Techniques for Efficient Event Routing

Leonardo Querzoni
Techniques for Efficient Event Routing

Thesis Committee

Prof. Roberto Baldoni (Co-Advisor)
Prof. Ricardo Jiménez-Peris (Co-Advisor)
Prof. Roberto Beraldi

Reviewers

Prof. Bettina Kemme
Prof. Anne Marie Kermarrec
Prof. Philip K. McKinley
Author’s address:
Leonardo Querzoni
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
E-mail: querzoni@dis.uniroma1.it
www: http://www.dis.uniroma1.it/~querzoni/
# Contents

## 1 Publish/Subscribe Systems

1.1 Introduction ................................................. 1  
1.2 The publish/subscribe interaction paradigm ................. 2  
1.3 Data model ................................................. 4  
1.4 Architectural model ......................................... 6  
  1.4.1 Network Layer ......................................... 7  
  1.4.2 Overlay Layer ......................................... 9  
  1.4.3 Routing Layer ........................................ 11  
  1.4.4 Matching Engine ...................................... 12  
  1.4.5 Interface Layer ...................................... 13  
1.5 Routing in Distributed Infrastructures ...................... 13  
  1.5.1 Flooding-based Algorithms ............................ 13  
  1.5.2 Selective Algorithms ................................. 15  
1.6 The event-filtering and interest clustering techniques .... 17  
1.7 Research Contribution .................................... 21  

## 2 Interest Clustering for Filter-based Systems

2.1 Measuring similarity of interests among brokers ............ 24  
2.2 Self-organizing Algorithm .................................. 25  
  2.2.1 Algorithm Details ..................................... 26  
  2.2.2 Addressing Network Awareness ....................... 32  
2.3 Evaluation .................................................. 33  
  2.3.1 Experimental model .................................... 33  
  2.3.2 Performance metrics ................................... 35  
  2.3.3 Algorithm convergence ................................ 36  
  2.3.4 Algorithm stability ................................... 37  
  2.3.5 Performance results ................................... 39  
  2.3.6 Taking network latency into account ................ 41  
2.4 Related Work ............................................... 42  


Chapter 1

Distributed Event Routing through Publish/Subscribe Systems

1.1 Introduction

The huge development of the Internet in the 1990s gave birth to the first large scale network-based applications. All these systems were based on the client/server interaction paradigm, a communication model that has dramatically contributed to the development of distributed computing.

In the 1990s the client/server paradigm has been extensively used to develop a huge number and variety of applications. The reasons are simple: it is a very clean model which adheres well to software modularity and usability requirements.

Even if the client/server model has encountered a widespread adoption, it is far from being perfect. The first problem is that it requires participants to know each other, i.e. the client must know in advance the server with which it will interact. A second problem lies in the fact that the interaction is inherently synchronous: a client requests some services from a server and waits for the reply. This form of interaction is also known as request/reply.

The tight coupling between interacting parties is an undesirable characteristic for all those applications that run on a large scale, whose participants are heterogeneous, whose components are not reliable, etc. For these reasons various alternatives to the client/server model have been introduced that try to address the cited shortcomings and satisfy the needs of ever more demanding applications: RPC, RMI, tuple spaces, messages queues and publish/subscribe.

The publish/subscribe interaction paradigm has been recognized, in the last years, as a valid alternative to the client/server model, and has attracted the attention of both the industry and academia. With publish/subscribe, communications are no more
based on the notion of a source/destination channel, but are instead focused on the content of the information exchanged: consumers just express their interests on the type of information they expect to receive, regardless of the source of such information. Thanks to the radically different perspective on communication it offers, the publish/subscribe interaction paradigm can address many issues raised by advanced distributed applications and, more importantly, it is claimed to provide the loosely coupled form of interaction required in large scale dynamic distributed systems.

