Semi-probabilistic Routing for Highly Dynamic Networks

Paolo Costa and Gian Pietro Picco

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
{costa,picco}@elet.polimi.it

Abstract. In this paper we describe a semi-probabilistic routing approach designed to enable content-based publish-subscribe on highly dynamic networks, e.g., mobile, peer-to-peer, or wireless sensor networks. We present the rationale and high level strategy of our approach, and then show its application in a link-based graph overlay as well as in a broadcast-based sensor network. Simulation results confirm that, in both scenarios, our semi-probabilistic approach strikes a balance between entirely deterministic and entirely probabilistic solutions, achieving high reliability with low overhead.

Keywords: Epidemic Algortihms, Publish/Subscribe, Peer-to-peer, Sensor Networks

1 Introduction

Modern distributed applications exhibit increasing degrees of dynamicity, as evidenced by the emergence of mobile computing, peer-to-peer networks, and wireless sensor networks. Programming distributed applications becomes therefore increasingly complex. In this context, publish-subscribe middleware is advocated by many as a viable solution thanks to its simple programming interface, and to its inherently decoupled interaction paradigm.

Publish-subscribe middleware is organized as a collection of *client* components, which interact by *publishing* messages and by *subscribing* to the classes of messages they are interested in. The core component of the middleware, the *dispatcher*, is responsible for collecting subscriptions and forwarding messages from publishers to subscribers. In the *content-based* incarnation of this publish-subscribe model, the filtering of relevant events is specified by the subscriber using predicates on the event content (e.g., using regular expressions and logic operators), therefore providing additional expressiveness and flexibility.

However, the potential of the publish-subscribe *model* can be fully unleashed in dynamic scenarios only if the underlying *system* is compatible with their requirements. Unfortunately, mainstream systems are typically geared to large-scale settings, and therefore focus on a distributed implementation of the event dispatcher—typically organized as a tree-shaped overlay network for improved scalability—but provide few or no mechanisms for dealing with topological reconfiguration. Moreover, these systems typically base their routing decisions on

information that is disseminated deterministically—a strategy that, in a highly dynamic environment, is likely to frequently lead to stale routes.

The alternative approach described in this paper departs from the mainstream along two dimensions. First of all, we do not rely on the existence of a tree overlay, but only on the ability of a dispatcher to communicate with its neighbors. Moreover, our routing strategy exploits deterministic information only in the immediate vicinity of the subscriber, resorting to probabilistic forwarding otherwise. As our simulation results show, this semi-probabilistic approach strikes a balance between a fully deterministic approach (efficient, but not very resilient to reconfigurations) and a fully probabilistic one (very resilient to reconfiguration but characterized by a higher overhead).

The paper is structured as follows. Section 2 discusses the motivation and rationale behind our approach. Section 3 first illustrates the high-level idea underlying our semi-probabilistic approach, and then shows in detail how to exploit it in two different network scenarios: the first characterized by link-based communication on a graph-shaped overlay network, and the other by broadcast-based communication in a wireless sensor network. Section 4 reports on the evaluation through simulation of the aforementioned protocols, in their respective scenarios. Section 5 places our work in the context of related efforts. Finally, Section 6 ends the paper with brief concluding remarks.

2 Rationale and Motivation

In the application scenarios we target, the connectivity among hosts, and therefore dispatchers, can change freely and frequently. This characteristic is typical of mobile ad hoc networks, peer-to-peer networks, and wireless sensor networks. Earlier work on topological reconfiguration of publish-subscribe from our research group successfully tackled the problems posed by the topological reconfigurations occurring in this scenarios [13], e.g., showing that it is possible to reconcile routing information [22] and recover events lost during reconfiguration [9] efficiently. However, these efforts still assumed the availability of an underlying tree-shaped overlay network, as the vast majority of available content-based publish-subscribe systems relies on this assumption to provide high scalability. Nevertheless, this assumption is likely to be challenged when dynamicity is high. In fact, not only the tree maintenance protocols are likely to cause considerable overhead, but the very structure of the tree, providing exactly one route among any two dispatchers, is ill-suited to provide reliability.

In the work described here, instead, we abandon the tree-shaped overlay network and simply assume that the dispatchers are able to communicate with their neighbors. The notion of neighbor clearly depends on the network characterizing the application scenario at hand, and in this paper we consider two very common scenarios, representative of the kind of dynamicity we address. The first one assumes the existence of a graph-shaped overlay network, like those often characterizing peer-to-peer networks. In this scenario, the neighbors of a dispatcher are defined by its links on the overlay. Instead, the second scenario relies solely

on the existence of wireless broadcast communication, and therefore defines the neighbors of a dispatcher in terms of its communication range. While this assumption encompasses several scenarios, in this paper we focus preminently on wireless sensor networks.

Besides the assumptions about the network, however, the defining feature of our approach is its peculiar approach to routing. Conventional content-based publish-subscribe systems adopt a deterministic routing strategy, where events are routed according to the information disseminated at subscription time. An example is the widely adopted *subscription forwarding* strategy [7], where a subscription is sent to all the dispatchers along the tree, and events follows the reverse path from the publisher to the subscriber. A direct application of this strategy on a graph overlay would frequently create loops, and is therefore impractical. Moreover, we contend that virtually *any* fully deterministic strategy is going to experience severe drawbacks in the highly dynamic scenario we target, where routing information quickly becomes stale. At the other extreme, probabilistic approaches like *epidemic* (or *gossip*) algorithms [4, 14] are known to satisfy many of the aforementioned requirements in the context of multicast communication. Inspired by the spreading of diseases, these algorithms forward information at random towards a small subset of available nodes, and rely on the availability of multiple routes to ensure that the "infection" carrying the information extends to a sufficient percentage of the receivers. Epidemic algorithms essentially trade the absolute guarantees provided by deterministic approaches for probabilistic ones, yielding in turn increased scalability and resilience to change, as well as reduced complexity. Unfortunately, these algorithms are well-versed for group communication or broadcast, where a message must be sent to *all* the members of a predetermined set of intended recipients. Instead, in our scenario subscribers can be a small fraction of the overall dispatcher network; moreover, each subscriber may be subscribed to a different set of subscriptions. In this case, a purely epidemic approach generates unnecessary overhead, since it proceeds by "blindly" infecting all the network.

