Content-based Publish-Subscribe in a Mobile Environment

Gianpaolo Cugola, Amy L. Murphy, Gian Pietro Picco*

Abstract

Content-based publish-subscribe is emerging as a communication paradigm able to cope with the needs of scalability, flexibility, and reconfigurability typical of highly dynamic distributed applications. However, very few efforts address dynamic changes in the topology of the publish-subscribe distributed dispatching infrastructure—a fundamental challenge in mobile computing scenarios. In this chapter we illustrate the problems posed by mobility in the context of publish-subscribe, discuss protocols and integrated solutions proposed by our research group, and survey the state of the art in this research area.

*G. Cugola and G.P. Picco ([cugola,picco]@elet.polimi.it) are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. A.L. Murphy (amy.murphy@unisi.ch) is with the Faculty of Informatics, University of Lugano, Switzerland.

Contents

1 Introduction

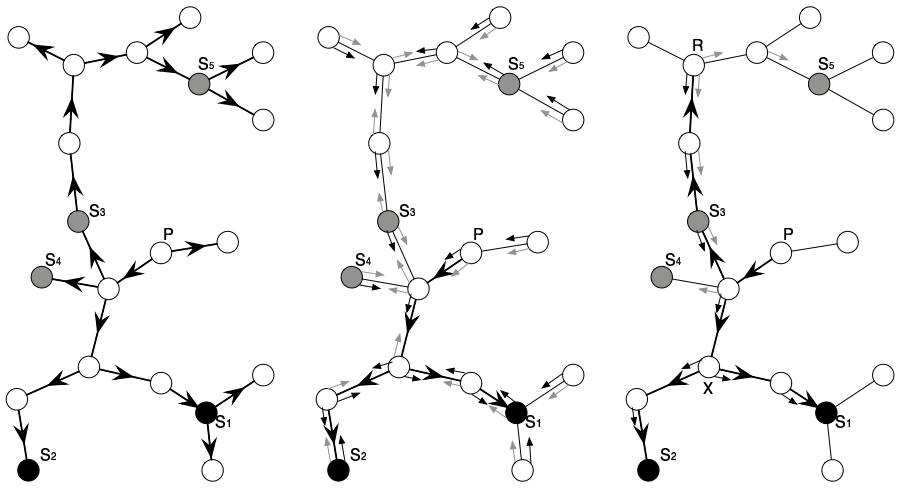
Modern distributed computing demands not only scalability, as witnessed by the Internet, but also an unprecedented degree of adaptability to dynamic conditions. Mobile computing is evidence of this trend. The mobility of network nodes undermines many of the traditional assumptions of distributed systems: topology becomes fluid as hosts move and yet retain the ability to communicate wirelessly; communication occurs over a shared media that is not only unreliable, but also largely unpredictable as it strongly depends on the characteristics of the local environment; hosts, and therefore applications, frequently experience disconnection, which is no longer just a network accident, rather is often induced deliberately for long periods of time to save power. Other modern distributed scenarios raise similar issues in terms of dynamicity: peer-to-peer networks and sensor networks come to mind.

Coping with these demands is a challenging task. In recent years, the *publish-subscribe* paradigm has emerged as a promising and effective way to tackle many of these issues. The implicit and asynchronous communication paradigm that characterizes publish-subscribe supports a high degree of decoupling among the components of a distributed application. In principle, it is possible to add or remove one component without affecting the others—only the dispatcher, the element in charge of collecting subscriptions and routing messages, needs to be aware of the change. Clearly, this form of decoupling would is desirable in a scenario where the set of available components undergoes continuous change as in the mobile one. Nevertheless, much of the potential of the publish-subscribe *model* still remains to be unleashed by publish-subscribe *systems*. Indeed, many of the available distributed publish-subscribe middleware exploit a dispatching network arranged in a tree overlay for increased scalability, but whose design usually does not tolerate any form of topological reconfiguration. Therefore, paradoxically, these systems cannot be exploited precisely in those application scenarios where decoupling would be most beneficial.

In this chapter, we discuss challenges of and solutions for content-based publish-subscribe in a mobile scenario. Although we focus on our own research in the field [1,13–16,18–20,27,33,36], we also provide the reader with a discussion of related and alternative approaches, therefore covering the whole spectrum of the state of the art.

2 Publish-Subscribe: An Overview

Distributed applications exploiting publish-subscribe middleware are organized as a collection of autonomous components, the *clients*, 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. This scheme results in a high degree of decoupling among the com-



(a) Message forwarding. (b) Subscription forwarding. (c) Hierarchical forwarding.

Figure 1: Publish-subscribe routing strategies.

municating parties. These ideas have been recently popularized by a wealth of systems, each interpreting the publish-subscribe paradigm in a different way[1].

A first point of differentiation is the expressiveness of the subscription language, drawing a line between *subject-based* and *content-based* systems. In the first case, subscriptions contain only the name of a class of messages—usually called subject, channel, or topic—chosen among a set of pre-defined classes. Instead, in content-based systems the selection of a message is determined entirely by the client, which uses expressions (often called *filters*) that allow sophisticated matching on the message content.

The second point of differentiation is the architecture of the dispatcher, which can be either centralized or distributed. In this paper, we focus on the latter. In this middleware, a set of *brokers* (see Figure 1) are interconnected in an overlay network and cooperatively route subscriptions and messages sent by clients connected to them, therefore increasing the scalability of the system. In this context, the main design decisions concern the topology of interconnection and the routing strategy. Although the first approaches based on a graph topology are starting to appear (e.g., [1,15]), most of the available systems are based on a tree topology, as this simplifies routing (e.g., by avoiding the possibility of routing loops) and provides a high degree of scalability.

Several tree-based routing strategies can be found in the literature (e.g., [5, 8,17]), with the most basic ones shown and compared in Figure 1. The simplest approach is *message forwarding* in which a published message is forwarded by a broker to all the others along the dispatching tree. Instead, subscriptions are

[1]For more detailed comparisons see [8,17,22,40].

never propagated beyond the broker receiving them. This broker stores these subscriptions in a *subscription table* that is used to determine which clients, if any, should receive incoming messages.

Message forwarding may generate high overhead since messages are sent to all brokers regardless of the interests of the clients attached to them. An alternative strategy, called *subscription forwarding*, limits this overhead by spreading knowledge about subscriptions throughout the system. When a broker receives a subscription from one of its clients, not only does it store the associated filter in its subscription table as in message forwarding, but it also forwards it to all the neighboring brokers. During this propagation, each dispatcher behaves as a subscriber with respect to its neighbors. Consequently, each of them records the filter associated with the subscription in its own subscription table and re-forwards it to all its neighboring dispatchers except the one that sent it[2]. This process effectively sets up routes for messages through the reverse path followed by subscriptions.

Finally, *hierarchical forwarding* strikes a balance between the two aforementioned strategies by assuming a rooted tree topology. Subscriptions are forwarded towards the root to establish the routes that published messages follow "downstream" towards subscribers. Messages, in fact, are always propagated "upstream" up to the root, and flow "downstream" along the tree only if a matching subscription has been received from the corresponding sub-tree.

Figure 1 provides a graphical representation of the three strategies. Brokers S_1 and S_2 subscribed (through their clients, not shown in the figure) to the same "black" filter, while S_3, S_4, and S_5 subscribed to "gray". Colored arrows represent the content of subscription tables for the corresponding filters. Broker P published a message matching the black filter but not the gray one. The path followed by this message is shown through thick, directed lines. In hierarchical forwarding, broker R is the root of the dispatching tree.

