Publish/Subscribe Tree Construction in Wireless Ad-Hoc Networks

Yongqiang Huang and Hector Garcia-Molina

Stanford University, Stanford, CA 94305
{yhuang, hector}@cs.stanford.edu

Abstract. Wireless ad-hoc publish/subscribe systems combine a publish/subscribe mechanism with wireless ad-hoc networking. The combination, although very attractive, has not been studied extensively in the literature. This paper addresses an important problem of such systems: how to construct an optimal publish/subscribe tree for routing information from the source to all interested recipients. First we precisely define the optimality of a publish/subscribe tree by developing a metric to evaluate its "efficiency." The optimality metric takes into account both the goal of a publish/subscribe system (i.e., to route a set of events), and the characteristics of an ad-hoc network (for example, devices are resource limited). We propose a greedy algorithm, SHOPPARENT, which builds the publish/subscribe tree in a fully distributed fashion. A key feature is that this algorithm can be "subscription-aware", allowing it to use publication/subscription information in order to find a better outcome. Our simulations show that SHOPPARENT's performance is within 15% of optimal under normal configurations. We also study the effect of geographically localized subscriptions.

1 Introduction

A *publish/subscribe* system connects information providers with consumers by delivering *events* from sources to interested users. A user expresses interest in receiving certain types of events by submitting a predicate defined on the event contents. The predicate is called the user's *subscription*. When a new event is generated and *published* to the system, the publish/subscribe infrastructure is responsible for checking the event against all current subscriptions and delivering it efficiently and reliably to all users whose subscriptions match the event.

Many problems related to publish/subscribe have been tackled and solved. However, almost all of the research so far has concentrated on publish/subscribe systems in a fixed network. With increasing popularity of wireless handheld devices, there is a pressing need to extend publish/subscribe to a wireless environment [1,2,3,4]. In this paper, we study publish/subscribe systems in a wireless ad-hoc network. Such a network is formed by wireless devices communicating without the benefit of a fixed network infrastructure, either because the infrastructure has been wiped out by a natural disaster, or because it is impractical to build one. As an example, in a military battlefield, thousands of wireless and mobile sensors such as satellites and equipment sensors report all kinds of information ranging from the location of enemy troops to whether the engine of a tank has overheated. There are also many parties interested in receiving certain types of information. An individual soldier may need to know the location of the nearest enemy troops, or whenever a missile has been fired. The above scenario requires the deployment of a highly scalable and efficient communication infrastructure, for which publish/subscribe is an ideal candidate.

In a wireless ad-hoc publish/subscribe system, nodes cooperate to deliver events from their publishers to interested subscribers. In this paper, we focus on constructing a *publish/subscribe tree* for such a system. A publish/subscribe tree is a node hierarchy along which new publications are sent in order to reach their subscribers. A tree construction algorithm produces such a tree, given information such as how the nodes are interconnected, and what each node has subscribed to.

A publish/subscribe tree is similar in function to a multicast tree in traditional multicast networks. Existing multicast protocols often aim to produce an "optimal" multicast tree which seeks to optimize one of a few possible metrics. For example, in order to minimize the total number of nodes participating in multicast, some protocols generate (or heuristically approximate) Steiner trees. In this paper we propose a new optimality criterion suitable for wireless publish/subscribe systems. Our metric measures how "efficiently" a publish/subscribe tree can transmit all publications to the interested nodes. Efficiency is important because wireless devices in ad-hoc networks typically have very limited resources, such as battery power. Hence it is desirable to accomplish the same goal with minimum work. Furthermore, our metric is tailored to a publish/subscribe system because it takes into account information such as user subscriptions and events, which are not applicable in regular multicast.

We present tree construction algorithms to produce efficient publish/subscribe trees based on the above metric. Our algorithms exploit knowledge of expected traffic on the publish/subscribe tree. For example, if a new node can connect to the tree in say two ways, it will select the connection that generates less new traffic for its parent, given the current subscriptions. As we will see, this "intelligent" tree construction can yield significant gains.

In addition to the algorithms, our other contribution is their evaluation. In particular, we devise models to simulate subscription and publication so that we can compare the performance of various algorithms.

We define the problem and our optimality metric in Section 2. We then give our algorithms for tree construction (Section 3). Finally, we discuss our evaluation model (Section 4) and present our results (Section 5).

2 Framework

Figure 1 illustrates a wireless ad-hoc publish/subscribe system. It consists of N wireless nodes, each identified by a globally unique id. The nodes communicate with each other wirelessly using radio, and cooperate to send, relay, and receive *events* (i.e., publications).

We define the *connectivity graph* G of a system as the graph whose vertices are the wireless nodes, and where an edge exists between two vertices if the two corresponding nodes are "neighbors." Two nodes are neighbors if they can talk to each other directly via radio, i.e., if they are a single hop away. See Figure 1 for an example G. Note that, as in [5], the connectivity graph is assumed to be undirected, precluding the situation where one node can hear another, but not vice versa.

Each node i has a subscription s_i, called its *inherent subscription*, which is expressed as a predicate defined on the contents of an event. A node i is interested in an

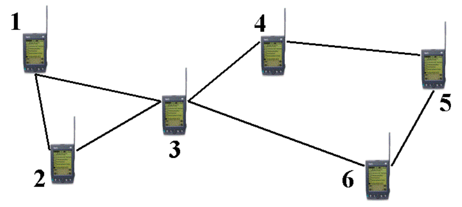


Fig. 1: A wireless ad-hoc publish/subscribe system. The lines form the connectivity graph G.

event e if and only if $s_i(e) = true$. Naturally, the inherent subscription is determined by the particular applications that are running on that node. Note that we do not make any assumptions about an event's content.

Of all the nodes in the system, one is designated as *root* of the publish/subscribe mechanism. Without loss of generality, we assume that the root is always labeled node 1. We assume that only the root node can publish new events. That is, a new event has to be generated at node 1, and then forwarded to other interested nodes. Although this seems like a limiting assumption, our algorithms can easily be modified when multiple nodes are allowed to publish. For example, we can construct one tree per potential publisher (per-source trees in multicast lingo). Alternatively, we can require any new event to be passed to the root first via unicast. However, a better method is to forward the event towards the root along the publish/subscribe tree itself, allowing shortcutting to tree branches along the way. The user is referred to [6] for a relevant discussion in the context of shared multicast trees.