As a consequence, the publish/subscribe interaction paradigm is nowadays becoming an essential part of large scale distributed systems design, and many software platforms have incorporated event-based communication mechanisms. Information buses are the basis of many systems [94, 19], and a large number of publish/subscribe systems have been developed (e.g. [29, 45, 120, 122, 99, 83, 84, 46, 35]) as well as integrated into modern component platforms such as Java 2 Enterprise Edition [82] or Corba Component Model [92]. A broad range of applications can potentially be favored by the introduction of event-based data diffusion: information dissemination (news tickers, real-time control systems, stock market monitoring, etc.), network monitoring, sensor networks, many applications based on mobile ad-hoc networks, enterprise application integration systems.

In the following of this chapter we will introduce the reader to the details of this interaction paradigm and to the architectures that implement it. Based on these premises the last section of the chapter will be devoted to clarify the contribution of this thesis.

### 1.2 The publish/subscribe interaction paradigm

![Figure 1.1: A schematic representation of a publish/subscribe system.](image)

The publish/subscribe interaction paradigm [52, 40, 18] provides users of a system with an alternative communication model with respect to the classical synchronous request/reply model. In a publish/subscribe system (see figure 1.1) users interact playing one of two roles:

**Publishers**: users that produce information and publish it into the system;
1.2. THE PUBLISH/SUBSCRIBE INTERACTION PARADIGM

Subscribers: users that consume information received from the system.

When a publisher publishes some information in the system, an event is generated that contains, as a sort of payload, the published information. An event notification service then notifies this event to the subscribers that, in this way, receive the published information.

Usually each subscriber is interested only in a subset of all the produced events; in order to limit which events it will be notified of, a subscriber can express its interests issuing one or more subscriptions. Each subscription works as a filter on the set of events produced in the system: the subscriber will be notified about events that satisfy the conditions expressed in one subscription at least. When an event $e$ satisfies the conditions contained in a subscription $s$ we say that $e$ matches subscription $s$, or, more briefly, $e \bowtie s$.

The event notification service can be seen from participants as a black box they interact with. It offers to each participant a standard interface composed by the functions publish, subscribe, unsubscribe and notify. The function publish is used by publishers to publish information and generate the corresponding events, while subscribe and unsubscribe are used by subscribers to issue and revoke subscriptions. notify is a callback function used by the event notification service to notify a subscriber about an event. Other functions can be exposed by the interface of the publish/subscribe system depending on the specific implementation.

The event notification service plays in the system a role of a mediator between publishers and subscribers, completely decoupling the interactions among them. More specifically the publish/subscribe interaction paradigm is able to provide an interaction characterized by:

Space decoupling: the interaction can be completely anonymous, i.e. each publisher does not need to know the identity of subscribers that will be notified for an event it produces and vice-versa.

Time decoupling: the interaction does not require publishers and subscribers to be connected at the same time. A Subscriber can be notified about an event produced by a publisher that is no more active.

Flow decoupling: the production and consumption of events from publishers and subscribers can happen asynchronously.

The lack of any form of coupling between participants renders the publish/subscribe interaction paradigm well suited for large scale and dynamic settings where indirection during interactions helps in the development of more robust systems.
1.3 Data model

The publish/subscribe interaction paradigm is a communication model completely based on message content. The way information published in the system is structured and then selected through subscriptions characterizes various flavors of publish/subscribe. It influences, in fact, the expressiveness of the language used by subscribers to issue subscriptions, and thus, indirectly affects the complexity of the mechanism used to match events against subscriptions. Various proposals have been presented in the literature, but only two of them have been widely recognized and employed: topic-based and content-based selection. Here we introduce these two forms of publish subscribe giving only few details for other minor variants.

**Topic-based selection.** With this data model each piece of data published in the system is tagged by the publisher with a *topic* (or *subject*) that completely characterize its content for the event notification service; in the same way each subscription contains a single topic the subscriber is interested in. In topic-based systems matching between an event $e$ and a subscription $s$ reduces to a simple comparison between the respective topics: if the two topics are equal then $e \odot s$. The actual content of the event (if present) remains hidden to the system and cannot be used for selection.

