Full Mobility and Fault Tolerance in Content-Based Publish/Subscribe*

Ugaitz Amozarrain and Mikel Larrea
Department of Computer Architecture and Technology
University of the Basque Country UPV/EHU
{ugaitz.amozarrain,mikel.larrea}@ehu.eus

October 13, 2019

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, where mobility and dynamicity are the norm. This paper presents a fully mobile and fault tolerant content-based publish/subscribe protocol. We prove the validity of our solution by experimentation, and compare it with other routing approaches of mobile ad hoc networking.

1 Introduction

The publish/subscribe paradigm provides a anonymous, loosely coupled communication between event producers and interested subscribers [43]. This paradigm has been initially used for large scale systems. e.g., the Internet [21, 22, 76]. Recently, it has been used in new scenarios, e.g., wireless sensor networks [9, 18, 88] and the Internet of Things [1, 49, 50]. In this regard, several systems can be mentioned as relevant in the field, e.g., SIENA [21], JEDI [32], REBECA [66] and REDS [35] for both publish/subscribe in general and for their client mobility support in particular.

In content-based publish/subscribe protocols *subscribers* register to filters, *publishers* produce events, and *brokers* route and deliver events matching positively to interested subscribers. Phoenix [78, 80, 81] is a recent protocol supporting client mobility in content-based publish/subscribe systems targeting wireless sensor networks and the Internet of Things. Observe that content-based publish/subscribe naturally provides publisher mobility, so the Phoenix protocol handles subscriber mobility. In order to accomplish this, it assumes a static and reliable broker topology. Basically two tasks must be solved when a subscriber migrates [52]: update the routing tables of the corresponding brokers such that

*Research supported by the Spanish Research Council (MINECO), grant TIN2016-79897-P, and the Department of Education, Universities and Research of the Basque Government, grant IT980-16.

new events are properly routed, and deliver the events published during the migration. Phoenix solves both tasks in a communication-efficient manner, i.e., without flooding the network.

Besides client mobility, in this paper we propose two approaches to handle broker mobility in a wireless ad hoc network. The first approach is more modular and pedagogical, while the second approach is more efficient as it reduces network traffic. In both approaches 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 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, and describes briefly the Simple Routing and Phoenix protocols. Section 4 addresses the creation of the network overlay to route events among brokers. Section 5 presents a first, pedagogical protocol to handle broker migration. Section 6 presents an alternative, more efficient protocol. Section 7 presents performance results of both approaches and compares them to other routing protocols. Finally, Section 8 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 [41], improving the performance or increasing the fault tolerance [96]. Some work try to improve on a typical tree structure for event delivery. In [38] 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 [77] propose using gossiping so that the system can keep working while the tree is being repaired due to a node failure. Our proposal uses a tree created by a leader election algorithm that is constantly being updated in order to support fault tolerance.

Another topic is the support for mobility. Though there are various protocols for publish/subscribe middleware, few of them support mobility [83]. There has been some effort on trying to adapt current protocols for mobile environments [64], detecting when a broker has lost its connection to the rest of the network and storing the messages until that connection is reestablished. In [3] a possible solution is suggested for highly mobile environments, though only edge brokers are mobile, while the backbone is cloud based. Different to these solutions or work allows for full mobility of all the devices on the network.

Another possible solution to support mobility if the use of information-centric networks [39, 89]. Since this kind of networks support mobility natively, authors propose exploiting this property instead of using traditional TCP/IP communications. The work presented in this paper is similar to these last ones,

but it also allows for direct communication between any two nodes on the network.

Gryphon [87] is a large scale content-based publish/subscribe system that provides high throughput and low latency. Gryphon integrates the following technologies: event matching, multicasting, graph transformations, fault tolerance, ordered delivery, optimistic delivery, compression, reconfiguration, reflection, and security. Bayeux [98] is an efficient application-level multicast system that scales to arbitrarily large receiver groups while tolerating failures in nodes and network links. It includes specific mechanisms for load balancing across replicated nodes. Bayeux leverages the architecture of Tapestry [95], a fault tolerant, wide area overlay routing and location network.

In [34], Cugola and Jacobsen investigate the use of publish/subscribe middleware for large scale mobile applications, identifying two key problems: scalability and the ability to support changes in the topology which result from mobility. In [28], Cilia et al. address the use of publish/subscribe notification services in pervasive applications where mobility plays a prime role. They propose two enhancements of the publish/subscribe platform REBECA that provide applications with sequences of past notifications.

In [15], Burcea et al. study the factors that affect the performance of distributed publish/subscribe systems supporting subscriber mobility. In [67], Mühl et al. address client mobility in the REBECA publish/subscribe system. In [68], Muthusamy et al. study the effects of routing computations in content-based routing networks with mobile data sources. Additionally, the paper identifies the factors that affect the performance of a distributed publish/subscribe architecture supporting mobile publishers. In [92], Xylomenos et al. propose the publish/subscribe Internet (PSI) architecture, a information-centric networking approach designed to satisfy the user demands for pervasive content delivery. PSI supports seamless user mobility.

Mobile XSiena [79] is a publish/subscribe platform which seeks to extend the XSiena [58] 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. In [20], Caporuscio et al. propose a support service for mobile, wireless clients of a distributed publish/subscribe system. The goal is to support applications on mobile, wireless host devices in such a way that the applications can be oblivious to the mobility and intermittent connectivity of their hosts as they move from one broker to another. In [44], Fiege et al. investigate the support of mobility in content-based publish/subscribe middleware. They distinguish two orthogonal forms of mobility: the system-centric physical mobility and an application-centric logical mobility (where users are aware that they are changing location).

In [5], Baldoni et al. introduce a self-configurable and adaptive peer-to-peer architecture for implementing content-based publish/subscribe communications on top of structured overlay networks. In [4], Baldoni et al. propose a self-organizing algorithm executed by brokers whose aim is to directly connect, through overlay links, pairs of brokers matching same events. In this way, on average, the number of brokers involved in an event dissemination decreases.

HoP-and-Pull (HoPP) [47] is a robust publish/subscribe scheme for Internet of Things scenarios that targets networks consisting of hundreds of resource constrained devices at intermittent connectivity. In [70], Pham and Huh propose an

efficient edge-cloud publish/subscribe model that coordinates brokers in the system for large scale Internet of Things applications. They show that their broker overlay network can support low delay data delivery in a high scalable manner. In [84], Siegemund and Turau propose a self-stabilizing publish/subscribe middleware for Internet of Things applications. The middleware eventually provides safety and liveness properties such as the guaranteed delivery of all published messages to all subscribers and the correct handling of subscriptions and unsubscriptions. In [40], Dominguez et al. propose an algorithm for distributing hot-topics, subscribers and the process of notifying events over mobile and fixed brokers, focusing on the Internet of Things.

MQTT-S [53] is a data-centric publish/subscribe protocol for wireless sensor networks. It is designed in such a way that it can be run on low-end and battery-operated sensor/actuator devices and operate over bandwidth-constraint wireless sensor networks. In [13], Boonma and Suzuki propose La Nina, a self-adaptive framework for event routing in TinyDDS, which is a publish/subscribe middleware for wireless sensor networks. Mires [86] is a publish/subscribe middleware for wireless sensor networks. It has been implemented on top of TinyOS, an event-based operating system explicitly designed for sensor networks. Mires supports sudden topology changes and individual node crashes.

In [82], Schönherr et al. present a method to realize a self-organizing and self-stabilizing publish/subscribe middleware for wireless actuator and sensor networks. Their approach combines a content-based publish/subscribe algorithm and a clustering scheme. The publish/subscribe clusters are able to adapt themselves to changing network topologies supporting moderately dynamic settings. PS-QUASAR [26] is a lightweight, topic-based, Quality of Service aware publish/subscribe middleware for wireless sensor and actor networks. PS-QUASAR handles reliability and supports a many-to-many exchange of messages between nodes in a fully distributed way by means of multicasting techniques.

P2S [24] is a fault tolerant publish/subscribe infrastructure for building large scale distributed event notification systems. P2S contributes a practical solution by replicating the broker in a replicated architecture based on Paxos [62]. In [57], Jehl and Meling propose a Byzantine fault tolerant publish/subscribe system, on a tree-based overlay, tolerating a configurable number of failures in any part of the system. In [25], Chang and Meling address the design of a Byzantine fault tolerant publish/subscribe system suitable for use as a cloud computing infrastructure for building large scale distributed event notification systems. In [11], Berger and Reiser address Byzantine fault tolerance for resilient interactive web applications. They explore the possibility of using web-based clients for interaction with a Byzantine fault tolerant service.

In [60], Kazemzadeh and Jacobsen propose reliable distributed publish/subscribe algorithms that can tolerate concurrent failure of brokers and communication links. They propose protocols to address three problems in presence of broker or link failures: (i) subscription propagation, (ii) publication forwarding, and (iii) broker recovery. In [59], Kazemzadeh and Jacobsen propose a reliable distributed publish/subscribe approach that provides availability of service in the face of concurrent crash failure of up to δ brokers. They also propose a recovery procedure that recovering brokers execute in order to re-enter the system, and synchronize their routing information. XNET [23] is a reliable content-based publish/subscribe system. It gracefully handles broker and link failures and maintains the system global state consistent at all times. The key principle

underlying the recovery mechanism is that the local state of a broker can be reconstructed from the state of its neighbor brokers.