Although neighbors of the root node can receive newly published events directly from the root, other nodes will have to rely on the help of intermediate nodes to relay events. For example, in Figure 1, node 3 needs to forward those events of interest to node 4. We use the term *publish/subscribe tree* (PST) to describe the tree rooted at node 1 that is formed by paths traveled by the events to reach their destinations.[1] We observe that the PST is a spanning tree of the connectivity graph G.[2]

We assume that every node in the system participates in the publish/subscribe protocol.[3] In particular, nodes that are not interested in any event will simply have an empty subscription. We believe that the above is a reasonable assumption if publish/subscribe is used as a fundamental underlying communication mechanism for such a system. The assumption also implies that our algorithms will not rely on a separate ad-hoc unicast protocol to work. However, if the assumption does not hold, additional steps may be necessary in the algorithms, which will be considered in extensions of this work. For

[1] Similar to a multicast tree, the PST is a tree and not a graph because we force the events to always travel the same route between the root node and any subscriber node.

[2] The PST is a spanning tree and not a Steiner tree because, as explained next, all nodes participate in the publish/subscribe protocol.

[3] This is a fundamentally different and much less limiting assumption than assuming that all nodes are part of a particular multicast tree in traditional multicasting. While there may be many concurrent active multicast groups at any time and only a small portion of nodes are interested in any particular one, we envision a single publish/subscribe session where different node interests are represented by different subscriptions.

example, new subscribers may need to perform expanding ring searches to look for the nearest node that is also part of publish/subscribe.

Suppose node i is the parent of another node j in a PST. Then any event that j subscribes to will need to pass through i first. We define node i's *effective subscription* (S_i) as the "combined" subscription of itself and all its children. Specifically, let s'_i denote i's *proxied subscription*, which is the disjunction of the (effective) subscriptions of all i's children. Then S_i is simply the disjunction of i's inherent subscription (s_i) and proxied subscription (s'_i). In other words, $S_i(e) = s_i(e) \vee s'_i(e)$, where $s'_i(e) = S_{i1}(e) \vee S_{i2}(e) \ldots$ for all of i's children $i1, i2, \ldots$

When a new event e is published at the root node, the wireless nodes run a distributed *publish/subscribe protocol* to forward e along the branches of the PST until it reaches all nodes that have subscribed to it. Under the protocol, every node i listens for new events broadcast by its parent node in the tree. Then, depending on an event's content, some or all of a set of actions can be performed on it, as shown in Table 1. In particular, receiving an event means capturing it in the air and transferring it to an internal buffer, logging it, etc. Processing implies that an event has matched i's inherent subscription and will be displayed to the user in a pop-up alert window, for example. Finally, forwarding an event involves calculating the set of children nodes that will be interested in the event and re-broadcasting it in the air for them. For example, the first row of Table 1 shows that events satisfying $s_i(e) \wedge \neg s'_i(e)$ are of interest to node i only, and not to its children. Consequently, these events will be received and processed, but not forwarded.

Satisfies	**Of interest to**	**Recv**	**Proc**	**Fwd**
$s_i \wedge \neg s'_i$	i only	$\surd$	$\surd$	X
$\neg s_i \wedge s'_i$	i's children only	$\surd$	X	$\surd$
$s_i \wedge s'_i$	both i & children	$\surd$	$\surd$	$\surd$
$\neg S_i$	neither	X	X	X

Table 1: Classification of new events seen by node i and whether they will be "received", "processed", and/or "forwarded".

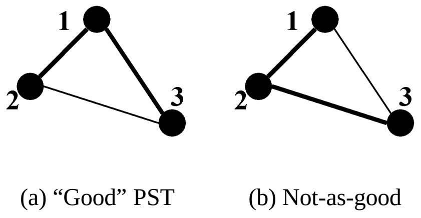


Fig. 2: Two PSTs of the same connectivity graph. Lines represent the connectivity graph, while thick lines constitute a PST.

We do not focus on reliability in this paper. Instead, we develop best-effort algorithms where nodes may occasionally miss events, especially when they move around geographically. If desired, reliability can be added to our algorithms via logging and retries, etc.[7] However, we suspect that such a guaranteed delivery system may be costly in a dynamic wireless ad-hoc environment, and we leave a detailed study to future work.

2.1 PST evaluation metric

There usually can be many possible PSTs in any given system (actually, as many as there are spanning trees in the connectivity graph G). Some trees can be "better" than others, however, as the following example shows.

Example 1. Figure 2 shows a simple system with three nodes. Assume both nodes 2 and 3 can receive events directly from the root, node 1. If they both do, the resulting PST is given by the thick lines in Figure 2(a). If, on the other hand, node 3 chooses to rely on node 2 to forward events, we have the PST in Figure 2(b). Intuitively, the PST in Figure 2(a) is "better" than 2(b) since the extra work by node 2 to forward events is avoided.

In order to meaningfully compare different PSTs, we define a metric called the *cost* of a publish/subscribe tree. A PST's cost with respect to a set E of events, denoted $C(E)$, is the total amount of work performed by all tree nodes in order to publish E. Cost measures the "efficiency" of a PST, as a tree with a lower cost will need less overall work to deliver all the events in E. We believe that this is a useful and realistic criterion due to severely limited resources in a wireless ad-hoc environment. We have $C(E) = \sum_i C_i(E)$, where $C_i(E)$ is the amount of work node i has to perform to publish E, called the cost of i.

Let us assume that receiving an event constitutes r units of work, and processing and forwarding need p and f units, respectively. Let $E(\alpha)$ represent the subset of E satisfying the predicate α, and let $\mathit{\Phi} E(\alpha)$ denote the number of events in that subset. According to Table 1, we get $C_i(E) = (r + p) \cdot \mathit{\Phi} E(s_i \wedge \neg s'_i) + (r + f) \cdot \mathit{\Phi} E(\neg s_i \wedge s'_i) + (r + p + f) \cdot \mathit{\Phi} E(s_i \wedge s'_i)$.

Note that, similar to some previous work on ad-hoc multicast/broadcast [5], we have assumed the cost (f) to forward an event to be constant, regardless of how many of i's children want this event. This is a reasonable assumption because of the broadcast nature of radio. With one broadcast operation, a node can normally send a message to all its neighbors. In our case, specifically, when node i broadcasts an event in the air, all nodes within i's radio range will be able to hear the transmission. However, only the intended recipients of the event will accept and receive it; others will simply ignore it. Furthermore, since we concentrate on best-effort algorithms as noted before, we do not assume the broadcast primitive to be reliable, which may be expensive to enforce.