Let us give an example to clarify this form of event selection. Suppose an application where publishers publish in the system data about new music albums. If the topic-based model is employed, each album should be tagged (for example) with the genre of the music it contains. In this way subscribers could select the events they are interested in simply issuing subscriptions containing the corresponding topics (i.e. the music genres).

The main drawback of the topic-based model is the *coarseness* of the selection model: a subscriber can decide to receive either all the events tagged with a topic or none. There is no way for a subscriber to refine the selection inside the set of events published for a specific topic, apart from discarding undesired events once they have been notified.

This problem can be partially addressed using a very large number of highly specific topics, but this solution can negatively affect the scalability of the whole system as each event will be probably published several times for different topics, and each subscriber will subscribe a lot of different topics to cover all its interests. A different solution is the employment of a topic hierarchy; with this variant topics are organized in a tree, and a subscription for a specific topic implies a subscriptions for all its children topics. This hierarchical organization allows to consider a relation of containment among topics that enriches the expressive power of the subscription mechanism; nevertheless, each topic is constrained to a single *super*-topic, thus limiting topic classification to a single point of view. Multiple classifications can be obtained considering various trees for the same set of topic, but this can pose other problems like multiple notifications for the same event.
1.3. DATA MODEL

Despite these drawbacks the topic-based model is widely adopted both in research prototypes [35, 104, 4] and commercial products [94, 83, 90] thanks to its simplicity.

**Content-based selection.** With the content-based data model subscribers express their interest by specifying conditions over the content of events they want to be notified about. In other words, a filter in a subscription is a query formed by a set of constraints over the event content composed through disjunction or conjunction operators. Possible constraints depend on the event content and on the subscription language. Most subscription languages comprise equality and comparison operators as well as regular expressions [29, 113, 56]. The content-based model permits subscribers a finer selection on the set of events they want to receive, that strictly depends on the expressiveness of the subscription language. This aspect obviously also influences the complexity of matching operation: the higher expressive power of content-based publish/subscribe comes at the price of the higher resource consumption needed to calculate for each event the set of interested subscribers [28, 51].

If we consider the example previously introduced, with the content-based selection model each published event would be constituted by a set of attributes (e.g. the author name, the album title, the number of songs, the musc genre, etc.). In this case each subscription could express constraints over each single attribute characterizing the event.

It is important to note that a topic-based scheme can be emulated through a content-based one simply considering filters comprising a single equality constraint, while the opposite is, in general, not possible. From this point of view the content-based model can be considered as the most general scheme of event selection.

**Other forms of selection.** Various other data models for publish/subscribe have been presented that are usually more specialized variants of the topic-based or content-based model.

The type-based selection model [53] extends the topic-based model considering each event as an object of a certain type and supporting subtype inclusion and multiple inheritance. In this way it enhances the simple tree-based hierarchical structure allowing the usage of type lattices. Thanks to the object nature of events, selection can be refined filtering events on the basis of their attributes (or event methods if present). In some sense the type-based model lies somewhere in between the topic-based and the content-based models, but it also contains characteristics that are not present in the two most used models.

Some research work [36, 37, 116] describe publish/subscribe systems supporting a semistructured data model, typically based on XML documents. XML is not merely a matter of representation but differs in the fact that it introduces the possibility of hierarchies in the language, thus differentiating from a flat content-based
model in terms of an added flexibility. Moreover, it provides natural advantages such as interoperability, independence from implementation and extensibility.

The underlying implicit assumption within all the above-mentioned data models is that participants have to be aware of the structure of produced events, both under a syntactic and a semantic point of view. Concept-based addressing [38] allows to describe event schema at a higher level of abstraction by using ontologies, that provide a knowledge base for an unambiguous interpretation of the event structure, by using metadata and mapping functions.