In [29, 30], Costa et al. address the problem of reliable event delivery in content-based publish/subscribe in dynamic environments by means of epidemic algorithms, which are by nature decentralized, scalable, and resilient to topological changes. Sub-2-Sub [90] is a self-organizing content-based publish/subscribe system for dynamic large scale collaborative environments. Sub-2-Sub deploys an unstructured overlay network, and relies on an epidemic-based algorithm in which peers continuously exchange subscription information to get clustered to similar peers.

In [73], Pongthawornkamol et al. analyze the reliability and timeliness of fault tolerant distributed publish/subscribe systems. They propose a model to predict Quality of Service in terms of event delivery probability and timeliness based on statistical attributes of each component in the distributed publish/subscribe system. In [37], García and Calveras propose a mechanism that establishes different Quality of Service levels, based on the publish/subscribe model for wireless networks to meet application requirements, to provide reliability and timeliness.

Mist [85] is a reliable and delay-tolerant publish/subscribe middleware for dynamic networks. It provides publish/subscribe with guaranteed message delivery in dynamic topologies with high mobility. In [55], Jaeger et al. present a self-organizing broker overlay infrastructure that adapts dynamically to achieve a better efficiency in large scale publish/subscribe systems.

In [6], Baldoni et al. study distributed event routing in publish/subscribe systems, addressing the issue of achieving scalable information dissemination in mobile ad hoc networks. COMAN [65] is a protocol to organize the nodes of mobile ad hoc networks in a tree-shaped network able to self-repair to tolerate the frequent topological reconfigurations typical of mobile ad hoc networks. COMAN builds upon the tree maintenance algorithm of the MAODV multicast protocol [10]. PSAMANET [75], is a publish/subscribe architecture for mobile ad hoc networks, which properly adapts to the highly dynamic topology of such networks using the nodes' movement to disseminate publications to the whole network with few transmissions. In [51], 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.

In [36], Cugola et al. propose an algorithm to dynamically reconfigure the topology of a distributed publish/subscribe dispatching infrastructure. The algorithm strikes a balance between simplicity and efficiency by tolerating frequent reconfigurations at the cost of moderate overhead. In [71], Picco et al. study the efficient content-based event dispatching problem in the presence of topological reconfigurations by means of a strawman approach whose simplicity is often outweighed by its inefficiency. In [12], Bhola et al. propose an efficient and scalable protocol for exactly-once delivery to subscribers in a content-based publish/subscribe system. The protocol requires persistent storage only at the publishing site, and is tolerant to message drops, node failures, and link failures.

PADRES [54] is a robust and scalable content-based publish/subscribe system providing redundancy in routes between publishers and subscribers and thus tolerating load imbalances, congestion, and broker and network failures. In [61],

Kazemzadeh and Jacobsen propose an approach to adapt a publish/subscribe overlay based on publication traffic and network conditions by selectively creating special links that boost the network connectivity and provide a large number of end-to-end forwarding paths, improving the system's throughput and efficiency.

In [94], Zhang et al. investigate the issue of total ordering in content-based publish/subscribe systems, proposing a solution using reliable FIFO channels. The solution does not require any global knowledge and is implemented directly in the brokers, eschewing the need for an external sequencing service.

Herald [17] is a highly scalable global event notification system that transparently scales in the numbers of publishers and subscribers, and event delivery rates. Herald has been designed to operate correctly in the presence of broken and disconnected components. Hermes [72] is a robust distributed event-based middleware architecture to build large scale publish/subscribe systems. It relies on a scalable and fault tolerant routing algorithm using an overlay network that avoids global broadcasts.

In [56], Jafarpour et al. propose a fast and robust content-based publish/subscribe architecture based on clustering brokers and forming rings between these clusters. The proposed approach can mask broker failures and increase dissemination service availability. It can also provide load balancing. In [27], Cheung and Jacobsen propose a load balancing solution specifically tailored to the needs of content-based publish/subscribe systems that is distributed, dynamic, adaptive, transparent, and accommodates heterogeneity.

In [33], Cugola et al. study how to minimize the reconfiguration overhead in content-based publish/subscribe, in order to adapt to changes which impact the topology of the dispatching infrastructure. They propose an algorithm that minimizes the portion of the system affected by the reconfiguration.

In [31], Costa and Picco propose an approach where event routing relies on deterministic decisions driven by a limited view on the subscription information and, when this is not sufficient, resorts to probabilistic decisions performed by selecting links at random. They show that mixing deterministic and probabilistic decisions is very effective at providing high event delivery and low overhead in highly dynamic scenarios, without sacrificing scalability.

In [63], Li et al. propose a content-based publish/subscribe algorithm to support general overlay topologies, as opposed to traditional acyclic or tree-based topologies. Content-based routing in a general overlay improves the scalability and robustness of publish/subscribe systems by offering routing path alternatives.

Dynamoth [45] is a dynamic, scalable, channel-based publish/subscribe middleware targeted at large scale, distributed and latency constrained systems in the cloud. In [8], Barazzutti et al. propose an elastic content-based publish/subscribe engine, E-STREAMHUB, running as a service on cloud environments.

DYNATOPS [97] is a dynamic topic-based publish/subscribe architecture that provides efficient scalable large scale event notifications for dynamic subscriptions. In DYNATOPS, brokers are repositioned on the overlay structure for efficient event notifications, to adapt to the publications and subscription dynamics.

Meghdoot [48] is a middleware for a content-based publish/subscribe system, which leverages peer-to-peer based Distributed Hash Tables for scalable

dissemination of events to subscribers distributed across the network. PubSub-Coord [2] is an autonomous and dynamic coordination and discovery service for large scale publish/subscribe applications.

In [91], Wang at al. propose a solution for supporting mobile clients in publish/subscribe systems. The key of their solution is the two-phase handover protocol, which can guarantee the exactly-once and ordered event delivery to mobile clients with little extra overhead, and allow mobile clients to get the undelivered events in a very short time after reconnection.

Trinity [74] is a distributed publish/subscribe broker with blockchain-based immutability for Internet of Things applications. Through the use of a blockchain network, Trinity can guarantee persistence, ordering, and immutability across trust boundaries. HyperPubSub [99] is a decentralized, permissioned, publish/subscribe service using blockchains, which provides secure and privacy preserving messaging. In [14], Bu et al. propose a publish/subscribe system, also named HyperPubSub, where the broker role is played by a private blockchain.

In [93], Yoon et al. investigate the foundations for highly available content-based publish/subscribe overlays. They propose a set of primitive operations to allow arbitrary transformations of a broker overlay to an optimal one, and to enable on-demand adjustments when there are permanent or transient failures.

In [42], Esposito et al. present a state-of-the-art on reliability in publish/subscribe services, addressing aspects like efficient and robust multi-point data dissemination, performance, and scalability.

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 the ones responsible of storing the subscriptions issued by the subscribers and route the published events to the matching subscribers. At any moment a broker will have a set of neighbouring 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 neighbour set evolves. There is no need of having previous knowledge

Message	Payload	Client/Broker	Meaning
SUB	$f \in F$	$s \in S$	Subscribe s to filter f
UNS	$f \in F$	$s \in S$	Unsubscribe s from filter f
PUB	$e \in E$	$p \in P$	Publish event e

Table 1: Simple Routing message description

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 [7] 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 $z \in S$).

Table 1 shows the three types of messages used in the Simple Routing protocol, for subscribing to a filter, unsubscribing from a filter and publishing an event respectively.

3.2 Phoenix

The Phoenix protocol handles subscriber mobility in content-based publish/subscribe. 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. Whenever the subscriber re-connects to the system, possibly to another broker, it will issue a MIG message, whose propagation allows updating routing tables and delivering published messages for the subscriber.

Message	Payload	Client/Broker	Meaning
MIG	—	$s \in S$	Notify the migration of s
REP	$e \in E$	$s \in S$	Replay event e towards s

Table 2: Phoenix message description

Table 2 shows the two extra types of messages used by Phoenix, for notifying the migration of a subscriber, and for replaying queued events to a migrated subscriber respectively.

4 Creating the network overlay

In order for the devices on the network to communicate efficiently we must create a logical overlay. 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 [46]. 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 [16], 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 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.

5 A modular protocol

The first version of the protocol for broker migration follows a modular approach. 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 this each broker only knows the nodes that are directly connected to it, and the broker might not have been notified of a change further down on the connection tree. For this reason a broker has to maintain an updated list of subscribers that are directly connected to it. And, when it migrates, it is only responsible for the migration of those subscribers. Every other broker connected to the one that has migrated has to be notified of this migration so they can migrate too. This way the whole branch of the tree that the migrating broker is responsible for has to migrate.

