36

Enabling Publish/Subscribe Services in Sensor Networks

Duc A. Tran[1] and Linh H. Truong[2]

[1] *University of Massachusetts, Boston, USA*
[2] *IBM Zurich Research Laboratory, Switzerland*

36.1 Introduction

We have seen in this decade sensor technologies increasingly deployed in many applications. Sensors are used to monitor the surrounding environment for important events such as climate changes, chemical leaks, early warnings of a natural disaster, or violations in a no-trespassing zone. For a large area being monitored, we need a sensor network that allows for efficient search and dissemination of the sensor data.

A sensor network for monitoring purposes basically involves two types of nodes: the “query" nodes and the “sensor" nodes. The query nodes are those that send queries into the network to inquire about sensor data of interest. A query node can be a sink node that needs to collect the data from the network or an actuator node whose operation is triggered based on events detected in the network. The sensor nodes are those that capture event data and need to send them, or notify of their existence, to interested query nodes. In many applications, a query needs to be submitted to the network in advance waiting for notification of the events that match the query. Because a sensor node does not know who may be interested in its data, and, vice versa, a query node does not know where in the network its events of interest may occur, a challenging problem is to design an effective mechanism for query subscription and the event notification so that an event can notify its inquirers quickly and efficiently. In this publish/subscribe problem, the sensor nodes

and query nodes are respectively called the *publisher* nodes and *subscriber* nodes.

Many Internet-based publish/subscribe systems have been designed, e.g., PADRES [24], REBECA [25], SIENA [12], and XNET [14]. Some concepts of these systems might be useful, but enabling publish/subscribe services in sensor networks is fundamentally different due to unique constraints on communication, storage, and computation capacities. On the other hand, despite a lot of research and development efforts made for sensor networks that provide search mechanisms, most of them are focused on retrieving sensor data that have *already* been stored by the network. The publish/subscribe model poses a different challenge as a query of this model is to inquire about *future* events. Thus, the query must be stored proactively in the network to wait for those events. When such an event occurs, it needs to be published to the network to search for the matching queries among the stored. With the “proactive" storing of queries, events can be delivered “timely" to the subscribers, while in the “retrieving" case, subscribers only receive “past" events. This is of importance to applications that require a real-time monitoring of the events.

To enable publish/subscribe services in sensor networks, a desirable design should consider the following important issues:

- Routing design: To subscribe a query, without any knowledge about where a matching event could occur, the simplest way is to broadcast it to the entire network. This way, an event can find its matching queries immediately at the local node. Because the traffic due to broadcasting can be overwhelming for a large-scale network, other efforts have been attempted to reduce this communication with an efficient routing design. A key approach is to replicate a query to a set of select nodes and to publish an event to another corresponding set of select nodes such that, if the event matches the query, there exists a rendezvous node, certainly or at least with a high probability. This guarantee needs to take into account the communication cost due to transmissions of queries and events.
- Query aggregation: Because a sensor node’s storage capacity is usually limited, it is desirable that a node stores as small a number of queries as possible. The broadcast approach aforementioned incurs a high storage cost because each node has to store every query. It is thus important to reduce the number of replicas for any given query. By doing so, it also helps reduce the amount of traffic in the network due to query forwardings. Query aggregation serves this purpose. For example, queries can be merged to produce fewer queries. Or, if a new query arrives at

a node finding itself covered by a locally stored query, the new query may not necessarily be forwarded further. This is so because any event matching the new query will match the existing query as well, and thus this event will be returned anyway as a result of the existing query's earlier subscription.

- Event matching: When a node receives a publication of an event, the node may need to evaluate its locally stored queries to find those matching the event. Because a sensor node has limited processing capability, this procedure if not properly designed may create a computational burden too heavy for the node. Several techniques have been proposed to organize the queries into some indexing architecture that is convenient for event matching. Alternatively, one may employ a multi-phase checking algorithm where the earlier phases are to determine quickly whether a query should be ignored and the later phases are to evaluate the queries that pass the earlier phases.

While most techniques emphasize the above algorithmic issues, e.g., [34, 31, 22, 40, 48, 16, 27, 49], publish/subscribe middleware techniques aimed at high-level service abstraction have also been proposed [44, 33, 28, 52]. They provide a convenient middleware layer serving as an API for the application developer, hiding all the underlying network complexities and the implementation details of the publish/subscribe mechanisms.

A survey of representative publish/subscribe techniques for sensor networks is presented in this chapter, with attention given to the issues of routing design, query aggregation, event matching, and middleware development.

36.2 Preliminaries

36.2.1 Event and Query Representation

An event in a publish/subscribe system is usually specified as a set of d attribute-value pairs $\{(attr_1, v_1), (attr_2, v_2), ..., (attr_d, v_d)\}$ where d is the number of attributes, $\{attr_1, attr_2, ..., attr_d\}$, associated with the event. For example, if the sensor network is used to monitor temperature, humidity, wind speed, and air pressure of some area, d is four – representing these four sensor data attributes.

The constraints in a query, in general, can be specified in a predicate of the disjunctive normal form – a disjunction of one or more condition clauses, each clause being a conjunction of elementary predicates. Each elementary predicate, denoted by $(attr_i \ \gamma \ p_i)$, is a condition on some attribute $attr_i$ with γ

being the filtering operator. A filtering operator can be a comparison operator (one of $\{=, <, >\}$) or a string operator such as "*prefix of*", "*suffix of*", and "*substring of*" if the attribute is of *string* type. However, for simplicity of implementation, most schemes assume that a query is a single conjunctive clause of elementary predicates that can only use the comparison operators. This form of query can be called the *rectangular form* because if an event is modeled as a point in a d-dimensional coordinate system, each dimension representing an attribute, a query can be considered a d-dimensional box with the vertices defined based on the attribute constraint values provided in the query clause. Sometimes it can be a tedious process to specify all the lower and upper bounds for all the attributes of a query. In such a case, it is more convenient to specify a query in terms of an event sample and ask to be notified of all the events similar to this sample. For example, consider a camera-sensor remote surveillance network deployed over many airports to detect criminal suspects. If a particular suspect is searched for, his or her picture is submited as a subscription to the network in hopes of finding the locations where similar images are captured. A query of this kind can be represented by a sphere, in which the sample is the center of the sphere and similarity is constrained by the sphere's radius. This query is said to have the *spherical form*.