1.4 Architectural model

![Event Notification Service](image)

Figure 1.2: Event Notification Service with a distributed infrastructure.

In section 1.2 we introduced the event notification service as a component whose purpose is correctly notify subscribers about published events. The event notification service is seen by publishers and subscribers as a black box they interact with, completely ignoring the way it works. Nevertheless, its internal structure strongly affects non functional attributes of the whole publish/subscribe system like efficiency, scalability and availability.

While early publish/subscribe systems were proposed as centralized architectures, the demand for more reliable systems, better performance and higher scalability moved the interest of both researchers and industry toward distributed architectures. In a distributed publish/subscribe system the event notification service is constituted by a set of $N$ processes $B_1, \ldots, B_N$, called event brokers, that communicate through message exchanges. These processes, each running on a different host (or node), collaborate in order to realize the functionalities of a publish/subscribe system. Figure 1.2 shows an example of the internal structure of a distributed ENS where various
brokers (B) collaborate to serve the requests coming from publishers (P) and subscribers (S).

### 1.4 ARCHITECTURAL MODEL

The general architecture of a distributed publish/subscribe system is depicted in figure 1.3 and is constituted of five layers: **Network, Overlay, Routing, Matching** and **Interface**. In the following we will detail the functionalities associated to each layer as well as the possible different solutions for its design.

#### 1.4.1 Network Layer

Network protocols anchor a publish/subscribe system to the underlying network by allowing transmission of data among participants. Due to the fact that a publish/subscribe system could span over heterogeneous networks (e.g., LANs, WANs, mobile networks, etc.), it could employ more than a single network protocol either to cope with different software/hardware conditions that could be found in a given part of the network or to maximize performance.

**Transport Level.** Publish/subscribe systems are usually built exploiting the functionalities of common transport-level protocols. That is, nodes in the overlay infrastructure communicate directly through TCP or UDP sockets or using specific TCP-based middleware protocols (like IIOP or SOAP). This choice allows greater flexibility and ease of deployment, though in some situations the deployment over a wide-area network can be limited by the presence of network firewalls or private networks, requiring the intervention of an administrator for configuration.

**Network-level Multicast.** Directly exploiting local-area or wide-area multicast and broadcast facilities is an alternative solution to efficiently realize many-to-many event
CHAPTER 1. PUBLISH/SUBSCRIBE SYSTEMS

diffusion. For example, IP Multicast can be directly used in wide-area topic-based systems, as each topic can be mapped exactly to one multicast group. Using IP multicast for content-based systems is instead not straightforward because subscribers can be mapped to multicast groups only when events are published, and groups can completely change from event to event. This problem inspired some research work targeting at organizing subscribers in clusters, where subscribers in the same cluster contain most of the subscriptions in common [95, 106, 107, 61]. The main drawback of IP multicast is however its lack of a widespread deployment [49, 114]. Hence, network-level multicasting cannot in general be considered as a feasible solution for applications deployed over a WAN (for example TIB/RV [94] or the CORBA Notification Service [91] uses network multicast only for diffusing notifications inside a local area network).

*Mobile Networks.* Several work on publish/subscribe system address the mobile network scenario, and do it in two different fashions. Some work consider systems where all nodes can freely move, forming a mobile ad-hoc network (MANET) [65, 3, 116, 80]. In other work, part of the nodes forms a fixed infrastructure and only clients can roam, being always at one-hop away from the fixed infrastructure [57, 27, 89]. In both cases the network interface for mobile nodes can be either a transport protocol built on the top of a data link layer specific for mobile nodes, such as 802.11b, or the protocol 802.11b itself. Obviously mobility induces specific resource constraints over the overall architecture design such as battery drain, limited bandwidth etc. Also phenomena such as temporary disconnections and node unavailability should be considered common and have to be dealt with. Moreover, mobile clients typically have to include the support for location-aware subscriptions, i.e. subscriptions whose constraints are based on the current position of the user. Note that one-hop mobile networks do not pose, indeed, important research issues being considered as an engineering of wired solutions. Publish/subscribe systems built over MANETs have instead to cope with constantly changing topologies and partitions that makes the problem of event notification non-trivial.

