Wireless Networks
https://doi.org/10.1007/s11276-023-03233-8

ORIGINAL PAPER

Using publish/subscribe for message routing in mobile environments

Ugaitz Amozarrain[1] · Mikel Larrea[2]

Accepted: 30 December 2022

Abstract
Publish/subscribe is a mature communication paradigm to route and deliver events from publishers to interested subscribers. Initially conceived for large scale systems, e.g., the Internet, it has been used more recently in new scenarios, e.g., wireless sensor networks and the Internet of Things (IoT), where mobility and dynamicity are the norm. The loose-coupling and asynchronicity of publish/subscribe makes it an interesting choice for IoT scenarios, i.e., each node in an IoT network can choose a different role depending on its location, capabilities, etc. This paper presents MFT-PubSub, a fully mobile and fault tolerant content-based publish/subscribe protocol. Our proposal is a purely reactive solution for mobility in a publish/subscribe system without any kind of limits on the mobility patterns of the nodes. A wireless ad hoc network is created without the need of any previous connections or knowledge on the nodes. Handling the mobility, be it physical or logical, of both clients and brokers. We prove the validity of our solution by experimentation, and compare it with AODV, a routing protocol for mobile ad hoc networking. The simulations show an improvement on message delivery rate over previously used protocols.

Keywords Fault tolerance · Mobile ad hoc networks · Publish/subscribe systems · Wireless sensor networks

1 Introduction

The publish/subscribe paradigm provides a anonymous, loosely coupled communication between event producers and interested subscribers [1]. This paradigm has been initially used for large scale systems. e.g., the Internet [2–4]. Recently, it has been used in other scenarios, e.g., wireless sensor networks [5–7] and the Internet of Things [8–10]. In this regard, several systems can be mentioned as relevant in the field, e.g., SIENA [2], JEDI [11], REBECA [12] and REDS [13] for both publish/subscribe in general and for their client mobility support in particular.

The main idea behind the publish/subscribe paradigm is to separate the devices that generate content from those that consume it. The content generated can range from a temperature reading of a sensor to an access notification on a web page, or even the distribution of a live television broadcast through the internet. In a publish/subscribe system the processes that generate and send content to the network are called the *publishers*, and those that consume the events are called *subscribers*. The decoupling is complete between both such processes. A publisher does not need to know which is the subscriber that is receiving the information it is sending to, nor do both of them need to be communicating at the same time in order for the message exchange to happen.

In this paper we propose an approach that not only supports client mobility, but also handles broker mobility in a wireless ad hoc network. In this approach the amount of devices connected to the network, and the connections between them are able to change. Whenever a broker moves physically a change is made in the network topology. Due to the wireless nature of the network the broker might lose connections or be able to connect to new devices. The protocol we propose tries to minimize these changes in topology to help with stability. We can also simplify a fault in one of the brokers as a loss of

✉ Mikel Larrea
mikel.larrea@ehu.eus

Ugaitz Amozarrain
ugaitz.amozarrain@unavarra.es

[1] Department of Statistics, Computer Science and Mathematics, Public University of Navarre, Pamplona, Spain

[2] Computer Architecture and Technology Department, University of the Basque Country UPV/EHU, Leioa, Spain

Published online: 03 February 2023

connectivity, making it easier to the protocol to also be fault tolerant.

The rest of the paper is organized as follows. Section 2 introduces the related work on publish/subscribe and the mobility and fault tolerance in these systems. Section 3 presents the model and definitions. Section 4 addresses the creation of the network overlay to route events between brokers. Section 5 presents a protocol to handle broker migration in publish/subscribe systems. Section 6 presents performance results of our approach and compares it to AODV. Finally, Sect. 7 concludes the paper.

2 Related work

Most of the research done in publish/subscribe systems is centered on improving current solutions, be it the reliability of delivering an event [14], improving the performance or increasing the fault tolerance [15]. Some work tries to improve on a typical tree structure for event delivery. In [16] authors propose the creation of a tree for each topic a subscriber can subscribe to with the publisher being the root of the tree for optimal message delivery. In some cases a communication tree might be too weak against node failure and the authors of [17] propose using gossiping so that the system can keep working while the tree is being repaired due to a node failure.

Another topic is the support for mobility. Though there are various protocols for publish/subscribe middleware, few of them support mobility [18]. In [19], authors mention some possible solutions for mobility support in publish/subscribe. Strategies are suggested to extend existing solutions, both in centralized and decentralized networks. In the case of a mobile network, nodes will need to adapt to disconnections, partitions of the network or the merging of those partitions, and the storage of undelivered events. In [20], Huang and Garcia-Molina study the tree construction problem in wireless ad hoc publish/subscribe systems. They define the optimality of a publish/subscribe tree by developing a metric to evaluate its efficiency, and propose a greedy algorithm that builds the publish/subscribe tree in a fully distributed fashion. Several works also address the different factors that affect the performance of a system with mobile nodes [21, 22], mostly based on mobile clients. A proposal to create self-configurable and adaptive peer-to-peer architecture for implementing content-based publish/subscribe communications on top of structured overlay networks has also been made [23, 24].

Another possible solution to support mobility is the use of information-centric networks [25–27]. Since this kind of network supports mobility natively, authors propose exploiting this property instead of using traditional TCP/IP communications.

Internet of Things (IoT) and Wireless Sensor Networks (WSN) also constitute an area that is still pushing research towards new topics [28, 29]. A number of recent contributions have also been made in the area [30–33]. Most of the approaches support mobility through the inclusion of gateway nodes and the separation of the publish/subscribe system from the WSN. The gateway nodes receive messages from any number of sensors and act as a publisher to the publish/subscribe system. This allows for the sensors to be mobile devices that send events to the gateway they are connected to, but does not fully compose a mobile publish/subscribe system.

2.1 Mobile clients

Most of the research carried out to support the mobility of nodes in a publish/subscribe system has been done with regard to supporting mobile clients, be they publishers or subscribers.

The first system to support client mobility was called JEDI [11], named for Java Event-Based Distributed Infrastructure. In JEDI a node must notify of its intention to migrate to the broker to which it is connected, before the migration happens. This is done by the use of explicit *moveOut* and *moveIn* messages, which a subscriber sends in order to start and finalize the migration process.

SIENA [2, 30] was a system developed at the same time as JEDI that also allowed for client mobility. It also uses explicit *moveOut* and *moveIn* messages and it uses flooding, that has been found to be excessive [34].

Another framework that added support for client mobility is the REBECA [12, 35] publish/subscribe system. In this case the moving node does not need to send an explicit *moveOut* message, a broker will detect when one of its connected subscribers has disconnected. The broker will then create a virtual counterpart of the roaming subscriber that will be merged with the real one once the migration finishes.