36.2.2 Subject-based vs. Content-based

There are two main types of publish/subscribe designs [20]: subject-based or content-based. In the subject-based design, events are categorized into a small number of known *subjects*. There must be an event attribute called *'subject'*, or something alike, that represents the type of the event and a query must include a predicate *('subject' = s)* to search only events belonging to some known subject *s*. The occurence of any event of subject *s* will trigger a notification to the query subscriber. The subscription and notification protocols are mainly driven by subject match rather than actual-content match.

The content-based design offers a finer filtering inside the network and a richer way to express queries. A subscriber wants to receive only the events that match its query content, not all the events that belong to a certain subject (which could be too many). A node upon receipt of a query or event message needs to extract the content and makes a forwarding decision based on this content. We can think of the subject-based model as a special case of the content-based model and because of this simplification, a subject-based system is less challenging than a content-based system to design.

36.2.3 Sensor Network Assumptions

There are different types of sensor networks that have been considered by existing publish/subscribe techniques. The techniques in [39, 40] assume that a sensor node knows its location (e.g., by a built-in GPS-like device or by running a localization protocol [47]). A mapping from the query/event space to the location space can be designed, based on which a query and a matching event are sent to the corresponding nodes who can find each other within a short distance. For example, if a query is replicated at all the nodes along the vertical line crossing the subscriber node's location, and if an event is published to all the nodes along the horizontal line crossing the publisher node's location, a rendezvous node exists for every pair of queries and events. This is so because every vertical line must meet every horizontal line.

Several techniques do not require any knowledge of location information (e.g., [34, 31, 16, 45, 27, 49]). Besides the broadcast approach, another approach used by such techniques is based on some form of randomized dissemination to subscribe a query or publish an event. The main intuition is that if a query is replicated to a set of random nodes, an event published to its corresponding set of also random nodes, and if these two sets are large enough, then there exists a rendezvous node with a high probability. For example, one can use a random walk or a gossip-based protocol [16, 45] to visit these large sets of nodes. Some other techniques [27, 49] use a naming scheme to assign names to the network nodes and routing is driven by node names instead of nodes being chosen in random.

Most techniques assume that nodes are stationary. There are some techniques addressing publish/subscribe in general mobile networks that could apply to a mobile sensor network [4, 53, 26, 23, 32, 18, 15]. This chapter discusses techniques designed for stationary networks only.

36.3 Routing Design

Routing protocols for query subscription and event notification form the most important component of any distributed publish/subscribe system. While traditional routing protocols for sensor networks are designed for synchronous communication and are address-driven, routing for publish/subscribe purposes is asynchronous because subscribers and publishers are not aware of each other and the timing of the subscription and publication. There are various publish/subscribe routing approaches which are discussed below.

36.3.1 Centralized-based

A simple approach is to employ a central brokerage station that receives the subscription of every query and publication of every event. This approach keeps the system design lightweight on each node. Routing of queries and events becomes trivial because the central broker is the only destination. Queries may be aggregated at the intermediate nodes on the way to the central broker. Event matching is conducted at the central broker to find all the queries matching a newly published event. The centralized-based approach, which has been implemented in the MQTT and MQTT-S middleware at IBM [33], can work with queries and events of any form. However, the central broker can easily be a severe communication bottleneck if queries and events are produced at high rates.

36.3.2 Broadcast-based

Directed Diffusion [34] is one of the earliest publish/subscribe techniques for sensor networks. Given a new query, the first step in this technique is for the subscriber node to broadcast the query to all the nodes in the network. Each node upon receipt of this query creates a “gradient" entry in the routing table to point toward the neighboring node from which the query is received. Using a gradient path, a matching event can be sent toward the subscriber. Since there may be more than one such gradient path, the choice of which path to use is made based on some performance factors to improve energy efficiency.

[31] proposes building a broadcast tree, called PST, spanning all the nodes in which events will be disseminated from a publisher (root node) to all their subscribers. Each node in PST that has a subscription needs to let its parent node know of this subscription. Thus a node can compute a combination of all the subscriptions downstream. When an event is published by the root, the event will follow the branches of PST that lead to all the matching subscribers. If there are multiple publishers, multiple trees are built each for a publisher, or alternatively, a shared root is selected that receives all the events from the publishers and a single tree is built based on this root to send the events to all the nodes. PST does not incur the cost to replicate queries at every node, but it is efficient only when there are a smaller number of publisher nodes.

In a different effort, [41] proposes a hierarchical structure to organize the network into multiple clusters, in which each cluster forms a publish/subscribe sub-network and has a representative node to appear in the next layer of the

hierarchy; the representative nodes form a layer of clusters which again are used to build the next layer in the hierarchy. The representative nodes are responsible for forwarding a query or event from one sub-network to another. Although this clustering idea can potentially reduce the subscription and notification traffic in the network compared to the pure broadcast-based approach, its complexity lies in the maintenance of the hierarchy under network dynamics.

36.3.3 Location-based

Flooding the network can be expensive, especially for systems where there are a large number of queries and events generated by many potential subscriber and publisher nodes. Consequently, many distributed techniques have been proposed. If the location is known for every node, the location information can be useful. [39] proposed the method of Geographic Hash Tables (GHT) – to hash the data space to locations in the location space. Each data value is respresented by an one-dimensional identifier called a “key" k, which correponds to a geographic coordinate $h(k)$ based on the hash function. Thus, if both events and queries each can be identified by a single key, we can use GHT. For example, a subject-based system is a natural candidate for this technique because the subject value can be used as the key value for GHT. A query with key k will be stored at the node closest to the location $h(k)$. When an event with the same key k emerges, it will be routed to the node $h(k)$ and thus can find its matching queries. For routing from a sensor location to another sensor location, we can use a geographic routing protocol designed for sensor networks such as [35]. An extension of the GHT technique in combination with landmark-based routing to improve routing efficiency is proposed in [22].