In the rest of this paper we describe a semi-probabilistic routing strategy that borrows from both the aforementioned approaches to provide reliable and efficient routing in the context of content-based publish-subscribe. On one hand, we still maintain deterministic information about subscriptions but only in the vicinity of a dispatcher, therefore reducing the likelihood of loops and yet providing accurate—albeit limited—information for routing events. On the other hand, in the portion of the network where this localized information is unavailable we complement it with probabilistic routing decisions, by forwarding events at random towards neighbors. Essentially, we use an epidemic approach complemented by deterministic information. The former addresses reconfiguration, while the latter reduces indiscriminate propagation by "steering" events towards the subscribers. As we demonstrate in Section 4, our mix of deterministic and probabilistic routing enables high reliability and low overhead in both the scenarios considered in this paper.

3 Semi-probabilistic Routing

In our semi-probabilistic approach, routing is governed by two parameters. The *subscription horizon* ϕ represents the number of hops a subscription is propagated away from the subscriber. Therefore, in the area of radium ϕ around a subscriber, all dispatchers are aware of its interests. Instead, the parameter τ represents the *event propagation threshold*, which determines to what extent events are disseminated in the network. The higher the value of τ, the higher the number of copies of an event that are forwarded by a dispatcher.

In a nutshell, our approach works as follows. When an event gets routed through the network, the local subscription table of the dispatcher is examined at each hop. If it contains some subscription coming from subscribers nearby and $\phi \neq 0$, the event is forwarded towards them. Otherwise, the decision about whether to forward the event and to what extent is taken at random, based on the value of τ. If $\phi = 0$ no subscription is ever transmitted by the subscriber node, and therefore our approach degenerates in an entirely probabilistic one.

Clearly, a real protocol is slightly more complex. In the rest of this section we describe how the high level strategy we just described is instantiated into real protocols for the two network scenarios we mentioned in the previous section. The presentation is kept concise due to space constraints: more details are available in [10, 11].

3.1 Link-based Communication: Graph Overlay

In this section we describe how our approach can be exploited in the context of link-based communication. In particular, we assume here that communication takes place along the links of an undirected graph-shaped overlay network. We further assume that the overlay network layer is able to inform our protocol when the connectivity changes, i.e., when a new link appears or an old one vanishes.

Base Routing Scheme

Subscription Propagation Propagation occurs similarly to subscription forwarding. When a subscription request is issued by a dispatcher, the corresponding message is forwarded to all of its neighbors, which update their subscription tables accordingly. If $\phi = 1$, no further action is taken. Otherwise, each dispatcher forwards the subscription message to all of its neighbors, except the one who sent the message. A subscription is never forwarded twice along the same link, unless an unsubscription occurs in between the two. Differently from subscription forwarding, however, the subscription tables maintain information not only about which subscription was received on which link, but also about the distance of the subscriber, ranging between zero (for a local subscription) and ϕ. Figure 1 shows the layout of subscriptions for a case where $\phi = 1$.

Topological reconfigurations, i.e., the appearance of a new link or the vanishing of an existing one, must induce a proper reconfiguration of subscription

information. However, this is easily accomplished by relying on (un)subscription operations, as discussed in [22]. When a dispatcher detects the presence of a new link, it simply sends a subscription message along that link. Similarly, when a link vanishes, the dispatcher behaves as if it received an unsubscription message for all patterns associated to that link. These (un)subscriptions are then propagated based to the extent determined by ϕ.

Event Propagation Event propagation is where probabilistic decisions may come into play. Upon receiving an event, in principle[1] the following processing occurs. First, the subscription table is inspected for subscriptions matching the event. If a match is found, the event is routed along the link associated to the subscription. Subscriptions selection is prioritized according to ϕ: an event is forwarded based on a subscription at distance d only if there is no matching subscription at distance $d-1$. As we verified through simulation, this strategy reduces the likelihood of forwarding the event along a stale route.

This step is iterated until the number of links used for propagation is greater than f. If the number of matching subscriptions is not sufficient, the propagation threshold is met by forwarding the event along as many links as needed to reach f, randomly selected among those that have not been used in the current forwarding step. The only exception is constituted by links associated to subscriptions at distance $d = 1$; a matching event is forwarded along *all* of these links, regardless of the propagation threshold. The rationale is the fact that subscriptions at $d = 1$ represent the most accurate routing information, and the most direct route towards the corresponding subscribers. Finally, it is important to note that, during the overall process, an event is never forwarded twice along the same link.

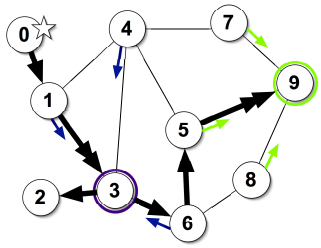


Fig. 1. Semi-probabilistic routing with $\phi = 1$ and $\tau = 0.5$. Numbered circles represent dispatchers. A colored circle around a dispatcher denotes it as a subscriber. The short colored arrows represent subscription information, and indicate the forwarding path for matching events. Dispatcher 0 publishes an event that gets forwarded either deterministically (double-headed thick arrows) or probabilistically (single-headed thick arrows).

Figure 1 shows an example. Let us assume that $\tau = 0.5$ and that an event matching both subscriptions is published by dispatcher 0. This dispatcher has only one link and no subscription information: therefore, the event gets forwarded to dispatcher 1 as this is the only alternative. At dispatcher 1, two links are available. Nevertheless, the link towards 3 is associated with subscription information: it is therefore selected for forwarding and no further action is taken since the threshold is met. At dispatcher 3, the event is delivered locally. Moreover, the links towards 2, 4 and 6 are all viable routing options, and the event must be forwarded along $f = 2$ links. No deterministic information is

[1] This sequence of steps serves only for illustration purposes: a number of optimizations are possible in reality.

available, therefore the decision is done entirely at random. The figure shows the case where the event is forwarded towards 2, where it stops propagating, and 6. There, the same situation occurs, with the links towards 5 and 8 as viable options. The figure shows the case where 5 is selected. At this dispatcher, the presence of deterministic information "captures" the events and steers it towards 9, where it gets locally delivered.

It is interesting to note that, at each hop, an event may be routed according to different criteria. As we already mentioned, "holes" in the dissemination of subscription information are bypassed by relying on random selection of links. However, the very nature of content-based systems is an asset for our routing approach, because an event matching multiple subscriptions may leverage of a bigger set of subscription information during its travel. Again, this is exemplified in Figure 1, where the event not only is routed by a mixture of deterministic and non-deterministic decisions, but deterministic ones (i.e., the hops from 1 to 3, and from 5 to 9) are generated by different subscriptions.