Message	Payload	Client/Broker	Meaning
FILTERS	$f : f \in F$	$b \in B$	Send active subscriptions
BMIG	C_b	$b \in B$	Notify the migration of b
BTAB	R_b	$b \in B$	Updated routing table of b
FMIG	—	$b \in B$	Force the migration of a broker

Table 3: Broker mobility message description

Table 3 shows the additional messages used in order to support broker mobility, for sending the set of filters a subscriber has issued, for notifying the migration of a broker, for updating the routing table based on the information of the primary partition (i.e., the partition containing the leader), and for forcing the migration of a broker respectively.

For this version of the protocol we have simplified the code of Phoenix, for example by removing the timestamp values from messages. In Phoenix, when a subscriber migrates it can request for the events that it has lost to be resent by specifying the timestamp of the last received message. When the broker where it was connected previously receives this message, it will resend all the messages that are newer than that timestamp. 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 changes are reflected on Algorithms 1 and 2.

To complicate things more we may have 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 of 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 3 shows how the broker migration works. Whenever a broker migrates it will send a $BMIG$ message with two parameters: a list of the subscribers connected to it and the identifier of the broker that is migrating. Any broker that receives this message will first replace the next hop of those subscribers in the routing table to the sender of the message and then resend the $BMIG$ message to the old next hops so that they are notified of the change. This behavior can be seen on lines 2-8 on Algorithm 3 and it is similar to what happens when a subscriber migrates. Then if the broker is the first one receiving the message it will answer with a $BTAB$ message containing its whole routing table. Finally the broker will send any stored messages for those subscribers.

Whenever a broker migrates it can only trust the subscribers that are connected directly. For this reason when after sending a $BMIG$ message it receives

```
1  when receive(SUB, f, s) from z ∈ I_i do
2      if ∄(f, _, s) ∈ R_i then
3          R_i ← R_i ∪ {(f, z, s)}
4      foreach b ∈ N_i where b ≠ z do
5          send(SUB, f, s) to b

6  when receive(UNS, f, s) from z ∈ I_i do
7      if ∃(f, _, s) ∈ R_i then
8          R_i ← R_i \ {(f, _, s)}
9      foreach b ∈ N_i where b ≠ z do
10         send(UNS, f, s) to b

11 when receive(PUB, e) from z ∈ I_i do
12     X ← ∅
13     foreach (f, y, _) ∈ R_i where y ∉ X ∧ y ≠ z do
14         if f(e) = true then
15             X ← X ∪ {y}
16     foreach y ∈ X do
17         if y ∉ I_i then
18             foreach (f, y, s) ∈ R_i where f(e) = true do
19                 enqueue {e} in Q_i(s)
20         else
21             send(PUB, e) to y

22 when receive(MIG, s, b) from z ∈ I_i do
23     if z = s then
24         X ← ∅
25         foreach (f, _, s) ∈ R_i do
26             X ← X ∪ {f}
27         send(FILTERS, X) to s
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) to b_j
32     foreach (_, _, s) ∈ R_i do
33         replace (_, _, s) with (_, z, s) in R_i
34     sendQueuedMessages(s, z)
```

Algorithm 1: Simple Routing with mobile clients

```
35 function sendQueuedMessages(s, y)
36     while Q_i(s) is not empty do
37         dequeue {e} from Q_i(s)
38         if y ∉ I_i then
39             enqueue {e} in Q_i(s)
40             return
41         else
42             send(REP, e, s) to y

43 when receive(REP, e, s) from z ∈ I_i do
44     y ← x ∈ I_i where (_, x, s) ∈ R_i
45     if y ∉ I_i then
46         enqueue {e} in Q_i(s)
47     else
48         send(REP, e, s) to y
```

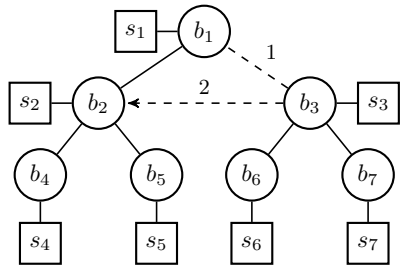
Algorithm 2: Simple Routing with mobile clients - Message replay

a $BTAB$ message back it will empty its routing table except the entries of the local subscribers. And then it will store all the information that comes with the message as shown on lines 14-17. On lines 18-21 the broker checks for any information that is missing on the other broker's routing table and sends the necessary SUB and UNS messages to fix it. After this, the broker will force the migration of all the brokers that are connected to it with an $FMIG$ message. As soon as a broker receives a $FMIG$ message it will answer back with a $BMIG$ message starting the process again. And finally, as before, the broker will send any stored messages for all subscribers.

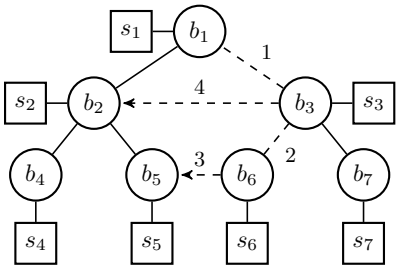
On Figure 1 we show two examples of migrations with a sequence of messages for each of the migrations explained on Figure 2. Independently of the changes that have happened on the topology a migrating broker will always try to force all the other brokers that it is connected with to also migrate. If there was not a forced migration in the example b_3 would not be able to differentiate between both examples.

```
1  when receive(BMIG, C_j, b_j) from z ∈ N_i do
2      X ← ∅
3      foreach s ∈ C_j do
4          X ← X ∪ {b ∈ N_i where (_, b, s) ∈ R_i ∧ b ≠ z}
5          foreach (_, _, s) ∈ R_i do
6              replace (_, _, s) with (_, z, s) in R_i
7      foreach y ∈ X do
8          send(BMIG, C_j, b_j) to y
9      if z = b_j then
10         send(BTAB, R_i) to b_j
11     foreach s ∈ C_j do
12         sendQueuedMessages (s, z)

13 when receive(BTAB, R_j) from b_j ∈ N_i do
14     foreach (_, _, s) ∈ R_i where s ∉ C_i do
15         R_i ← R_i \ {(_, _, s)}
16     foreach (_, _, s) ∈ R_j where s ∉ C_i do
17         R_i ← R_i ∪ {(_, b_j, s)}
18     foreach (f, _, s) ∈ (R_i − R_j) do
19         send(SUB, f, s) to b_j
20     foreach (f, _, s) ∈ (R_j − R_i) do
21         send(UNS, f, s) to b_j
22     foreach b ∈ N_i where b ≠ b_j do
23         send(FMIG) to b
24     foreach (_, _, s) ∈ R_j where s ∉ C_i do
25         sendQueuedMessages (s, b_j)

26 when receive(FMIG) from z ∈ N_i do
27     send(BMIG, C_i, b_i) to z
```

Algorithm 3: Simple Routing with mobile brokers



(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 after $b_6 \leftrightarrow b_3$ is also lost. $b_6 \leftrightarrow b_5$ is connected and allowed to completelly migrate before $b_3 \leftrightarrow b_2$ is created.

Figure 1: Two migration examples, Figure 1a shows a straightforward migration of one broker whereas Figure 1b has two migrating nodes. Numbers in links refer to the order of events.

1. $b_3 \rightarrow b_2 \Rightarrow BMIG : \{C_j = [s_3], b_j = b_3\}$
2. $b_2 \rightarrow b_3 \Rightarrow BTAB : \{R_j = R_2\}$
3. $b_2 \rightarrow b_1 \Rightarrow BMIG : \{C_j = [s_3], b_j = b_3\}$
4. $b_3 \rightarrow b_6 \Rightarrow FMIG$
5. $b_3 \rightarrow b_7 \Rightarrow FMIG$
6. $b_6 \rightarrow b_3 \Rightarrow BMIG : \{C_j = [s_6], b_j = b_6\}$
7. $b_7 \rightarrow b_3 \Rightarrow BMIG : \{C_j = [s_7], b_j = b_7\}$
8. $b_3 \rightarrow b_6 \Rightarrow BTAB : \{R_j = R_3\}$
9. $b_3 \rightarrow b_7 \Rightarrow BTAB : \{R_j = R_3\}$
10. $b_3 \rightarrow b_2 \Rightarrow BMIG : \{C_j = [s_6], b_j = b_6\}$
11. $b_3 \rightarrow b_2 \Rightarrow BMIG : \{C_j = [s_7], b_j = b_7\}$
12. $b_2 \rightarrow b_1 \Rightarrow BMIG : \{C_j = [s_6], b_j = b_6\}$
13. $b_2 \rightarrow b_1 \Rightarrow BMIG : \{C_j = [s_7], b_j = b_7\}$

(a) Message sequence when the migration on Figure 1a occurs.