Mobile XSiena [36] is a publish/subscribe platform which seeks to extend the XSiena [37] content-based publish/subscribe system in order to support user mobility. The key mobility-related features of Mobile XSiena are mobile device integration, seamless networking, reconnection support, location-based matching, and persistent events. This was later integrated into the Phoenix framework [38–40].

MQTT is a commonly used protocol that also has received improvements in order to support client mobility. Though MQTT offers the support for subscriber mobility by allowing a subscriber to be connected to a subset of brokers, creating backups in case of a link failure, it does not allow for network reconfiguration in the case of a new connection, the subscriber will have to issue the subscriptions again. In [41] authors extend the protocol to support publisher mobility, by

detecting a disconnection in the publisher node, and storing undelivered messages while the system is reconfigured. This approach guarantees the delivery order of the messages to be the same as that of the creation.

$\mathcal{PSVR}$ [42] is a routing algorithm for a publish/subscribe system in a WSN. Siegemund et. al. mention the cost of maintaining a communication overlay in a dynamic environment, that is often really high or is omitted [43, 44], where systems usually recreate the overlay completely. The proposed algorithm is designed for systems with highly dynamic subscribers and publishers.

2.2 Mobile brokers

The scenario of mobility inside the event notification service is the most difficult to handle [45]. In this case the algorithms need to be able to handle the migration of not only clients but also reconfiguration on the subscription delivery path. There are few solutions that support full mobility on publish/subscribe systems.

In [24] an extension to SIENA is introduced where a self-organizing algorithm executed by brokers will try to optimize message delivery. Mechanisms are introduced to allow the reconfiguration caused by changes in topology, mostly to minimize the notification cost, but it could also be a first step towards supporting mobile nodes. Though the complexity of the algorithm, together with the need for a human administrator in case of a broker failure during the topology change procedure, makes it unsuitable for a highly mobile environment where a broker might start the topology change, but be disconnected by the time it finishes.

EMMA [46] is an extension to MQTT that not only handles client and broker migration in a transparent way, it also uses its migration mechanism in order to optimize QoS. It uses a controller node that is constantly monitoring the network and is informed of any change in device connectivity. The controller will then try to optimize event delivery and issue migrations to both clients and brokers to load balance the system. The requirement of a device that needs to know the connectivity of each node in the system prevent this solution to be used in a fully mobile environment where it might sometimes be unreachable.

2.3 Proposed solution

In this paper, we propose a protocol where any node, at any time, can migrate in the network without the need to notify neighboring nodes or a single central controller. In a truly mobile environment nodes might not know they need to migrate before a connection is lost. Using our protocol, nodes (publishers, subscribers or brokers) will be able to join or leave the publish/subscribe service at any time.

Unlike existing solutions, in our protocol, if the network is partitioned, the service will not have to wait for connections to restore. Any partition will work as an independent service, and after some time if connections recover the partitions will merge and any messages that were not delivered yet will be reach their destination.

3 Model and definitions

In a publish/subscribe system we might find two different components. *Clients* will produce and consume events while the *notification service* handles the subscriptions issued by the clients and assures the correct delivery of events to the interested clients.

We can further divide the clients into two subsets: *subscribers* that will register their interests and consume events, and *publishers* that will produce those events. We will use $s \in S$ to refer to a subscriber belonging to the set of subscribers S and $p \in P$ to refer to a publisher that belongs to the set of publishers P. Any clients in the system may behave as a subscriber, publisher or even both at the same time. We will also use the nomenclature $f \in F$ when referring to a *filter* that belongs to the set of filters F.

The notification service is composed of a set of brokers which we will call B and refer to individually as $b \in B$. The brokers will be connected at the logical level by an acyclic graph or a spanning tree. The brokers are responsible for storing the subscriptions issued by the subscribers and routing the published events to the matching subscribers. At any moment a broker will have a set of neighboring brokers in the graph, that it can communicate with. We will refer to this set as N_i for broker b_i. A broker will also be able to communicate with clients that are connected to it. For this reason we will refer to the set of interfaces, be it other brokers or clients, that a broker b_i can communicate with at any moment as I_i.

All communications are by point-to-point message passing over FIFO channels. Since participants are mobile, the set of channels linking them, as well as the neighbor set evolves. There is no need to have previous knowledge of the sets, i.e., initially each participant knows only itself and the amount of participants on each set might change as time passes.

3.1 Simple routing

The Simple Routing [47] protocol assumes a static system where brokers are connected in an acyclic graph, and clients are permanently bound to a single broker. This routing strategy is based on the propagation of subscription (*SUB*) and unsubscription (*UNS*) messages to all of the brokers in the system. Every broker b_i maintains a routing table R_i

that is based on the received *SUB* and *UNS* messages and models the subscriptions in the system. The routing tables enable brokers to filter incoming events received as *PUB* messages, and forward them only towards those subscribers with matching subscriptions.

The routing table R_i at every broker b_i contains, for every subscription in the system, a routing entry (f, z) where $f \in F$ and $z \in I_i$, to indicate that the publication of an event e matching f must either be forwarded towards broker z (if $z \in B$) or delivered to subscriber z (if $z \in S$).

3.2 Phoenix

The Phoenix [40] protocol handles subscriber mobility in content-based publish/subscribe systems. In order to do so, the routing table at brokers also stores the identity of the subscriber that issued each subscription. This way, when a subscriber migrates, the broker to which it was connected can be notified of the change. There are two extra types of messages used by Phoenix, one for notifying the migration of a subscriber(*MIG*), and another for replaying queued events to a migrated subscriber(*REP*). Whenever the subscriber re-connects to the system, possibly to another broker, it will issue a *MIG* message, whose propagation allows routing tables to be updated and published messages for the subscriber to be delivered.

4 Creating the network overlay

In order for the devices on the network to communicate efficiently we must create a logical overlay over a wireless ad hoc network. We need a way to create an acyclic graph (a spanning tree) in order to correctly route the messages. We also need a mechanism that detects when a change in the topology has occurred so a new link will be created when an old one disappears. The algorithm that creates this graph must also support the formation of several partitions in the network, each one working independently until they can merge together again. Lowering the changes made to the graph caused by physical changes on the network will also help to reduce the migrations needed to synchronize the publish/subscribe system.

We can use any algorithm that gives us these properties. In our case, we have chosen a leader election algorithm that has a heartbeat mechanism in order to keep the leader stable [48]. Once a leader has been elected, this node will keep sending messages so that all the other nodes will have this one as their leader. When a node receives one of this messages it will know the path to the leader [49], and it will broadcast it so the message spreads to all nodes within communication range. With this we create the overlay we need for constructing the publish/subscribe system.

Using this algorithm, in the event that the network is partitioned, each of the partitions will choose a leader. And eventually when the network becomes connected again both partitions will merge choosing a single leader and maintaining a single graph. Furthermore, with this heartbeat message, when a node first receives the message of a new round it will store the sender as the next hop to the leader. This next hop might be modified by any physical change in the location of a node or by a failure since the heartbeat message will arrive via another node. With this we can detect when the topology has changed and notify the publish/subscribe system so that it can migrate accordingly.