*Sensor Networks.* Another kind of application the publish/subscribe interaction paradigm is perfectly suited for are sensor networks. A sensor network is composed by small devices capable of taking various measurements from the environment, and transmitting them toward applications hosted into specific base stations. Sensors communicate (among each other and with base stations) through broadcast-based facilities over wireless protocols such as 802.15.4. It is evident that the publish/subscribe interaction paradigm fits naturally this context: sensors publish measurements, that are received by subscribers placed in base stations. With respect to a general publish/subscribe system, a further assumption can be made in this context: the number of subscribing nodes is very low, as sensors are exclusively publishers and they are
predominant in number with respect to base stations. From an architectural point of view, sensor networks are similar to MANETs, regarding aspects such as the topology determined by the devices transmission range and the limited power supply. Obviously, the fact that a sensor network is a fixed network reduces the complexity related to dynamic topology changes, though dynamicity has still to be taken into account, because devices are in general failure-prone and they frequently rely on stand-by periods for power saving. Work presenting pub/sub solutions specifically tailored to sensor networks are [64, 41].

1.4.2 Overlay Layer

The communication primitives provided by the network layer are used by brokers to exchange messages. Each message is sent from a broker $B_i$ to a broker $B_j$ through an application-level channel, or link, $l_{ij}$ connecting them. These links are pure abstractions as they are not required to represent permanent, long-lived connections, so that the neighborhood of a broker in the overlay network is determined purely by a knowledge relation.

All the brokers, together with the links connecting them, form an application-level network (overlay network) that incarnates the distributed event notification service. This overlay network can be managed through an ad-hoc solution, specifically designed for the considered publish/subscribe system, or employing a generic overlay maintenance protocol (OMP).

The first solution has been widely adopted for both commercial products and research prototypes [94, 123, 29, 45, 46]. In these systems brokers are usually dedicated servers that act in behalf of several remote clients. The topology of the overlay network can assume various forms (cliques, random graphs, trees, rings, etc.), depending on the algorithms employed on the routing layer, but usually it is assumed to be managed by an administrator, based on technical or administrative constraints. For this reason, a broker overlay is inherently static: topology changes are considered to be rare, mainly to face events such as addition of new brokers or repairing after a failure. This assumption also limits in some way the maximum size the event notification service can reach: the more the event notification service size grows, the more its administration becomes difficult.

Systems where a centralized management is impractical (e.g. peer-to-peer scenarios) must employ self-management techniques to maintain the overlay network. These systems are usually assumed to be constituted by a huge number of brokers, each acting in behalf of a single client that usually resides on the same host. Brokers are assumed volatile, i.e. they can join and leave the system, have heterogeneous capabilities, are unreliable, etc. In such environment the broker overlay is inherently dynamic. For this reason publish/subscribe systems designed with these assumptions usually employ some generic overlay maintenance protocol that take care of all these aspects, hiding their complex management to the broker and providing some power-
ful communication primitive. Overlay maintenance protocols can be broadly divided in two families: protocols for structured and unstructured systems. A case apart is represented by publish/subscribe system built over mobile ad-hoc networks or sensor networks.

**Peer-to-peer Structured Overlay.** A peer-to-peer structured overlay infrastructure is a self-organized application-level network composed by a set of nodes forming a structured graph over a virtual key space; each key of this space is mapped to a node. The structure imposed to the graph permits efficient discovery of data items and this, in turns, allows the implementation of efficient uni/multicast communication primitives.