1. $b_6 \rightarrow b_5 \Rightarrow BMIG : \{C_j = [s_6], b_j = b_6\}$
2. $b_5 \rightarrow b_6 \Rightarrow BTAB : \{R_j = R_5\}$
3. $b_5 \rightarrow b_2 \Rightarrow BMIG : \{C_j = [s_6], b_j = b_6\}$
4. $b_2 \rightarrow b_1 \Rightarrow BMIG : \{C_j = [s_6], b_j = b_6\}$
5. $b_3 \rightarrow b_2 \Rightarrow BMIG : \{C_j = [s_3], b_j = b_3\}$
6. $b_2 \rightarrow b_3 \Rightarrow BTAB : \{R_j = R_2\}$
7. $b_2 \rightarrow b_1 \Rightarrow BMIG : \{C_j = [s_3], b_j = b_3\}$
8. $b_3 \rightarrow b_7 \Rightarrow FMIG$
9. $b_7 \rightarrow b_3 \Rightarrow BMIG : \{C_j = [s_7], b_j = b_7\}$
10. $b_3 \rightarrow b_7 \Rightarrow BTAB : \{R_j = R_3\}$
11. $b_3 \rightarrow b_2 \Rightarrow BMIG : \{C_j = [s_7], b_j = b_7\}$
12. $b_2 \rightarrow b_1 \Rightarrow BMIG : \{C_j = [s_7], b_j = b_7\}$

(b) Message sequence when the migration on Figure 1b occurs.

Figure 2: Message sequences when the migration on Figure 1 occurs following the modular protocol. R_2, R_3 and R_5 contain the whole routing table of the corresponding broker, b_2, b_3 and b_5 respectively.

6 A more efficient protocol

In this section we will describe the changes made to the protocol in order to avoid the cascade of messages that a migrating broker might cause. For this we add a timestamp to any message sent by a subscriber. This 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. 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 4. 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.

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

Table 4: Message description

Table 4 shows the messages used in order to support broker mobility without forcing migrations, the new messages will be $BQUERY$ 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. These messages replace the $BTAB$ message of the previous approach. We also keep the $BMIG$ message for notifying when a broker migrates, but change its behaviour.

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 5.

A $BMIG$ message has three parameters; two lists of subscribers with their timestamps, separating what the sending broker believes that 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 on lines 15-30 on Algorithm 6. 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 59-63 of Algorithm 7). 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 of children subscribers is created with

```
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      foreach b ∈ N_i where b ≠ z do
6          send(SUB, f, s, t, h+1) to b

7  when receive(UNS, f, s, t, h) from z ∈ I_i where (t, h) > T_i(s) do
8      T_i(s) ← (t, h + 1)
9      if ∃(f, _, s) ∈ R_i then
10         R_i ← R_i \ {(f, _, s)}
11     foreach b ∈ N_i where b ≠ z do
12         send(UNS, f, s, t, h+1) to b

13 when receive(MIG, s, b, t, h) from z ∈ I_i where (t, h) > T_i(s) do
14     T_i(s) ← (t, h + 1)
15     if z = s then
16         X ← ∅
17         foreach (f, _, s) ∈ R_i do
18             X ← X ∪ {f}
19         send(FILTERS, X) to s
20     if b ≠ b_i then
21         if ∃(_, _, s) ∈ R_i then
22             b_j ← y ∈ N_i where (_, y, s) ∈ R_i
23             send(MIG, s, b, t, h+1) to b_j
24     foreach (_, _, s) ∈ R_i do
25         replace (_, _, s) with (_, z, s) in R_i
26     sendQueuedMessages(s, z)
```

Algorithm 4: Simple Routing with mobile clients

the corrected information. The same procedure is followed for all subscribers in O_j on lines 31-35, 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 36-38 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, on lines 40-43, 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 message will be forwarded to the updated subscribers.

```
1  function migrate(b_o, b_n)
2      C ← ∅
3      O ← ∅
4      foreach (s, z, _) ∈ R_i do
5          if z = b_o then
6              O ← O ∪ {(s, T_i(s).t, T_i(s).h)}
7              replace (s, _, _) with (s, b_n, _)
8          else
9              C ← C ∪ {(s, T_i(s).t, T_i(s).h)}
10     send(BMIG, C, O, 0) to b_n
11     foreach ((s, _, _) ∈ O) do
12         sendQueuedMessages (s, b_n)
```

Algorithm 5: Migration of broker b_i

$$1.\ b_3 \to b_2 \Rightarrow BMIG: \left\{ \begin{array}{l} C_j = [(s_3,1,1),(s_6,1,2),(s_7,1,2)] \\ O_j = [(s_1,1,2),(s_2,1,3),(s_4,1,4),(s_5,1,4)] \\ h = 0 \end{array} \right\}$$

$$2.\ b_2 \to b_1 \Rightarrow BMIG: \left\{ \begin{array}{l} C_j = [(s_3,1,1),(s_6,1,2),(s_7,1,2)] \\ O_j = \varnothing \\ h = 1 \end{array} \right\}$$

Figure 3: Message sequence when the migration on Figure 1a occurs following the more efficient protocol.

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 49-53 of Algorithm 7. 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 57-58 of Algorithm 7.

A sequence of the messages sent whenever a broker migrates can be seen on Figures 3 and 4, referring to the migrations that happened on Figure 1. The number of messages is completely different in both cases. In the first example no changes have been made to the topology before the migration. With the help of the timestamps and hop numbers the brokers are able to detect it and no further messages are needed. Instead on the second example b_6 is the one that realizes some error on the message with respect to the number of hops of s_6. And answers back with a $BSUB$ message that is propagated to the whole network.

```
13 when receive(BMIG, C_j, O_j, h) from b_j ∈ N_i do
14     C_i ← ∅
15     foreach (s, (t, h_j)) ∈ C_j do
16         if t > T_i(s).ts then
17             //If j is newer than i
18             send(BQUERY, s, T_i(s).ts, T_i(s).h) to b_j
19             C_i ← C_i ∪ {(s, (T_i(s).ts, h_j))}
20         else if t < T_i(s).ts then
21             //If i is newer than j
22             sendSubscriptions(s, b_j)
23         else if t = T_i(s).ts then
24             if h ≤ T_i(s).hops then
25                 //If j is closer than i
26                 C_i ← C_i ∪ {(s, (T_i(s).ts, h_j))}
27                 T_i(s) ← (t, h + h_j + 1)
28             else if h > T_i(s).hops then
29                 //If i is closer than j
30                 sendSubscriptions(s, b_j)
31     foreach (s, (t, _)) ∈ O_j do
32         if t > T_i(s).ts then
33             send(BQUERY, s, T_i(s).ts, T_i(s).h) to b_j
34         else if t < T_i(s).ts then
35             sendSubscriptions(s, b_j)
36     if h = 0 then
37         foreach (s, _, _) ∈ R_i where s ∉ (C_j ∪ O_j) do
38             sendSubscriptions(s, b_j)
39     X ← ∅
40     foreach s ∈ C_i do
41         X ← X ∪ {b ∈ N_i where (_, b, s, _) ∈ R_i ∧ b ≠ b_j}
42         foreach (_, _, s) ∈ R_i do
43             replace (_, _, s) with (_, b_j, s) in R_i
44     foreach y ∈ X do
45         send(BMIG, C_i, ∅, h+1) to y
46     foreach s ∈ C_i do
47         sendQueuedMessages (s, b_j)
```

Algorithm 6: Mobile brokers with timestamps

```
48 when receive(BSUB, s, S_j, T_j(s), h) from z ∈ N_i where
   T_j(s) > T_i(s) do
49     foreach (_, _, s) ∈ R_i do
50         R_i ← R_i \ {(_, _, s)}
51     foreach f ∈ S_j do
52         R_i ← R_i ∪ {(f, z, s)}
53     T_i(s) ← {(T_j(s).ts, T_j(s).h + h + 1)}
54     foreach b ∈ N_i where b ≠ z do
55         send(BSUB, s, S_j, T_j(s), h + 1) to b
56     sendQueuedMessages (s, z)

57 when receive(BQUERY, s, t, h) from z ∈ N_i where (t, h) < T_i(s)
   do
58     sendSubscriptions(s, z)

59 function sendSubscriptions(s, z)
60     S_i ← ∅
61     foreach (f, _, s) ∈ R_i do
62         S_i ← S_i ∪ {f}
63     send(BSUB, s, S_i, T_i(s), 0) to z
```

Algorithm 7: Messages to fix inconsistent routing tables

1. $b_6 \to b_5 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_6, 1, 1)] \\ O_j = \left[\begin{array}{l} (s_1, 1, 3), (s_2, 1, 4), (s_3, 1, 2), \\ (s_4, 1, 5), (s_5, 1, 5), (s_7, 1, 3) \end{array} \right] \\ h = 0 \end{array} \right\}$

2. $b_5 \to b_2 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_6, 1, 1)] \\ O_j = \varnothing \\ h = 1 \end{array} \right\}$

3. $b_2 \to b_1 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_6, 1, 1)] \\ O_j = \varnothing \\ h = 2 \end{array} \right\}$

4. $b_3 \to b_2 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_3, 1, 1), (s_6, 1, 2), (s_7, 1, 2)] \\ O_j = [(s_1, 1, 2), (s_2, 1, 3), (s_4, 1, 4), (s_5, 1, 4)] \\ h = 0 \end{array} \right\}$