Another location-based technique called Double-Ruling is proposed in [40], in which each node P is mapped to a point $f(P)$ on the surface of a 3D sphere and a key k is hashed to a point $h(k)$ also on this spherical surface. A query with key k subscribed by a node P will be sent to the node corresponding to the point $h(k)$. The routing follows the great circle connecting the points $h(k)$ and $f(P)$ on the 3D surface. An event is published to the corresponding node in a similar fashion and thus every event with key k will be sent to node $h(k)$ and can find all the matching queries there. A property of Double-Ruling is that it is distance-sensitive: the length of the notification path from a publisher to a matching subscriber is guaranteed to be within a

small constant factor of the direct path connecting them. The GHT method does not make this guarantee.

The above distributed methods (GHT and Double-Ruling) should not work efficiently with a content-based publish/subscribe system because it is difficult to represent a complex query by a single key. For events and queries of any dimensionality d, one can use the technique proposed by [48]. This technique assumes the spherical form for the queries. Given a query, a random-projection method is used to hash this query into a 2D rectangle in the location space; the query will be replicated at all the nodes inside this rectangle. Using the same random projection, an event is hashed a single location; the event will be sent to the node closest to this location. The random project method guarantees that this event will find all the matching queries. To reduce the number of query replicas, the subscription covering relationship is taken into account to avoid further replications of those queries that are subsumed by previously-subscribed queries.

36.3.4 Gossip-based

Without location information, a common idea is to use some form of gossip to disseminate queries and events. A simple design is to use random walks [3, 9]. Each query is replicated on all the nodes visited by a random walk starting from the subscriber node, and each event also follows a random walk from the publisher node to find the queries. If these random walks are long enough, it is highly likely that an event will find all the matching queries. To shorten the notification delay, multiple random walks can be used to propagate an event (or to replicate a query) [9].

Another approach is proposed in [16], where both queries and events are “selectively" broadcast to the network. A query is broadcast to an extent defined by the subscription horizon ϕ which limits the number of times the query is rebroadcast. In the broadcast of an event, only a fraction τ of the neighbor links at each current node is used to forward the event. By choosing appropriate values for ϕ and τ, we can control the overhead and effectiveness of the system.

We can also design a publish/subscribe mechanism by adopting BubbleStorm – a gossip idea proposed in [45]: each query is replicated in a random-walk based tree of $q = O(\sqrt{n})$ nodes rooted at the subscriber node, and each event is sent along a similarly-built tree of $c^2 n/q$ nodes rooted at the publisher node where n is the number of nodes and $c \geq 1$ is a cer-

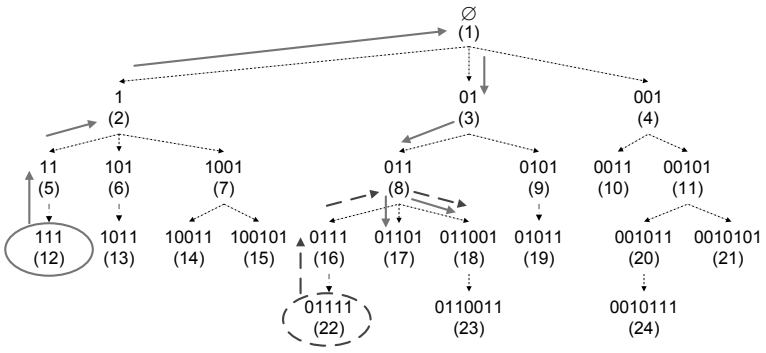


Figure 36.1 The Pub-2-Sub$^+$ scheme: Nodes are each assigned a name and the names form a prefix tree. Solid-bold path represents the subscription path of query ['0110001', '0110101'] initiated by node 12; dashed-bold path represents the notification path of event '0110010' published by node 22

tainty constant. It is shown that the probability to have a rendezvous node is $r = 1 - \exp(-c^2)$ (e.g., $c = 2$ means a hit probability of $r = 0.98$).

36.3.5 Naming-based

Despite its simplicity, the gossip-based approach incurs significant communication, storage, and computation costs because each query or event needs to be disseminated to many nodes to have a good chance to meet its matches at rendezvous nodes. Also, the guarantee that a rendezvous node exists for every pair of queries and events, including those that do not match each other, is unnecesssarily strong and thus leads to unnecessary traffic. Therefore, alternative techniques that requires no location information but is not based on gossiping are proposed [27, 49]. The common idea is to design a naming scheme that assigns *names* to the nodes in a way convenient for routing purposes. The technique in [27] clusters the network into a multi-level hierarchy and assigns to each node a name based on its position in this hierarchy. The resultant hierarchical naming scheme is used for routing during event publication and query subscription. Each node requires only $O(\log n)$ bits per node to store auxiliary routing information. Any event can find its subscribers in a distance-sensitive way. Further, by visiting only $O(k)$ nodes, the subscriber can collect all occurrences of a particular type of data within a similarity radius k (for spherical queries). The above technique works for the subject-based model only. The Pub-2-Sub$^+$ technique [49] is also based

on a naming scheme but designed for content-based services. Pub-2-Sub$^+$ maintains a set of *m* spanning trees each rooted at a node in the network. The root nodes are dedicated reliable nodes placed at random network positions. Each tree correponds to a naming tree assigning a binary-string name to each node; hence, a node has *m* names. The names on a tree form a prefix tree. Based on the naming scheme, each node is assigned a “zone" of binary strings to own. The zone of a node is the set of all binary strings starting with this node’s name but not with any child node’s name. A query is subscribed to a random tree and an event is published to all the trees. Pub-2-Sub$^+$ formats an event as a binary string (e.g., ‘0110010’) and a query an interval of binary strings (e.g., [‘0110001’, ‘0110101’]). On the randomly chosen tree, a query is routed to, and stored at, all the nodes whose zone overlaps with the query’s interval. On each tree, an event is published to the node whose name is the longest prefix of the event string. Figure 36.1 provides an example where *m* = 1. The query [‘0110001’, ‘0110101’] is stored at nodes 3, 8, 17, and 18. The event ‘0110010’ is published to node 18; it will find all the matching queries there. In general, the notification path is bounded by two times the tree height which should be $O(\log n)$ in most cases. Also, because there are multiple paths for the event notification, the disconnection of a path due to some failure does not stop an event from finding its way to the matching queries.