Additional Protocol Details

Dealing with loops With reference to Figure 1, a choice of $\phi = 2$ would have created a routing loop among the nodes 4, 5, 7, and 9. Loops can be detected easily by relying on a unique identifier for every event, trivially implemented using the identifier of the event publisher and the value of a counter incremented at the publisher each time it publishes an event. Therefore, an event received is actually propagated by a dispatcher only if it has never been received before.

More sophisticated loop avoidance and detection algorithms are available in the literature. However, on one hand they are likely to be impractical in the highly dynamic scenario we target, while on the other hand they would introduce a lot of complexity in our algorithm, which instead we want to keep as lightweight as possible.

Avoiding unnecessary propagation In Figure 1, we note how event forwarding does not really stop at dispatcher 9, since there is no way to know that no other subscriber exists in the system. Without a way to stop forwarding, events would be forwarded indefinitely—more precisely, until a loop is detected. Indefinite propagation is dealt with by attaching a time-to-live (TTL) field to each event message, and by decrementing its value at every hop. When an event is duplicated at a dispatcher along multiple routes, all the copies retain the same TTL. Therefore, an event is propagated only if its TTL is greater than zero.

A more refined mechanism consists of associating different TTLs to the two form of routing we exploit, therefore defining a *deterministic TTL* (TTL_d) and a *probabilistic TTL* (TTL_p), each limiting only the corresponding routing component. With this scheme, propagation ceases when both TTL values reach zero. The advantage of this scheme is that it provides a direct way to control both aspects of propagation, therefore enabling a more accurate tuning of the performance of our approach.

3.2 Broadcast-based Communication: Wireless Sensor Networks

In this section we show how to adapt our approach for a wireless sensor network scenario. In the following we assume wireless broadcast is the only communication media used, and also assume that each (active) sensor takes part in routing, regardless of whether it is currently interested in publishing or subscribing.

Base Routing Scheme

Subscription Propagation When the application running on a node issues a subscription, our protocol broadcasts the corresponding filter. This information is rebroadcast by the subscriber neighbors to an extent defined by the subscription horizon ϕ. In the link-based approach, ϕ was measured as the number of hops travelled by a subscription message along the links of the graph overlay. Here, instead, ϕ represents the number of times the subscription message is (re)broadcast. Moreover, in Section 3.1, we exploited the standard technique of dealing with (un)subscriptions explicitly, by using control messages propagated whenever a node decides to (un)subscribe. The same technique is used to deal with appearing or vanishing links, by treating the disappearing endpoint as if it were, respectively, subscribing or unsubscribing. Here, we use a different strategy that associates *leases* to subscriptions, and requires the subscriber to refresh subscriptions by re-propagating the corresponding message[2]. If no message is received before a lease expires, the corresponding subscription is deleted.

Clearly, there are tradeoffs involved. Without a leased approach the (un)subscription traffic is likely to be significant, due to the need to reconcile routing information whenever a link appears or disappears. The leased approach remarkably reduces the communication overhead, by removing this need. On the other hand, if subscriptions are stable, bandwidth is unnecessarily wasted for refreshing leases. However, in sensor networks the former case is much more likely to happen than the latter, since nodes typically alternate work and sleep periods to save energy. Moreover, the combination of leased subscriptions and broadcast communication remarkably simplifies the management of the subscription table, and drastically reduces the associated computational and memory overhead. In the previous section, to properly reconcile subscription information upon connectivity changes, we kept a different table for each value of ϕ, where each row contained the subscription filter and the link the subscription referred to. Here, instead, all we need is to store the subscription filter together with a timestamp used for managing leases. Differentiating according to ϕ is no longer needed, since subscriptions simply expire, and broadcast removes the need for maintaining information about links.

Event Propagation In the link-based approach, the event propagation threshold τ controls the effectiveness of event routing by specifying a fraction of the links

[2] Optimizations are possible, e.g., to broadcast the subscription hash, and transmit the entire one only if missing on the receiving node.

available at a given dispatcher. Nevertheless, here we assume broadcast communication, therefore this parameter assumes a different meaning. When an event is received[3] for which a matching filter exists in the subscription table, the event is simply rebroadcast. On the other hand, if no matching subscription is found, the event is rebroadcast with a probability τ. The parameter τ, therefore, still limits the extent of propagation, but more indirectly than in Section 3.1, as it comes into play only when no deterministic information is available.

The effectiveness of our approach is clearly proportional to the number of forwarders F, i.e., the neighbors receiving and retransmitting an event. In absence of deterministic information, in our approach $F = \tau \cdot \eta$ holds, being η the number of neighbors. As a consequence, a small value of η (e.g., in sparse networks) must be compensated by increased values of τ.

Moreover, using a link abstraction the event always got routed along the fraction of links mandated by τ, here instead we have a non-zero probability that none of the neighbors will rebroadcast the event. More precisely, in absence of deterministic information, if η is the average number of neighbors, the probability of stopping the propagation of the event is $(1 - \tau)^\eta$. If no subscriber is in the immediate vicinity of the event publisher and τ is small, there is a significant possibility that event propagation immediately stops. To ensure that a reasonable amount of event messages are injected into the network, we mark event messages with a flag stating whether they have been just published or instead they already travelled through the network. In the first case, the receiver behaves as if $\tau = 1$ and rebroadcasts the event in any case. This mechanism guarantees that at least η copies of the event message are injected in the network and propagate independently.

Additional Protocol Details

Dealing with Collisions Wireless broadcast is subject to packet collisions, which occur when two or more nodes in the same area send data at the same time. Since in our approach the propagation of subscriptions and events both rely on wireless broadcast, it becomes crucial to reduce the impact of collisions and avoid wasting precious energy on useless retransmissions.

TinyOS [16] adopts a very simple scheme to recover from collisions where, after a broadcast message has been sent, the sender waits for an acknowledgment from at least one of its neighbors. If none is received before the associated timeout expires, the message is resent. The evident weakness of this solution is that it does not take into account the actual number of neighbors. If only one neighbor received and acknowledged successfully the message, the transmission is assumed successful, regardless of the possibly many nodes that did not receive the message. Moreover, it does not try to limit in any way the number of collisions. More sophisticated MAC protocols has been proposed in literature [21] but none is currently supported by the Crossbow MICA2 [1], our target platform.

[3] Clearly, events that have already been processed and that are received again because of routing loops are easily discarded based on their identifier.