Algorithm 1 Simple Routing with mobile clients

```
1:  When RECEIVE(SUB, f, s, t, h) from z ∈ I_i
    where (t, h) > T_i(s) do
2:      T_i(s) ← (t, h + 1)
3:      if ∄(f, _, s) ∈ R_i then
4:          R_i ← R_i ∪ {(f, z, s)}
5:      end if
6:      for each b ∈ N_i where b ≠ z do
7:          SEND(SUB, f, s, t, h+1) to b
8:      end for
9:  End

10: When RECEIVE(UNS, f, s, t, h) from z ∈ I_i
    where (t, h) > T_i(s) do
11:     T_i(s) ← (t, h + 1)
12:     if ∃(f, _, s) ∈ R_i then
13:         R_i ← R_i \ {(f, _, s)}
14:     end if
15:     for each b ∈ N_i where b ≠ z do
16:         SEND(UNS, f, s, t, h+1) to b
17:     end for
18: End

19: When RECEIVE(MIG, f, s, t, h) from z ∈ I_i
    where (t, h) > T_i(s) do
20:     T_i(s) ← (t, h + 1)
21:     if z = s then
22:         X ← ∅
23:         for each (f, _, s) ∈ R_i do
24:             X ← X ∪ {f}
25:         end for
26:         SEND(FILTERS, X) to s
27:     end if
28:     if b ≠ b_i then
29:         if ∃(_, _, s) ∈ R_i then
30:             b_j ← y ∈ N_i where (_, y, s) ∈ R_i
31:             SEND(MIG, s, b, t, h+1) to b_j
32:         end if
33:     end if
34:     for each (_, _, s) ∈ R_i do
35:         replace (_, _, s) with (_, z, s) in R_i
36:     end for
37:     SENDQUEUEDMESSAGES(s, z)
38: End
```

5 The MFT-PubSub protocol

In this section we will describe the Mobile Fault Tolerant publish/subscribe (MFT-PubSub) protocol, and the changes made to Phoenix. Since the brokers are moving any change

in the topology can happen at any time. These changes can range from a simple client migration to the migration of multiple brokers at the same time. Due to the changing nature of the communication tree brokers might not have been notified of a change further down on the connection tree.

In order to take this into account we add a timestamp to any message sent by a subscriber. Previously on Phoenix a timestamp was also used so that a subscriber could request all the messages it lost during migration, and these messages were stored on a single broker. But, in our case the subscribers are not the only ones that are migrating, brokers will also migrate. As a broker migrates it has no knowledge of the last received message by a subscriber. Furthermore since the broker network is also changing, we cannot designate a single broker as the one responsible for storing the events. In order to solve this we need a mechanism that tells us if a message has been delivered. With this, if an error occurs, the broker will store the message as undelivered. When a broker receives a migration message, from a subscriber or another broker, it will send all messages stored for the subscribers that migrate. This is why we decided to repurpose the timestamp concept. This new timestamp will consist of a sequence number that increases each time a subscriber sends a message. We will also include a hop count to the messages, this way any broker will know on how many hops it can reach a subscriber. This is referred to as (t, h) in the algorithms. With these two values we have useful information when a migration occurs in order to find how the topology is changing. Any broker with a higher sequence number will be deemed to have the latest information and correct path on that subscriber, if the timestamps are equal the one that reports being the closest will have a higher probability of being correct.

If we want to include this information we have to modify the previously defined *SUB*, *UNS* and *MIG* messages. The changes can be seen on Algorithm 1. When a broker b_i receives one of these messages it will first check if the message contains new information by comparing the timestamps. Then it will store the new value and before propagating the message to the rest of the network it will increase the hop count of the message by one. Algorithm 2 shows how the event replay works.

Algorithm 2 Handling message replay

```
39: function sendQueuedMessages(s, y)
40:     while Q_i(s) is not empty do
41:         dequeue e from Q_i(s)
42:         if y ∉ I_i then
43:             enqueue e in Q_i(s)
44:         else
45:             send(REP, e, s) to y
46:         end if
47:     end while
48: end function

49: When receive(REP, e, s) from z ∈ I_i do
50:     y ← x ∈ I_i where (_, x, s) ∈ R_i
51:     if y ∉ I_i then
52:         enqueue e in Q_i(s)
53:     else
54:         send(REP, e, s) to y
55:     end if
56: End
```

Algorithm 3 Migration of broker b_i

```
57: function migrate(b_o, b_n)
58:     C ← ∅
59:     O ← ∅
60:     for each (_, z, s) ∈ R_i do
61:         if z = b_o then
62:             O ← O ∪ {(s, T_i(s).t, T_i(s).h)}
63:             replace (_, _, s) with (_, b_n, s)
64:         else
65:             C ← C ∪ {(s, T_i(s).t, T_i(s).h)}
66:         end if
67:     end for
68:     send(BMIG, C, O, 0) to b_n
69:     for each ((_, _, s) ∈ O do
70:         sendQueuedMessages(s, b_n)
71:     end for
72: end function
```

We also have to take into account the possibility of a subscriber migrating from one partition of the network to another, and since both partitions function individually the subscriber will have different subscriptions in each of them. We added a new message called *FILTERS* to fix this issue. We can see how this message is sent on lines 23–27 on Algorithm 1. Whenever a subscribers sends a *MIG* message the broker it migrates to will answer with a *FILTERS* message. This message contains all the subscriptions of that subscriber that the broker has in its routing table. Using this information the subscriber may decide that the subscriptions are outdated and issue *SUB* or *UNS* messages to fix and update the routing tables of the brokers on that partition.

Algorithm 4 Mobile brokers with timestamps

```
73: When RECEIVE(BMIG, C_j, O_j, h) from b_j ∈ N_i do
74:     C_i ← ∅
75:     for each (s, (t, h_j)) ∈ C_j do
76:         if t > T_i(s).ts then
77:             SEND(BQUERY, s, T_i(s).ts, T_i(s).h) to b_j
78:             C_i ← C_i ∪ {(s, (T_i(s).ts, h_j))}
79:         else if t < T_i(s).ts then
80:             SENDSUBSCRIPTIONS(s, b_j)
81:         else if t = T_i(s).ts then
82:             if h ≤ T_i(s).hops then
83:                 C_i ← C_i ∪ {(s, (T_i(s).ts, h_j))}
84:                 T_i(s) ← (t, h + h_j + 1)
85:             else if h > T_i(s).hops then
86:                 SENDSUBSCRIPTIONS(s, b_j)
87:             end if
88:         end if
89:     end for
90:     for each (s, (t, _)) ∈ O_j do
91:         if t > T_i(s).ts then
92:             SEND(BQUERY, s, T_i(s).ts, T_i(s).h) to b_j
93:         else if t < T_i(s).ts then
94:             SENDSUBSCRIPTIONS(s, b_j)
95:         end if
96:     end for
97:     if h = 0 then
98:         for each (s, _, _) ∈ R_i where s ∉ (C_j ∪ O_j) do
99:             SENDSUBSCRIPTIONS(s, b_j)
100:        end for
101:    end if
102:    X ← ∅
103:    for each s ∈ C_i do
104:        X ← X ∪ {b ∈ N_i where (_, b, s, _) ∈ R_i ∧ b ≠ b_j}
105:        for each (_, _, s) ∈ R_i do
106:            replace (_, _, s) with (_, b_j, s) in R_i
107:        end for
108:    end for
109:    for each y ∈ X do
110:        SEND(BMIG, C_i, ∅, h+1) to y
111:    end for
112:    for each s ∈ C_i do
113:        SENDQUEUEDMESSAGES(s, b_j)
114:    end for
115: End
```