5. $b_2 \to b_1 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_3, 1, 1), (s_6, 1, 2), (s_7, 1, 2)] \\ O_j = \varnothing \\ h = 1 \end{array} \right\}$

6. $b_2 \to b_5 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_3, 1, 1), (s_6, 1, 2), (s_7, 1, 2)] \\ O_j = \varnothing \\ h = 1 \end{array} \right\}$

7. $b_5 \to b_6 \Rightarrow BMIG : \left\{ \begin{array}{l} C_j = [(s_3, 1, 1), (s_6, 1, 2), (s_7, 1, 2)] \\ O_j = \varnothing \\ h = 2 \end{array} \right\}$

8. $b_6 \to b_5 \Rightarrow BSUB : \{s = s_6, S_j = [f_6], T_j(s) = (1, 1), 0\}$

9. $b_5 \to b_2 \Rightarrow BSUB : \{s = s_6, S_j = [f_6], T_j(s) = (1, 1), 1\}$

10. $b_2 \to b_4 \Rightarrow BSUB : \{s = s_6, S_j = [f_6], T_j(s) = (1, 1), 2\}$

11. $b_2 \to b_1 \Rightarrow BSUB : \{s = s_6, S_j = [f_6], T_j(s) = (1, 1), 2\}$

12. $b_2 \to b_3 \Rightarrow BSUB : \{s = s_6, S_j = [f_6], T_j(s) = (1, 1), 2\}$

13. $b_3 \to b_7 \Rightarrow BSUB : \{s = s_6, S_j = [f_6], T_j(s) = (1, 1), 3\}$

Figure 4: Message sequence when the migration on Figure 1b occurs following the more efficient protocol.

7 Performance evaluation

This section presents the performance evaluation of the protocols presented in the previous sections. Results have been obtained by simulation, using the OMNeT++[1] tool with the Castalia simulation framework.

We choose to only implement the protocol introduced in Section 6 since it is a direct improvement over the first one. We have named this protocol "Mobile Fault-Tolerant Publish/Subscribe", or *MFT-PubSub* for short.

Table 5 presents the different scenarios simulated. 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.

Configuration	#publishers	#susbscribers	#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

Table 5: Simulation configurations.

The duration of the simulations is set at 700 seconds, with a publication rate by publishers of 1 message every second. 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 opportunity, to be delivered. The mobility of nodes follows a random waypoint model [19], 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. All possible combinations of size and speed are repeated 10 times with a different seed for the mobility pattern and the results are averaged.

We compare our protocol with Ad hoc On-Demand Distance Vector (AODV) [69], which is a more general communication protocol for ad-hoc networks. In order to better compare both of them we also use the same roles that can be seen on Table 5. With AODV publishers are informed of the subscriber identifiers via a configuration file and all nodes work as brokers.

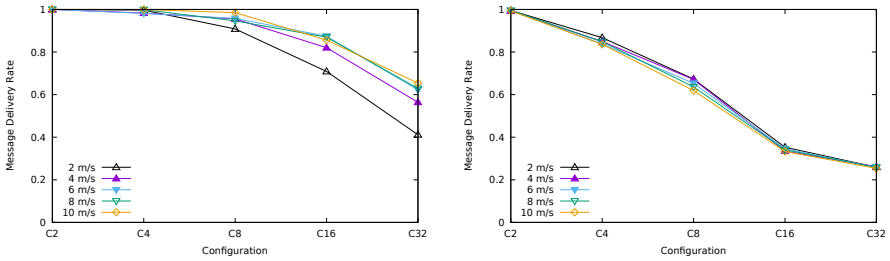
Delivery rate

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.

On Figure 5 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.

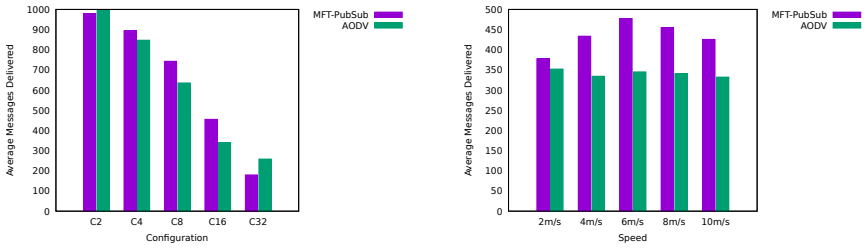
[1] https://omnetpp.org/

In addition, if we look at Figure 6 we can see how many messages are actually delivered. The behavior we see on Figure 5a, where we see an improvement of delivery rate for higher speeds can be further analyzed with Figure 6b. Here we can see a slight increase on 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 on Figure 5 with the amount of messages delivered on Figure 6a, we might think that the difference on 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.

Figure 5: Message delivery rate comparison of both algorithms depending on the size of the simulation.



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

Figure 6: Average number of messages correctly delivered. In Figure 6a we show the results for a node speed of 8 m/s on different configurations. And, in Figure 6b we show the results of all speeds for the C16 configuration.

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. On Figure 7 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 with keeping updated routing tables.

We also observed that some of these values in the case of AODV are higher than they should. If we look at Figure 8, that shows the average and maximum number of hops for a message to be delivered, there is something unusual happening on C4 and C8. On Figure 8b for C4 the maximum number of hops on the simulations was 13, and taking into account that in that configuration there are only 10 nodes, this would mean that the the nodes are not able to correctly route the message.

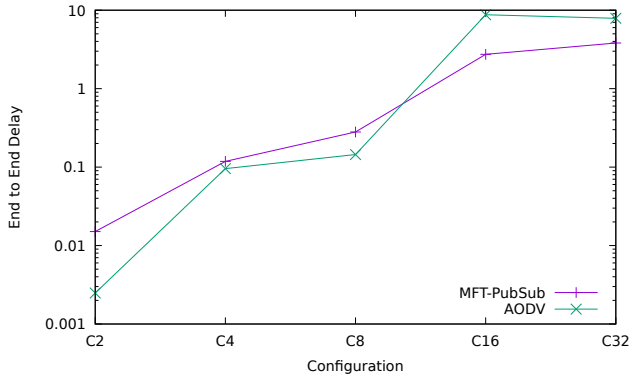
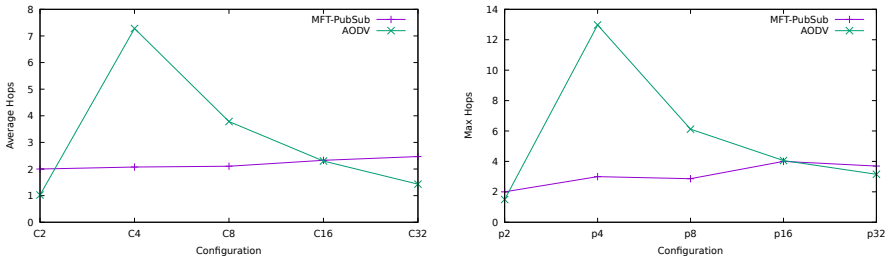


Figure 7: Comparison of end-to-end delay, in seconds, for data messages for a node speed of 8 m/s. Note the logarithmic scale on the y axis.



(a) Average number of hops.

(b) Maximum number of hops.

Figure 8: Number of hops a message does on the network before it is delivered for a node speed of 8 m/s.

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. On Figure 9 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 on our algorithm, AODV has a huge increase in the number of messages it needs to find the correct routes.

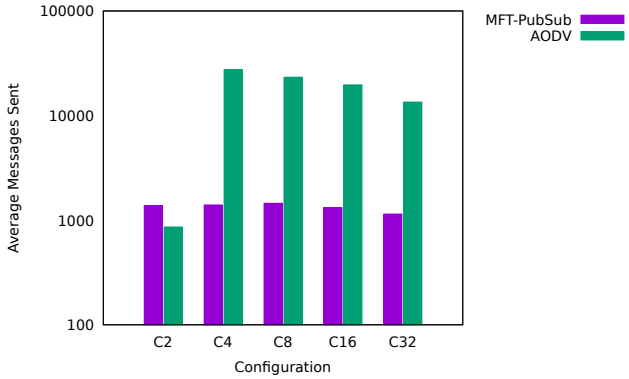


Figure 9: Comparison of the average number of messages sent by each node in order to correctly route messages for a node speed of 8 m/s. Note the logarithmic scale on the y axis.

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.

8 Conclusion

We have presented two approaches to introduce mobility support for a publish/subscribe system. Both of them are based on a spanning tree created via a leader election algorithm that works in situations where we do not know how many nodes there are. This algorithm also gives us a mechanism to detect the movement of nodes as a migration.

The first of the protocols is simpler, but whenever a broker migrates it creates a cascade of messages for all the nodes in its branch on the spanning tree, and those messages contain the whole routing table of the broker. The second protocol, MFT-PubSub, improves on the first one by reducing the number of messages for any migration and only exchanging information when asked.

We have simulated MFT-PubSub on Castalia and compared it to AODV, to compare 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 of 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.