The cost calculation not only depends on E, the set of events being published, but also on the connectivity graph and user subscriptions. We will propose models to simulation these factors in Section 4. In general, if tree T_1 has a lower cost than T_2, T_1 will incur less overall work given the user subscriptions and event distribution that are used to calculate the cost.

Although the cost metric can be used directly to compare two PSTs, we observe that a component in the cost formula is unchanged in all possible trees. Specifically, regardless of i's children, the work needed to receive and process the sets $E(s_i \wedge \neg s'_i)$ and $E(s_i \wedge s'_i)$ is always needed. Therefore, to highlight the differences between two PSTs and to simplify calculation, we next define another derivative metric, the overhead of a PST.

Node i's *overhead* with respect to E, denoted $O_i(E)$, is the additional amount of work that i needs to perform on behalf of its children in a PST (i.e., extra work in addition to what i would always need to do even without any children). In other words, the overhead is the cost of a node minus the amount of work it needs to do for itself. The *overhead* of a publish/subscribe tree, $O(E)$, is simply the sum of overheads of all nodes, namely, $O(E) = \sum_i O_i(E)$.

Based on above analysis, we get $O_i(E) = (r + f) \cdot \mathit{\Phi} E(\neg s_i \wedge s'_i) + f \cdot \mathit{\Phi} E(s_i \wedge s'_i)$. Without loss of generality, we assume that f corresponds to one unit of work, i.e., $f =$

1. To simply calculation, we also assume that $r = 1$. That is, it takes the same amount of work to receive an event as to transmit one. This is reasonable if "work" is measured in terms of processing time, for example. On the other hand, since radio transmission often consumes much more energy than reception, we will also study the impact of letting $r \ll f$ in Section 5. Given $r = f = 1$, we have, $O_i(E) = 2\Phi E(\neg s_i \wedge s'_i) + \Phi E(s_i \wedge s'_i)$.

Example 2. Referring back to Figure 2, we assume $E = \{e_1, e_2\}$, where e_1 matches node 2's subscription, while e_2 matches node 3's. For the PST in Figure 2(a), we have $O_2(E) = 2\Phi E(\neg s_2 \wedge s'_2) + \Phi E(s_2 \wedge s'_2) = 2 \cdot 0 + 0 = 0$ since node 2 has no children. Likewise, $O_3(E) = 0$. Also, $O_1(E) = 2\Phi E(\neg s_1 \wedge s'_1) + \Phi E(s_1 \wedge s'_1) = 2 \cdot 2 + 0 = 4$. Hence, $O(E) = \sum_i O_i(E) = 4 + 0 + 0 = 4$.

In Figure 2(b), however, node 3 is node 2's child. Thus node 2's effective subscription includes that of 3. We get $O_2(E) = 2\Phi E(\neg s_2 \wedge s'_2) + \Phi E(s_2 \wedge s'_2) = 2 \cdot 1 + 0 = 2$. As before, we also have $O_1(E) = 4$ and $O_3(E) = 0$. Overall, $O(E) = \sum_i O_i(E) = 4 + 2 + 0 = 6$. Hence, the left PST is indeed "better" according to our metric.

3 Algorithms

A *tree construction algorithm* is the protocol run by the nodes of a wireless publish/subscribe system in order to determine which PST to use. The algorithm aims to produce an *optimal PST*, namely, to minimize the overhead of the resulting PST based on the metric presented earlier. In this section, we give several such algorithms.

Our first algorithm, OPT, is a centralized algorithm that is guaranteed to be optimal. Specifically, Algorithm OPT runs on a single node with global knowledge about the system. That is, a controller node gathers all inputs needed, including the entire connectivity graph G and user subscription information, before it calculates the optimal PST, and distributes the tree structure back to other nodes.

When finding the optimal PST, OPT exhaustively searches through all spanning trees of the connectivity graph G, and selects the one with the smallest overhead. Since a PST must be a spanning tree of G, OPT guarantees optimality of the resulting PST. Note that an optimal PST differs from a minimum spanning tree of G in the graph theoretical sense, because the "weight" of an edge is not fixed, but is dependent on what events flow through it, which in turn depends on the shape of other parts of the tree. Hence one cannot use the more efficient minimum spanning tree algorithms in this step. Due to space limitations, we will omit further details of this algorithm.

Because of its exhaustive and centralized nature, OPT is very inefficient, which makes it particularly unsuitable to a wireless ad-hoc operating environment. Nevertheless, we have included it here to establish a basis for comparison, because OPT gives the best case of what any tree construction algorithm can achieve. Next, we follow with a distributed algorithm, which, although not optimal, is efficient and practical.

3.1 Algorithm ShopParent

Algorithm ShopParent is a greedy distributed algorithm that avoids the disadvantages of the centralized OPT. The PST is constructed by each node running one instance of the algorithm and making its own decision about which other node to select as its parent. The parent is chosen from all the node's neighbors according to criteria aimed to minimize overhead. No node needs global knowledge about the system, only information (e.g., subscriptions) about itself and its parent or children.

When a new node i comes into an existing system, it broadcasts a PARENT-PROBE to all its immediate neighbors, looking for the "best" potential parent. If a recipient of the probe, say node k, currently has a route to the root publisher, k will reply with a PARENT-ADVERTISE message. The reply contains information which allows node i to calculate a *routing metric*, which we will discuss promptly. For example, in addition to k's id, PARENT-ADVERTISE may contain k's current route to the root (to avoid a routing loop), and its effective subscription etc.

From all the PARENT-ADVERTISE replies it receives, node i selects the one, say node k's, that gives the smallest routing metric, and connects to that node as its parent in the publish/subscribe tree. Node k, in turn, may need to notify its own parent (or even "shop around" for a new one) if S_k has been expanded by the joining of node i. On the other hand, if node i does not receive any PARENT-ADVERTISE from its PARENT-PROBE, that means it is currently partitioned from the publishing tree, and it will periodically re-attempt to connect later.

To deal with connectivity changes in the network as a result of nodes' moving around or failing, for example, each node i periodically broadcasts its PARENT-ADVERTISE message. At the same time, it constantly listens for these messages from neighboring nodes, and calculates the routing metric and compares against that of the current parent. This periodic "beaconing" serves two purposes. First, if a new potential parent comes into range that offers a "better" route to the root, node i can take advantage of it by initiating a procedure to switch its parent in the PST to the new node. The switching may involve notifying the new and old parents, and possibly i's descendants. Second, if a child does not receive PARENT-ADVERTISE from its current parent during some timeout period, it expires the parent because the parent must have failed or gone out of range. After expiration, the node probes for a new parent, essentially as if it had just joined the network.