Algorithm 5 Messages to fix inconsistent routing tables

```
116: When RECEIVE(BSUB, s, S_j, T_j(s), h) from z ∈ N_i
     where T_j(s) > T_i(s) do
117:     for each (_, _, s) ∈ R_i do
118:         R_i ← R_i \ {(_, _, s)}
119:     end for
120:     for each f ∈ S_j do
121:         R_i ← R_i ∪ {(f, z, s)}
122:     end for
123:     T_i(s) ← {(T_j(s).ts, T_j(s).h + h + 1)}
124:     for each b ∈ N_i where b ≠ z do
125:         SEND(BSUB, s, S_j, T_j(s), h + 1) to b
126:     end for
127:     SENDQUEUEDMESSAGES(s, z)
128: End

129: When RECEIVE(BQUERY, s, t, h) from z ∈ N_i
     where (t, h) < T_i(s) do
130:     SENDSUBSCRIPTIONS(s, z)
131: End

132: function SENDSUBSCRIPTIONS(s, z)
133:     S_i ← ∅
134:     for each (f, _, s) ∈ R_i do
135:         S_i ← S_i ∪ {f}
136:     end for
137:     SEND(BSUB, s, S_i, T_i(s), 0) to z
138: end function
```

Table 1 shows the messages used in order to support broker mobility without forcing migrations. The main message for this protocol is called *BMIG* and it is used for notifying the migration of a broker. We also add two helper messages to fix the routing tables in case of a migration; *BQUERY* and *BSUB*, for asking about the subscriptions a broker has of an specific subscriber and for sending the subscriptions issued by a single subscriber that a broker has in its routing table respectively.

When a broker b_i migrates from b_o to b_n, it calculates two sets of subscribers, *C* for its children and *O* for the rest, based on their next hop. It updates its routing table for the subscribers in *O* with b_n as their new next hop. After sending a *BMIG* message to b_n, it sends any queued message for subscribers in *O* through b_n. The code that describes this behavior can be seen on Algorithm 3.

A *BMIG* message has three parameters; two lists of subscribers with their timestamps, separating what the sending broker believes are children nodes C_j, and the rest O_j, and a hop count for the message. Upon reception of this message from another broker b_j, a broker b_i first goes through the C_j set in order to find inconsistencies, as shown in lines 75–89 on Algorithm 4. If b_i has a newer timestamp or a lower hop count to the subscriber than what is shown on C_j b_i will send a message containing the subscriptions of that subscriber with the correct timemstamp and hop count to b_j (lines 132–138 of Algorithm 5). On the other hand if the timestamp is lower it will ask b_j to send updated information on the subscriber. At the same time a new list

Table 1 New Message descriptions

Message	Payload	Client/Broker	Meaning
FILTERS	$f : f \in F$	$b \in B$	Send active subscriptions
BMIG	$\{C_b, O_b\} \in S_b$	$b \in B$	Notify the migration of b
BQUERY	$s \in S$	$b \in B$	Ask for the subscriptions of s
BSUB	$f : f \in F$	$b \in B$	Send active subscriptions

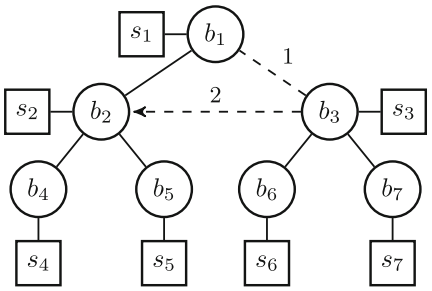
of children subscribers is created with the corrected information. The same procedure is followed for all subscribers in O_j on lines 90–96, in this case hop counts are ignored and b_i will only check the timestamps. To finish checking for inconsistencies on b_j's routing table on lines 97–101 the first broker that receives a *BMIG* message will check if both sets C_j and O_j contain all the subscribers b_i knows. For any subscriber that is not in the combination of both sets, b_i will send a message back to b_j with its subscriptions.

Once all inconsistencies have been fixed, in lines 103–108, b_i updates its routing table to show the change in topology for the subscribers that are children of b_j and previous next hops for those subscribers are stored. The corrected children list will be forwarded to those stored brokers. Finally any queued messages will be forwarded to the updated subscribers.

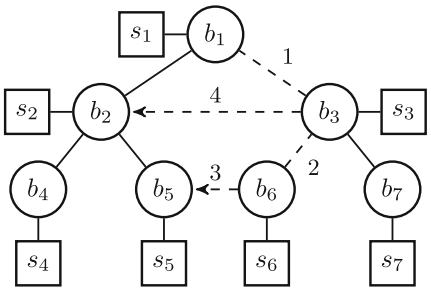
Once a broker b_i receives a *BSUB* message it will check if it has a newer timestamp for the subscriber than what b_i itself has, if it is older b_i will ignore the message. Then b_i will first remove all entries for that subscriber from its routing table and add the ones that came with the message updating the subscriber's timestamp as shown on lines 117–123 of Algorithm 5. This message will be forwarded as if it were a *SUB* message issued by any subscriber. Finally any queued message will be sent to the subscriber.

When a broker receives a *BQUERY* message, with a timestamp older than what it has, it will directly answer back with the subscriptions of the subscriber the message is asking for, lines 129–131 of Algorithm 5.

Figure 1 contains two examples of possible migrations. If we were to take a look into the messages needed to co mplete the migration shown in Fig. 1a we would only see two messages. A *BMIG* message that is sent by b_3 and needs to be routed to b_1 through b_2. Whereas, the migration that takes place in Fig. 1b is more complicated. In this case we need to notify b_3 that b_6 has migrated before it so it can update its routing table during its migration.



(a) $b_3 \leftrightarrow b_1$ link is lost and $b_3 \leftrightarrow b_2$ is created.