A structured overlay infrastructure ensures that a correspondence always exists between any key and an active node in the system despite churn (the continuous process of arrivals and departures of nodes of the overlay) and node failures. These self-organizing properties make structured solutions well suited for those environments where human administration interventions cannot be considered a feasible solution.

As a consequence of the popularity of structured overlays, many such systems have been developed, we cite among the others Pastry [110], Chord [119], Tapestry [139], CAN [104], I3 [118]. The simple data-lookup services offered by these systems have been later extended to provide more complex communication and coordination mechanisms for advanced applications [47, 22, 111, 63, 16].

Structuring a publish/subscribe system over an existing overlay maintenance protocol means leveraging its self-organization capabilities, by building a pub/sub interface over it. Examples of systems using this solution are Bayeux [141] and Scribe [35], for what concerns topic-based systems, and Meghdoot [62], Hermes [100] and Rebeca [122], for what concerns content-based systems.

**Peer-to-peer Unstructured Overlays.** Unstructured overlay networks [96, 2, 55, 73] organize nodes in a random graph characterized by some desired properties like low-diameter, high connectivity, constant-node degree. Each node in the overlay network maintains a partial view of the whole system, containing network addresses of other nodes participating to the same network. The union of all the nodes together with their partial views, constitutes the whole overlay. Due to the large scale of such systems, partial views are usually limited in their size to a constant degree or to a size that varies only logarithmically with the overlay size.

Nodes joining the system use some randomized algorithm that starts at a bootstrap node chosen in some way among those already in the system. This bootstrap node is used to build an initial partial view through which a new node effectively becomes part of the overlay. Node leaves can be treated with two different approaches: a first approach, named active leave, requires leaving nodes to execute some protocol before actually leaving the system in order to guarantee connectivity despite their
departure. A different approach, named passive leave, treats leaving nodes as failed nodes: the system does not expect leaving nodes to execute any special action, as the remaining nodes are able to withstand their departure thanks to the high connectivity of the graph. Random structure and resource positioning force the usage of resource location algorithms based on flooding or random walks [58].

Overlay maintenance protocols for unstructured systems has been considered for the design of publish/subscribe system only by few research proposals appeared in the last years. Among them we cite [43], [4], and Sub-2-Sub [131].

**Overlays for Mobile Networks.** In a mobile setting, topology changes in the overlay are due to the mobility pattern as well as churn and node failures. Mobility determines the topology of the network. Moreover, specific algorithms are required for creating and maintaining the conditions under which routing algorithms can work, such as connectivity or consistency of routing data structures [65, 3, 98]. The maintenance of such structures can become cumbersome in environments characterized by high mobility and low resources. Hence, typical solutions to event diffusion in MANETs are directly mapped on the physical network topology and rely on the MAC layer (e.g. 802.11b) by exploiting its beaoning system and its broadcast characteristics [5].

1.4.3 Routing Layer

In a distributed publish/subscribe system events and subscriptions can be issued on any of the brokers constituting it. Therefore the system must be equipped with a routing mechanism whose aim is to let events flow inside the event notification service and reach all the intended target subscribers.

More formally: given an event \( e \) published on broker \( B_i \) and a subscription \( s \) issued on broker \( B_j \), if \( e \odot s \), then the routing mechanism must route both \( e \) and \( s \) on a same broker \( B_k \) where matching will take place. This routing mechanism is often referred to, in the literature, as event routing, even if it affects both events and subscriptions.

Note that routing of \( e \) and \( s \) must not necessarily take place at the same time: usually a subscription is routed as soon as it is issued to a set of destination brokers where it will be hosted during its whole lifetime. Each event is routed as soon it is published toward a set of destination brokers (target brokers) that ideally should host all the subscriptions matched by the event. The difficulties in defining a global temporal ordering between a subscription and a publication that occurred at two different nodes makes this definition of routing rather ambiguous. A discussion on this point as well as formal specifications of the event routing problem can be found in [7].