References

[1] Sven Akkermans, Rafael Bachiller, Nelson Matthys, Wouter Joosen, Danny Hughes, and Malisa Vucinic. Towards efficient publish-subscribe middleware in the IoT with IPv6 multicast. In *2016 IEEE International Conference on Communications, ICC 2016, Kuala Lumpur, Malaysia, May 22-27, 2016*, pages 1–6. IEEE, 2016.

[2] Kyoungho An, Shweta Khare, Aniruddha S. Gokhale, and Akram Hakiri. An Autonomous and Dynamic Coordination and Discovery Service for Wide-Area Peer-to-peer Publish/Subscribe: Experience Paper. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 239–248. ACM, 2017.

[3] Aleksandar Antonić, Martina Marjanović, Krešimir Pripužić, and Ivana Podnar Žarko. A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things. *Future Generation Computer Systems*, 56:607–622, 2016.

[4] Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, and Antonino Virgillito. Efficient Publish/Subscribe Through a Self-Organizing Broker Overlay and its Application to SIENA. *Comput. J.*, 50(4):444–459, 2007.

[5] Roberto Baldoni, Carlo Marchetti, Antonino Virgillito, and Roman Vitenberg. Content-Based Publish-Subscribe over Structured Overlay Networks. In *25th International Conference on Distributed Computing Systems (ICDCS 2005), 6-10 June 2005, Columbus, OH, USA*, pages 437–446. IEEE Computer Society, 2005.

[6] Roberto Baldoni, Leonardo Querzoni, Sasu Tarkoma, and Antonino Virgillito. Distributed Event Routing in Publish/Subscribe Systems. In Benoît Garbinato, Hugo Miranda, and Luís E. T. Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 219–244. Springer, 2009.

[7] Guruduth Banavar, Tushar Deepak Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems, Austin, TX, USA, May 31 - June 4, 1999*, pages 262–272. IEEE Computer Society, 1999.

[8] Raphaël Barazzutti, Thomas Heinze, André Martin, Emanuel Onica, Pascal Felber, Christof Fetzer, Zbigniew Jerzak, Marcelo Pasin, and Etienne Riviere. Elastic Scaling of a High-Throughput Content-Based Publish/Subscribe Engine. In *IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, Madrid, Spain, June 30 - July 3, 2014*, pages 567–576. IEEE Computer Society, 2014.

[9] Kai Beckmann and Marcus Thoss. A wireless sensor network protocol for the OMG Data Distribution Service. In *Proceedings of the 10th International Workshop on Intelligent Solutions in Embedded Systems, WISES 2012, Klagenfurt, Carinthia, Austria, July 5-6, 2012*, pages 45–50. IEEE, 2012.

[10] Elizabeth M. Belding-Royer and Charles E. Perkins. Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol. In Harel Kodesh, Victor Bahl, Tomasz Imielinski, and Martha Steenstrup, editors, *MOBICOM '99, The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, Washington, USA, August 15-19, 1999.*, pages 207–218. ACM, 1999.

[11] Christian Berger and Hans P. Reiser. WebBFT: Byzantine Fault Tolerance for Resilient Interactive Web Applications. In Silvia Bonomi and Etienne Rivière, editors, *Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings*, volume 10853 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2018.

[12] Sumeer Bhola, Robert E. Strom, Saurabh Bagchi, Yuanyuan Zhao, and Joshua S. Auerbach. Exactly-once Delivery in a Content-based Publish-Subscribe System. In *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*, pages 7–16. IEEE Computer Society, 2002.

[13] Pruet Boonma and Junichi Suzuki. La Nina: framework in self-adaptive publish/subscribe middleware for wireless sensor networks. *International Journal of Autonomous and Adaptive Communications Systems*, 4(2):180–201, 2011.

[14] Gewu Bu, Thanh Son Lam Nguyen, Maria Potop-Butucaru, and Kim Tha. HyperPubSub: Blockchain based Publish/Subscribe. *CoRR*, abs/1907.03627, 2019.

[15] Ioana Burcea, Hans-Arno Jacobsen, Eyal de Lara, Vinod Muthusamy, and Milenko Petrovic. Disconnected Operation in Publish/Subscribe Middleware. In *5th IEEE International Conference on Mobile Data Management (MDM 2004), 19-22 January 2004, Berkeley, CA, USA*, page 39. IEEE Computer Society, 2004.

[16] Unai Burgos, Ugaitz Amozarrain, Carlos Gómez-Calzado, and Alberto Lafuente. Routing in Mobile Wireless Sensor Networks: A Leader-Based Approach. *Sensors*, 17(7):1587, 2017.

[17] Luis-Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a Global Event Notification Service. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany*, pages 87–92. IEEE Computer Society, 2001.

[18] Hakan Cam, Ozgur Koray Sahingoz, and Ahmet Coskun Sonmez. Wireless Sensor Networks Based on Publish/Subscribe Messaging Paradigms. In Jukka Riekki, Mika Ylianttila, and Minyi Guo, editors, *Advances in Grid and Pervasive Computing - 6th International Conference, GPC 2011, Oulu, Finland, May 11-13, 2011. Proceedings*, volume 6646 of *Lecture Notes in Computer Science*, pages 233–242. Springer, 2011.

[19] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2(5):483–502, 2002.

[20] Mauro Caporuscio, Antonio Carzaniga, and Alexander L. Wolf. Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications. *IEEE Trans. Software Eng.*, 29(12):1059–1071, 2003.

[21] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[22] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony I. T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[23] Raphaël Chand and Pascal Felber. XNET: A Reliable Content-Based Publish/Subscribe System. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 264–273. IEEE Computer Society, 2004.

[24] Tiancheng Chang, Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. P2S: a fault-tolerant publish/subscribe infrastructure. In Umesh Bellur and Ravi Kothari, editors, *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, pages 189–197. ACM, 2014.

[25] Tiancheng Chang and Hein Meling. Byzantine Fault-Tolerant Publish/Subscribe: A Cloud Computing Infrastructure. In *IEEE 31st Symposium on Reliable Distributed Systems, SRDS 2012, Irvine, CA, USA, October 8-11, 2012*, pages 454–456. IEEE Computer Society, 2012.

[26] Jaime Chen, Manuel Díaz, Bartolomé Rubio, and José M. Troya. PS-QUASAR: A publish/subscribe QoS aware middleware for Wireless Sensor and Actor Networks. *Journal of Systems and Software*, 86(6):1650–1662, 2013.

[27] Alex King Yeung Cheung and Hans-Arno Jacobsen. Load Balancing Content-Based Publish/Subscribe Systems. *ACM Trans. Comput. Syst.*, 28(4):9:1–9:55, 2010.

[28] Mariano Cilia, Ludger Fiege, C. Haul, Andreas Zeidler, and Alejandro P. Buchmann. Looking into the past: enhancing mobile publish/subscribe middleware. In Hans-Arno Jacobsen, editor, *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, DEBS 2003, Sunday, June 8th, 2003, San Diego, California, USA (in conjunction with SIGMOD/PODS)*. ACM, 2003.

[29] Paolo Costa, Matteo Migliavacca, Gian Pietro Picco, and Gianpaolo Cugola. Introducing reliability in content-based publish-subscribe through epidemic algorithms. In Hans-Arno Jacobsen, editor, *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, DEBS 2003, Sunday, June 8th, 2003, San Diego, California, USA (in conjunction with SIGMOD/PODS)*. ACM, 2003.

[30] Paolo Costa, Matteo Migliavacca, Gian Pietro Picco, and Gianpaolo Cugola. Epidemic Algorithms for Reliable Content-Based Publish-Subscribe: An Evaluation. In *24th International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March 2004, Hachioji, Tokyo, Japan*, pages 552–561. IEEE Computer Society, 2004.

[31] Paolo Costa and Gian Pietro Picco. Semi-Probabilistic Content-Based Publish-Subscribe. In *25th International Conference on Distributed Computing Systems (ICDCS 2005), 6-10 June 2005, Columbus, OH, USA*, pages 575–585. IEEE Computer Society, 2005.

[32] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[33] Gianpaolo Cugola, Davide Frey, Amy L. Murphy, and Gian Pietro Picco. Minimizing the reconfiguration overhead in content-based publish-subscribe. In Hisham Haddad, Andrea Omicini, Roger L. Wainwright, and Lorie M. Liebrock, editors, *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004*, pages 1134–1140. ACM, 2004.

[34] Gianpaolo Cugola and Hans-Arno Jacobsen. Using publish/subscribe middleware for mobile systems. *Mobile Computing and Communications Review*, 6(4):25–33, 2002.

[35] Gianpaolo Cugola, Amy L. Murphy, and Gian Pietro Picco. Content-Based Publish-Subscribe in a Mobile Environment. In Paolo Bellavista and Antonio Corradi, editors, *The Handbook of Mobile Middleware*, pages 257–285. Auerbach Publications/CRC, 2006.