Therefore, we conceived a simple yet effective solution that decreases significantly the number of collisions, without requiring any synchronization among nodes. The idea can be regarded as a sort of simplified TDMA protocol where each node, upon startup, sets a timer whose value is a global configuration parameter. Sending messages (i.e., subscriptions and events) takes place only upon timer expiration, while receiving is in principle always enabled. Since each node in the network bootstraps at a different time, it is highly unlikely that two nodes in range of each other end up with synchronized timers. The simulations in Section 4 show that this trivial idea goes a long way in drastically reducing the amount of collisions.

Avoiding Unnecessary Propagation In Section 3.1 we limited the propagation of events using a TTL. However, our simulations showed that this solution is much less effective with broadcast propagation. In fact, even when an event travels for a small number of hops, the number of nodes it reaches is great, and therefore the impact of TTL is limited.

To address this issue, we modified slightly the retransmission strategy we just described. Let us assume a node A waiting to broadcast an event e hears one of its neighbors, say B, transmitting e before A's timer expires. If the set of A's neighbors partially overlaps with B's neighbors, it is likely that most of A's neighbors receive the event from B's transmission, therefore making A's broadcast largely useless. Some of A's neighbors may not hear about e from B but, given the epidemic nature of our algorithm, they are very likely to get it through other routes. Based on this observation, in our approach (which we called *delay-drop*) we would simply let A safely remove e from its transmission queue. In doing this, not only we limit propagation—our initial rationale for this modification—but also reduce communication and therefore save battery power. A downside of this approach is a potentially higher latency, as the event may go through longer routes before reaching its recipients. Nevertheless, in principle this delay-drop mechanism could be only one of many alternatives specified at the application or middleware layer, therefore enabling to tradeoff latency for overhead as needed.

4 Evaluation

In this section we report about the evaluation through simulation of our approaches in a wired, link-based scenario as well as a wireless, broadcast-based one. The original and complete evaluations can be found in [10, 11].

The metrics we analyze are event delivery rate and overhead. The former is defined as the ratio between the number of subscribers that should receive a given event and those who actually get it. The overhead is constituted by subscription messages and by event messages that are either duplicated, never received by a subscriber, or routed along unnecessarily long routes. These contributions are difficult to separate, and in any case do not provide significant insights. Therefore, we analyze overhead by simply plotting the overall number of messages flying in the system.

In our simulations, an event is represented as a randomly-generated sequence of integers, determined using a uniform distribution. An event pattern associated to a subscription is represented by a single number. An event matches a subscription if it contains the number specified by the event pattern in the subscription. Each dispatcher is subscribed to two event patterns, drawn randomly from the overall number of patterns available in the system. For each event, the percentage of receivers is about 10% of the overall number N of dispatchers in the system as this is a commonly accepted "rule of thumb" for content-based systems (see e.g., [8]). Finally, simulations are run with dispatchers continuously publishing events on a network with stable subscription information, i.e., where no (un)subscriptions are being issued.

4.1 Graph Overlay

In this section we evaluate the protocol described in Section 3.1 over a graph overlay network. The graph is built with a constant degree $l = 5$, to eliminate as much as possible the bias induced by a random shape. To accommodate dynamicity, however, the actual number of links of a dispatcher is allowed to vary between $l - 1$ and $l + 1$. A topological reconfiguration consists of a link breakage, followed by the appearance of a new link. Graph repair is performed after a time interval (that we set to $0.1s$) modeling the delay necessary to the underlying layers to find the replacement link. When a link breakage occurs, our simulator looks for two nodes with a degree lesser than or equal to l, to maintain the average degree as steady as possible. Likewise, a link is selected for removal only if its endpoints' degree is greater than or equal to l. The time between two reconfigurations is $\rho = 0.03s$ and since a link is always replaced after $0.1s$ the system is undergoing continuous and frequent reconfiguration[4]. Also, we initially set TTL=∞ to first analyze the behavior of the system without limiting event propagation. Each simulation was run 10 times with different seeds and the values averaged. Simulations were run with the OMNeT++ discrete event simulator [26].

Network Size We first analyze the performance of our approach when the size of the system grows in a setting where the network topology undergoes reconfiguration. The left chart in Figure 2 shows the event delivery rate for a configuration with an event propagation threshold $\tau = 0.25$ and a subscription horizon varying between $\phi = 0$ (purely probabilistic) and $\phi = 3$. The chart evidences that indeed deterministic information boosts delivery, which almost doubles when moving from a purely probabilistic routing to one with a 1-hop subscription information, while there is no appreciable difference against $\phi > 1$. The reason for the overlapping of these latter curves is that, unlike $\phi = 1$, they are subject to the limitation on propagation set by τ.

[4] Each reconfiguration involves two dispatchers, the link endpoints. At 300 reconfigurations per second, with a network size of $N = 300$ each dispatcher changes two neighbors per second. Since our simulations are run for over two seconds, at least 4 neighbors out of 5 get replaced.

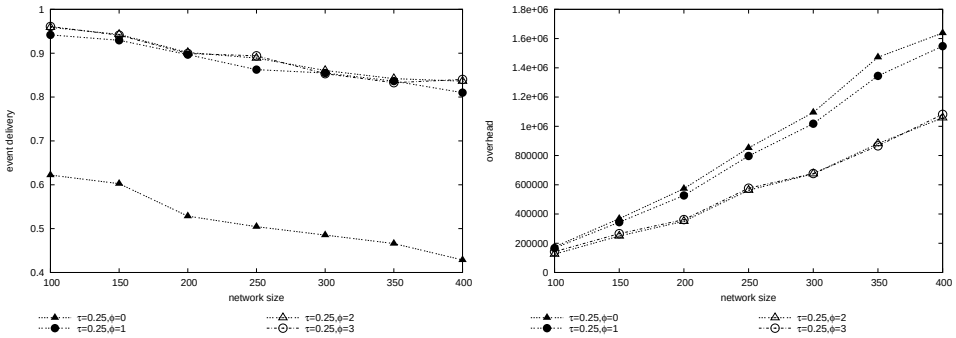


Fig. 2. Event delivery and overhead in a network with reconfigurations every $\rho = 0.03s$.

It is worth noting that, as discussed in [11], the performance of our semi-probabilistic routing on a static topology is similar to the one we just described. Indeed, the perturbation induced by dynamicity is easily absorbed by the probabilistic component of routing and by the redundancy of the graph. Moreover, as known from probabilistic algorithms, the continuous restructuring of connectivity among dispatchers effectively helps spreading information, by allowing nodes to suddenly become in contact with a different set of neighbors, and spread messages from another point.