(b) First $b_3 \leftrightarrow b_1$ link is lost and $b_6 \leftrightarrow b_3$ after that. $b_6 \leftrightarrow b_5$ is connected and allowed to completelly migrate before $b_3 \leftrightarrow b_2$ is created.

Fig. 1 **a** shows a straightforward migration of one broker whereas **b** has two migrating nodes. Numbers in links refer to the order of events

6 Performance evaluation

This section presents the performance evaluation of the MFT-PubSub protocol presented in the previous section. Results have been obtained by simulation, using the OMNeT++ [50] tool with the Castalia [51] simulation framework. Table 2 presents the different simulated scenarios. The area has been calculated for a node density of 0,005 nodes per square meter, which is adequate for wireless sensor networks, i.e., giving an area of 200 square meters per node. We also define a role (publisher, subscriber or broker) for each node. We are interested in seeing how the protocol behaves in different scenarios, not in fully stress testing it. For this reason we have chosen to use only 2 publishers for all scenarios whereas the number of subscribers and brokers increases. The protocol does not

Table 2 Simulation Configurations

Configuration	#pubs	#susbs	#brokers	Area
C2	2	2	2	35x35 m^2
C4	2	4	4	45x45 m^2
C8	2	8	8	60x60 m^2
C16	2	16	16	80x80 m^2
C32	2	32	32	110x110 m^2

store any information on the publishers, a publisher simply sends a message and the brokers are in charge of correctly delivering it. By increasing the subscribers and brokers we increase the amount of nodes the protocol has to take into account. Subscribers will also subscribe to 2 filters and publishers will randomly choose to send messages matching one or the other.

The duration of the simulations is set at 700 seconds, with a publication rate by publishers of 1 message every second. This message generation rate is enough to test if the protocol works without overloading the communication buffers of the low power devices of the simulation. The messages have a constant data payload of 100 bytes. At the end there is a 200 second period where no new messages are sent so that messages that are still in buffers have time, and the opportunity, to be delivered. The mobility of nodes follows a random waypoint model [52], with speeds of 2-4-6-8-10 meters per second. Using this mobility model nodes will choose a random point in the simulation area and move towards it at a constant speed. Once the point is reached the process will be repeated. For the radio module we choose to use one that is already configured in the Castalia framework, the CC2420 chip, with a transmission power of 0dbm and an additive collision model. With respect to the MAC layer, we have used the Carrier-Sense Multiple Access (CSMA) configuration that comes with the Castalia installation. All possible combinations of size and speed are repeated 10 times with a different random seed for the mobility pattern and the results are averaged. The combination of node density and movement speed for the simulations means that all iterations have large amount of migrations and moments where the network is partitioned. The total number of migrations range from 10 in the smallest ones to over 600 in the largest simulations. The low transmission and reception power, and speed of the radio module chosen for the simulation means that in the larger scenarios the amount of message collision increases, causing over 600 migrations due to nodes trying to find better connections.

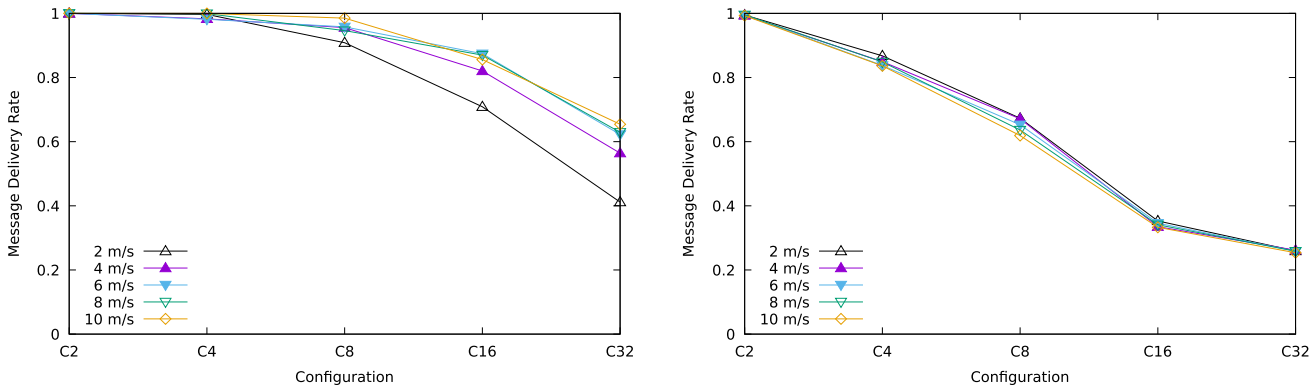
Due to the difficulty of finding an algorithm that allows for full broker mobility in a publish/subscribe system we had to choose a more general communication protocol for ad hoc networks. We compare our protocol with Ad hoc On-Demand Distance Vector (AODV) [53]. AODV uses a reactive approach to route creation to compensate for the dynamic nature of the network, where routes are created only whenever a node wants to send a message. In order to better compare both of them we also use the same roles that can be seen in Table 2. With AODV publishers are informed of the subscriber identifiers via a configuration file and all nodes work as brokers.

6.1 Delivery rate

One of the metrics that is able to tell us how well our protocol works is the delivery rate of messages. We consider the delivery rate as the number of messages a subscriber receives with respect to the ones that were originally sent to it. In MFT-PubSub messages that are not yet delivered are stored on the brokers waiting to be sent as soon as it receives new information about the subscriber. Eventually all messages will be delivered, but in the case of our simulations we consider that any message not yet delivered at the end on the simulation as undelivered.

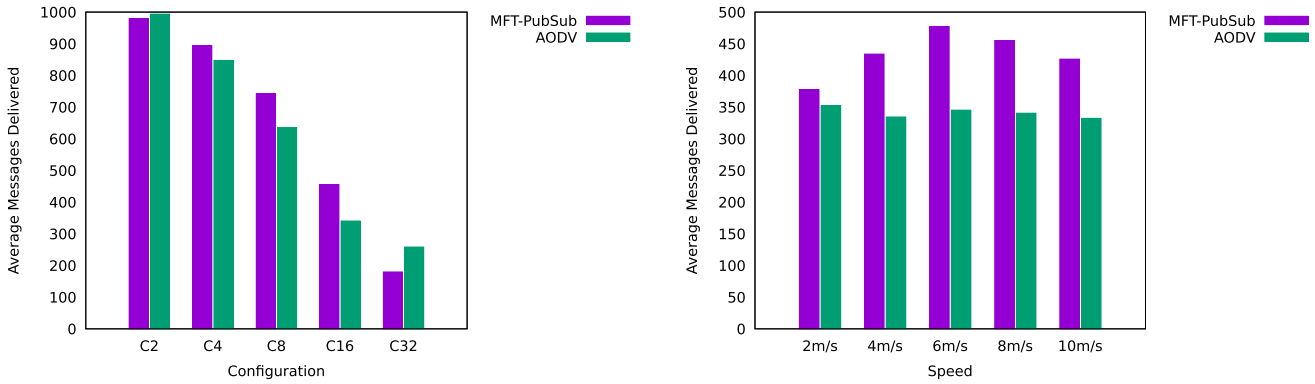
In Fig. 2 we can see a comparison between our protocol and AODV. MFT-PubSub seems to have better resilience to speed, even improving the delivery rate as the speed goes up. Both protocols are strongly affected by the network size, the bigger the network, the harder it is to correctly deliver a message.