During the routing of an event throughout the event notification service, it can be received by some brokers that do not host any subscription matched by it; messages containing such events are considered by those brokers as spam messages, as
they only “waste” local resources. Spam messages usually exist because the routing mechanism is not always precise, or maybe because the receiving broker must re-route them toward other destinations; in any case spam messages are common in many publish/subscribe systems, and their presence can be reduced but rarely eliminated.

The main issue with the routing layer is scalability, i.e. an increase of the number of brokers, subscriptions and publications should not cause a serious (e.g., exponential) degradation of performance. This requires controlling the publication process on one hand, in order to avoid as much as possible the presence of spam messages. On the other hand, reducing the amount of routing information that must be maintained at brokers, in order to support and flexibly allow subscription changes. These two aspects are evidently conflicting and reaching a balance between them is the main aim of a publish/subscribe systems designer.

Section 1.5 offers a more in depth analysis of the various solutions available to implement this layer.

1.4.4 Matching Engine

Matching is the process of checking if an event satisfies the conditions expressed in a subscription. Matching is usually performed by the publish/subscribe system in order to determine whether notifying an event to a subscriber or not, but, depending on the solution employed for the routing layer, it can sometimes be used even during the event diffusion process. Given the central role played by the matching process inside the event notification service, its efficiency is a fundamental issue for the overall performance of the publish/subscribe system. Techniques for efficiently performing the matching operation are then one important research issue related in the pub/sub field. They can be grouped in two main categories [108], namely predicate indexing algorithms and testing network algorithms. Predicate indexing algorithms are structured in two phases: the first phase is used to decompose subscriptions into elementary constraints and determine which constraints are satisfied by the notification; in the second phase the results of the first phase are used to determine the filters in which all constraints match the event. Matching algorithms falling into the predicate indexing family are [135, 97, 56, 31]. Testing network algorithms ([1, 59, 25]) are based on a pre-processing of the set of subscriptions that builds a data structure (a tree in [1] and [59] or a binary decision diagram in [25]) composed by nodes representing the constraints in each filter. The structure is traversed in a second phase of the algorithm, by matching the event against each constraint. An event matches a filter when the data structure is completely traversed by it. Simpler data models, like the topic-based selection model, can be treated with easier matching strategies.

A formal complexity analysis and comparison of matching algorithms can be found in [72].
1.5. ROUTING IN DISTRIBUTED INFRASTRUCTURES

1.4.5 Interface Layer

This layer represents how clients, i.e. publishers and subscribers, interact with the event notification service. Two opposite strategies are usually considered.

With the first strategy brokers constituting the event notification service are specialized servers deployed and maintained by some organization; each broker serves as an access point for a very large number of clients, storing their subscriptions and publishing their data. In this case, fairly common for all those publish/subscribe systems that are not designed as peer-to-peer services, each broker interacts remotely with the clients that uses it as an access point to the system.

A second strategy requires each client to actively participate to the system acting also as a broker. Each broker thus only serves requests coming from local applications that want to access the system. This strategy is obviously perfectly suited for peer-to-peer applications where the system itself is made up of the users themselves, but it is also widely used in MANET-based publish/subscribe systems, where every mobile host is part of the system.

Note that also a mixed approach can be considered, but currently none of the existing proposals adopt it.

1.5 Routing in Distributed Infrastructures

Various approach to event and subscription routing in a distributed publish/subscribe system can be classified on the basis of two design choices:

- The strategy used to route subscriptions
- The strategy used to route events

These two strategies, put together, completely define the behavior of the routing mechanism, uniquely identifying, given an event $e$ and a subscription $s$ such that $e \circ s$, the broker where the matching between $e$ and $s$ will take place.

1.5.1 Flooding-based Algorithms

A lot of different routing algorithms for publish/subscribe systems have been proposed in the literature, but most of them can be in some way recognized as improvements of two basic algorithms: event flooding and subscription flooding.