The overhead is shown in the right chart of Figure 2. The upper bound is provided by flooding ($\tau = 1$), which is not plotted because it generates an extremely high number of messages (e.g., 11,882,777 at $N = 400$). Therefore, Figure 2 shows that our overhead is extremely far from the upper bound. Indeed, more deterministic information ($\phi > 1$) enables savings up to 35% w.r.t. pure probabilistic and $\phi = 1$. We also verified that higher values of τ quickly bring the system to 100% delivery. This is clearly true for flooding. Also, $\tau = 0.5$ already brings all the curves to full delivery except for $\phi = 0$, which remains at about 96%. Nevertheless, in this latter case the overhead is five times higher w.r.t. to the case with $\tau = 0.25$ and $\phi = 1$ (around 5.5 million messages instead of 1.6 at $N = 400$). The relative performance among the curves with $\tau = 0.5$ is unchanged, with $\phi > 1$ providing the smaller overhead.

Looking at Figure 2, one could notice how delivery drops as the scale increases. Nevertheless, it is worth noting that in the charts above we assume that each dispatcher added to the system is also a publisher emitting 5 events per second, and that the fraction of receivers for each event is always 10% of the dispatchers in the system. Instead, both the dispatcher's degree l and the event propagation threshold τ remain constant: the fanout f (i.e., the number of links along which events are forwarded) therefore remains constant as well. As a consequence, while the number of dispatchers and receivers increases the ability of the system to spread messages decreases. In a real deployment setting,

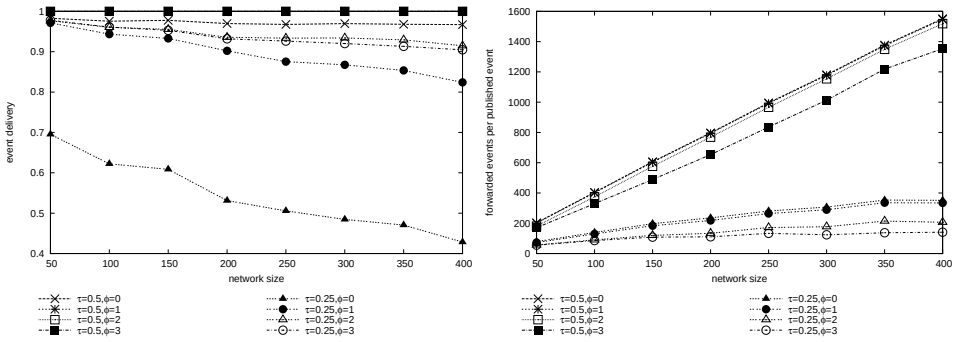


Fig. 3. Event delivery and overhead in a network with reconfigurations every $\rho = 0.03s$ and a fixed number of receivers.

the increase in scale should be compensated by increasing f, i.e., by intervening either on τ or l. In [11] we showed how increasing the degree to $l = 9$ boosts delivery which becomes close to (and for $\phi = 1$ exactly) 100%, due to the ability to spread messages over more links. On the other hand, the overhead is also largely increased and reaches the same order of magnitude of the configuration with $l = 5$ and $\tau = 0.5$. This is not surprising, since the product $\tau \cdot l$, which defines the number f of links available for routing and therefore ultimately constrains the effectiveness of routing, is roughly the same in both scenarios.

Density of Receivers vs. System Scale To evaluate how the density of receivers in the network affects our strategy we studied a scenario where the number of dispatchers (still all publishing at 5 event/s) is increased while the number of receivers per event (10 in our case) remains constant. The density of receivers therefore decreases linearly, from 20% for $N = 50$ down to 2.5% for $N = 400$. This scenario elicits new issues w.r.t. the one we examined previously. With a constant density of receivers and a growing scale, we need to increment the number of forwarded events to reach a larger set of receivers, and therefore increasing the fanout is a viable solution. Instead, here the number of receivers does not change: therefore, what we are assessing is how "selective" is our routing towards the receivers.

Results are in Figure 3, with the same simulation parameters as in Figure 2. The chart shows both the event delivery and the number of forwarded events divided by the number of published events, where the latter characterizes the effort, in terms of forwarded events, required to deliver a single event to a fixed set of receivers in a growing network[5].

[5] We do not consider the traffic generated by (un)subscriptions since in our heavy publishing scenario it is negligible w.r.t. the number of events.

Figure 3 shows that a high fanout ($\tau = 0.5$) always achieves high delivery but basically saturates the network by reaching almost every dispatcher, thus increasing the traffic linearly with scale. However, even in this case deterministic information ($\phi = 3$) achieves some savings, as it enables a more selective routing. Instead, a lower fanout ($\tau = 0.25$) yields a very different behavior. Event delivery is a lower than with $\tau = 0.5$, since the probability to reach a receiver depends on the product of the probabilities to select the right neighbor at each hop—which in turn depends on fanout, $f = 1$ in this case—and therefore decreases as the routes connecting publishers to receivers become longer. However, while this negative effect is evident for pure probabilistic routing, it is almost entirely compensated in terms of delivery by the deterministic information available when $\phi > 0$, which reduces the reliance on random forwarding by "steering" forwarded events more efficiently through the network. This increased efficiency is mirrored in the overhead chart, where the number of forwarded events per published event still increases, due to the longer routes towards receivers, but this time remains well below a linear trend.

Limiting Propagation As we discussed in Section 3.1, introducing a TTL enables considerable savings in overhead. Clearly, low TTL values may constrain propagation too much, and negatively impact event delivery. In our simulation scenario we found out that with TTL=14 the event delivery with $\phi > 0$ remains about the same, while the purely probabilistic routing gets about 20% worse. This is not surprising, given that the deterministic component leads to significantly shorter routes, and therefore is not affected significantly by the TTL. Moreover, the overhead drops considerably, as we expected. For $\phi > 1$, overhead is reduced of about 8%, while for the others reduction is around 30%.

The use of two different values TTL_d and TTL_p we mentioned in Section 3.1 enables further optimizations, as shown in Figure 4 for TTL_d =10 and TTL_p =8. Differently from the chart we showed thus far, this one represents a single run plotted against (simulated) time, and $N = 300$. The values for TTL_d and TTL_p enable only minor—albeit positive—variations over the single TTL=14. Instead, the tradeoff in terms of overhead is profoundly different. The overhead of $\phi > 1$ is slightly increased, but justified by the small increase in the delivery rate. On the other hand, the overhead of $\phi = 1$ is greatly improved, dropping from about 820,000 messages to about 640,000 (more than 20% less), while $\phi = 2$ and $\phi = 3$ are at about 680,000 and 730,000, respectively. This configuration makes routing with $\phi = 1$ more appealing than in earlier scenarios, making it a valid alternative to $\phi = 2$. A different choice for TTL_d and TTL_p, e.g., further increasing the gap between the two, would favor routing with $\phi > 1$.