36.4 Query Aggregation

Queries in a publish/subscribe system need to be stored in advance. Due to the resource constraints of a typical sensor node, it is desirable to limit the replication of each query in the network. Although the routing design dictates how to disseminate a query, its integration with a query aggregation mechanism may lead to more efficient query forwarding. Queries arriving at a node can be merged and/or pruned to produce fewer queries representing equivalent constraints. Query aggregation based on covering relationship is useful to deciding whether a query needs to be forwarded further. This section describes some common aggregation strategies.

36.4.1 Subscription Covering

Used in various Internet-based publish/subscribe systems [24, 25, 12, 14], one strategy is that a node, upon receipt of a new subscription query, does not forward it if it is already covered by an existing locally stored query. Figure

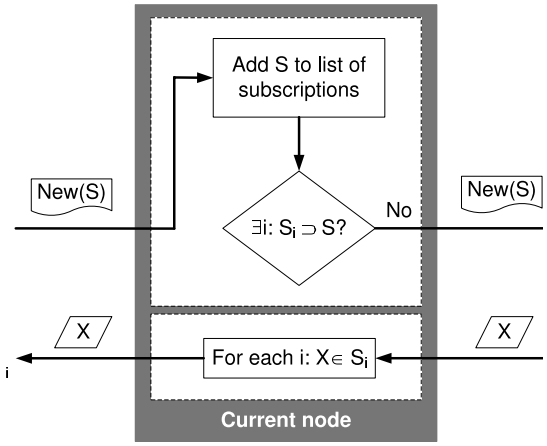


Figure 36.2 Subscription Covering: Do not forward a new subscription s if it is already covered by an existing subscription s_i

36.2 illustrates this strategy. Typically, when a new query s is routed to a node P, the node will store a copy of this query and forward it to the next node P_{next} according to the subscription's routing protocol. When the next node P_{next} receives a publication of an event x that matches s, it will forward it back to P, which in turn forwards x to the node that previously sent s to P. Now, let us assume that there is an existing query s_i stored at P such that s_i covers s (i.e., $s_i \supset s$). In this case, P may opt not to forward s to P_{next} because P knows that if there is an event x matching s, it must be returned to P as a result of P's forwarding s_i earlier. By not forwarding s, we avoid the costs of disseminating s further in the network and collecting duplicate events that satisfy both s and s_i.

Detecting coverings in a large set of queries can, however, be a computationally expensive procedure for any given sensor node. On the other hand, it is not mandatory that covered queries must not be forwarded. Techniques [38, 43, 46] have been proposed to detect *approximately* subscription coverings. Although such a technique may still forward a query even if the query is covered by an existing one, it is guaranteed that no matching event will be missed. These techniques are still much more efficient than having to detect all coverings.

The technique in [43] assumes that queries adopt the rectangular form. It is observed that if we map a rectangular query $s = [l_1, r_1] \times [l_2, r_2] \times ... \times [l_d, r_d]$ into a $2d$-dimensional point $p(s) = (-l_1, r_1, -l_2, r_2, ..., -l_d, r_d)$, the

subscription covering problem in d dimensions becomes the point dominance problem in $2d$ dimensions. A point p is said to *dominate* a point p' if every coordinate of p is no less than that of p'. Instead of solving this problem directly, [43] proposed to solve an approximate version of the problem:

> "Find a data structure for a set Σ of n points in the m-dimensional box $[0,MAX_1] \times [0,MAX_2] \times ... \times [0,MAX_m]$ so that the following question can be answered efficiently: given a constant $\varepsilon \in (0,1)$, and a query point $p = (p_1, p_2, ..., p_m)$, search a subset of the box $[p_1,MAX_1] \times [p_2,MAX_2] \times ... \times [p_m,MAX_2]$, whose volume is at least $(1-\varepsilon)$ of the volume of the box, and report any point of Σ if it is in the subset."

A data structure was proposed that sorts the input points based on its positions on the Z space-filling curve [2]. For each query point, only a subset of segments of the Z curve is searched for the existence of any point of Σ that dominates the query point p.

A different approach has been proposed in [46] which can work with both spherical and retangular queries. Unlike [43] which increases efficiency by searching only a subset of the set of queries, [46] searches all queries but for each visited query the covering condition is quickly, however, tolerated by a probability of error. To illustrate the idea, suppose that queries are spherical. Firstly, a constant $k < d$ is pre-determined. Secondly, k random orthonormal vectors, $\{u_1, u_2, ..., u_k\}$, are generated. Then, each query centered at s with radius r is mapped to the following k-dimensional rectangle: $u(s,r) = u_1(s,r) \times u_2(s,r) \times ... \times u_k(s,r)$ where each side $u_i(s,r)$ of this rectangle is the projection of the subscription on unit vector u_i, which is the following interval: ($\circ$ is the inner product) $u_i(s,r) = [u_i \circ s - r,\ u_i \circ s + r]$. To check whether a query s covers a query s', we check whether the projection rectangle $u(s)$ covers the projection rectangle $u(s')$. If it is found that $u(s)$ covers $u(s')$ (or not), it is concluded that s covers s' (or not). Although this conclusion is not always correct, it is shown that if queries are uniformly distributed in both center points and radii, the probability of error is roughly bounded by $(2/\pi)^k$. This probability approaches zero quickly as k increases.

The case of rectangular queries is handled similarly. Each original d-dimensional rectangular query s with 2^d vertices v_1, v_2, ..., v_{2^d} is mapped to a k-dimensional rectangle where each side is the following interval: $u_i(s) = \left[\min_{j \le 2^d}(v_j \circ u_i), \max_{j \le 2^d}(v_j \circ u_i)\right]$. Using a similar random projection, with a low probability of error, the subscription covering problem in d dimensions can be mapped into that in $k << d$ dimensions. The value of k can be tuned to achieve any given success rate.