3 Mobility and Publish-Subscribe: The Issues

The overview in the previous section shows how several approaches enable distributed content-based publish-subscribe. However, most of the research efforts focus either on how to provide efficient pattern matching and message forwarding, or on efficient routing strategies for pushing scalability. Research essentially aims at improving the *performance* of content-based publish-subscribe in large-scale settings, implicitly assuming a static dispatching infrastructure. This is unfortunate because, as mentioned in the introduction, the characteristics of the publish-subscribe model and more specifically the high degree of decoupling it enables, makes it amenable to highly dynamic scenarios such as those defined by mobility. Nevertheless, this is possible only if the

[2]This scheme is optimized to avoid forwarding the same message filter in the same direction. Moreover, some systems (e.g., [8,9,26,45]) perform even more aggressive optimizations by exploiting "coverage" relations among filters.

systems embodying the model are expressly designed to take into account the assumptions and challenges posed by the target dynamic scenario.

Mobility poses several challenges to the design of publish-subscribe middleware. The most evident is that the topology of the system, usually assumed static by existing systems, now becomes dynamic and undergoes continuous reconfiguration as the mobile nodes move. Depending on the mobility scenario, this may have different impact.

In many cases, mobility is relegated to the periphery of the system. For instance, this is the case of the *nomadic* scenarios many of us experience while traveling, or even when moving from office to home. The user detaches from one network (e.g., office) and reconnects to a different one (e.g., home, or a hotel room). The entry point to the network is different and yet, thanks to dedicated protocols (e.g., DHCP, VPN), the user retains access to the basic networking services. Similar considerations hold for those scenarios where the user changes its network entry point while in movement, and protocols such as Mobile IP [34] that transparently maintain connectivity at the network level. Notably, while in the first case wireless communication is a nice but unnecessary feature, in the second one it becomes key to enable unconstrained and continuous movement. However, in both these scenarios, only the end nodes are mobile: the networking infrastructure, handling routing and other functionality, is assumed to be stable.

The same concepts can be applied to the typical architecture of a publish-subscribe system by observing that clients play the role of end nodes, since they do not provide network functionality, while instead brokers play the role of routers and switches. Similar to networking, the impact of mobility turns out to be limited to modifications that shield clients from the complexity of dealing with mobility, but leave the behavior of the infrastructure largely unaffected. Interestingly, the same applies when the system exhibits logical mobility of code or agents [28], e.g., because the publish-subscribe clients are mobile agents, which detach and reattach to the closest broker during migration.

At the other extreme, *mobile ad hoc networks* (MANETs) [35, 44] define the most radical mobility scenario, where no assumption is made about the dynamic topology of the system and the networking infrastructure itself is assumed to be mobile. The impact of mobility in this case is disruptive, and no longer limited to the clients dwelling at the fringes of the system, since the intermediate nodes in charge of routing and other network functions are now assumed to be mobile. Moreover, most applicative scenarios for MANETs actually blur the distinction between end nodes and intermediate ones, assuming that *all* the network nodes possess the functionality required to cooperate to enable routing. As a consequence, networking protocols must be rethought from the ground up to accommodate the new deployment assumptions, as witnessed by the appearance of entirely new routing protocols (e.g., those described in [35]).

Again, publish-subscribe faces similar problems, demanding significant and radical changes to the behavior of the dispatching infrastructure. For instance, subscription information can no longer be associated permanently to the link

where it came from, because the subscriber can move and become connected through a route involving a different set of links. Moreover, as in networking scenarios, the distinction between infrastructure and application nodes becomes blurred, effectively introducing a different application model where all client hosts are also brokers [30].

Interestingly, analogous considerations hold for scenarios where the communication topology is dynamic, although not caused by mobility and wireless communication. For instance, in peer-to-peer networks the hosts and physical communication links are fixed, but the *logical* topology of the overlay network along which file searches are disseminated undergoes continuous change as peers join and leave. Exploiting a publish-subscribe system in this scenario faces challenges similar to the one discussed thus far, where the architecture of the peer-to-peer network (e.g., based on a hierarchical supernode infrastructure or totally decentralized) determines the level of dynamicity required in the dispatching infrastructure.

Nevertheless, other challenges are peculiar to mobility, such as those related to the physical communication media. Wireless communication removes the need for cables and therefore is a key enabler of mobility. However, the price to pay for this freedom is lower performance and reduced reliability, which must often be taken into account at the higher application layers. For instance, unicast communication is the fundamental building block for many distributed applications in fixed environments, where it enjoys an efficient network implementation. On the other hand, in mobile scenarios multi-hop unicast is often expensive, in that it often requires several local broadcasts (and corresponding replies) to find a suitable route [35]. Therefore, it should be used sparingly in the development of middleware for these scenarios.

Similarly, the communication links of a conventional distributed system are often thought of as fairly reliable. For instance, the fault model assumed by many systems and protocols (notably including TCP) is one where communication failures are rare and transient, i.e., the communication target is assumed to become reachable again. Instead, in mobile scenarios disconnections are frequent, not only because the communication media is more sensitive (e.g., to the fluctuations in the propagation of radio waves as induced by the environment) but also because disconnection is no longer an *accident*, rather it is often deliberately induced by the user or application, e.g., to save battery power[3]. Moreover, failures are not guaranteed to be transient: for instance, cars moving on a highway in opposite directions may never meet again. Reliability is usually taken into account at the network level. However, in the field of mobility it is often useful to reduce the size of the network stack by blurring the distinction among levels, for the sake of reducing the system footprint and enable optimizations (e.g., reduce the use of unicast). Moreover, in the specific case of publish-subscribe, another challenge to reliability comes directly from the application level, where the messages being routed on the overlay network

[3]Power management is another relevant issue in mobility, which affects not only the host, but also network communication. However, in this paper we do not have the opportunity to touch upon these issues.

may get lost along stale routes due to the topological reconfiguration induced by mobility. The net result of these considerations is that the design of publish-subscribe middleware must often deal directly with reliability.

Finally, besides posing challenges to the implementation of the core communication layer, mobility brings along the necessity of a new way to approach the development of distributed applications, i.e., one that is *context-aware*. By definition, mobile hosts change their location in the physical space, and in doing so experience a different *context*, in terms of physical (e.g., temperature, light, reachable hosts) or logical (e.g., application services) constituents of the environment. Devising programming abstractions to properly capture, disseminate, and exploit context is an open research problem. Publish-subscribe, and in particular content-based systems, appear to provide a sound foundation for many approaches, thanks to their decoupling and reactive paradigm of interaction.

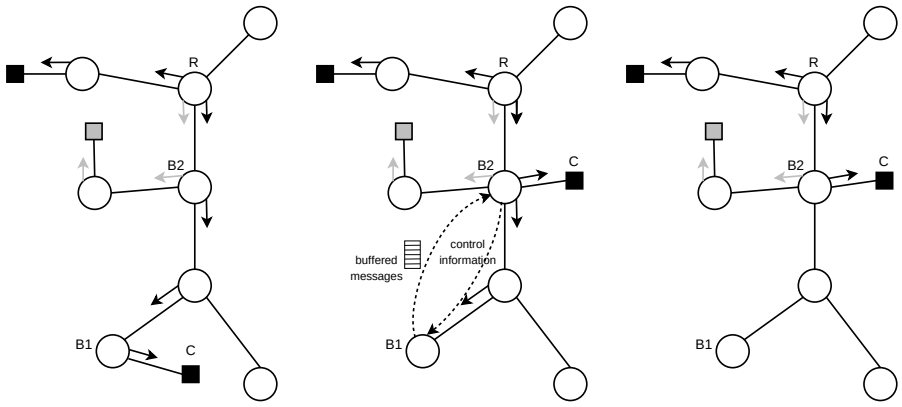
The rest of paper analyzes many of these issues in more detail.

4 Dealing with Mobile Clients

The first and simplest form of mobility that should be supported by a publish-subscribe middleware tailored to mobile scenarios is that of clients, by offering them the possibility to disconnect from the dispatching infrastructure and reconnect from a different place at a later time. This facility is fundamental to effectively support those scenarios of mobility, such as nomadic computing, which involve only the leaves of the system, i.e., the clients, as described in Section 3. In these situations, the publish-subscribe middleware must offer appropriate mechanisms to make mobility transparent to the other components, reconfiguring routing and storing messages addressed to the moving clients until they reconnect.