In addition, if we look at Fig. 3 we can see how many messages are actually delivered. The behavior we see in Fig. 2a, where we see an improvement of delivery rate for higher speeds can be further analyzed with Fig. 3b. Here we can see a slight increase in the total number of messages delivered related to the speed, but as the speed reaches 6 m/s it starts to drop. This behavior can be explained by the way our algorithm buffers the messages. Whenever a broker cannot find the path to a subscriber it will store it and wait for new information on that subscriber, as the speed goes there are more opportunities for a subscriber to pass by a broker that has a message for it. If we compare the delivery rate differences in Fig. 2 with the amount of messages delivered in Fig. 3a, we might think that the difference in delivered messages is not as big as the delivery rate might suggest. This difference is due to how a publish/subscribe system works, in order to deliver a message to a subscriber that subscriber has to first subscribe to some content. In these simulations we only take into account the messages that are routed to a subscriber as having to be delivered to that subscriber. If a broker receives a *PUB* message but does not know a subscriber on the other side of the network is interested on it, the message will not be considered a loss.



(a) MFT-PubSub (b) AODV

Fig. 2 Message delivery rate comparison of both algorithms depending on the size of the simulation



(a) Node speed of 8 m/s (b) C16 configuration

Fig. 3 Average number of messages correctly delivered. In **a** we show the results for a node speed of 8 m/s on different configurations. And, in **b** we show the results of all speeds for the C16 configuration

6.2 End-to-end delay

Another metric is the time it takes a message to reach its destination, we call this the end-to-end delay. In Fig. 4 we can see how network size affects the end-to-end delay. Even though MFT-PubSub uses buffering of messages to be delivered at a later date, it still keeps up with AODV, that tries to deliver a message as soon as possible, even obtaining better results on bigger networks where AODV struggles to keep routing tables updated.

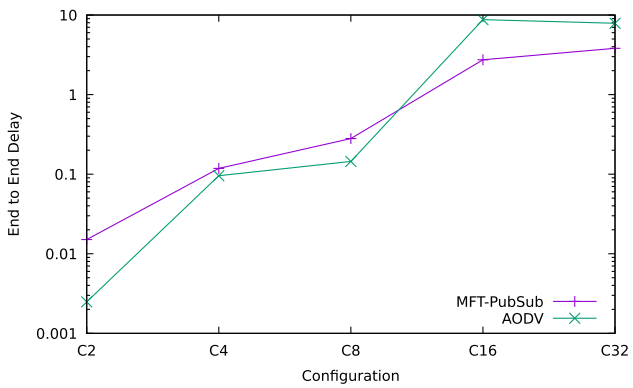


Fig. 4 Comparison of end-to-end delay, in seconds, for data messages for a node speed of 8 m/s. Note the logarithmic scale on the y axis

6.3 Number of messages exchanged

Finally, an interesting metric is the total number of messages exchanged in the network. This gives us insight into how efficiently a protocol is able to route messages, and how much overhead the protocol creates. In Fig. 5 we have this data as the average number of messages sent by each node, be it to find a route, delivery of a publication or any other kind of message. For the smallest configuration AODV has better performance than our protocol, since MFT-PubSub has to maintain a communication tree. But, whereas the number of messages needed as the network size gets bigger barely changes in our algorithm, AODV has a huge increase in the number of messages it needs to find the correct routes.

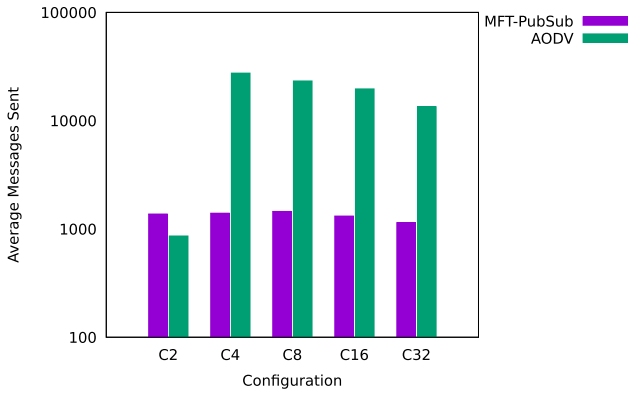


Fig. 5 Comparison of the average number of messages sent by each node in order to correctly route messages for a node speed of 8 m/s. Note the logarithmic scale on the y axis

In the case of MFT-PubSub we also observed that there is a big difference between the publishers/subscribers and brokers in the number of messages. The former only need to send a few messages in total to keep connected to the spanning tree and the brokers do most of the work.

7 Conclusion

In this paper we have presented an approach to introduce full mobility support (i.e., not only clients but also brokers) for a publish/subscribe system. It is based on a spanning tree created via a leader election algorithm that works in situations where it is not known a priori how many nodes there are. This algorithm also gives us a mechanism to detect the movement of nodes as a migration.

Our protocol, named MFT-PubSub, uses a mechanism to reduce the number of messages for any migration and only exchanges information when explicitly asked.

We have simulated MFT-PubSub on Castalia and compared it to AODV, to analyze the performance with respect to mobility support and number of devices supported. We improve on the message delivery rate of AODV, though the performance is significantly reduced for networks up to 66 nodes. We have also shown that the number of messages exchanged increases linearly with the number of nodes in the system and is an order of magnitude lower than AODV for simulations of more than 10 nodes.

MFT-PubSub allows for any node in the network to behave as any role of a publish/subscribe system; be it publisher, subscriber or broker. In the future we want to further test this approach and use it as a solution for multicast communication in mobile environments.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. Research supported by grant TIN2016-79897-P funded by MCIN/AEI/10.13039/501100011033 and by the European Union, and by the Department of Education, Universities and Research of the Basque Government, grant IT-1437-22 (ADIAN).

Data availability The datasets generated during and/or analysed during the current study are available in the following GitHub repository, https://github.com/ugaitz/MFTPubSubData.