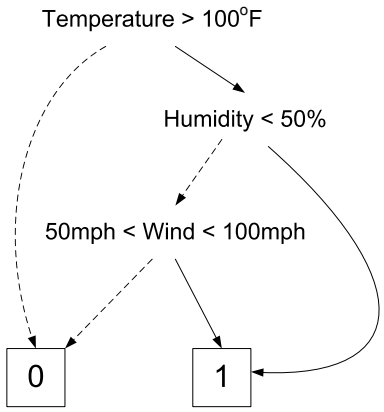
[38] addresses a more generic covering problem; that is, to find whether a new query s is covered by *the set* of existing queries $s_1 \cup s_2 \cup ... \cup s_n$, rather than by a single existing query. The basic idea is as follows. First, k points $\{x_1, x_2, ..., x_k\}$ that satisfy s are selected in random. Second, we check whether each of these points satisfies any of the existing queries. It is concluded that s is covered if all the selected points satisfy the set of existing queries, and otherwise not covered. Thus, the probability of erronously concluding that a query is covered is upper-bounded by $(1-p_w)^k$, where p_w is the probability that a random point x_i satisfying s also satisfies the set of existing queries. This error probability is quickly improved as k increases.

36.4.2 Subscription Merging/Pruning

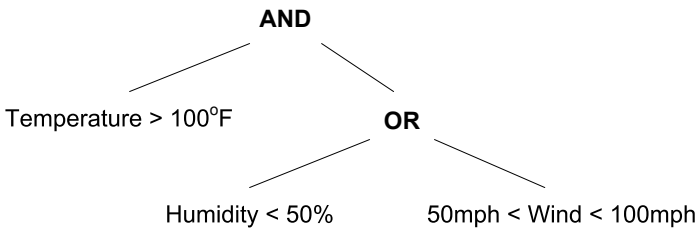
Another technique to reduce the number of query replicas in the network is via *subscription merging* [36, 25, 50, 17]. Given a set of queries S, we need to merge them to create a new set of queries S' such that (1) $\mid S \mid < \mid S' \mid$ and (2) the events satisfying S are the same as that satisfying S'. Instead of forwarding the queries in S, we forward the queries in S', thus reducing the amount of subscriptions in the network.

Although subscription merging is theoretically helpful, we cannot always find a perfect merging for a given set of queries. Even if such a merging exists, its algorithm can be computationally expensive. Indeed, [17] shows that the merging problem is NP-complete. Consequently, imperfect merging algorithms are suggested [36, 50, 17]. These algorithms aim to merge the queries in S into a new set of queries S' such that (1) $\mid S \mid < \mid S' \mid$ and (2) the events satisfying S are a *subset* of that satisfying S'.

One such an algorithm [50] clusters S into groups of similar queries and then finds a merging for each group. In another algorithm [36], the representation of a query is based on the concept of ordered Binary Decision Datagram (BDD) [10]. A BDD is a rooted, directed acyclic graph designed for easy manipulations of Boolean functions. A query is expressed as a BDD in which each node is a predicate of the query and a solid (dashed) link from a node means that the corresponding predicate is satisfied (unsatisfied). For example, Figure 36.3(a) shows a BDD for the query *{('temperature' $>$ 100) AND ('humidity' $<$ 50)} OR {('temperature' $>$ 100) AND ('humidity' $\geq$ 50) AND ('wind speed' $>$ 50) AND ('wind speed' $<$ 100)}*. Given an event, it traverses the BDD and if terminal node 1 is reached, it is concluded that the event satisfies the query. To address a large number of queries, thus a large number of BDDs, a modified version of BDD called MBD is introduced. A MBD



(a) A query as a binary decision diagram



(b) A query as a boolean tree

Figure 36.3 Examples of query representations

is a forest of BDDs, each representing a query, with the property that if a predicate occurs in multiple queries only one common node is shared among these queries to represent the predicate; this predicate is evaluated only once. A MBD thus represents the merging of multiple queries. Winthin a MBD, two predicate nodes can also be merged to create a single node to represent the same condition, resulting in a simpler MBD. An event is evaluated on a MBD rather than on each individual query.

Imperfect merging has its tradeoff. At any node P where an imperfect merging of S into S' is performed, we have the problem that many false events, that satisfy S' but not S, could be returned to node P. This is unnecessary

traffic which can be costly for sensor networks. In addition, the computational and space costs to perform subscription merging (perfect or imperfect) may exceed that a sensor node can afford. Thus, subscription merging is recommended only when its scope is small.

Subscription pruning [7] is another strategy to reduce the query storage load in the network. Each query is represented as a tree [6] in which each inner node represents a Boolean operator (e.g., AND, OR) and each leaf being a single-attribute predicate (e.g., *('temperature' > 100)*, or *('humidity' < 20)*. For example, Figure 36.3(b) shows the Boolean tree representing the same query in Figure 36.3(a). The tree is then simplified by pruning off some predicates or by replacing them with simpler ones. Since the queries become simpler, data structures and algorithms for processing them require less memory and computation. On the other hand, similar to imperfect subscription merging, subscription pruning also results in notification of false events.

36.5 Event Matching

When a publication of some event reaches a node where the locally stored queries need to be evaluated against this event, simply testing every query and predicate may be computationally expensive for a sensor node, especially when there are numerous, complex queries and high volumes of events. Consequently, it has been proposed that the queries are organized into some data structure that enables faster than linear-time event matching.

The Matching Tree in [1] is such a data structure, where the matching time is sub-linear and the space complexity is linear. This tree allows incremental query updates (insert/delete) and is most suitable where events are published at a fast rate. Each tree node is a test on some of the attributes and each link eminating a node is labeled with a result of the correponding test. A link with label '*' means a "do not care" link. Each query corresponds to a leaf node and the path from the root to this leaf node consists of all the tests whose conjunction is equivalent to the query. For example, Figure 36.4 shows a simple matching tree storing three queries: sub_1*: (*$attr_1 = v_1$*) AND (*$attr_2 = v_2$*) AND (*$attr_3 = v_3$*)*, sub_2*: (*$attr_1 = v_1$*) AND (*$attr_3 = v'_3$*)*, and sub_3*: (*$attr_1 = v'_1$*) AND (*$attr_2 = v_2$*) AND (*$attr_3 = v_3$*)*. To find the queries matching an event, at each node starting at the root, the corresponding test is performed and a link to the next node is followed if its label matches the result of the test. This step is repeated at the next node. The leaves that are finally visited correspond to the queries that match the event. In the case where a query consists of equality tests on the attributes, the expected time to match a random event is

$O(n^{1-\lambda})$ where n is the number of queries and λ is a parameter dependent on the number and type of attributes (in some cases, $\lambda = 1/2$). The constants hidden behind the big-O notation are quite reasonable.