In the presence of a centralized dispatcher, supporting mobile clients is just a matter of buffering messages addressed to disconnected clients until they reconnect. The problem becomes much harder in the presence of a distributed dispatcher. In this case, a client must be allowed to disconnect from the broker currently acting as its entry point to the dispatching network, and to later reconnect to a broker potentially different from the previous one, which is usually chosen as the closest one to the new location of the client. To support this form of mobility, not only the publish-subscribe middleware must buffer messages addressed to the client while it is disconnected, but it must also be able to change the brokers' subscription tables (see Section 2) when the client reconnects. This requires a distributed protocol that coordinates the involved brokers and avoids loosing (or duplicating) the messages sent while the reconnection process is running.

Jedi [17] was the first distributed publish-subscribe middleware to offer this form of mobility. It adopts a hierarchical forwarding routing strategy and expects clients to proactively inform the middleware when moving away from or arriving at a broker. Figure 2 illustrates the procedure taking place when a



(a) Before. (b) During. (c) After.

Figure 2: Dealing with client mobility in Jedi: the situation before, during, and after the migration of a client C.

client C moves, and therefore detaches from broker B_1 and reattaches to B_2. Upon disconnection, B_1 starts buffering the messages addressed to C. When C reconnects at B_2, the latter initiates and coordinates the distributed protocol to rearrange the routing information and retrieve the messages buffered at B_1. This protocol consists of the following steps. First, B_2 re-propagates the subscriptions held by C, to set up the new routes that will steer messages toward the new location of C. Any message received as a consequence of these subscriptions is buffered at B_2 until the reconnection process ends. After the new routes are in place, B_2 asks B_1 to stop buffering messages, to remove C's subscriptions, and to forward the buffered messages. These messages, together with those buffered at B_2, represent the whole set of messages circulated in the system during the migration of C. Some duplicates may be present in this set because the old routes and the new ones co-existed for a short time, as shown in Figure 2(b). However, these duplicates are easily detected and discarded at B_2. The filtered set of messages is finally sent to C, ending the reconnection process.

Similar distributed protocols, albeit in the context of a subscription forwarding routing strategy, are adopted by the extended version of Siena in [7], Elvin [43], REBECA [24], and by the system described in [39].

5 Dealing with Mobile Brokers: An Integrated Approach

As mentioned in Section 3, dealing with mobility scenarios that make no assumptions about the stability of the infrastructure requires an entirely differ-

ent approach from the one discussed in the previous section. The topological reconfiguration induced by mobility disrupts the very dispatching infrastructure, therefore new solutions are required to preserve its operation and yet support its dynamicity.

The reconfiguration problem we address in this section can be defined informally as *to adapt the dispatching infrastructure of a distributed publish-subscribe system to changes in the topology of the underlying physical network, and to do so without interrupting the normal system operation*. In the following, we focus on content-based systems that adopt a subscription forwarding strategy and an unrooted tree overlay, since these are assumed by the majority of existing systems. For these systems, the reconfiguration problem stated previously can be broken down into three subproblems, namely:

1. repairing the overlay dispatching network, to retain connectivity among brokers without creating loops;
2. reconciling the subscription information held by each broker and used for routing messages, to keep it consistent with the topological changes above, without interfering with the normal processing of subscriptions and unsubscriptions;
3. recovering messages lost during reconfiguration.

In this section we present solutions to these problems, based on our own research on the topic [13, 14, 18, 20, 27, 33, 36]. To reduce complexity, each problem is addressed separately by leveraging off the fact that the three problems are orthogonal. When the techniques we describe here to solve each problem are combined in a single coherent system (e.g., the REDS system we describe in Section 6), they provide an integrated solution to the overall problem of dealing with mobile brokers.

5.1 Repairing the Overlay

Given that the overlay network we consider is a tree, there are two options to consider for repairing: to allow cycles to form and remove them later, or to disallow the formation of cycles. We choose the second approach because it is most appropriate when considering updates to the subscription tables, as seen in the next section.

We also consider two different types of failures: link and broker. From a theoretical perspective, link failure creates two trees with exactly the same nodes as before the link break. Repair, therefore, involves adding a link with endpoints in each of the two trees. Failure of a node with n neighbors results in n partitions, which require the addition of $n - 1$ new links.

The main challenge to address in repair of the overlay network is the selection of these links to repair the tree. We have developed two approaches, the first specifically for mobile ad hoc networks and the second for dynamic networks in which connectivity exists between each pair of brokers (e.g., peer to peer networks). In the following, we consider these two scenarios, separately.

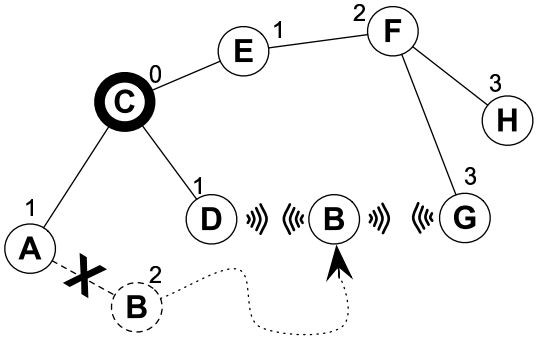


Figure 3: Overlay network in a mobile environment where B moves out of contact with A and into contact with D and G. All brokers are labeled with their numerical depth from the leader, C.

5.1.1 Mobile Networks

Our work to build and maintain an overlay network in a mobile environment is based on the prior study of multicast in MANETs. In particular, we started from the MAODV (Mobile Ad hoc On-Demand Distance Vector) protocol [41], since it focuses on building and maintaining a single tree, containing the mobile nodes participating in a multicast group.

In MAODV, one node is identified as the leader, and all nodes know their distance from it in the tree. When a link breaks (or a node fails, causing several links to break), the node(s) farther from the leader initiate the reconnection process, searching for a link that will reconnect their subtree to the subtree of the parent through a node with depth less than or equal to their own. This constraint guarantees that insertion of a link will not create a cycle. For example, in Figure 3, if the link fails between A and B, B will search for a new link between its subtree and the subtree of its parent A. D satisfies the depth constraint, consequently the link (B, D) is added and the depths of all nodes are updated.

Identification of potential links is accomplished by broadcasting a route request (RREQ) message a small number of hops from B. Any node with a depth less than or equal to B's depth responds with a route reply (RREP) message that follows the reverse path of the RREQ, identifying the path between the two trees. In MAODV, nodes not participating in the multicast group may also serve as routers in the multicast tree. Our approach [33], instead, assumes that all nodes act as brokers in the publish-subscribe network; a similar assumption as made in [30]. With this assumption, we have designed more efficient mechanisms for reconnecting the tree, specifically changing the propagation rules of the RREQ message and altering the selection criteria for the new link.

In our approach, members of the tree are allowed to propagate the RREQ message, an act disallowed in MAODV to prevent the formation of loops. This

forwarding of the RREQ extends the limits of the search for a broker with suitable depth, and the identification of a path between it and the requesting broker. To prevent the introduction of loops, we prohibit the RREQ from being propagated across a non-tree link more than once. In other words, the RREQ cannot propagate from B to G (a non-tree link) and then to H (a second non-tree link). This disallows the addition of links (B, G) and (G, H) a situation that forms cycles in the overlay tree. However, when G receives the RREQ from B, it is forwarded to F along the overlay tree, a situation explicitly prohibited in MAODV. In this case, F meets the depth criteria and responds to the RREQ message, indicating the possible addition of the link between B and G, an option for restoring the tree that MAODV does not identify.

Our second extension to MAODV is the selection criteria for the new link. In Figure 3, both D and F reply to the RREQ message with options for repairing the tree and B must choose one. The main selection criteria in MAODV is the length of the path to the tree, which in MAODV may include several nodes that are actually not part of the multicast group. Instead, in our approach where all nodes acting as brokers, the distance to the tree is always one hop. Therefore, we adopt a selection criteria that minimizes the effect of the reconfiguration on the subscription tables. Specifically, the reply with the shortest path between the endpoint of the old link on the tree (A) and the new endpoint (D or G) is selected. In the example, D's reply is selected because the path length from A to D is shorter than from A to G.