Stability of Event Delivery Figure 4 enables us to evidence also another interesting phenomenon whose generality goes beyond the use of TTL, that is, the event delivery is quite stable over time. This is particularly relevant especially if compared against similar results obtained by approaches that rely on a tree as in Figure 5. Although the comparison is entirely qualitative, since the simulators and scenarios differ, it is worth observing how event delivery has wide and very

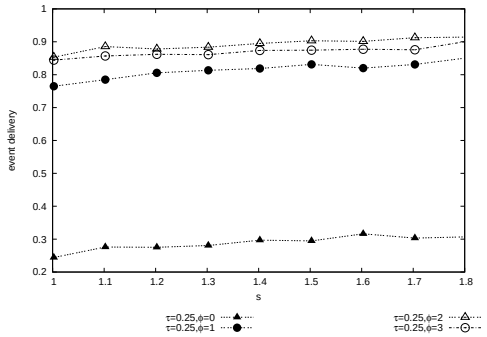


Fig. 4. Event delivery with TTL_d =10 and TTL_p =8.

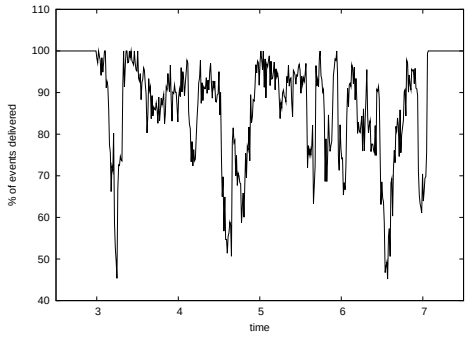


Fig. 5. Event delivery on a dynamic tree topology (from [22]).

frequent changes, ranging from 100% down to 40%. The reason for the remarkable improvement of our approach can be attributed to the use of a graph and the ability to exploit alternative routes thanks to its probabilistic component.

4.2 Wireless Sensor Networks

In this Section, we report about the performance of the approach described in Section 3.2 using TOSSIM [18], the simulation tool provided with TINYOS [16]. TOSSIM emulates all the operating systems layers and therefore works by reusing directly the code deployed on the sensor nodes—Crossbow's MICA2 motes [1] in our case. Here, we report only results for $\tau = 0.5$, as it yields the best tradeoff in our simulations. Each simulation run lasted 60 simulated seconds, with an extra second devoted to "booting" the network, as performed automatically by TOSSIM. Transmission occurs by using our simple delay technique to avoid collisions. The impact of this technique, as well as of its delay-drop variant, is analyzed later in this section. Simulations are performed on a stable network, except for the analysis of the behavior of the sensors duty cycle. Finally, we assume each node has $\eta = 5$ neighbors.

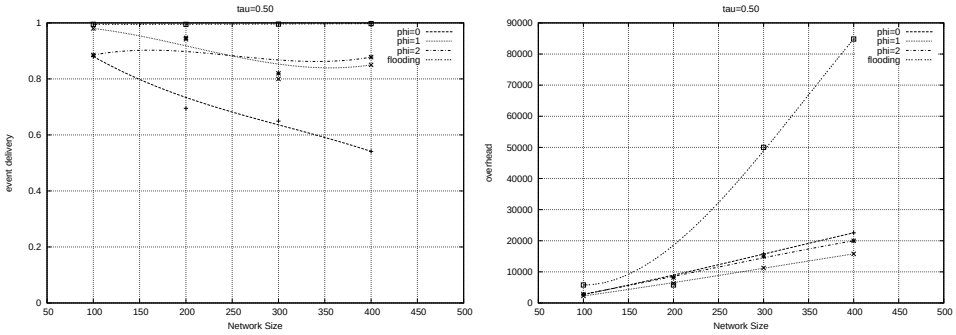


Fig. 6. Event delivery and overhead using $\tau = 0.5$.

Network Size The first parameter we analyze is the size of the network, which we ranged from 100 to 400. To maintain a steady publishing load and receiver density, we increased them proportionally by ranging the former from 1 to 4 published events per second, and keeping the latter at 10% (yielding from 10 to 40 receivers).

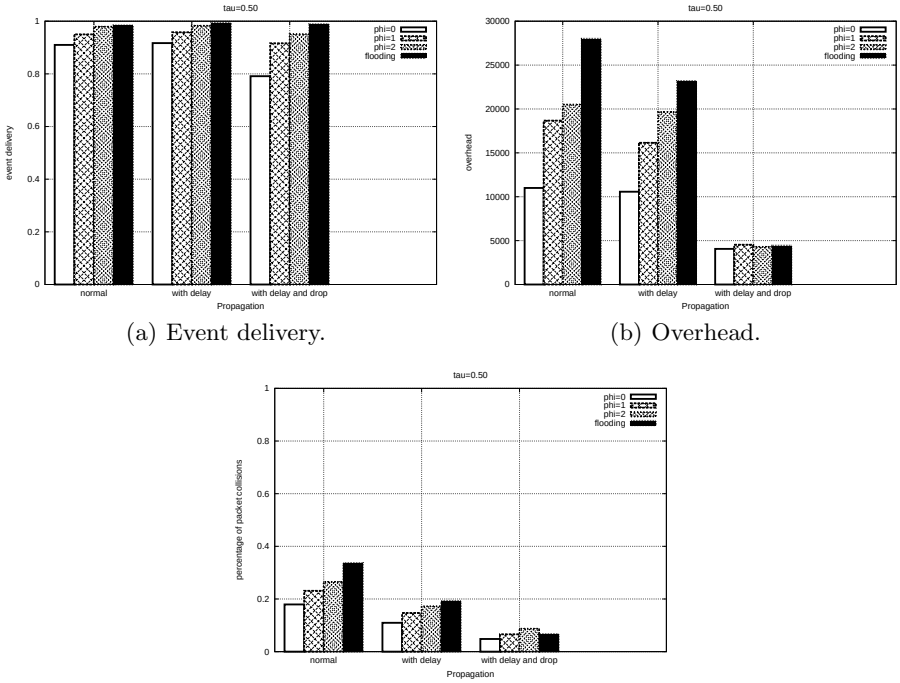
The results in Figure 6(a) show[6] that event delivery depends only marginally on network size. This is not surprising, since the probabilistic component of our approach tends to distribute the load equally on each node and, therefore, the more the network grows (and the more receivers need to be reached), the more nodes participate in delivering the events. Notably, in some cases event delivery is even increased as more routes become available. On the other hand, as shown in Figure 6(b) the overhead increases too, since the number of receivers and the publishing load augments linearly, i.e., there are more events to deliver to more recipients. Nevertheless, the two increments share the same trends, that is, no additional overhead is introduced by the size. This, again, stems from the fact the the effort imposed on each node by our algorithm is constant.