The matching tree structure follows the approach that the search for queries matching a given event starts from the attribute constraints derived from the full set of queries and moves through them consulting the attributes appearing in the event. The binary decision datagram (BDD) structure discussed in Section 36.4 also follows this approach. Alternatively, we can start with the attributes appearing in the event and move through them consulting the query constraints. An early method adopting this approach is the SIFT system [51], which is the basis for subsequently designed structures [21, 13]. SIFT is limited to strings only and the equality operator over strings. Le Subscribe [21] allows the integer type and its associated operators. [13] adds the prefix, suffix, and substring operators for strings and, especially, allows a query to be expressed as a disjunction of conjunctive clauses, not just a single conjunctive clause. The matching algorithm in this work is based on a counting algorithm also used in [21, 51] and when tested on a 950Mhz computer for a 10-attribute event and 20 queries consisting of 25 conjunctions could find the matching queries in 3 milliseconds. It took 48 bytes of storage for each elementary predicate. This algorithm is therefore simple and efficient enough to be implemented on a sensor node.

Another strategy for building an efficient data structure is based on the query covering relationship. It is observed that if an event x matches query s_1 (denoted by $x \in s_1$) and if we already know that $s_1 \subset s_2$ for some query s_2, then it must be true that $x \in s_2$ and we need not check whether x matches s_2. Based on this observation, we should build a data structure that captures the covering relationship among subscriptions. This is similar to the problem of *Orthogonal Range Search* in Computational Geometry, for which several structures exist such as the kd-tree [5] and the layered range tree [19]. Using the kd-tree, the complexities would be $O(n^{1-1/d})$ for time and $O(n)$ for storage, while that using the layered range tree would be $O(log^d n)$ for time and $O(nlog^{d-1}n)$ for storage. If one of these two structures must be used, due to the high storage cost of the layered range tree, the kd-tree should be a better choice for sensor networks.

The random projection approach in [48], which was discussed earlier in Section 36.4.1, can be used to expedite the matching procedure. Using a random projection from d dimensions to k dimensions (d is the number of event attributes, k is some small constant), a query is mapped to a k-dimensional rectangle. To check whether an event x matches a query s (center u, radius

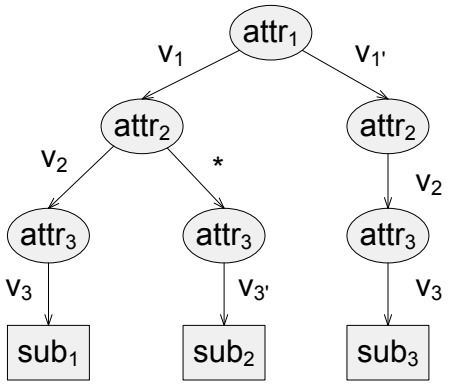


Figure 36.4 Example of a matching tree

r), a quick check can be to verify whether the projection x' of x in the k-dimensional space is inside the k-dimensional rectangular projection s' of s. If $x' \notin s'$, we can immeditately ignore s; otherwise, we perform the check whether $x \in s$ as usual. The first check is performed in a k-dimensional space, thus much quicker than the second check which is in a d-dimensional space. A nice property of this method is that the number of queries that can be ignored after the first check increases quickly if a larger value for k is chosen.

36.6 Middleware Development

It is important to have a middleware component that can be integrated on top of a sensor network to allow for easy deployment of publish/subscribe applications. An application developer should know just how to call the publish/subscribe functions, not having to worry about the complexity of the underlying network and the implementation details of the publish/subscribe mechanisms. There have been various approaches toward providing middleware services in sensor networks, and according to the categorization in [30], these approaches can be placed in one of the following groups: (1) database-inspired approaches (TinyDB [37], COUGAR [8], SINA [42]); (2) tuple space approaches (TinyLIME [11]); (3) event-based approaches (Mires [44], MQTT-S [33], TinyCOPS [28]; and (4) service discovery based ap-

proaches (MiLAN [29]). Publish/subscribe middleware belongs to the group of event-based approaches.

Mires [44] is such a publish/subscribe middleware design, which has been implemented using nesC on top of TinyOS 1.x. Mires supports *subject-based* publish/subscribe applications and provides an architecture that allows sensor nodes to advertise the subject of sensor data they can provide, user applications to select subjects of interest from the advertised services, and sensor nodes to publish their data to the corresponding subscribers. The subscription and notification protocols integrated in this middleware are similar to that of Directed Fusion. Mires also offers data aggregation services for some common aggregation functions, such as min, max, and average, to help reduce the event traffic in the network.