We introduce three variations of algorithm SHOPPARENT, each using a different routing metric when "shopping" for the best parent. The first, called SP-NHOP, uses H_k as the routing metric for node k. H_k is k's distance to the root node, measured in number of hops. This algorithm is designed to mimic some existing distributed network routing protocols, and results in a shortest-path spanning tree of the connectivity graph. Note that for this to work, each node should keep track of its own hop count, which can be propagated and refreshed by PARENT-ADVERTISE messages.

The routing metric used by SP-OVHD is $O_k^{+i}(E) - O_k^{-i}(E)$, where $O_k^{+i}(E)$ is node k's overhead if k takes on i as one of its children, while $O_k^{-i}(E)$ is if it does not. The metric measures the increase in $O_k(E)$ if k becomes i's parent. Since $O_k(E)$ is part of the PST overhead, this metric indirectly estimates the impact of k's parenting i.

Note that in order to calculate the overhead metric above, the algorithm needs the "event distribution" of the set E of events to be published. Essentially, an event distribution allows one to calculate $\Phi E(\alpha)$, the expected number of events in E matching the subscription predicate α. This distribution can be computed fairly accurately in certain applications, e.g., if the events of interest to a user are generated by a sensor with a regular pattern. In situations where the precise distribution is not known a priori, however, we often find that the absolute numbers are not needed to run the algorithm correctly. Because the node only uses this information to choose among several alternative parents, it is usually sufficient to know the relative distribution of events in several categories. For example, even though the exact rate of new postings in each newsgroup varies over time, the relative frequencies of different newsgroups may well stay comparatively fixed, especially over longer periods of time. Thus, for example, the groups in `comp.os.linux` will always have more traffic than `comp.os.mach`. Finally, nodes can also project future event distributions using recorded history of past events. Section 5 studies the resilience of the algorithm to skew in the approximation.

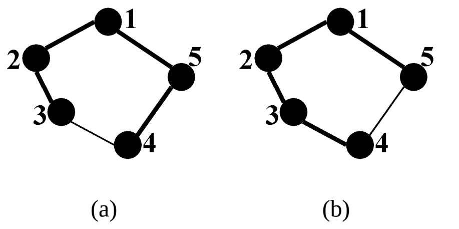


Fig. 3: Two PSTs of the same connectivity graph.

The last algorithm, SP-COMBO, simply uses the product of the two metrics used above ($(O_k^{+i}(E) - O_k^{-i}(E)) \cdot H_k$). Intuitively, SP-OVHD underestimates the impact on the overall PST overhead if k parents i, since not only k, but some of its ancestors as well, are likely going to be affected. SP-COMBO is intended as a better estimate by assuming that all H_k nodes on the path from k to the root are equally affected. Naturally, SP-COMBO may overestimate in some cases. Section 5 studies the actual performance of the three SHOPPARENT algorithms in detail.

Example 3. Figure 3 shows a simple system with 5 nodes. Node 4 is joining the system, and is trying to select either node 5 (Figure 3(a)) or node 3 (Figure 3(b)) as its parent. Let E contain 10 events, all of which match s_2 and s_4, while half of which (5 events) match s_5. Moreover, assume no events match s_1 or s_3.

When node 4 sends out its PARENT-PROBE, it will receive two replies, from nodes 3 and 5, respectively. Assume SP-COMBO is being used. The routing metric calculated from 3's PARENT-ADVERTISE will be $(O_3^{+4}(E) - O_3^{-4}(E)) \cdot H_3 = (20 - 0) \cdot 2 = 40$ because, for example, $O_3^{+4}(E) = 2\Phi E(\neg s_3 \wedge s_4) + \Phi E(s_3 \wedge s_4) = 2 \cdot 10 + 0 = 20$. On the other hand, $(O_5^{+4}(E) - O_5^{-4}(E)) \cdot H_5 = (15 - 0) \cdot 1 = 15$. Therefore, node 4 will eventually pick 5 as its parent, resulting in the PST of Figure 3(a). (Note that, in this particular instance, all three SHOPPARENT variants give the same outcome.) The reader can verify that this is indeed the best PST in this scenario.

SHOPPARENT is a greedy local algorithm because each node makes its own decision without global information. Omitted here due to space, examples can indeed be constructed to show that the resulting publish/subscribe tree may not always be optimal. Section 5 will investigate how close the algorithm comes to the global optimum.

For comparison purposes, we introduce another simple algorithm called RANDOM. RANDOM is a distributed algorithm similar to SHOPPARENT. However, each node randomly selects its parent from the pool of candidates. Effectively, the algorithm generates a random spanning tree of G as the PST. This behavior is similar in nature to techniques used in some existing ad-hoc multicast tree protocols [8] and fixed network publish/subscribe systems [9,10,11], where a new node joining the system often connects to a random existing member.

4 Evaluation Model

We use simulation to study the effectiveness of our algorithms. Recall that the algorithms need as inputs the connectivity graph G, each node's subscription and the distribution of the set E of events to be published. In this section, we present models to simulate these factors.

4.1 Connectivity graph

To obtain the connectivity graph G, we first place a total of N wireless nodes in a two-dimensional space. Instead of randomly generating an x- and a y-coordinate for each node, we use a clustered model, which we believe is more representative of a real wireless ad-hoc deployment. In particular, the nodes are partitioned into N_c clusters, and each cluster consists of N_n nodes, so that $N = N_c \cdot N_n$. Nodes belonging to the same cluster are expected to be geographically closer to each other in general.

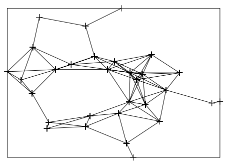
Each cluster has a cluster center, which is a point in the x-y plane. The set of all cluster centers are generated with respect to the origin in a polar coordinate system. The angle ρ of all the centers follows a uniform random distribution between 0 and 2π. The distance r to the origin follows a normal distribution with mean D_c.

Once we have the cluster centers, the node coordinates in each cluster are generated in a similar fashion. In particular, the nodes' ρ relative to their cluster center also follows a uniform distribution, while their r follows a normal distribution with mean D_n.

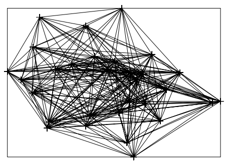
Finally, we obtain the connectivity graph G by picking a value for R, which denotes the transmission *radio range* of the wireless devices. A pair of nodes are neighbors (i.e., they are adjacent in G) if the distance between them in the x-y plane is less than R. We assume that R is the same for all wireless devices to ensure that G is undirected.