A more detailed description of this approach, together with experimental results collected from our implementation, can be found in [33].

5.1.2 Fixed Networks

As an alternative to networks where connectivity is determined by broker proximity, we have also devised a protocol [27] for a fixed network scenario, as found in peer-to-peer networks. In this scenario, dynamicity comes from addition and removal of brokers, not links, and the presence of a fixed network enables the addition of a link between any pair of connected brokers. Our solution is again inspired by MAODV, but adapted to the aforementioned scenario requirements. Furthermore, since we have greater control over connectivity, we also enforce a maximum degree on each broker, therefore limiting its message forwarding burden.

In our approach, we exploit three types of repair procedures: local, global, and root-specific. In a local recovery, only brokers close to the recovering one are involved. Global recovery reaches brokers anywhere in the tree. Finally, root-specific protocols come into play only when the root broker fails.

Local recovery exploits the fact that all brokers know the identities of their siblings as well as the identities of some of their direct ancestors (brokers on the path between itself and the root). When a broker fails, the tree can be reconnected by linking its former children to each other and at least one of these children to an ancestor. We have developed protocols that balance the broker degree, preventing all brokers from connecting to the same ancestor (creating

a star network with high broker degree) and similarly preventing all but one child from connecting to one another (creating a line with low broker degree).

Our local recovery procedure also allows a broker to refuse a request if the addition of the broker as a child will increase its degree beyond a predefined limit. In this case, the request to find a parent is forwarded downstream from the refusing broker in hopes of finding a broker that has not yet reached its maximum degree. This technique is surprisingly effective, exploiting the trend that brokers farther from the root have lower degrees.

Global recovery comes into play when local recovery fails because broker degree requirements cannot be met or because a new parent cannot be identified among the siblings and the ancestors, a case that arises when a cluster of brokers fails. In these situations, the broker seeks a new parent from a cache that it maintains of other brokers in the tree. This cache is populated, for example, by recording the source of messages propagated over the tree. To find a new parent, a broker is selected from the cache and sent a request to allow the requesting broker to become a child. To avoid loops, we adopt an algorithm based on the notion of tree depth, similar to MAODV. If the broker has a lower depth than the requesting broker, it can accept to become the new parent. Otherwise, it can forward the request upstream to find a broker with lower depth, downstream to find a broker with lower broker degree, or simply reject the request.

It may still happen that a broker cannot identify a new broker to serve as its parent. In this case, it declares itself to be a new root, creating its own tree. Because our goal is to maintain a *single* overlay tree, we need a mechanism to merge trees when they discover each another. For this, we assign an identifier to each tree. After a broker has declared itself to be a root, it periodically contacts brokers in its global cache. If a broker is found with a different tree identity, the two trees are merged.

This notion of merging trees can also be exploited in the case of root failure. When the root fails, all its former children declare themselves to be roots of their subtrees. They then exploit their global cache to identify the other subtrees and re-merge the tree. Unfortunately, this may take a long time, during which message routing on the tree is disrupted. Therefore, we defined a protocol specific to root-failure, essentially electing a new root among the former children of the old root and allowing the remaining children to connect to this new root or to one another.

By combining local, global, and root-specific protocols we can keep a tree connected despite brokers frequently being added and removed. A full evaluation of the effectiveness of these techniques is available in [27].

5.2 Reconciling Routing Information

After ensuring the maintenance of the overlay tree, the next step is maintaining the subscription tables to allow messages to continue to reach the subscribers. Here we consider protocols that address link loss rather than broker loss, since the latter case can be addressed as a combination of several link repair actions.

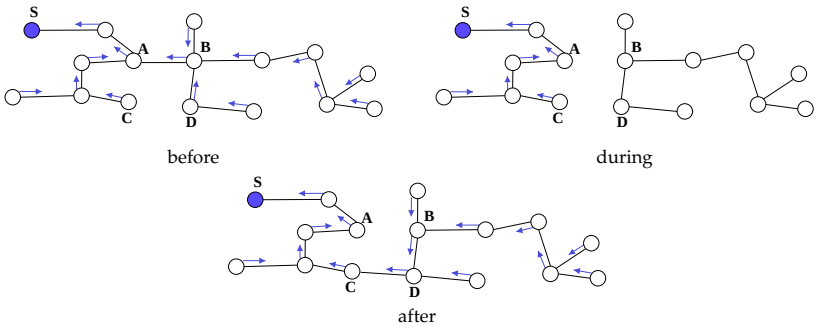


Figure 4: A dispatching tree of before, during and after a reconfiguration performed using Strawman. The shaded broker is a subscriber. Arrows indicate the propagation direction for messages.

When a link fails between a pair of brokers the overlay management protocols described in the previous section take on the responsibility of finding the replacement link. Once this link is found, the subscription tables must be updated so that all messages that traversed the now broken link are sent across the new link to reach the subscribers on the other subtree. We have developed a series of protocols to accomplish this, each with different requirements from the overlay management protocols and with different assumptions about the environment [18,20,36].

The first solution, which we refer to as Strawman protocol, is the only prior proposal in the literature [8]. This approach utilizes only the usual publish-subscribe subscription and unsubscription messages. When a link disappears, a broker behaves as if it received unsubscription messages from the former neighbor, updating its subscription table and propagating the unsubscription message if necessary. This has the effect of stopping message forwarding across a broken link. When the new link is added, its endpoints send subscriptions to one another for all entries in their subscription table, allowing messages to flow across the new link.

While this approach successfully reconfigures the subscription tables, it may cause unnecessary overhead. For example, consider the scenario in Figure 4 in which only one broker in a subtree is a subscriber. When the link breaks between A and B, the unsubscription process removes all entries in the subscription tables of the brokers in B's subtree. When the subscription process begins across the link (C, D), it reinserts most of these entries exactly as before, creating unnecessary overhead to remove many subscriptions that are immediately reinserted. To overcome this, we experimented with delaying the unsubscription process until the subscription process is complete. This reversal technique, that we termed Deferred Unsubscription, is effective in reducing the overhead of reconfiguration, up to 50% over Strawman in simulations studies characterized by a large number of reconfigurations. In the simple example

above, it prevents the removal and replacement of all subscriptions on the subtree to the right of B and D as well as the broker above B. Details about two different mechanisms for deferring subscriptions can be found in [36] and [20].

Analysis of the publish-subscribe behavior reveals that reconfiguration is restricted to the brokers on the path between the endpoints of the old and new links, termed *reconfiguration path* [18]. The subscription tables of all other brokers remain unchanged. In Figure 4, the reconfiguration path is composed of the brokers from A to C, across the new link from C to D, and from D to B. To exploit this property, we designed a protocol [18] that starts at one endpoint of the old link and moves along the reconfiguration path, updating the subscription tables as it progresses. One drawback of this protocol is the requirement that the path remains intact during the entire reconfiguration. If a second link fails on the reconfiguration path, the reconfiguration messages stop propagating and the system is left with inconsistent subscription tables. A second drawback is the need to know the identity of the brokers on the reconfiguration path, an additional requirement for the overlay management protocol. Finally, this protocol is complex when considering the details to address the subscription and unsubscriptions that occur during the reconfiguration. On the other hand, this protocol can achieve overhead reductions up to 78% over Strawman in scenarios where reconfigurations do not overlap.

To bridge between the resilient delayed unsubscription protocol and the efficient reconfiguration path protocol, we have designed a new protocol that exchanges information among the brokers on the the old and new links, called Informed Link Activation. Specifically, the endpoints on the old link send the contents of their subscription tables to the endpoints of the new links. Combining this with their own subscription tables, the endpoints of the new link calculate which subscriptions to send across the new link. Again, this is complicated by the insertion and removal of subscriptions during reconfiguration, still the protocol is not as complex as Reconfiguration Path protocol. With these approaches that share information between the old and new link, we have shown that few brokers outside the reconfiguration path are affected by reconfiguration, enabling an overhead reduction by up to 76%, similar to Reconfiguration Path but in the presence of concurrent reconfigurations.