Another middleware design is the MQTT-S architecture by IBM [33]. MQTT-S is an extension of MQTT originally developed for telemetry applications using constrained devices[1]. This architecture is applicable to *subject-based* publish/subscribe applications that run in a network integrating multiple wireless sensor networks (WSN). Its aim is to hide end-point details: an application running on either the backbone network or inside a WSN does not know whether the data is coming from a device in a WSN or the backbone network. A query can be submitted anywhere in the global network subscribing to events that may belong to one or more participating WSNs. Broker nodes are placed in the backbone network to provide a publish/subscribe service to the nodes in the backbone network; the broker nodes run the original MQTT middleware. MQTT-S is used to provide a publish/subscribe service within a single WSN, with two main entities: MQTT-S Client running on a sensor node and MQTT-S Gateway running on a gateway node connecting its WSN to the global network. An MQTT-S Client contains both a publisher and a subscriber, thus allowing sensor nodes to not only publish their data, but also receive e.g. control information sent by nodes residing in the global network. The main function of the gateway is to translate between MQTT-S and MQTT protocols. This architecture is illustrated in Figure 36.5. MQTT-S provides a method to search for a local gateway and allows for multiple gateways used per WSN, thus increasing the reliability of the publish/subscribe system under deployment. MQTT-S has been implemented in a testbed which comprises two wireless sensor networks of different types, a ZigBee-based and a TinyOS-based one. Both implementations are lightweight with about 12kB of code. The testbed devices each have only 64kB of program memory

[1] http://mqtt.org

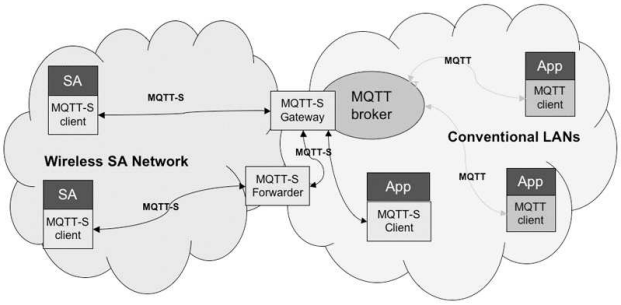


Figure 36.5 The MQTT-S middleware architecture

available. The MQTT-S Gateway is written in Java and uses the Java Communications API to communicate with the gateway devices over the serial port. The Gateway connects to a broker using the MQTT protocol. Nodes residing on the TinyOS-based network can communicate with nodes on the ZigBee network via the broker.

Mires and MQTT-S focus on architectural and networking issues and provide support for simple subject-based subscriptions and publications. The middleware development in [52] is aimed for a rich expressiveness of query and event description. A set of filtering operators is proposed to support not only standard comparison and string operators, but also allow for spatiotemporal constraints. For example, a query can be subscribed to receive the average temperature at a specific location during a future period of time. A problem with this middleware development is due to the complexity involved in the implementation of the subscription and notification protocols. Indeed, a complex timestamping scheme is needed to support the temporal operators and, further, the spatial operators requires location information which is not always available for every sensor network [30].

The aforementioned middleware designs tightly integrate filtering, routing and forwarding mechanisms resulting in more optimized, but less flexible solutions. TinyCOPS [28] is aimed to be a unified middleware architecture enabling *content-based* publish/subscribe applications, not just subject-based, that gives the application developer a wide range of orthogonal choices about the communication protocol components to use for subscription and notification, the supported data attributes, and a set of service extension components. This allows the adaptation of the publish/subscribe service to the specific needs of the application to deploy. TinyCOPS also introduces the

concept of *metadata* included in the description of each query to influence the communication and sensing process. For example, the metadata can specify the expiration time for a given query or to request a sampling rate events should be sent to the subscriber. TinyCOPS has been implemented to run on top of TinyOS 2.0, in which two communication protocols are integrated (broadcast-based or gossip-based).

36.7 Concluding Remarks

The publish/subscribe paradigm represents a large class of applications in sensor networks as sensors are designed mainly to detect and notify upon events of interests. This important paradigm allows a user to subscribe in advance a query specifying the early warnings of a wildfire, so that any event matching these warnings when detected by a sensor can be published to the network quickly to notify the subscriber. In industrial applications, we can provide a more secure working environment by deploying a sensor network that warns workers upon detection of dangerous events.

Many publish/subscribe techniques for sensor networks have been inspired by that for traditional Internet-based networks. For example, some gossip-based routing schemes, query aggregation schemes, and event matching algorithms that have been designed for the Internet can also be used in sensor networks, as we discussed earlier in the chapter. There, however, remains much room for future research. For a large-scale sensor network where broadcast-based and gossip-based routing approaches may not be the best fit, the routing design should be driven by the content of the message being routed so as to limit the scope of propagation. In networks where location information is available, it should be a main factor in the routing design. In other networks without location information, the naming-based approach seems a promising direction, the challenge, however, being how to maintain the naming structure efficiently under network dynamics. Since sensor networks may be of different types (small vs. large, location-aware vs. location-unaware, static vs. dynamic) and the application to deploy may also have its own characteristic (low vs. high query rate, low vs. high event rate, subject-based vs. content-based, etc.), it is difficult to choose a publish/subscribe design that works well in every case. Thus, rather than trying to find a universally “perfect" design, it would be better to categorize the networks and applications into similarity-based groups and design the “best" technique for each group.

It is then natural for the next step to be developing a publish/subscribe middleware package that provides a set of common services to most publish/subscribe network/applications no matter their categories, and another set of services each customized toward a specific category. This middleware should provide convenient tools for the middleware designer to add new service components to the existing architecture, such as a new language for query and event description and a new implementation for routing, data aggregation, or an event matching algorithm. It should also give the application developer freedom and a convenient API to choose the publish/subscribe service configuration that is best for the context of the deployment. Middleware development for publish/subscribe applications in sensor networks remains ad hoc and isolated. It should be a high-priority item in the future research towards publish/subscribe services in sensor networks.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 53–61. ACM Press, 1999.

[2] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science*, 181(1):3–15, 1997.

[3] C. Avin and B. Krishnamachari. The power of choice in random walks: An empirical study. *Comput. Networks*, 52(1):44–60, 2008.

[4] M. Avvenuti, A. Vecchio, and G. Turi. A cross-layer approach for publish/subscribe in mobile ad hoc networks. In *MATA*, volume 3744 of *Lecture Notes in Computer Science*, pages 203–214. Springer, 2005.

[5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[6] S. Bittner and A. Hinze. On the benefits of non-canonical filtering in publish/subscribe systems. In *ICDCSW '05: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05)*, pages 451–457, Washington, DC, USA, 2005. IEEE Computer Society.

[7] S. Bittner and A. Hinze. Pruning subscriptions in distributed publish/subscribe systems. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pages 197–206, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[8] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *2nd International Conference on Mobile Data Management (MDM)*, pages 3–14, 2001.

[9] D. Braginsky and D. Estrin. Rumor routing algorthim for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31. ACM, 2002.