Our evaluation will assume that G is connected, since otherwise, some nodes will not be able to receive all necessary events simply because no route exists to the publisher. Consequently, we only use values of R large enough to guarantee the connectedness of G.

By varying R, we can obtain multiple connectivity graphs with differing degrees of "connectedness." A more connected graph implies a scenario where users are more clustered together, while a less connected one means they are farther apart. Figure 4 shows two G's obtained from the same set of nodes, but with different values for R.



(a) $R = 60$



(b) $R = 140$

Fig. 4: A bigger R results in a more"connected" G.

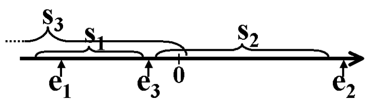


Fig. 5: The Number Intervals subscription model, with 3 sample subscriptions (intervals) and 3 events (points). As shown, e_1 matches s_1 and s_3, e_2 matches nothing, while e_3 matches s_3.

4.2 Subscription model

We next present models to simulate each node's inherent subscription, s_i. The subscription model should allow for diverse interests for different users. Otherwise, if every user subscribes to the same things, the system essentially performs broadcast flooding. On the other hand, the model should also allow for overlap of interests between different users. Just like in the real world, there is often some sharing of interest between users. Note that we are not trying to model specific applications. We want simple models that, with few parameters, capture the essential features of subscriptions. With such models, we can understand why a particular algorithm works well, or what is the nature of overhead.

Our first subscription model (Figure 5), called Number Intervals (NI), represents a user's subscription by an interval on the number axis, e.g., $[3.0, 4.0]$, or $[-100, \infty)$. An event is represented by a real number. An event e matches node i's subscription if and only if e's number falls within range s_i. For example, if $s_i = [3.0, 4.0]$, and a new event e_1 is published as the number 3.2, then e_1 matches. On the other hand, if $e_2 = 5.2$, then e_2 does not match.

The NI model can be mapped directly to many real world applications. For example, in a nuclear reactor temperature monitoring application, new events report the current temperature of the reactor. Interested users want to receive these temperature readings only when they fall within a certain range. For instance, an overheating warning system is interested in receiving new readings only when they are above 2000 degrees. In such case, the subscription interval of the warning system is $[2000, \infty)$.

Within the NI model, the actual interval of node i is generated by picking a center and a length. The lengths of the intervals are selected randomly according to a normal distribution. The NI model is further divided into several submodels depending on how the centers of s_i's are determined. In the first submodel, called *random-center* (NI-R), we pick a random number (within a certain range) as s_i's center. The *x-center* (NI-X) submodel uses node i's x-coordinate (in the plane used to generate G) as the center. The idea is to introduce some correlation between a user's interest and its geographical location, so nodes that are nearby subscribe to similar events. The impact of such "geographical correlation of subscriptions" will be inspected in detail in Section 5. Finally, the *modified x-center* submodel (NI-Xmod) adds a random offset to i's x-coordinate before using it as the center. The offset is picked randomly from a normal distribution with mean 0 and standard deviation σ. By increasing σ, we can dilute the geographical correlation of subscriptions in NI-Xmod. Therefore, NI-Xmod represents a middle ground between NI-R and NI-X. When comparing the submodels, we use the same set of random numbers as the interval lengths. This way the total overhead of the system under different submodels should be comparable.

We feel that the NI model is fairly general and flexible enough to capture many important tradeoffs. Specifically, different users can have subscriptions of different sizes, corresponding to different lengths of the intervals. Moreover, we can simulate the different amount of overlap between two users' interests by adjusting the overlap of their two corresponding intervals. Nevertheless, our studies also use a second subscription model, the Topic Tree (TT), which simulates a newsgroup type of environment with a hierarchical tree of topics. We omit the details of the TT model in this paper since the conclusions drawn from the TT experiments are similar to those from the NI model.

4.3 Event distribution model

For simplification, we assume that an event has an equal probability of being represented by any real number (within a range of possible numbers) in the Number Intervals model. Although we realize that in practice this assumption does not always hold (e.g., a nuclear reactor has a higher probability of being in a certain temperature range), we also believe that a skewed event distribution function will not affect our conclusions.[4] Based on the assumption, the number of events in E matching a subscription α, i.e., $\Phi E(\alpha)$, is proportional to the length of α's interval.

For most of our simulations, we assume that the event distribution used by the nodes when running the algorithm will be the same as that of events actually published during the simulation. In Section 5, however, we will also study what happens if this assumption does not hold.

Table 2 lists all the parameters used in the simulation, and their default values.

Sym	**Meaning**	**Base value**
N_c	# clusters	3
N_n	# nodes per cluster	10
N	total # nodes	$= N_c \cdot N_n$
D_c	avg. dist. btw. cluster centers & origin	100
D_n	avg. dist. btw. nodes & cluster center	80
R	radio transmission range	80
l	avg. interval length (NI)	25

Table 2: Parameters used in analysis

5 Results

In this section we study and compare the performance of different algorithms. Due to space limitations, we will not present all of the studies we have done, but only give a few representative results. We have opted to run simulation at a higher level than the detailed MAC-layer simulators used, for example, by some earlier performance studies to compare multicast protocols [8]. As a result, we have not studied aspects of the systems such as control overhead or link contention. We believe, however, that our simulator is adequate given the high level issues we are trying to study at this stage, such as the appropriateness of our new metric and the fundamental difference between optimal PST algorithms and traditional multicast protocols.

For every algorithm except OPT, we do several simulation runs and take the average as the final performance number for that algorithm in that particular configuration. During each run, nodes take turn to execute the algorithm, i.e., to send messages and receive replies. The order in which nodes take turn is random, and is different for each run. This

[4] In a sense, the effect of a skewed event distribution can also be modeled by a skew in the subscription interval length, for example.

simulates, for example, nodes joining the publish/subscribe system in different orders. We run the simulation long enough to reach a steady state, when nodes stop switching to better parents between successive iterations. We then take a snapshot of the PST to calculate our evaluation metric: total tree overhead O. Since we do not simulate node mobility or failures except for our dynamics study in Section 5.6, the tree structure is guaranteed to eventually converge.