Each of these protocols operates with varying expectations from the overlay management protocol and tolerance for change during tree repair. This leads to a number of tradeoffs that must be considered when selecting the protocol for a given system. For example, although Reconfiguration Path has clear advantages with respect to reduction of overhead, it adds the burden to the overlay management protocol to identify all nodes on the path, and requires the environment to keep the path stable during reconfiguration. On the other side, Deferred Unsubscription makes no assumptions about either stability or knowledge passed from the overlay management protocol, however its overhead reduction is not as significant. The Informed Link Activation protocol is in between Reconfiguration Path and Deferred Unsubscription both in terms of overhead reduction and required knowledge. Notably, the endpoints of the old link must be informed of the identities of the endpoints of the new link in

order to send information to aid reconfiguration.

In summary, our suite of protocols provides many options to the system designer, who can select the most appropriate protocol based on the characteristics of the deployment environment.

5.3 Recovering Lost Messages

The last problem hampering content-based publish-subscribe on a dynamic topology is to recover lost messages. Even in the presence of reliable links, messages can be lost due to the reconfiguration of the dispatching network, as routing tables are changed while a message is in transit and therefore may cause its forwarding along stale routes. In this section, we describe a solution based on epidemic algorithms which does not make any assumptions about the cause of message loss and therefore enjoys general applicability.

The idea behind *epidemic* (or *gossip*) algorithms [6, 21] is for each process to communicate periodically its partial knowledge about the system "state" to a random subset of other processes, thus contributing to build a shared view of the global state. The interaction between hosts can exploit a push or pull style. In a *push* style, each process gossips periodically to disseminate its view of the system. Instead, in a *pull* style each process requests the transmission of information from other processes. Usually, a push approach exploits gossip messages containing a *positive* digest, while a pull approach exploits a *negative* digest (i.e., containing the portion of the state known to be missing). Regardless of the scheme adopted, the probabilistic and decentralized nature of epidemic algorithms brings many desirable properties: a constant, equally distributed load on the processes in the system, therefore improving scalability; resilience to changes in the system configuration, including topological ones; a simple implementation and low computational overhead.

In our case, the state to be reconciled is the set of messages that have appeared in the system. Nevertheless, the nature of content-based publish-subscribe systems brings additional complexity because, unlike subject-based publish-subscribe and IP multicast, not only are messages not bound to a subject or group determining their routing, but they may also match multiple subscriptions instead of a single group. Together, these features greatly complicate the task of identifying the subset of brokers that may hold a missing message.

The solutions we describe share a common structure. Each broker periodically starts a new "gossip round", during which it contacts other brokers potentially holding a copy of the lost message. The broker playing this *gossiper* role builds a gossip message and sends it along the dispatching tree. The content of the gossip message and its routing by the other brokers along the tree vary according to the solutions we describe next. We assume that each broker caches the messages received, and that a unicast mechanism is available for sending missing messages (e.g., using the reverse path of gossip messages along the tree, or through an out-of-band transport layer).

Push. Our first solution uses proactive gossip push with positive digests. At

each gossip round, the gossiper chooses randomly a filter p from its subscription table, constructs a digest of the identifiers[4] of all the cached messages matching p, builds a gossip message containing the digest, and labels it with p. The gossip message is then propagated along the dispatching tree as if it were a normal message matching p. The only difference is that, to limit overhead, the gossip message is forwarded only to a random subset of the neighbors subscribed to p. To increase the chance of eventually finding all the brokers interested in the cached messages, therefore speeding up convergence, p is selected from the whole subscription table instead of just the local subscriptions.

When a broker receives a gossip message labeled with p, it checks if it is subscribed to this filter and if all the identifiers contained in the digest correspond to previously received messages. The identifiers of the missed messages are included in a request message sent to the gossiper, which replies by sending a copy of the messages. Both messages are exchanged through the unicast channel mentioned above.

Pull. A pull approach implies the ability to detect lost messages. In subject-based systems, this is easily achieved by using a sequence number per source and per subject. In content-based systems this task is complicated by the absence of a notion of subject and by the fact that each broker receives only those messages whose content matches the filters it is subscribed to. As detailed in [14], this problem can be solved by tagging each message with (1) the identifier of its source, (2) information about all the filters matched by the message and, (3) for each filter, a sequence number incremented at the source each time a message is published for that filter. This information is bound to each message at its source—an opportunity enabled by subscription forwarding, where subscriptions are known to all brokers. Event loss is detected when a broker receives a message matching a filter p whose sequence number, associated to p in the message identifier, is greater than the one expected for p from that message source.

Based on this detection technique, we defined two approaches exploiting different routing strategies: one steers gossip messages towards the subscribers, while the other steers them towards the publisher.

- *Subscriber-based Pull.* Upon detecting a lost message, a broker inserts the corresponding information (i.e., source, matched filter, and sequence number associated to filter and source) in a buffer $Lost$. When the next gossip round begins the broker now a gossiper, picks a filter p among those associated to local subscriptions[5], selects the messages in $Lost$ matching p, and inserts the corresponding information in a digest attached to a new gossip message. Finally, the gossip message is labeled with p and routed as in the push solution. A broker receiving the gossip message

[4]The pair given by the source identifier and a monotonically increasing sequence number associated to the source is sufficient.

[5]Unlike push, subscriptions are not drawn from the whole subscription table, since here the goal is to retrieve messages relevant to the gossiper rather than disseminating information about received messages.

checks its cache against the requested messages and, if any are found, sends them back to the gossiper. Note how the replying broker need not be subscribed to p. In fact, the broker could have received the gossip message because it sits on a route towards a subscriber for p, and could have received (and cached) some of the messages missed by the gossiper because they match also a filter $p' \neq p$ the broker is subscribed to.

- *Publisher-based Pull.* This scheme requires that published messages are cached not only by the brokers that received them but also by the source, and that the address of each broker encountered on the route towards a subscriber is appended to the published message. Processing occurs similarly to the previous one, but gossip messages are routed towards publishers instead of subscribers.

These three solutions are described in greater detail as well as formalized in [14]. Moreover, in [13] we evaluated their performance through simulation. The results confirmed that the approach is effective, and provided insights about how to tune the parameters (most notably, the interval between two gossip rounds and the size of the message cache) to achieve the desired level of reliability. Interestingly, we discovered that neither of the pull solutions alone guarantees a satisfactory performance. Instead, the combination of the two, performed by randomly choosing subscriber- or publisher-based pull according to a given probability, performs similarly to push albeit with lower overhead in the case of infrequent reconfiguration.

6 REDS: Mobile Publish-Subscribe in Practice

Developed at Politecnico di Milano, REDS (REconfigurable Dispatching System) [19] puts the mechanisms and algorithms described in the previous sections into practice. REDS is publicly available at `zeus.elet.polimi.it/reds` as open source under the LGPL license, and is implemented entirely in Java. Its distinctive and innovative feature is reconfigurability, a property made available on two different planes. The first concerns the configuration of the middleware architecture, enabling the selection of different mechanisms (e.g., the format of messages and filters or the routing strategy) for different deployment scenarios. The second concerns the dynamic reconfiguration of the topology of the REDS distributed dispatcher, addressing the problems of maintaining the overlay network of REDS brokers in the face of topological changes, efficiently restoring stale subscription information, and recovering messages lost during the reconfiguration, as described in Section 5.

