition and interface design, it is indeed possible to achieve a good balance between efficient resource usage and reusable software design. We observe that publish-subscribe, a distribution system implementation of the implicit-invocation architectural style, promotes reuse and extensibility. We have shown in this paper that publish-subscribe demonstrates some very attractive qualities as a middleware for wireless sensor networks systems.

REFERENCES

AKYILDIZ, F., SU, W., SANKARASUBRAMANIAM, Y. AND CAYIRCI, E. 2002. A survey on sensor networks. *IEEE Communications Magazine*, 40 (8), 102-114.

BERKELEY, U.C., 2006. TinyOS operating system [online]. Available from: http://www.tinyos.net/ [Accessed 15 August 2008]

BONNET, P., GEHRKE, J. E. AND SESHADRI, P., 2001. Towards sensor database systems. *In 2nd International Conference on Mobile Data Management (MDM),* Lecture Notes in Computer Science, Springer, 3–14.

CARZANIGA, A., WOLF, A.L. 2003. Forwarding in a content-based network. *In: Proc. of ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

CILIA, M., BORNHOEVD, C. AND BUCHMANN, P. 2003. CREAM: An infrastructure for distributed, heterogeneous event-based applications. In *Proceedings of the Intl Conference on on Cooperative Information Systems (CoopIS'03)*, Nov 2003.

COSTA, P. AND PICCO, G., 2005. Semi-probabilistic Content-based Publish/subscribe. In *Proc. of the 25th Int. Conf. on Distributed Computing Systems (ICDCS05)*, 575–585, Columbus (OH, USA), June 2005. IEEE Computer Society Press.

CURINO, C., GIANI, M., GIORGETTA, M., GIUSTI, A., MURPHY, A.L. AND PICCO G.P., 2005. TinyLIME: Bridging mobile and sensor networks through middleware. *In 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom),* IEEE Computer Society, 61–72.

EUGSTER, P., FELBER, P., GUERRAOUI, R., AND KERMARREC, A., 2003. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131.

EUGSTER, P., GUERRAOUI, R. AND DAMM, C. 2001. On objects and events. In Linda Northrop and John Vlissides, editors, *Proceedings of the OOPSLA '01 Conference on Object Oriented Programming Systems Languages and Applications*, 254–269, Tampa Bay, FL, USA, 2001. ACM Press.

GELERNTER, D., 1985. Generative communication in Linda. *ACM Computing Surveys*, 7(1), 80–112.

HALL, C.P., CARZANIGA, A., ROSE, J., WOLF, A.L. 2004. A content-based networking protocol for sensor networks. *Technical Report CU-CS-979-04,* Department of Computer Science, University of Colorado, August 2004.

HEIDEMANN, J., SILVA, F., ESTRIN, D. 2003. Matching data dissemination algorithms to application requirements. *In: SenSys '03: Proc. of the 1st international conference on Embedded networked sensor systems*, New York, NY, USA 2003.

HEINZELMAN, W.B., MURPHY, A.L., CARVALHO, H.S. AND PERILLO, M.A., 2004. Middleware to support sensor network applications. *IEEE Network*, 18 (1), 6–14.

INTANAGONWIWAT, C., GOVINDAN, R., ESTRIN, D., HEIDEMANN, J., SILVA, F. 2003. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking (TON),* 11(1), 2003.

LEVIS, P., BREWER, E., CULLER, D., GAY, D., MADDEN, S., PATEL, N., POLASTRE, J., SHENKER, S., SZEWCZYK, R., WOO, A., 2008. The emergence of a networking primitive in wireless sensor networks. *Communications of the ACM*, 51 (7), 99-106, July 2008.

LI, S., LIN, Y., SON, S.H., STANKOVIC, J.A. AND WEI, Y., 2004. Event detection services using data service middleware in distributed sensor networks. *Telecommunication Systems*, 26(2-4), 351–368.

LIU, C.M., LEE, C.H. AND WANG, L.C., 2004. Power-efficient communication algorithms for wireless mobile sensor networks. In *PE-WASUN '04: Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, 121–122, New York, NY, USA, 2004. ACM Press.

MADDEN, S., FRANKLIN, M.J., HELLERSTEIN, J.M. AND HONG, W., 2005. TinyDB: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30 (1), 122–173.

MÜHL. G. 2001. Generic constraints for content-based publish/subscribe systems. In *Proc. of CoopIS '01*, volume 2172 of *LNCS*, 211–225. Springer-Verlag, 2001.

MUHL, G., FIEGE, L. AND BUCHMANN, P. 2002. Filter similarities in content-based publish/subscribe systems. In H. Schmeck, T. Ungerer, and L. Wolf, editors, *International Conference on Architecture of Computing Systems (ARCS),* 224–238, Karlsruhe, Germany, 2002.

OBJECT MANAGEMENT GROUP, 2000. CORBA event service specification, version 1.0. OMG Document formal, 2000-06-15, 2000.

OKI, B., PFLUEGL, M., SIEGEL, A. AND SKEEN, D., 1993. The information bus— an architecture for extensible distributed systems. In Barbara Liskov, editor, *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 58–68, Asheville, NC, USA, December 1993. ACM Press.

SOUTO, E., GUIM˜ARAES, G., VASCONCELOS, G., VIEIRA, M., ROSA, N., FERRAZ, C. AND KELNER, J., 2006. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing,* 10(1), 37–44.

SUN, B., GAO, S., CHI, R., HUANG F., 2008. Algorithms for balancing energy consumption in wireless sensor networks. *In FOWANC '08: Proceeding of the 1st ACM international workshop on Foundations of wireless ad hoc and sensor networking and computing*, 53-60, Hong Kong, China, May 2008.

YONEKI E. AND BACON, J., 2005. Unified semantics for event correlation over time and space in hybrid network environments. *In IFIP International Conference on Cooperative Information Systems (CoopIS)*, Lecture Notes in Computer Science, Springer, 366–384.