It is interesting to see that $\phi = 1$ and $\phi = 2$ exhibit a different behavior. When $N = 100$, $\phi = 2$ performs worse than $\phi = 1$, most likely due to the fact that the smaller size increases the likelihood of creating loops. As N increases, however, the additional deterministic information provided by $\phi = 2$ becomes precious in steering events towards the receivers in a sparser network. Finally, event delivery with $\phi = 1$ and $\phi = 2$ is about the 90% of the delivery obtained through flooding, but the overhead is only 25% of the one introduced by flooding.

Collisions and Rebroadcast In Section 3.2 we described two simple techniques for, respectively, reducing collisions and avoiding useless rebroadcasts.

The effect of these techniques on the system is shown in Figure 7 for $\tau = 0.5$ and $\eta = 10$. Figure 7(a) shows that the delivery is largely unaffected, with a

[6] We use Bezier interpolation to better evidence the trends.



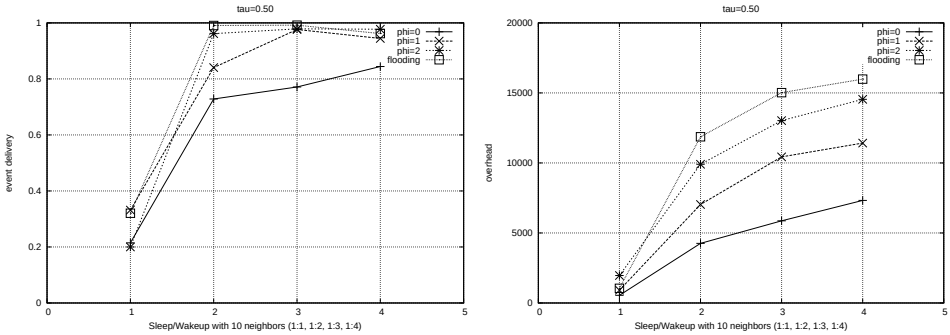
(a) Event delivery. (b) Overhead.

(c) Number of collisions.

Fig. 7. Collisions and delay-drop ($\tau = 0.5, \eta = 10$).

small decrease in the case of delay-drop. On the other hand, Figure 7(c) shows that our simple mechanism for avoiding collisions is very effective, since it more than halves the number of collisions. The delay-drop mechanism does not improve much in terms of collisions. Instead, by avoiding useless rebroadcasts, this latter technique drastically reduces overhead, as shown in Figure 7(b). Although we do not have simulations linking directly these results to the power consumption, it is evident how the combination of these two simple techniques not only improves the performance of our approach, but also yields remarkable savings in communication, therefore enabling a longer life of the overall sensor network.

Duty Cycle A prominent feature of our approach is the resilience to changes in the underlying topology and connectivity. Most approaches for content dissemination and group communication for sensor networks rely on exact routes that must be recalculated each time the topology is modified. This is an important limitation, since sensors are often supposed to regularly switch from active to sleeping, to preserve battery and extend the system lifetime. Therefore, unless some kind of synchronization is in place, routes become invalid and must be recomputed, with consequent overhead. Conversely, our approach does



(a) Delivery. (b) Overhead.

Fig. 8. Performance with sleeping nodes.

not make any assumption on the underlying topology, as it "explores" it semi-probabilistically. Therefore, it can tolerate sleeping (or even crashed, or moving) nodes, without any particular measure.

In the simulations in Figure 8, we used a simple model where each node is active for a period T_a, followed by a sleeping period T_s. All nodes are initially active: after a random time (which temporally scatters them) they are regularly switched off and reactivated after T_s. To obtain meaningful results, sleeping nodes are not considered in the event delivery, which is then computed by taking into account only the active subscribers. Also, since the temporal scattering among nodes is completely random, it may happen that under certain combination of T_a, T_s and η, the network becomes not connected. Then, a delivery of 100% is not meaningful because, if no path exists among two nodes, there is no way to correctly deliver the event. Consequently, our upper bound is represented by the delivery of the flooding approach.

In most scenarios found in literature, sensor nodes sleep for most time and switch on only for a short amount of time. However, in our scenario, sensor nodes are essential not only to acquire data from the environment but also to participate in their propagation. Hence it seems reasonable that the ratio $\frac{T_a}{T_s}$ is greater than (or at least equal to) 1. Clearly, if too many nodes are sleeping at the same time, delivery falls abruptly since the number of forwarders is too low. However, the delivery of flooding also falls abruptly, and some of our solutions remain comparable to it.

These results are not surprising, since what we stated earlier about density holds here as well. Indeed, the effect of sleeping nodes is to reduce the density, expressed in terms of the number η of neighbors. Therefore, since our algorithm tolerates low densities up to a given extent, it is resilient to sleeping nodes as well.

5 Related Work

The majority of content-based publish-subscribe systems are built upon a tree-shaped overlay, with some of them addressing the easier problem of supporting client mobility. Recent work by the authors' research group [9, 12, 22] deals instead with reconfigurations affecting the dispatchers in the tree. Other recent approaches [8, 23] exploit a graph-based topology upon which a set of dispatching trees are superimposed. However, these papers do not provide any detail about if and how dynamicity is taken into account, and at which cost.

In the related area of MANET routing, none of the approaches is directly reusable because of the peculiar challenges posed by content-based routing, but some rely on similar ideas. In the Zone Routing Protocol (ZRP) [15] for unicast, a node proactively maintains routing information about its neighborhood, and reactively requests information about destinations outside of it. In our approach, long distance propagation is instead achieved in a probabilistic way. On the other hand, route driven gossip [19] exploits epidemic algorithms to maintain and disseminate a localized view of the system, enhanced with routing information.

In the context of sensor networks, researchers mostly focused on efficiently delivering the sensed data from the sensors to a fixed base station or, alternatively, on enabling communication from the base station towards all the sensors (e.g to perform a query or force a network re-programming). Our work, instead, is directly applicable to more general scenarios where multiple data sinks (e.g., multiple base stations, but also actuators as in *wireless sensor and actor networks* [2]) are present.