To achieve these goals, REDS is conceived of as a framework (in the object-oriented sense) of Java classes, which allows programmers to easily build a publish-subscribe middleware explicitly tailored to their application domain. In particular, REDS defines the architecture of a generic broker organized as a set of components implementing well-defined interfaces that represent several

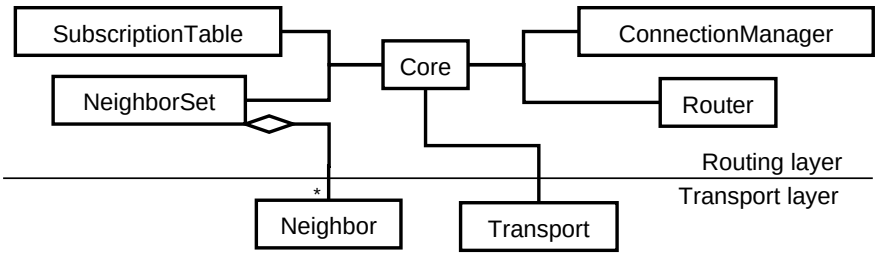


Figure 5: The architecture of a REDS broker.

aspects of a publish-subscribe system. For instance, for what concerns messages and filters, REDS defines two interfaces encompassing the minimal set of methods required by a publish-subscribe broker to operate. By implementing these interfaces, developers are free to define their own message formats and, even more important, filters, without the need to change the rest of the system. Here, however, we are interested in the components used to build the brokers constituting the REDS distributed dispatcher. As shown in Figure 5, each REDS broker is organized as a set of components grouped into two layers, the *transport* and the *routing* layer.

Transport. The transport layer encapsulates the mechanisms used to transport messages, (un)subscriptions, and any other kind of broker-specific control messages through the network. In performing this task, it hides the wire protocol adopted to move data around and the mechanisms used to address and access brokers and clients, and to setup the dispatching network in case the dispatcher is distributed.

It includes an instance of the `Transport` component and a set of `Neighbors`. The `Transport` component is in charge of receiving incoming requests from neighboring brokers or clients (e.g., connection requests, subscriptions, etc.), interpreting them, and calling the appropriate methods on the `Core` component that, as we describe below, is the pivot of the whole architecture. Similarly, `Neighbor` instances are used internally by the components of the routing layer as proxies to interact with the broker's neighbors, therefore hiding the details concerned with accessing the underlying network layer.

The current version of REDS provides two different implementations for the transport layer, and consequently for the two components `Transport` and `Neighbor`: one using TCP links to connect each broker with its neighbors, and one based on UDP datagrams.

Routing. The routing layer includes three components: `Core`, `Router`, and `ConnectionManager`, which share two data structures: `SubscriptionTable` and `NeighborSet`. `SubscriptionTable` plays a critical role in recording the subscriptions received by the broker's neighbors. By encapsulating

the algorithm used to efficiently match messages against a set of filters, its implementation has a great impact on the performance of the broker. The `NeighborSet` is a convenience data structure provided as a way to simplify the task of accessing, from within the routing components, information concerned with the broker's neighbors represented by `Neighbor` instances.

As its name suggests, the `Core` is the central element of a REDS broker. It holds the two aforementioned data structures and mediates the communication among the other components, which therefore do not have direct visibility to each other. This design choice yields two benefits: it increases the decoupling among components, which need to be aware only of the existence of the `Core`, and provides a central place through which all communication is funneled, therefore opening opportunities for transparently intercepting and modifying messages before redirecting them to the intended components. As an example, we are currently exploiting this possibility in the implementation of the epidemic algorithm described in Section 5.3.

The `Router` is in charge of implementing the specific routing strategy, e.g., one of those discussed in Section 2. Its methods are invoked by the `Transport`, through the `Core`, to notify it that a subscription, unsubscription, or message have been received from one of the broker's neighbors and must be routed according to the strategy it encapsulates.

The `ConnectionManager` is the key component of our architecture enabling support for publish-subscribe on a dynamic topology. It is in charge of (i) maintaining the overlay dispatching network connected and (ii) efficiently rearranging the brokers' subscription tables when the topology of the network changes. It operates in a reactive way, being notified by the `Transport` (through the `Core`) when a new client or broker requests to connect or disconnect, and, most important, when the transport cannot reach a neighbor that was formerly connected. The protocols described in Section 5.1 and 5.2 are currently implemented in REDS as specializations of this component.

7 Related Approaches

In this section we discuss other approaches to publish-subscribe in mobile scenarios, by focusing on the few other publish-subscribe middleware capable of tolerating reconfigurations of the dispatching network as well as faulty links, on those providing solutions specific to content-based publish-subscribe on MANETs and, finally, touching on how location—a fundamental aspect in mobile scenarios—can be introduced into the publish-subscribe communication model.

7.1 Reconfigurable and Fault-Tolerant Publish-Subscribe

The ability to deal with dynamic reconfiguration of the dispatching network topology is not common in content-based publish-subscribe middleware.

The most relevant exception is Hermes [37, 38], a scalable and reconfigurable publish-subscribe middleware that uses peer-to-peer techniques to build and maintain its overlay routing network. Hermes provides a slightly limited form of content-based routing, termed "type and attribute based" routing [23]. Type-based routing is comparable to subject-based routing, but preserves inheritance among message types. On top of this routing mechanism, Hermes adds content-based filtering on message attributes. Each message type is associated to a rendezvous point, which takes the same role as the core node in core-based tree multicast [2]. The Hermes peer-to-peer substrate associates a specific Hermes broker to any rendezvous point and helps in building the dispatching tree associated to the associated message type. The self-organization and stabilization features of this peer-to-peer substrate allow Hermes to handle dynamic addition, removal, and failure of brokers. However, Hermes does not address the problem of recovering messages lost during reconfiguration.

This latter problem is instead addressed by the developers of Joram [3] and Gryphon [5], which focus on fault-tolerance and reliability by allowing a set of brokers to operate as a redundant cluster, but not specifically for mobility. A new feature in Joram 4.2 allows a set of brokers to be grouped together and operate as a single, redundant cluster to transparently handle network handover and broker fail-over. The Joram brokers that are part of the same cluster communicate and coordinate by using JGroups [4], a toolkit for reliable multicast communication developed at Cornell University. Similarly, an approach based on a redundant network of brokers to deal with link failures and broker crashes has been recently proposed for the Gryphon [46] system.

7.2 Publish-Subscribe on MANETs

In Sections 4 and 5 we described a comprehensive set of approaches to dynamically reconfigure a tree of brokers and clients to efficiently support publish-subscribe interactions in different mobile scenarios, exhibiting various degrees of mobility. However, in these approaches message dispatching still relies on a tree-shaped overlay tree. In a MANET environment, and especially if hosts move frequently, the overhead of maintaining the broker tree may overcome the advantages offered by this topology. However, very little literature addresses this problem.

In [29] the spectrum of approaches (from centralized to distributed) to publish-subscribe is described, and some possible extensions to mobile environments and particularly MANETs are briefly analyzed. The paper does not provide any complete solutions to the problem, but offers a good starting point by eliciting the problems involved. Similarly, the preliminary work described in [42] analyzes the issues involved in designing a publish-subscribe middleware for MANETs, focusing specifically on the routing problem.

Instead, the work in [30] describes a distributed protocol to build optimized publish-subscribe trees in wireless networks. The authors make rather constraining assumptions, namely they expect to have knowledge about the placement of publishers (which must become the root of the dispatching tree) and

about the statistical ddistribution of messages with respect to subscriptions (to satisfy the optimality requirements of routing). Moreover, they consider only quasi-static scenarios where nodes move only occasionally and then settle down for a period on the order of minutes. Another proposal [12] describes a mechanism to reconfigure an overlay dispatching network depending on the changes in the physical topology of the MANET and on the current brokers' load. However, each broker must be provided with a global view of the network topology, the mechanism does not handle partitions, and it requires an underlying unicast protocol.