[36] Gianpaolo Cugola, Gian Pietro Picco, and Amy L. Murphy. Towards Dynamic Reconfiguration of Distributed Publish-Subscribe Middleware. In Alberto Coen-Porisini and André van der Hoek, editors, *Software Engineering and Middleware, Third International Workshop, SEM 2002. Orlando, FL, USA, May 20-21, 2002, Revised Papers*, volume 2596 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2002.

[37] Ernesto García Davis and Anna Calveras Augé. Publish/Subscribe Protocol in Wireless Sensor Networks: Improved Reliability and Timeliness. *KSII Transactions on Internet and Information Systems*, 12(4):1527–1552, 2018.

[38] João Paulo de Araujo, Luciana Arantes, Elias P. Duarte, Luiz A. Rodrigues, and Pierre Sens. VCube-PS: A causal broadcast topic-based publish/subscribe system. *Journal of Parallel and Distributed Computing*, 125:18–30, 2019.

[39] Andrea Detti, Dimitri Tassetto, Nicola Blefari Melazzi, and Francesco Fedi. Exploiting content centric networking to develop topic-based, publish–subscribe MANET systems. *Ad hoc networks*, 24:115–133, 2015.

[40] Augusto Morales Dominguez, Tomás Robles, Ramón Alcarria, and Edwin Cedeño. A Hot-topic based Distribution and Notification of Events in Pub/Sub Mobile Brokers. *Network Protocols & Algorithms*, 5(1):90–110, 2013.

[41] C. Esposito, M. Platania, and R. Beraldi. Reliable and Timely Event Notification for Publish/Subscribe Services Over the Internet. *IEEE/ACM Transactions on Networking*, 22(1):230–243, Feb 2014.

[42] Christian Esposito, Domenico Cotroneo, and Stefano Russo. On reliability in publish/subscribe services. *Computer Networks*, 57(5):1318–1343, 2013.

[43] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[44] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler. Supporting Mobility in Content-Based Publish/Subscribe Middleware. In Markus Endler and Douglas C. Schmidt, editors, *Middleware 2003, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 16-20, 2003, Proceedings*, volume 2672 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2003.

[45] Julien Gascon-Samson, Franz-Philippe Garcia, Bettina Kemme, and Jörg Kienzle. Dynamoth: A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud. In *35th IEEE International Conference on Distributed Computing Systems, ICDCS 2015, Columbus, OH, USA, June 29 - July 2, 2015*, pages 486–496. IEEE Computer Society, 2015.

[46] Carlos Gómez-Calzado, Alberto Lafuente, Mikel Larrea, and Michel Raynal. Fault-Tolerant Leader Election in Mobile Dynamic Distributed Systems. In *IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC 2013, Vancouver, BC, Canada, December 2-4, 2013*, pages 78–87. IEEE, 2013.

[47] Cenk Gündogan, Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. HoPP: Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things. In *43rd IEEE Conference on Local Computer Networks, LCN 2018, Chicago, IL, USA, October 1-4, 2018*, pages 331–334. IEEE, 2018.

[48] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In Hans-Arno Jacobsen, editor, *Middleware 2004, ACM/IFIP/USENIX International Middleware Conference, Toronto, Canada, October 18-20, 2004, Proceedings*, volume 3231 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2004.

[49] Akram Hakiri, Pascal Berthou, Aniruddha S. Gokhale, and Slim Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications. *IEEE Communications Magazine*, 53(9):48–54, 2015.

[50] Daniel Happ, Niels Karowski, Thomas Menzel, Vlado Handziski, and Adam Wolisz. Meeting IoT platform requirements with open pub/sub solutions. *Annales des Télécommunications*, 72(1-2):41–52, 2017.

[51] Yongqiang Huang and Hector Garcia-Molina. Publish/Subscribe Tree Construction in Wireless Ad-Hoc Networks. In Ming-Syan Chen, Panos K. Chrysanthis, Morris Sloman, and Arkady B. Zaslavsky, editors, *Mobile Data Management, 4th International Conference, MDM 2003, Melbourne, Australia, January 21-24, 2003, Proceedings*, volume 2574 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2003.

[52] Yongqiang Huang and Hector Garcia-Molina. Publish/Subscribe in a Mobile Environment. *Wireless Networks*, 10(6):643–652, 2004.

[53] Urs Hunkeler, Hong Linh Truong, and Andy J. Stanford-Clark. MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks. In Sunghyun Choi, Jim Kurose, and Krithi Ramamritham, editors, *Proceedings of the Third International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2008), January 5-10, 2008, Bangalore, India*, pages 791–798. IEEE, 2008.

[54] Hans-Arno Jacobsen, Alex King Yeung Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. The PADRES Publish/Subscribe System. In Annika Hinze and Alejandro P. Buchmann, editors, *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. IGI Global, 2010.

[55] Michael A. Jaeger, Helge Parzyjegla, Gero Mühl, and Klaus Herrmann. Self-organizing broker topologies for publish/subscribe systems. In Yookun Cho, Roger L. Wainwright, Hisham Haddad, Sung Y. Shin, and Yong Wan Koo, editors, *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*, pages 543–550. ACM, 2007.

[56] Hojjat Jafarpour, Sharad Mehrotra, and Nalini Venkatasubramanian. A Fast and Robust Content-based Publish/Subscribe Architecture. In *Proceedings of The Seventh IEEE International Symposium on Networking Computing and Applications, NCA 2008, July 10-12, 2008, Cambridge, Massachusetts, USA*, pages 52–59. IEEE Computer Society, 2008.

[57] Leander Jehl and Hein Meling. Towards Byzantine fault tolerant publish/subscribe: a state machine approach. In Christian Cachin and Robbert van Renesse, editors, *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems, HotDep 2013, Farmington, Pennsylvania, USA, November 3, 2013*, pages 5:1–5:5. ACM, 2013.

[58] Zbigniew Jerzak. *XSiena: The Content-Based Publish/Subscribe System.* PhD thesis, Dresden University of Technology, 2009.

[59] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. Reliable and Highly Available Distributed Publish/Subscribe Service. In *28th IEEE Symposium on Reliable Distributed Systems (SRDS 2009), Niagara Falls, New York, USA, September 27-30, 2009*, pages 41–50. IEEE Computer Society, 2009.

[60] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. Partition-Tolerant Distributed Publish/Subscribe Systems. In *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011), Madrid, Spain, October 4-7, 2011*, pages 101–110. IEEE Computer Society, 2011.

[61] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. Opportunistic Multipath Forwarding in Content-Based Publish/Subscribe Overlays. In Priya Narasimhan and Peter Triantafillou, editors, *Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings*, volume 7662 of *Lecture Notes in Computer Science*, pages 249–270. Springer, 2012.

[62] Leslie Lamport. The Part-Time Parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[63] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. Adaptive Content-Based Routing in General Overlay Topologies. In Valérie Issarny and Richard E. Schantz, editors, *Middleware 2008, ACM/IFIP/USENIX 9th International Middleware Conference, Leuven, Belgium, December 1-5, 2008, Proceedings*, volume 5346 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2008.

[64] Jorge E Luzuriaga, Juan Carlos Cano, Carlos Calafate, Pietro Manzoni, Miguel Perez, and Pablo Boronat. Handling mobility in IoT applications using the MQTT protocol. In *2015 Internet Technologies and Applications (ITA)*, pages 245–250. IEEE, 2015.

[65] Luca Mottola, Gianpaolo Cugola, and Gian Pietro Picco. A Self-Repairing Tree Topology Enabling Content-Based Routing in Mobile Ad Hoc Networks. *IEEE Trans. Mob. Comput.*, 7(8):946–960, 2008.

[66] Gero Mühl. *Large-Scale Content-Based Publish/Subscribe Systems.* PhD thesis, Darmstadt University of Technology, 2002.

[67] Gero Mühl, Andreas Ulbrich, Klaus Herrmann, and Torben Weis. Disseminating Information to Mobile Clients Using Publish-Subscribe. *IEEE Internet Computing*, 8(3):46–53, 2004.

[68] Vinod Muthusamy, Milenko Petrovic, and Hans-Arno Jacobsen. Effects of routing computations in content-based routing networks with mobile data sources. In Thomas F. La Porta, Christoph Lindemann, Elizabeth M. Belding-Royer, and Songwu Lu, editors, *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking, MOBICOM 2005, Cologne, Germany, August 28 - September 2, 2005*, pages 103–116. ACM, 2005.

[69] Charles E. Perkins and Elizabeth M. Belding-Royer. Ad-hoc On-Demand Distance Vector Routing. In *2nd Workshop on Mobile Computing Systems and Applications (WMCSA '99), February 25-26, 1999, New Orleans, LA, USA*, pages 90–100. IEEE Computer Society, 1999.

[70] Van-Nam Pham and Eui-Nam Huh. An Efficient Edge-Cloud Publish/Subscribe Model for Large-Scale IoT Applications. In Sukhan Lee, Roslan Ismail, and Hyunseung Choo, editors, *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication, IMCOM 2019, Phuket, Thailand, January 4-6, 2019*, volume 935 of *Advances in Intelligent Systems and Computing*, pages 130–140. Springer, 2019.