[10] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

[11] M. G. C. Curino, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. Tinylime: Bridging mobile and sensor networks through middleware. In *3rd IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 61–72, March 2005.

[12] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[13] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *ACM SIGCOMM*, pages 163–174, 2003.

[14] R. Chand and P. Felber. Xnet: A reliable content-based publish/subscribe system. In *SRDS '04: Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 264–273, Washington, DC, USA, 2004. IEEE Computer Society.

[15] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco. Socially-aware routing for publish-subscribe middleware in delay-tolerant mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 26(5), 2008.

[16] P. Costa, G. P. Picco, and S. Rossetto. Publish-Subscribe on Sensor Networks: A Semi-probabilistic Approach. In *Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS05)*, Washington DC, USA, November 2005.

[17] A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Query merging: Improving query subscription processing in a multicast environment. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):174–191, 2003.

[18] G. Cugola and H.-A. Jacobsen. Using publish/subscribe middleware for mobile systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):25–33, 2002.

[19] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[20] P. T. Eugster, R. G. P. A. Felber, and A.-M. Kernmarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114âĂŞ131, 2003.

[21] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 115–126, New York, NY, USA, 2001. ACM Press.

[22] Q. Fang, J. Gao, and L. J. Guibas. Landmark-based information storage and retrieval in sensor networks. In *IEEE INFOCOM*, 2006.

[23] U. Farooq, E. W. Parsons, and S. Majumdar. Performance of publish/subscribe middleware in mobile wireless networks. *SIGSOFT Softw. Eng. Notes*, 29(1):278–289, 2004.

[24] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The padres distributed publish/subscribe system. In *FIW*, pages 12–30, 2005.

[25] L. Fiege, M. Cilia, G. Muhl, and A. Buchmann. Publish-subscribe grows up: Support for management, visibility control, and heterogeneity. *IEEE Internet Computing*, 10(1):48–55, 2006.

[26] D. Frey and G.-C. Roman. Context-aware publish subscribe in mobile ad hoc networks. In *International Conference on Coordination Models and Languages (COORDINATION 2007)*, pages 37–55, 2007.

[27] S. Funke, L. J. Guibas, A. Nguyen, and Y. Wang. Distance-sensitive information brokerage in sensor networks. In *DCOSS*, pages 234–251, 2006.

[28] J.-H. Hauer, V. Handziski, A. Kaopke, A. Willig, and A. Wolisz. A component framework for content-based publish/subscribe in sensor networks. In *European Workshop on Wireless Sensor Networks (EWSN)*, Bologna, Italy, January 2008.

[29] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, August 2004.

[30] K. Henricksen and R. Robinson. A survey of middleware for sensor networks: State of the art and future directions. In *ACM Workshop on Middleware for Sensor Networks*, Melbourne, Australia, 2006.

[31] Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 122–140, London, UK, 2003. Springer-Verlag.

[32] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wirel. Netw.*, 10(6):643–652, 2004.

[33] U. Hunkeler, H.-L. Truong, and A. Stanford-Clark. MQTT-S: A publish/subscribe protocol for wireless sensor networks. In *Workshop on Information Assurance for Middleware Communications (IAMCOM '08)*, Bangalore, India, January 2008.

[34] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000.

[35] B. Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM Press.

[36] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 447–457, Washington, DC, USA, 2005. IEEE Computer Society.

[37] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[38] A. M. Ouksel, O. Jurca, I. Podnar, and K. Aberer. Efficient probabilistic subsumption checking for content-based publish/subscribe systems. In *ACM/IFIP/USENIX 7th International Middleware Conference*, pages 121–140, 2006.

[39] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl.*, 8(4):427–442, 2003.

[40] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. In *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 286–297, New York, NY, USA, 2006. ACM.

[41] J. H. Schönherr, H. Parzyjegla, and G. Mühl. Clustered publish/subscribe in wireless actuator and sensor networks. In *MPAC '08: Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, pages 60–65, New York, NY, USA, 2008. ACM.

[42] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8(4):52âĂŞ–59, August 2001.

[43] Z. Shen and S. Tirthapura. Approximate covering detection among content-based subscriptions using space filling curves. In *IEEE International Conference on Distributed Computing Systems*, Toronto, Canada, June 2007.

[44] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal Ubiquitous Comput.*, 10(1):37–44, 2005.

[45] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 49–60, New York, NY, USA, 2007. ACM.

[46] D. A. Tran and T. Nguyen. Subscription covering detection in publish/subscribe systems based on random projections. In *Proceedings of IEEE Int'l Conference on Collaborative Computing (COLLABORATECOM 2007)*, White Plains, NY, October 2007. IEEE Press.

[47] D. A. Tran and T. Nguyen. Localization in wireless sensor networks based on support vector machines. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):981–994, July 2008.

[48] D. A. Tran and C. Pham. Cost-effective multidimensional publish/subscribe services in sensor networks. In *IEEE Workshop on Localized Communications Algorithms and Networks - IEEE Conference on Mobile Ad hoc and Sensor Systems (MASS 2008)*, Atlanta, GA, September 2008. IEEE Press.

[49] D. A. Tran and C. Pham. A content-guided publish/subscribe mechanism for sensor networks without location information. *Journal on Computer Communications (COMCOM)*, 33(13):1515–1523, August 2010.

[50] Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, and P. Larson. Summary-based routing for content-based event distribution networks. *SIGCOMM Comput. Commun. Rev.*, 34(5):59–74, 2004.

[51] T. W. Yan and H. Garcia-Molina. The sift information dissemination system. *ACM Trans. Database Syst.*, 24(4):529–565, 1999.

[52] E. Yoneki and J. Bacon. Unified semantics for event correlation over time and space in hybrid network environments. In *IFIP International Conference on Cooperative Information Systems (CoopIS)*, pages 366–384, 2005.

[53] Q. Yuan and J. Wu. Drip: A dynamic voronoi regions-based publish/subscribe protocol in mobile networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 2110–2118, 2008.

Prasad, Buford, and Gurbani (Eds.), Advances in Next Generation Services and Service Architectures, 1–24.