The problem of providing content-based publish-subscribe for highly mobile scenarios without relying on a tree overlay has recently been tackled by our research group. In [15], the authors describe a semi-probabilistic routing algorithm that relies on an overlay network of brokers organized in an undirected connected graph. This topology is easier to maintain than a tree and it is intrinsically more tolerant to reconfigurations and faults since it provides multiple routes between any two brokers. Routing is partially deterministic and partially probabilistic. Subscriptions are forwarded as in subscription forwarding, but only up to a given distance (i.e., number of hops) from the subscriber, thus providing accurate routing information within a certain horizon from the subscriber. Along its route, a message is routed using this deterministic information—if available. If there is no such information to determine the next hop, the decision is taken probabilistically, by forwarding the message along a randomly selected subset of the available links. The simulations in [15] confirm that a proper tuning of the span of the subscription horizon and of the fraction of randomly selected links yields very good performance (in terms of event delivery and overhead) even in highly dynamic scenarios. In particular, the semi-probabilistic approach performs better than a purely probabilistic (or deterministic) approach.

Instead, the idea in [1] is to let each broker autonomously decide about forwarding messages, based on its estimated distance from the closest subscriber, and to perform forwarding by using the broadcast facility provided by wireless network cards. This allows other neighboring brokers to decide if they must forward the message or if they should cancel forwarding. In particular, in a MANET with quickly moving mobile nodes the distance between two such nodes can be estimated by measuring the time since they were most recently in communication range. Consequently, routing works as follows: each broker listens for messages broadcast by neighboring brokers. When it receives such a message, it stores it and delays forwarding for a time interval proportional to the estimation it made of its distance from the closest subscriber. During this time interval, forwarding is canceled if a message with the same identifier is forwarded by some neighboring broker (to avoid unnecessarily flooding the network). When the delay expires, if forwarding has not been canceled, the message is broadcast to the neighboring brokers, which reason similarly. The simulations in [1] confirmed that this is an efficient technique: it exploits the broadcast nature of wireless communication to send multiple copies of the same message via a single transmission; it avoids the burden of link breakage

detection and, even more important, it provides an intrinsic resilience to the topological changes caused by the mobility of the nodes.

7.3 Location and Context-Aware Publish-Subscribe

The very notion of mobility is tightly coupled with the notion of *location*. Indeed, in mobile scenarios, the ability to send messages only toward specific locations or that of subscribing to messages published by components located in specific areas could be beneficial to implement interactions that take into account mobility. Unfortunately, commonly available publish-subscribe middleware does not offer location-based services as part of the API and only few systems address the problem.

In [16] the authors provide a categorization of possible location-based publish-subscribe services and describe an algorithm to introduce them efficiently in a distributed publish-subscribe middleware, using a subscription forwarding routing strategy. In this scheme, each broker is provided with a *location table* used to route location-aware messages and subscriptions. Information about the actual location of publishers and subscribers is forwarded along the network of brokers to populate each broker's *location table*. This information is used both at subscription and publish time. If a component subscribes to messages coming from a specific area A, location tables are used to limit forwarding of the subscription only toward A. Similarly, if a component publishes a message toward area A, location tables are used (together with conventional subscription tables), to route the message only toward subscribers located in area A, if any. A similar approach is reported in [24], where the authors describe an extension to the REBECA [26] middleware to implement location-based subscriptions. The main difference between this approach and the previous one is that [24] does not take advantage of information about the actual location of clients to limit forwarding of location-based subscriptions. As a consequence, location-based subscriptions flood the entire network of REBECA's brokers, potentially reaching areas that are not relevant. A different approach is pursued in [25], where a general notion of *scope* is introduced to structure message availability and notification by restricting the visibility of published messages to a subset of subscribers in the system—those in the requested scope. In principle, scope can be defined using location or other forms of contextual information, therefore obtaining a form of context-awareness similar to those described above.

Instead, STEAM (Scalable Timed Events And Mobility) [32] is a publish-subscribe middleware designed for deployment over MANETs. STEAM targets application scenarios that include a large number of application components that communicate using wireless technology in an ad hoc scenario. In STEAM, messages are valid in a certain geographical area surrounding the publisher. In other words, STEAM provides a special form of location-based publishing service, in which location is expressed relative to the publisher. The STEAM implementation is specifically tailored to MANETs and takes advantage of a proximity-based group communication service [31], which uses the

number of hops traveled by messages at the MAC networking layer to approximate distance.

The work in [11] tackles the different, but related, problem of efficiently filtering a stream of messages representing the current location of clients against a set of spatial predicates. The goal is to determine the set of clients that could be interested in receiving some messages based on their position. The authors propose a middleware based on a centralized spatial matching engine, which collects subscriptions and delivers them to clients. Clients are in charge of matching those subscriptions against their current position. The results of this process are given back to the engine.

Finally, a complementary approach is taken by Solar [10], a distributed publish-subscribe middleware explicitly developed to disseminate location, and more in general contextual information, to a set of distributed components. Therefore, the emphasis is not on constraining the propagation of messages and subscriptions based on location, rather, on using the publish-subscribe infrastructure to efficiently disseminate contextual data (e.g., gathered by sensors) that can be processed and used by the distributed application. Solar abstracts context information as messages and allows components to subscribe to the kind of information to be notified of when their context changes. Moreover, components may use Solar services to aggregate low-level context information into more expressive and easier to manage high-level ones.

8 Conclusions

The publish-subscribe model holds the potential to become of fundamental importance in mobile computing, but only if it the technology supporting it embodies the mechanisms and algorithms necessary to cope with the dynamicity of this environment. In this chapter, we presented the challenges posed by the mobile environment, described our own solutions for bringing dynamicity in content-based publish-subscribe technology, and surveyed alternative state-of-the art proposals in the field.

References

[1] R. Baldoni, R. Beraldi, G. Cugola, M. Migliavacca, and L. Querzoni. Structure-less content-based routing in mobile ad hoc networks. In *Proc. of the IEEE Int. Conf. on Pervasive Services*, Santorini, Greece, July 2005. IEEE Computer Society Press.

[2] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees. In *Proc. of ACM SIGCOMM'93*, San Francisco, CA, August 1993. ACM Press.

[3] R. Balter. Joram: The open source enterprise service bus. Technical report, ScalAgent Distributed Technologies, March 2004. `www.scalagent.com/pages/en/datasheet/040322-joram-whitepaper-en.pdf`.

[4] B. Ban. Design and implementation of a reliable group communication toolkit for Java. Technical report, Cornell Univ., September 1998. `www.cs.cornell.edu/home/bba/`.

[5] G. Banavar et al. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. of the 19th Int. Conf. on Distributed Computing Systems*, 1999.

[6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. on Computer Systems*, 17(2):41–88, 1999.

[7] M. Caporuscio, A. Carzaniga, and A.L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Trans. on Software Engineering*, 29(12):1059–1071, December 2003.

[8] 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.

[9] R. Chand and P.A. Felber. A scalable protocol for content-based routing in overlay networks. In *Proc. of the 2nd IEEE Int. Symp. on Network Computing and Applications*, page 123, Cambridge, MA, April 2003. IEEE Computer Society Press.

[10] G. Chen and D. Kotz. Solar: An open platform for context-aware mobile applications. In *Proc. of the 1st Int. Conf. on Pervasive Computing*, pages 41–47, Zurich, Switzerland, June 2002.

[11] X. Chen, Y. Chen, and F. Rao. An efficient spatial publish/subscribe system for intelligent location-based services. In *Proc. of the 2nd Int. Workshop on Distributed Event-Based Systems (DEBS)*, June 2003.

[12] Y. Chen and K. Schwan. Opportunistic brokers: Supporting efficient content delivery in mobile ad hoc networks, 2005. Submitted for publication. Available at `http://www.cc.gatech.edu/~yuanchen/`.

[13] 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. IEEE Computer Society Press, March 2004.

[14] P. Costa, M. Migliavacca, G.P. Picco, and G. Cugola. Introducing reliability in content-based publish-subscribe through epidemic algorithms. In *Proc. of the 2nd Int. Workshop on Distributed Event-Based Systems (DEBS)*, June 2003.