[71] Gian Pietro Picco, Gianpaolo Cugola, and Amy L. Murphy. Efficient Content-Based Event Dispatching in the Presence of Topological Reconfiguration. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003), 19-22 May 2003, Providence, RI, USA*, pages 234–243. IEEE Computer Society, 2003.

[72] Peter R. Pietzuch and Jean Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *22nd International Conference on Distributed Computing Systems, Workshops (ICDCSW '02) July 2-5, 2002, Vienna, Austria, Proceedings*, pages 611–618. IEEE Computer Society, 2002.

[73] Thadpong Pongthawornkamol, Klara Nahrstedt, and Guijun Wang. Reliability and Timeliness Analysis of Fault-tolerant Distributed Publish/Subscribe Systems. In Jeffrey O. Kephart, Calton Pu, and Xiaoyun Zhu, editors, *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, pages 247–257. USENIX Association, 2013.

[74] Gowri Sankar Ramachandran, Kwame-Lante Wright, and Bhaskar Krishnamachari. Trinity: A Distributed Publish/Subscribe Broker with Blockchain-based Immutability. *CoRR*, abs/1807.03110, 2018.

[75] Cristiano G. Rezende, Bruno P. S. Rocha, and Antonio Alfredo Ferreira Loureiro. Publish/subscribe architecture for mobile ad hoc networks. In Roger L. Wainwright and Hisham Haddad, editors, *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 1913–1917. ACM, 2008.

[76] David S. Rosenblum and Alexander L. Wolf. A Design Framework for Internet-Scale Event Observation and Notification. In Mehdi Jazayeri and Helmut Schauer, editors, *Software Engineering - ESEC/FSE'97, 6th European Software Engineering Conference Held Jointly with the 5th ACM*

SIGSOFT Symposium on Foundations of Software Engineering, Zurich, Switzerland, September 22-25, 1997, Proceedings, volume 1301 of *Lecture Notes in Computer Science*, pages 344–360. Springer, 1997.

[77] Pooya Salehi, Christoph Doblander, and Hans-Arno Jacobsen. Highly-available Content-based Publish/Subscribe via Gossiping. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, DEBS 2016, pages 93–104, New York, NY, USA, 2016. ACM.

[78] Zigor Salvador. *Client Mobility Support and Communication Efficiency in Distributed Publish/Subscribe*. PhD thesis, University of the Basque Country UPV/EHU, Spain, 2012.

[79] Zigor Salvador, Aurkene Alzua, Mikel Larrea, and Alberto Lafuente. Mobile XSiena: towards mobile publish/subscribe. In Jean Bacon, Peter R. Pietzuch, Joe Sventek, and Ugur Çetintemel, editors, *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS 2010, Cambridge, United Kingdom, July 12-15, 2010*, pages 91–92. ACM, 2010.

[80] Zigor Salvador, Alberto Lafuente, and Mikel Larrea. Design and Evaluation of a Publish/Subscribe Framework for Ubiquitous Systems. In Kan Zheng, Mo Li, and Hongbo Jiang, editors, *Mobile and Ubiquitous Systems: Computing, Networking, and Services - 9th International Conference, MobiQuitous 2012, Beijing, China, December 12-14, 2012. Revised Selected Papers*, volume 120 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 50–63. Springer, 2012.

[81] Zigor Salvador, Mikel Larrea, and Alberto Lafuente. Phoenix: A Protocol for Seamless Client Mobility in Publish/Subscribe. In *11th IEEE International Symposium on Network Computing and Applications, NCA 2012, Cambridge, MA, USA, August 23-25, 2012*, pages 111–120. IEEE Computer Society, 2012.

[82] Jan Hendrik Schönherr, Helge Parzyjegla, and Gero Mühl. Clustered publish/subscribe in wireless actuator and sensor networks. In Sotirios Terzis, editor, *Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC 2008), held at the ACM/IFIP/USENIX 9th International Middleware Conference, December 1-5, 2008, Leuven, Belgium*, pages 60–65. ACM, 2008.

[83] Tarek R. Sheltami, Anas A. Al-Roubaiey, and Ashraf S. Hasan Mahmoud. A survey on developing publish/subscribe middleware over wireless sensor/actuator networks. *Wireless Networks*, 22(6):2049–2070, 2016.

[84] Gerry Siegemund and Volker Turau. A Self-Stabilizing Publish/Subscribe Middleware for IoT Applications. *ACM Transactions on Cyber-Physical Systems*, 2(2):12:1–12:26, 2018.

[85] Magnus Skjegstad, Frank T. Johnsen, Trude Hafsøe Bloebaum, and Torleiv Maseng. Mist: A Reliable and Delay-Tolerant Publish/Subscribe Solution for Dynamic Networks. In Albert Levi, Mohamad Badra, Matteo Cesana,

Mona Ghassemian, Özgür Gürbüz, Nafaâ Jabeur, Marek Klonowski, Antonio Maña, Susana Sargento, and Sherali Zeadally, editors, *5th International Conference on New Technologies, Mobility and Security, Istanbul, Turkey, NTMS 2012, May 7-10, 2012*, pages 1–8. IEEE, 2012.

[86] Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira, Nelson S. Rosa, Carlos André Guimarães Ferraz, and Judith Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, 10(1):37–44, 2006.

[87] Robert E. Strom, Guruduth Banavar, Tushar Deepak Chandra, Marc A. Kaplan, Kevan Miller, Bodhi Mukherjee, Daniel C. Sturman, and Michael Ward. Gryphon: An Information Flow Based Approach to Message Brokering. *CoRR*, cs.DC/9810019, 1998.

[88] Yasin Tekin and Ozgur Koray Sahingoz. A Publish/Subscribe messaging system for wireless sensor networks. In *Sixth International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2016, Konya, Turkey, July 21-23, 2016*, pages 171–176. IEEE, 2016.

[89] Agnese V. Ventrella, Giuseppe Piro, and L. Alfredo Grieco. Publish-subscribe in mobile information centric networks: Modeling and performance evaluation. *Computer Networks*, 127:317–339, 2017.

[90] Spyros Voulgaris, Etienne Riviere, Anne-Marie Kermarrec, and Maarten van Steen. Sub-2-Sub: Self-Organizing Content-Based Publish Subscribe for Dynamic Large Scale Collaborative Networks. In Emin Gün Sirer and Ben Y. Zhao, editors, *5th International workshop on Peer-To-Peer Systems, IPTPS 2006, Santa Barbara, CA, USA, February 27-28, 2006*, 2006.

[91] Jinling Wang, Jiannong Cao, and Jing Li. Supporting Mobile Clients in Publish/Subscribe Systems. In *25th International Conference on Distributed Computing Systems Workshops (ICDCS 2005 Workshops), 6-10 June 2005, Columbus, OH, USA*, pages 792–798. IEEE Computer Society, 2005.

[92] George Xylomenos, Xenofon Vasilakos, Christos Tsilopoulos, Vasilios A. Siris, and George C. Polyzos. Caching and Mobility Support in a Publish-Subscribe Internet Architecture. *IEEE Communications Magazine*, 50(7):52–58, 2012.

[93] Young Yoon, Vinod Muthusamy, and Hans-Arno Jacobsen. Foundations for Highly Available Content-Based Publish/Subscribe Overlays. In *2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20-24, 2011*, pages 800–811. IEEE Computer Society, 2011.

[94] Kaiwen Zhang, Vinod Muthusamy, and Hans-Arno Jacobsen. Total Order in Content-Based Publish/Subscribe Systems. In *2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, June 18-21, 2012*, pages 335–344. IEEE Computer Society, 2012.

[95] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *Computer Communication Review*, 32(1):81, 2002.

[96] Yaxiong Zhao and Jie Wu. Building a reliable and high-performance content-based publish/subscribe system. *Journal of Parallel and Distributed Computing*, 73(4):371–382, 2013.

[97] Ye Zhao, Kyungbaek Kim, and Nalini Venkatasubramanian. DYNATOPS: a dynamic topic-based publish/subscribe architecture. In Sharma Chakravarthy, Susan Darling Urban, Peter R. Pietzuch, and Elke A. Rundensteiner, editors, *The 7th ACM International Conference on Distributed Event-Based Systems, DEBS '13, Arlington, TX, USA - June 29 - July 03, 2013*, pages 75–86. ACM, 2013.

[98] Shelley Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Network and Operating System Support for Digital Audio and Video, 11th International Workshop, NOSSDAV 2001, Port Jefferson, NY, USA, June 25-26, 2001, Proceedings*, pages 11–20. ACM, 2001.

[99] Nejc Zupan, Kaiwen Zhang, and Hans-Arno Jacobsen. HyperPubSub: a Decentralized, Permissioned, Publish/Subscribe Service using Blockchains: Demo. In Parisa Jalili Marandi and Alessandro Margara, editors, *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos, Las Vegas, NV, USA, December 11 - 15, 2017*, pages 15–16. ACM, 2017.