5.1 Overhead of RANDOM

Figure 6 compares the performance of RANDOM with the three variants of SHOPPARENT. The y-axis gives the overhead metric (O) of the algorithms, while the x-axis is l, the average interval length in the Number Intervals subscription model. Note that the NI-R submodel is used, where subscriptions are random intervals on the number axis that are of varying lengths. A lower y value, naturally, implies a more efficient PST. A larger l implies that users are interested in more things (larger intervals). Hence the total overhead (overall work) rises with increasing l. Moreover, larger intervals also mean that there is more potential overlap between different user's interests. For example, at $l = 1$, the probability that two random user subscription intervals overlap is about 0.6%. The same probability increases to 36% when $l = 50$.

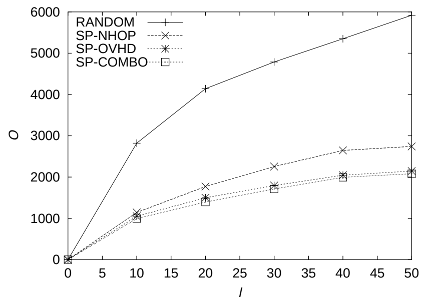


Fig. 6: Total overhead vs. average interval length l of NI-R.

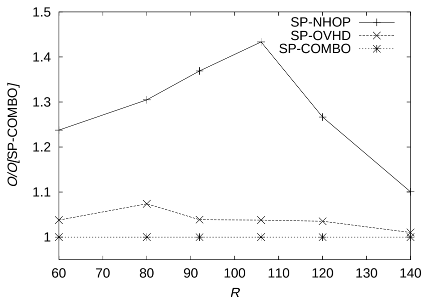


Fig. 7: Relative overhead of three SHOPPARENT algorithms vs. radio range R.

We see from the figure that RANDOM has a much larger overhead than the others. Because resources are extremely limited in a wireless ad-hoc environment, an efficient PST is especially important, and clearly RANDOM is not a good algorithm for this environment. Instead, a node should be discriminating in choosing its parent, as is the case with the SHOPPARENT algorithms.

Although all three SP algorithms work much better than RANDOM, SP-NHOP is out-performed by SP-OVHD and SP-COMBO by a significant percentage. (Our next figure highlights these differences.) We use the general term "subscription-aware" to describe algorithms that take into account user subscription information when constructing PSTs. SP-OVHD and SP-COMBO are two such examples, because user subscriptions are used in the routing metric to help a node select a "better" parent. SP-NHOP, on the other hand, is not subscription-aware. Figure 6 suggests that it is advantageous to use a subscription-aware algorithm. Furthermore, the advantage amplifies as l increases. In other words, subscription-aware algorithms are better suited to take advantage of increased overlap of user interests, marked by a bigger l. Intuitively, if a node can find another node which shares much of its own interest as its parent, the overall overhead is often reduced.

SP-COMBO is the best performing algorithm in the pack. We attribute SP-COMBO's superiority to its routing metric, which we believe more accurately estimates the impact of a local decision on the overall global outcome. However, as shown in the figure, the difference between SP-COMBO and SP-OVHD is usually quite small.

We also conducted the same set of experiments using different values for r in the overhead metric formula. At $r = 0$, for example, we ignore the cost of receiving a message, and concentrate only on the cost to transmit (f). We have found that, as expected, the curves all have smaller y values with smaller r. Moreover, the relative difference between them decreases (e.g., the difference between SP-COMBO and SP-NHOP is reduced from about 30% at $r = 1$ to about 23% at $r = 0$). However, the shape of the curves remains almost identical with varying r, so our conclusions are still valid.

5.2 SHOPPARENT variants

Figure 7 compares the three SHOPPARENT algorithms as network connectivity varies. Each curve represents the ratio of O of the corresponding SP algorithm to that of SP-COMBO, plotted against R, the radio transmission range. We plot the overhead ratio instead of absolute overhead numbers to more clearly show their relative performance. SP-COMBO is chosen as the "base" algorithm because it has the best performance. For example, we see in the graph that SP-NHOP generates about 30% more overhead than SP-COMBO at $R = 80$. We still use the NI-R model. As before, we observe that the curves follow the order SP-NHOP, SP-OVHD, and then SP-COMBO from worst to best performance.

Recall that the radio range R affects the connectivity graph G, with a larger R implying a more "connected" G. For example, at $R = 60$, each node in the 30-node system has 5.2 immediate neighbors on average. The average increases to 21.3 at $R = 140$. A striking feature of Figure 7 is that the performance gap first widens and then narrows as R increases. Initially, as G becomes more "connected", it contains more spanning trees, which are candidate PSTs, thereby giving a superior algorithm a better chance at excelling. However, as R continues to grow, G approaches a fully connected graph. In such case, nodes can simply broadcast events, and all algorithms can perform equally well.

5.3 Near-optimality of SP-COMBO

Figure 8 compares SP-COMBO to OPT, plotting their overhead against l, again using the NI-R model. Algorithm OPT is globally optimal, always producing the best PST. Because of the inherent intractability of OPT, we use, for this experiment only, a smaller scale system setup. Specifically, we let $R = 180$ to limit the potential number of spanning trees in G, thus to allow OPT to finish in a reasonable time. The figure shows that in many cases SP-COMBO comes fairly close to the optimal in terms of performance, with difference not more than 8%. In fact, after running simulations for several configurations, the most overhead we have observed is less than 15%. Of course, in larger scenarios, the difference could be larger. Yet, we expect SP-COMBO to continue to do very well, at a much lower computational cost.

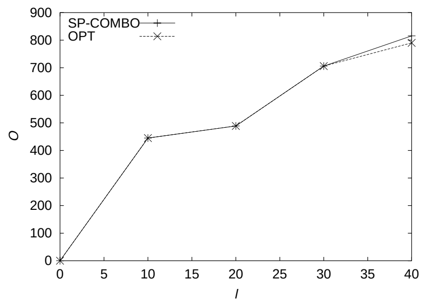


Fig. 8: O versus l, comparing SP-COMBO and OPT.

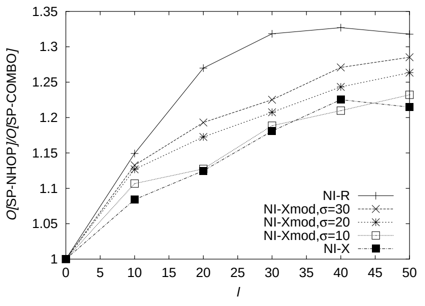


Fig. 9: Relative overhead of SP-NHOP over SP-COMBO, with different NI submodels.