Traditional approaches (e.g., [20,24]) rely on a tree-based structure to deliver messages. This approach minimizes data traffic, but tree maintenance and updates require many control messages and, more importantly, a stable network. Alternative approaches (e.g., [5,17]) spread the nodes' interests across the whole network to create a reverse path from a publisher to receivers. However, again, no details are provided about how to deal with a dynamic network, as in the case of mobile or sleeping sensors, and failures.

A recent work [3] exploits probabilistic forwarding combined with knowledge of the network topology to route messages from sensors to a special node acting as collector. The forwarding probability depends on various parameters, in particular the current distance from the collector. The probabilistic component allows to tolerate stale information on the global topology. Despite the different aim of the work, targetting at single sink application, this approach differs from ours in that we require a much smaller knowledge of the network, namely, only the subscribers ϕ hops away.

The possibility of temporarily switching off nodes is particularly amenable in sensor networks as the battery is not easily replaceable. At the same time, however, the network must maintain its functionality through a connected subnetwork, i.e., it should be able to correctly deliver events despite the lack of some nodes. Some works (e.g., [6, 25]) address this issue by introducing synchronization of the sleeping patterns to minimize the energy spent without affecting network connectivity. The weakness of this solution, however, is that other kinds

of topological reconfiguration (e.g., mobility or failures) are not tolerated. In these cases, the (expensive) synchronization procedure must be restarted, with added overhead. Conversely, our approach does not require any synchronization protocol and yet tolerates arbitrary reconfigurations.

6 Conclusions

Modern distributed applications exhibit increasing degrees of dynamicity, due to topological reconfigurations occurring at the physical or logical level. Supporting application development through middleware in dynamic scenarios in many cases demands new approaches to routing the applicative information managed by the middleware. Current approaches exploit either a deterministic approach, relying on the dissemination of routing information, or a probabilistic one, inspired by the diffusion of epidemics. However, both have drawbacks.

In this paper we described a semi-probabilistic routing approach targeted to publish-subscribe middleware for highly dynamic networks. Our routing strategy strikes a balance between the two aforementioned approaches, by combining the scalability and resilience to change of probabilistic approaches with the ability to quickly steer events towards the intended receivers typical of deterministic approaches. Our current experience with simulating this routing strategy in both link-based and broadcast-based network scenarios confirms that semi-probabilistic routing indeed achieves high delivery rates with low overhead in presence of frequent topological reconfigurations.

References

1. Crossbow Technology Inc. `http://www.xbow.com`.
2. I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal (Elsevier)*, 2(4):351–367, October 2004.
3. Christopher L. Barrett, Stephan J. Eidenbenz, Lukas Kroc, Madhav Marathe, and James P. Smith. Parametric probabilistic sensor network routing. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications (WSNA)*, 2003.
4. K. P. Birman et al. Bimodal multicast. *ACM Trans. on Computer Systems*, 17(2):41–88, 1999.
5. D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proc. of the 1st Int. Wkshp. on Wireless Sensor Networks and Applications*, pages 22–31, 2002.
6. J. Carle and D. Simplot. Energy-efficient area monitoring for sensor networks. *IEEE Computer*, 37(2):40–46, 2004.
7. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Trans. on Computer Systems*, 19(3):332–383, August 2001.
8. A. Carzaniga, M.J. Rutherford, and A.L. Wolf. A routing scheme for content-based networking. In *Proc. of INFOCOM*, March 2004.

9. P. Costa, M. Migliavacca, G.P. Picco, and G. Cugola. Epidemic Algorithms for Reliable Content-Based Publish-Subscribe: An Evaluation. In *Proc. of the 24th Int. Conf. on Distributed Computing Systems (ICDCS04)*, pages 552–561, 2004.
10. P. Costa, G. P. Picco, and S. Rossetto. Publish-subscribe on sensor networks: A semi-probabilistic approach. In *Proc. of the 2nd IEEE Int. Conf. on Mobile Ad-Hoc and Sensor Systems (MASS05)*, 2005. To appear.
11. P. Costa and G.P. Picco. Semi-probabilistic content-based publish-subscribe. In *Proc. of the 25th IEEE Int. Conf. on Distributed Computing Systems (ICDCS05)*, Columbus (Ohio, USA), June 2005.
12. G. Cugola, D. Frey, A.L. Murphy, and G.P. Picco. Minimizing the Reconfiguration Overhead in Content-Based Publish-Subscribe. In *Proc. of the 19th ACM Symp. on Applied Computing (SAC04)*, pages 1134–1140, March 2004.
13. G. Cugola, A.L. Murphy, and G.P. Picco. Content-Based Publish-Subscribe in a Mobile Environment. In P. Bellavista and A. Corradi, editors, *Mobile Middleware*. CRC Press, 2005. To appear.
14. A. Demers et al. Epidemic algorithms for replicated database maintenance. *Operating Systems Review*, 22(1):8–32, 1988.
15. Z. Haas and M. Pearlman. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. IETF draft, June 1999.
16. J. Hill et al. System architecture directions for networked sensors. In *Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104. ACM Press, 2000.
17. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proc. of MobiCom*, 2000.
18. P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems (SenSys'03)*, pages 126–137. ACM Press, 2003.
19. J. Luo, Patrick Eugster, and J.P. Hubaux. Route Driven Gossip: Probabilistic Reliable Multicast in Ad Hoc Networks. In *Proc. of INFOCOM'03*, April 2003.
20. S.R. Madden, M.J. Franklin, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD 2003*, 2003.
21. P. Naik and K.M. Sivalingam. A survey of MAC protocols for sensor networks. In *Wireless Sensor Networks*. Kluwer Academic Publishers, 2004.
22. G. P. Picco, G. Cugola, and A. L. Murphy. Efficient Content-Based Event Dispatching in the Presence of Topological Reconfigurations. In *Proc. of the 23rd Int. Conf. on Distributed Computing Systems (ICDCS03)*, pages 234–243, 2003.
23. P. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proc. of the 1st Wkshp on Distributed Event-Based Systems*, 2002.
24. C. Srisathapornphat, C. Jaikaeo, and C.-C. Shen. Sensor information networking architecture. In *Proc. of the Int. Workshop on Parallel Processing*, 2000.
25. D. Tian and N.D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proc. of the First ACM Int. Workshop on Wireless Sensor Networks and Applications*, pages 32–41, 2002.
26. A. Varga. OMNeT++ Web page. `www.omnetpp.org`.