References

1. Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys, 35*(2), 114–131.
2. Carzaniga, A., Rosenblum, D. S., & Wolf, A. L. (2001). Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS), 19*(3), 332–383. https://doi.org/10.1145/380749.380767.
3. Castro, M., Druschel, P., Kermarrec, A., & Rowstron, A. I. T. (2002). Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications, 20*(8), 1489–1499. https://doi.org/10.1109/JSAC.2002.803069
4. Rosenblum, D. S. & Wolf, A. L. (1997). A design framework for internet-scale event observation and notification. In *Software engineering-ESEC/FSE '97, 6th European software engineering conference held jointly with the 5th ACM SIGSOFT symposium on foundations of software engineering*, (Vol. 1301, pp. 344–360). Springer. https://doi.org/10.1007/3-540-63531-9_24
5. Beckmann, K. & Thoss, M. (2012). *A wireless sensor network protocol for the OMG data distribution service* (pp. 45–50). http://ieeexplore.ieee.org/document/6273603/.
6. Cam, H., Sahingoz, O. K., & Sonmez, A. C. (2011). *Wireless sensor networks based on publish/subscribe messaging paradigms* (pp. 233–242).
7. Tekin, Y. & Sahingoz, O. K. (2016). A publish/subscribe messaging system for wireless sensor networks. In *Sixth international conference on digital information and communication technology and its applications, DICTAP* (pp. 171–176). https://doi.org/10.1109/DICTAP.2016.7544022
8. Akkermans, S. et al. (2016). Towards efficient publish-subscribe middleware in the IoT with IPv6 multicast. In *2016 IEEE international conference on communications, ICC 2016*, Kuala Lumpur, Malaysia (pp. 1–6). https://doi.org/10.1109/ICC.2016.7511254.
9. Hakiri, A., Berthou, P., Gokhale, A. S., & Abdellatif, S. (2015). Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications. *IEEE Communications*

Magazine, 53(9), 48–54. https://doi.org/10.1109/MCOM.2015.7263372.
10. Happ, D., Karowski, N., Menzel, T., Handziski, V., & Wolisz, A. (2017). Meeting IoT platform requirements with open pub/sub solutions. *Annales des Télécommunications, 72*(1–2), 41–52. https://doi.org/10.1007/s12243-016-0537-4
11. Cugola, G., Di Nitto, E., & Fuggetta, A. (2001). The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering, 27*(9), 827–850.
12. Mühl, G., Ulbrich, A., Herrmann, K., & Weis, T. (2004). Disseminating information to mobile clients using publish-subscribe. *IEEE Internet Computing, 8*(3), 46–53. https://doi.org/10.1109/MIC.2004.1297273.
13. Cugola, G., Murphy, A. L., & Picco, G. P. (2006). Content-based publish-subscribe in a mobile environment. In P. Bellavista & A. Corradi (Eds.) *The handbook of mobile middleware* (pp. 257–285). Auerbach Publications/CRC. https://doi.org/10.1201/9781420013153.ch11.
14. Esposito, C., Platania, M., & Beraldi, R. (2014). Reliable and timely event notification for publish/subscribe services over the internet. *IEEE/ACM Transactions on Networking, 22*(1), 230–243. https://doi.org/10.1109/TNET.2013.2245144.
15. Zhao, Y., & Wu, J. (2013). Building a reliable and high-performance content-based publish/subscribe system. *Journal of Parallel and Distributed Computing, 73*(4), 371–382. https://doi.org/10.1016/j.jpdc.2012.12.014
16. de Araujo, J. P., Arantes, L., Duarte, E. P., Rodrigues, L. A., & Sens, P. (2019). VCube-PS: a causal broadcast topic-based publish/subscribe system. *Journal of Parallel and Distributed Computing, 125,* 18–30. https://doi.org/10.1016/j.jpdc.2018.10.011.
17. Salehi, P., Doblander, C., & Jacobsen, H.-A. (2016). Highly-available content-based publish/subscribe via gossiping. In *Proceedings ofthe 10th ACM international conference ondistributed and event-based systems, DEBS* (pp. 93–104). New York: ACM https://doi.org/10.1145/2933267.2933303.
18. Sheltami, T. R., Al-Roubaiey, A. A., & Mahmoud, A. S. H. (2016). A survey on developing publish/subscribe middleware over wireless sensor/actuator networks. *Wireless Networks, 22*(6), 2049–2070.
19. Huang, Y., & Garcia-Molina, H. (2004). Publish/subscribe in a mobile environment. *Wireless Networks, 10*(6), 643–652. https://doi.org/10.1023/B:WINE.0000044025.64654.65.
20. Huang, Y., & Garcia-Molina, H. (2003). Publish/subscribe tree construction in wireless Ad-Hoc networks. In *Mobile data management, 4th international conference, MDM 2003* (pp. 122–140). Melbourne, Australia. https://doi.org/10.1007/3-540-36389-0_9 .
21. Burcea, I., Jacobsen, H., de Lara, E., Muthusamy, V., & Petrovic, M. (2004). Disconnected operation in publish/subscribe middleware. In *5th IEEE international conference on mobile data management (MDM 2004)* (p. 39). Berkeley. https://doi.org/10.1109/MDM.2004.1263041.
22. Muthusamy, V., Petrovic, M., & Jacobsen, H. (2005). Effects of routing computations in content-based routing networks with mobile data sources. In *Proceedings of the 11th annual international conference on mobile computing and networking, MOBICOM 2005* (pp. 103–116). Cologne, Germany https://doi.org/10.1145/1080829.1080840.
23. Baldoni, R., Marchetti, C., Virgillito, A., & Vitenberg, R. (2005). Content-based publish-subscribe over structured overlay networks. In *25th international conference on distributedcomputing systems (ICDCS 2005)* (pp. 437–446). Columbus. https://doi.org/10.1109/ICDCS.2005.19.
24. Baldoni, R., Beraldi, R., Querzoni, L., & Virgillito, A. (2007). Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA. *The Computer Journal, 50*(4), 444–459. https://doi.org/10.1093/comjnl/bxm002.
25. Detti, A., Tassetto, D., Melazzi, N. B., & Fedi, F. (2015). Exploiting content centric networking to develop topic-based, publish-subscribe MANET systems. *Ad hoc Networks, 24,* 115–133.
26. Ventrella, A. V., Piro, G., & Grieco, L. A. (2017). Publish-subscribe in mobile information centric networks: Modeling and performance evaluation. *Computer Networks, 127*, 317–339. https://doi.org/10.1016/j.comnet.2017.08.022
27. Xylomenos, G., Vasilakos, X., Tsilopoulos, C., Siris, V. A., & Polyzos, G. C. (2012). Caching and mobility support in a publish-subscribe internet architecture. *IEEE Communications Magazine, 50*(7), 52–58. https://doi.org/10.1109/MCOM.2012.6231279.
28. Li, S., Da Xu, L., & Zhao, S. (2015). The internet of things: A survey. *Information Systems Frontiers, 17*(2), 243–259.
29. Whitmore, A., Agarwal, A., & Da Xu, L. (2015). The internet of things—A survey of topics and trends. *Information Systems Frontiers, 17*(2), 261–274.
30. Caporuscio, M., Carzaniga, A., & Wolf, A. L. (2003). Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transactions on Software Engineering, 29*(12), 1059–1071. https://doi.org/10.1109/TSE.2003.1265521.
31. Gündogan, C., Kietzmann, P., Schmidt, T. C., & Wählisch, M. (2018). HoPP: Robust and resilient publish-subscribe for an information-centric internet of things. In *43rd IEEE conferenceon local computer networks, LCN 2018* (pp. 331–334). Chicago, IL, USA https://doi.org/10.1109/LCN.2018.8638030.
32. Dominguez, A. M., Robles, T., Alcarria, R., & Cedeño, E. (2013). A Hot-topic based distribution and notification of events in Pub/Sub mobile brokers. *Network Protocols & Algorithms, 5*(1), 90–110. https://doi.org/10.5296/npa.v5i1.3326.
33. Hunkeler, U., Truong, H. L., & Stanford-Clark, A. J. (2008). MQTT-S - A publish/subscribe protocol for wireless sensor networks. In *Proceedings of the third international conference on communication system software and middleware (COMSWARE 2008)* (pp. 791–798). Bangalore. https://doi.org/10.1109/COMSWA.2008.4554519.
34. Tarkoma, S. (2006). *Efficient content-based routing, mobility-aware topologies, and temporal subspace matching*. Ph.D. thesis, University of Helsinki, Finland.
35. Fiege, L., Gärtner, F. C., Kasten, O., & Zeidler, A. (2003). Supporting mobility in content-based publish/subscribe middleware. In *Middleware 2003, ACM/IFIP/USENIX international middleware conference* (pp. 103–122). Rio de Janeiro, Brazil. https://doi.org/10.1007/3-540-44892-6.
36. Salvador, Z., Alzua, A., Larrea, M., & Lafuente, A. (2010). Mobile XSiena: Towards mobile publish/subscribe. In *Proceedings of the fourth ACM international conference on distributed event-based systems, DEBS 2010* (pp. 91–92). Cambridge, United Kingdom. https://doi.org/10.1145/1827418.1827434.
37. Jerzak, Z. (2009). *XSiena: The content-based publish/subscribe system*. Ph.D. thesis, Dresden University of Technology. http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-24628.
38. Salvador, Z. (2012). *Client mobility support and communication efficiency in distributed publish/subscribe*. Ph.D. thesis, University of the Basque Country UPV/EHU, Spain.
39. Salvador, Z., Lafuente, A., & Larrea, M. (2012). Design and evaluation of a publish/subscribe framework for ubiquitous systems. In *Mobile and ubiquitous systems: Computing, networking, and services - 9th international conference, MobiQuitous 2012. Lecture notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* (Vol. 120, pp. 50–63). Beijing, China. https://doi.org/10.1007/978-3-642-40238-8_5.