5.4 Subscription geographical correlation

Figure 9 plots the overhead ratio of SP-NHOP over SP-COMBO, against average interval length l. The curves use different submodels of the Number Intervals subscription model. The y value indirectly reflects the advantage of subscription-aware algorithms, represented by SP-COMBO, over non subscription-aware ones, represented by SP-NHOP. A higher y value implies a bigger performance gap between them. In general, performance difference magnifies as l increases, confirming the trend we had observed in Figure 6.

However, Figure 9 shows a surprising phenomenon that ran contrary to our intuition. We had originally expected the performance advantage of SP-COMBO to be bigger (larger y values) with the NI-X model than with NI-R. Since subscriptions are geographically correlated in NI-X, nodes closer together would also share more common interests. It seemed to us that the subscription-aware SP-COMBO should better be able to take advantage of the situation, resulting in a bigger edge over SP-NHOP. However, Figure 9 shows exactly the opposite. We explain this phenomenon next.

With geographical correlation of subscriptions, such as that provided by NI-X, a node has a much higher chance of finding that nodes around it share more or less the same interests as itself. This is a valid assumption for certain applications, e.g., one where a user subscribes to events pertaining to its local surroundings. Since the set of neighboring nodes is also where a node "shops" for a parent, geographical correlation increases the chance that a node's subscription overlaps with that of its parent. However, although this correlation results in less total overhead in the PST, surprisingly, it also *reduces* the advantage of a subscription-aware algorithm such as SP-COMBO over a non subscription-aware algorithm such as SP-NHOP. In other words, SP-NHOP is actually able to benefit more from geographical correlation. The reason is that, because there is no longer as much variation between the subscriptions of a node's neighbors, (after all, they are all similar to this node's) the potential benefit that can be reaped by a subscription-aware algorithm diminishes. In particular, picking a candidate with the fewest hop to the root is often just the right thing to do.

The above explanation is further validated by the three middle curves in Figure 9, which represent NI-Xmod submodels with different σ parameters. An NI-Xmod model is the same as NI-X except that the geographical correlation is diluted by subjecting the subscriptions to an arbitrary perturbation with standard deviation σ. The bigger the σ, the less geographical correlation there remains. As predicted by our hypothesis, the advantage of subscription-aware algorithms increases (larger y values) with decreasing amount of geographical correlation (larger σ). Moreover, as correlation gradually diminishes, the curve for NI-Xmod approaches that of a random model like NI-R.

5.5 Event distribution skew

An event distribution is used by certain SHOPPARENT algorithms such as SP-COMBO to calculate the routing metric, which in turn determines the best parent to attach to. So far our simulations have assumed that the projected event distribution is the same as the real published one. Next we study the resilience of SP-COMBO to skew in the projected event distribution, which can occur if, e.g., the characteristics of published events suddenly changes.

We introduce a variation of SP-COMBO called SP-COMBOerr, which is the same as SP-COMBO except that we deliberately inject a random error to the event distribution projection. Specifically, if the real value of an $O^*_k(E)$ term is v, we will substitute $v(1+\epsilon)$ with a random ϵ. The percentage error, ϵ, follows a normal distribution with mean 0, and standard deviation σ_ϵ. For example, if $\sigma_\epsilon = 0.1$, there is only a 68% chance[5] that ϵ will lie between $[-0.1, 0.1]$, and thus the actual value used to calculate the routing metric will be between $[0.9v, 1.1v]$.

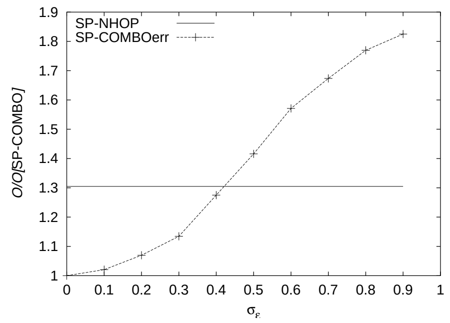


Fig. 10: Relative overhead of SP-COMBOerr and SP-NHOP over SP-COMBO vs. σ_ϵ, the standard deviation of injected error.

Figure 10 plots the relative overhead of SP-COMBOerr over SP-COMBO, against σ_ϵ. We also plot SP-NHOP over SP-COMBO for comparison. When $\sigma_\epsilon = 0$, there is no error, hence no difference between SP-COMBOerr and SP-COMBO (y value is 1). At small errors, the curve rises slowly with σ_ϵ. At $\sigma_\epsilon = 0.2$, e.g., SP-COMBOerr is still within 10% of SP-COMBO. The two curves cross at about $\sigma_\epsilon = 0.4$. Therefore, for a relatively wide range of errors (σ_ϵ between 0 and 0.4), SP-COMBO outperforms SP-NHOP. That is, even when expected traffic predictions can be off quite a bit, the predictions can still be helpful. On the other hand, if $\sigma_\epsilon > 0.4$, one is better off switching to SP-NHOP instead. Note that, however, even if significantly incorrect information is used at first with SP-COMBO, adaptive prediction methods can allow a node

[5] According to definition of standard deviation.

to detect shifts in event distribution based on observed events, and the SHOPPARENT algorithm enables a node to easily switch to a better parent when conditions change.

5.6 System dynamics

We conducted experiments to gauge how the algorithms cope with dynamism in the wireless ad-hoc environment, such as node mobility and failures. Here we report on our experience with mobility, as the conclusion is similar for other types of tests. We envision a mobility pattern where a node can move around occasionally, but it settles down for a period of time between moves. The settled period should be long enough (e.g., on the order of minutes) to allow the publish/subscribe tree to adapt and stabilize, and also to allow the nodes to take advantage of the new efficient tree. We do not consider, for example, scenarios where most of the nodes move constantly at high speeds. In fact, it has been shown in multicast research [12] that flooding usually is the most effective method of communication given such "fast mobility."

For our experiment, after all the nodes have joined the system and the tree has stabilized, we randomly select one node and move it to another random location. Naturally, while the move is taking place, old links time out, and some nodes will likely miss events as a result. Event losses are acceptable since, as stated before, we only deal with best-effort algorithms. For each experiment, we try to answer the following questions: whether the system eventually stabilizes after the node moves to its new location; how long it takes the system to converge to a new PST; and whether an efficient new PST is always formed.

We inject mobility as described above to a range of systems with different parameters. We find that in all cases the new system eventually converges to a new PST. The stabilization time varies, but is roughly proportional to the timeout period in SHOPPARENT (i.e., the time it takes for a child to realize that the parent has gone out of range), and also to the distance between the node in question and the root (i.e., the more hops the node is from the root, the longer it takes the PST to recover). Additionally, we are able to verify that in all cases, the overhead of the newly formed PST is comparable to that if the mobile node had originally been placed at its new location when it joined the system. In summary, we find that our algorithms perform very well in our target mobile environment.