[15] P. Costa and G.P. Picco. Semi-probabilistic content-based publish-subscribe. In *Proc. of the 25th Int. Conf. on Distributed Computing Systems*, Columbus (OH, USA), June 2005. IEEE Computer Society Press.

[16] G. Cugola and J.E. Munoz de Cote. On introducing location awareness in publish-subscribe middleware. In *Proc. of the 4th Int. Workshop on Distributed Event-Based Systems (DEBS)*, June 2005.

[17] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. on Software Engineering*, 27(9):827–850, September 2001.

[18] 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 Symposium on Applied Computing (SAC'04)*, pages 1134–1140. ACM Press, 2004.

[19] G. Cugola and G.P. Picco. REDS: A reconfigurable dispatching system. Technical report, Politecnico di Milano, March 2005. Submitted for publication. Available at `www.elet.polimi.it/upload/picco`.

[20] G. Cugola, G.P. Picco, and A.L. Murphy. Towards dynamic reconfiguration of distributed publish-subscribe systems. In *Proc. of the 3rd Int. Workshop on Software Engineering and Middleware (SEM)*, LNCS 2596, pages 187–202. Springer, May 2002.

[21] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *Operating Systems Review*, 22(1):8–32, 1988.

[22] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 2(35), June 2003.

[23] P.T. Eugster, R. Guerraoui, and C.H. Damm. On objects and events. In *Proc. of the OOPSLA'01 Conf. on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 254–269, Tampa Bay (FL, USA), October 2001.

[24] L. Fiege, F.C. Gartner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Proc. of the 4th ACM/IFIP/USENIX Int. Middleware Conf.*, Rio de Janeiro, Brazil, June 2003.

[25] L. Fiege, M. Mezini, G. Mühl, and A.P. Buchmann. Engineering event-based systems with scopes. In B. Magnusson, editor, *Proc. of the 16th European Conference on Object-Oriented Programming (ECOOP02)*, volume 2374 of *LNCS*, pages 309–333, Malaga, Spain, June 2002. Springer.

[26] L. Fiege, G. Mühl, and F.C. Gärtner. Modular event-based systems. *Knowledge Engineering Review*, 17(4):359–388, 2002.

[27] D. Frey and A.L. Murphy. Maintaining publish-subscribe overlay tree in large scale dynamic networks. Technical report, Politecnico di Milano, 2005. Submitted for publication. `www.elet.polimi.it/upload/frey`.

[28] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Trans. on Software Engineering*, 24(5), 1998.

[29] Y. Huang and H. Garcia-Molina. Publish/Subscribe in a mobile enviroment. In *MobiDe'01: Proc. of the 2nd ACM Int. Workshop on Data engineering for Wireless and Mobile access*, pages 27–34. ACM Press, 2001.

[30] Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *Proc. of the 4th Int. Conf. on Mobile Data Management (MDM '03)*, pages 122–140. Springer, 2003.

[31] M. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill. Towards group communication for mobile participants. In *Proc. of ACM Workshop on Principles of Mobile Computing (POMC'2001)*, pages 75–82, Newport, Rhode Island, USA, 2001.

[32] R. Meier and V. Cahill. STEAM: Event-Based Middleware For Wireless Ad Hoc Networks. In *Proc. of the 1st Int. Workshop on Distributed Event-Based Systems (DEBS)*, July 2002.

[33] L. Mottola, G. Cugola, and G.P. Picco. Tree-based overlays for publish-subscribe in mobile ad hoc networks. Technical report, Politecnico di Milano, 2005. Submitted for publication. `www.elet.polimi.it/upload/picco`.

[34] C. Perkins. IP Mobility Support. RFC 2002, The Internet Engineering Task Force, 1996.

[35] C.E. Perkins, editor. *Ad Hoc Networking*. Addison Wesley, 2000.

[36] G.P. Picco, G. Cugola, and A.L. Murphy. Efficient Content-Based Event Dispatching in Presence of Topological Reconfiguration. In *Proc. of the 23rd Int. Conf. on Distributed Computing Systems (ICDCS03)*, pages 234–243. ACM Press, May 2003.

[37] P.R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proc. of the 1^{st} Int. Workshop on Distributed Event-Based Systems (DEBS)*, July 2002.

[38] P.R. Pietzuch and J.M. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proc. of the 2^{nd} Int. Workshop on Distributed Event-Based Systems (DEBS)*, June 2003.

[39] I. Podnar and I. Lovrek. Supporting mobility with persistent notifications in publish-subscribe systems. In *Proc. of the 3^{rd} Int. Workshop on Distributed Event-Based Systems (DEBS)*, May 2004.

[40] D.S. Rosenblum and A.L. Wolf. A design framework for internet-scale event observation and notification. In *Proc. of the 6^{th} European Software Engineering Conf. held jointly with the 5^{th} Symp. on the Foundations of Software Engineering (ESEC/FSE97)*, LNCS 1301, Zurich (Switzerland), September 1997. Springer.

[41] E.M. Royer and C.E. Perkins. Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol. In *Proc. of the 5^{th} Int. Conf. on Mobile Computing and Networking (MobiCom99)*, pages 207–218, Seattle, WA, USA, August 1999.

[42] K.S. Skjelsvik, V. Goebel, and T. Plagemann. Distributed event notification for mobile ad hoc networks. *IEEE DSOnline*, 5(8), 2004.

[43] P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness—Transparent information delivery for mobile and invisible computing. In *Proc. of the IEEE Int. Symp. on Cluster Computing and the Grid*, May 2001.

[44] C-K. Toh. *Ad hoc mobile wireless networks*. Prentice Hall Inc., 2002.

[45] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proc. of the 24th Int. Conf. on Distributed Computing Systems*. IEEE Computer Society Press, March 2004.

[46] Y. Zhao, D. Sturman, and S. Bhola. Subscription propagation in highly-available publish/subscribe middleware. In *Proc. of the 5th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 274–293. Springer, 2004.

Gianpaolo Cugola received his Dr.Eng. degree in Electronic Engineering from Politecnico di Milano, where he spent most of his professional life. In 1998 he received the Prize for Engineering and Technology from the Dimitri N. Chorafas Foundation for his Ph.D. thesis on Software Development Environments. He is currently Associate Professor at Politecnico di Milano and also Guest Professor at University of Lugano. He collaborates as Information Director with the ACM Software Engineering Interest Group (SIGSoft). He has been involved in several projects financed by the EU commission and by the Italian governor. He is co-author of several scientific papers published in international journals and conference proceedings. His research interests are in the area of Software Engineering and Distributed Systems. In particular, his current research focuses on middleware technology for largely distributed and highly reconfigurable distributed applications.

Amy L. Murphy is an Assistant Professor in the Department of Informatics at the University of Lugano, Switzerland. She received a B.S. in Computer Science from the University of Tulsa in 1995, and M.S. and D.Sc. degrees from Washington University in St. Louis, Missouri in 1997 and 2000 respectively. She spent three years as an assistant professor at the University of Rochester, New York and one year as a visiting researcher at Politecnico di Milano, Italy before joining the department in Lugano. Her research interests include the design, specification, and implementation of middleware systems for mobile ad hoc, sensor, and dynamic peer to peer networks. The driving theme of the work is to enable the rapid development of dependable applications for these complex environments. More information at `www.inf.unisi.ch/murphy`.

Gian Pietro Picco is an Associate Professor at the Department of Electronics and Information of Politecnico di Milano, Italy. He received his M.Sc. degree in Electronic Engineering from Politecnico di Milano in 1993, and his Ph.D. in Computer Science from Politecnico di Torino in 1998. He visited Washington University in St. Louis, MO, USA as a research assistant (1996-98) and then as a Visiting Assistant Professor (1998-99). He is with Politecnico di Milano since September 1999. His research interests are in distributed systems that exhibit mobility (of code or hosts) and in general high degrees of dynamicity. His work in this area thus far has investigated several aspects spanning from theoretical models to systems research. More information at `www.elet.polimi.it/upload/picco`.