40. Salvador, Z., Larrea, M., & Lafuente, A. (2012). Phoenix: A protocol for seamless client mobility in publish/subscribe. In *11th IEEE international symposium on network computing and applications, NCA 2012* (pp. 111–120). Cambridge. https://doi.org/10.1109/NCA.2012.37.
41. Luzuriaga, J. E. et al. (2015). Handling mobility in IoT applications using the MQTT protocol. In *2015 Internet Technologies and Applications (ITA)* (pp. 245–250). https://doi.org/10.1109/ITechA.2015.7317403.
42. Siegemund, G., & Turau, V. (2018). A self-stabilizing publish/subscribe middleware for IoT applications. *ACM Transactions on Cyber-Physical Systems, 2*(2), 1–26. https://doi.org/10.1145/3185509.
43. Chen, C., Vitenberg, R. & Jacobsen, H.-A. (2016). Omen: Overlay mending for topic-based publish/subscribe systems under churn. In *Proceedings of the 10th ACM international conference on distributed and event-based systems, DEBS '16* (ACM, pp. 105–116). Irvine. https://doi.org/10.1145/2933267.2933305.
44. Chen, C., Jacobsen, H.-A., & Vitenberg, R. (2012). *Reinforce your overlay with shadows: Efficient dynamic maintenance of robust low fan-out overlays for topic-based publish/subscribe under churn*. Tech. Rep., University of Toronto, University of Oslo.
45. Huang, Y., & Garcia-Molina, H. (2001). Publish/subscribe in a mobile enviroment. In *Proceedingsof the second ACM international workshopon data engineering for wireless and mobileaccess* (pp. 27–34). Santa Barbara.
46. Rausch, T., Nastic, S., & Dustdar, S. (2018). Emma: Distributed qos-aware mqtt middleware for edge computing applications. In *2018 IEEEinternational conference on cloud engineering, IC2E 2018* (pp. 191–197). IEEE, Orlando.
47. Banavar, G. *et al.* (1999). An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th international conference on distributed computingsystems* (pp. 262–272). Austin: ACM.
48. Gómez-Calzado, C., Lafuente, A., Larrea, M. & Raynal, M. (2013). Fault-tolerant leader election in mobile dynamic distributed systems. In *IEEE 19th pacific rim international symposium on dependable computing, PRDC* (pp. 78–87). Vancouver, BC, Canada. https://doi.org/10.1109/PRDC.2013.17
49. Burgos, U., Amozarrain, U., Gómez-Calzado, C., & Lafuente, A. (2017). Routing in mobile wireless sensor networks: A leader-based approach. *Sensors, 17*(7), 1587.
50. Varga, A., & Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on simulation tools and techniques for communications, networks andsystems & workshops, SimuTools 2008,* (p. 60). Marseille, France:ICST/ACM. https://doi.org/10.4108/ICST.SIMUTOOLS2008.3027
51. Boulis, A. (2007). Castalia: Revealing pitfalls in designing distributed algorithms in wsn. In *Proceedings of the 5th international conference on embedded networked sensor systems, SenSys 2007* (p. 407–408). Sydney: ACM.
52. Camp, T., Boleng, J., & Davies, V. (2002). A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing, 2*(5), 483–502. https://doi.org/10.1002/wcm.72
53. Perkins, C. E., & Belding-Royer, E. M. (1999). Ad-hoc on-demand distance vector routing. In *2nd workshop on mobile computing systemsand applications (WMCSA '99)* (pp. 90–100). New Orleans. https://doi.org/10.1109/MCSA.1999.749281.

Ugaitz Amozarrain received the bachelor's degree in computer engineering from Mondragon University, Spain, and the Ph.D degree in Informatics from the University of the Basque Country UPV/EHU. He is currently a lecturer and researcher at the Public University of Navarre UPNA. His research interests include distributed systems with emphasis on mobility.

Mikel Larrea received the bachelor's degree in computer engineering from the Swiss Federal Institute of Technology, and the Ph.D degree in Informatics from the University of the Basque Country UPV/EHU. He is currently a professor at the UPV/EHU. His research interests include distributed algorithms and systems, fault tolerance and mobile and ubiquitous computing.