6 Related work

Our work combines a publish/subscribe mechanism with wireless ad-hoc networking. Naturally, we draw upon previous research in both areas. Yet, our work also differs from each in significant ways.

Publish/subscribe systems [13,14,10,11,15,16] have been researched and developed for many years, but as far as we know in fixed networks. Making them work in a wireless ad-hoc environment introduces new challenges [1,2,3]. Our system differs from fixed network systems in two major aspects. First, our algorithms take full advantage of the broadcast nature of wireless radio by building directly on top of lower level radio broadcast primitives. Traditional publish/subscribe systems are often built as overlay server networks over an IP-like networking infrastructure, which allows them to assume universal connectivity between any two nodes in the system, and at a roughly constant cost. Although this assumption is reasonable in fixed networks, it is wasteful in our environment because it hides the fact that a unicast is actually implemented with multi-hop broadcasts. Second, the goal of our system is different from most traditional publish/subscribe systems. Our algorithm tries to find the most “efficient” tree so that the least amount of work is needed to publish events. This is a direct result of the characteristics of a wireless ad-hoc environment. In contrast, publish/subscribe systems in the past have not traditionally been too concerned with this measure because of the relative low cost of communication in a fixed network.

Multicast routing protocols for traditional networks have been well studied. In recent years protocols have also been proposed for multicast in wireless ad-hoc networks [8]. Due to the “many-cast” nature of publish/subscribe, it is not surprising that our algorithm resembles some of these protocols to a certain extent, especially tree based protocols without unicast support [6]. The optimality criterion of [5] resembles to a certain extent the metric defined in this paper, while in many other cases, especially in fixed networks, the efficiency of the resulting multicast tree has not been a primary design consideration. While the combination of the publish/subscribe paradigm and wireless ad-hoc networks has allowed our algorithm to be simpler in certain respects (e.g., every node participates in publish/subscribe), it has also raised new challenges which have not been fully addressed in prior work. For example, our algorithm is able to take advantage of information unique to the publish/subscribe paradigm, such as user subscription information. As our simulation shows, subscription-aware algorithms can have significant benefits in certain situations.

Some ad-hoc multicast protocols use a mesh-based approach instead of tree-based, to increase route reliability. These systems trade efficiency of the routing structure for resilience in a highly dynamic and mobile environment. Our algorithms can deal with mobility as shown in Section 5.6, but we are not focusing on “unstable” systems. That is, our target operating environment is one with occasional reconfigurations, followed by periods of stability. In such an environment, our experiments have shown that a good routing tree can yield significant advantages.

Distributed sensor networks [4,17] coordinate a large array of small wireless sensors that monitor and interact with the physical world. Because such systems face many of the same challenges as our target environment, such as energy constraints and lack of existing infrastructure, their solutions have similarities with our algorithms.

7 Conclusion

In this paper we have studied the tree construction problem in wireless ad-hoc publish/subscribe systems. We have proposed a distributed algorithm SHOPPARENT, and developed an evaluation metric, total overhead of a publish/subscribe tree, to study it. We have found via simulations that our algorithm, despite being a non-optimal greedy algorithm, works quite well in normal situations. We have also found that subscription-aware variants of the algorithm give further performance improvements by considering user subscription information during tree construction. Furthermore, the effectiveness of this subscription-awareness depends on multiple factors such as the overall amount of overlap between user subscriptions, and the existence of geographical correlation.

References

1. Huang, Y., Garcia-Molina, H.: Publish/subscribe in a mobile enviroment. In: Proceedings of the Second ACM International Workshop on Data Engineering for Wireless and Mobile Access. (2001) 27–34
2. Cugola, G., Nitto, E.D., Picco, G.P.: Content-based dispatching in a mobile environment. In: Workshop su Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi. (2000)
3. Meier, R., Cahill, V.: STEAM: Event-based middleware for wireless ad hoc networks. In: Proceedings of the International Workshop on Distributed Event-Based Sytems. (2002) 639–644
4. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next century challenges: scalable coordination in sensor networks. In: Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking. (1999) 263–270
5. Lim, H., Kim, C.: Multicast tree construction and flooding in wireless ad hoc networks. In: Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems. (2000) 61–68
6. Wu, C., Tay, Y., C.-K.Toh: Ad hoc Multicast Routing protocol utilizing Increasing idnumberS (AMRIS) functional specification. Internet Draft (1998)
7. Garcia-Molina, H., Kogan, B.: An implementation of reliable broadcast using an unreliable multicast facility. In: Proceedings of the 7th IEEE Symposium on Reliable Distributed Systems. (1988) 101–111
8. Lee, S.J., Su, W., Hsu, J., Gerla, M., Bagrodia, R.: A performance comparison study of ad hoc wireless multicast protocols. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM). (2000) 565–574
9. Kantor, B., Lapsley, P.: Network News Transfer Protocol: A proposed standard for the stream-based transmission of news. Request for Comments: 977 (1986)
10. Banavar, G., Kaplan, M., Shaw, K., Strom, R.E., Sturman, D.C., Tao, W.: Information flow based event distribution middleware. In: Proceedings of the 1999 ICDCS Workshop on Electronic Commerce and Web-Based Applications. (1999)
11. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Achieving scalability and expressiveness in an Internet-scale event notification service. In: Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing. (2000) 219–227
12. Ho, C., Obraczka, K., Tsudik, G., Viswanath, K.: Flooding for reliable multicast in multi-hop ad hoc netowrks. In: Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M'99). (1999) 64–71
13. Oki, B., Pfluegl, M., Siegel, A., Skeen, D.: The Information Bus - an architecture for extensible distributed systems. Operating Systems Review **27.5** (1993) 58–68
14. Segall, B., Arnold, D.: Elvin has left the building: A publish/subscribe notification service with quenching. In: Proceedings of the 1997 Australian UNIX Users Group Technical Conference. (1997) 243–255
15. TIBCO Software Inc.: TIBCO Rendezvous. (`http://www.tibco.com/solutions/products/active_enterprise/rv/`)
16. Cabrera, L.F., Jones, M.B., Theimer, M.: Herald: Achieving a global event notification service. In: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII). (2001)
17. Bonnet, P., Gehrke, J., Seshadri, P.: Towards sensor database systems. In: Proceedings of the 2nd International Conference on Mobile Data Management. (2001) 3–14