Event Flooding

Event flooding is often considered as a naive solution to implement the publish/subscribe interaction paradigm. With event flooding subscriptions are hosted on the same broker where they are issued (i.e. no routing is needed for subscriptions), while each
event is forwarded to every broker in the system. The result of applying these two strategies is that $e$ and $s$ are always matched on the broker where $s$ was issued.

The main advantage of this solution is that it can be easily implemented on any overlay infrastructure, just relaying on a broadcast communication primitive. This primitive can be implemented at the routing layer or provided by the underlying layers. It can provide various degrees of quality of service: from a simple best-effort model to a completely reliable model.

Another point in favor of event flooding is the small memory overhead generated on brokers: each subscription is hosted only on one broker thus reducing the memory footprint required from the publish/subscribe system to a minimum. Note that here we do not consider the eventual memory overhead required to implement the broadcast primitive used to route events: depending on the desired type of service this primitive can require various amounts of memory overhead.

Despite these good characteristics the event flooding approach is usually considered unsuitable for a vast majority of applications based on publish/subscribe. With event flooding each event is, in fact, forwarded to every broker, regardless of the subscriptions it is hosting. This behavior usually generates a lot of spam messages as each broker can receive a lot of events that have no match with the subscriptions it hosts. Applications characterized by highly selective subscriptions experience in this case an enormous burden imposed on both nodes and the network from the proliferation of spam messages.

Another problem that can afflict event flooding is the latency experienced by the system from the moment an event is published on a broker up to the moment when the last subscriber, whose subscriptions are matched by the event, is notified. This latency is primarily due to the time needed by the employed broadcast primitive to correctly diffuse an event on all brokers.

Subscription Flooding

The subscription flooding approach is the dual of event flooding. In this case each subscription update is forwarded to every broker in the system, while events are not routed outside the broker where they are published. With subscription flooding matching between events and subscriptions thus happens on the broker where each event is published.

With subscription flooding ideally every broker has, at any time, a complete knowledge of the set of active subscriptions in the system. This knowledge can be used to directly notify subscribers from the broker where the event is published, without forwarding it, and without any lag.

However, both simulation studies [86, 29] and practical experiences [113] have shown how these premises are rarely maintained in realistic scenarios, where subscription flooding behaves very poorly. The strategy used to route subscription, in fact, quickly flood the system with a huge number of updates as soon as the set of
active subscriptions changes. Moreover the time needed to completely and correctly update a subscription on each broker can, in some cases, become quite large leading to inconsistent views of the active subscription set on different brokers. These inconsistencies can lead to the notification of unwanted events, or the missed notification of some recently subscribed event.

Moreover the memory footprint generated by the subscription flooding approach can be quite large, even for small applications, thus limiting its applicability in many environments where available resources are limited, like mobile ad hoc networks.

1.5.2 Selective Algorithms

Event flooding and subscription flooding are at the two ends of an imaginary line: on one end information contained in a subscription is maintained on a single broker, thus forcing a flooding approach for event diffusion; on the other end all the information contained in subscriptions is spread all over the event notification service avoiding the need for an event routing phase. Selective algorithms lie in between these two extremes mixing in various ways the goods of both.

These algorithms can be roughly grouped in two main families (minor exceptions exist): those employing a technique known as filtering, and those based on the rendezvous approach.

Event Filtering

In Filtering-based routing [29] events are forwarded only to nodes that lie on an overlay path leading to interested subscribers. Message overhead is reduced by identifying as soon as possible events that are not interesting for any subscriber and arrest their forwarding. This approach has been largely studied and used in the literature [87, 23].

The construction of diffusion paths requires routing information to be stored and maintained on the nodes. Routing information at a node is associated to each of its neighbors in the overlay and consists in the set of subscriptions that are reachable through that broker. This allows to build reverse paths to subscribers followed by events. In practice, copies of all the subscriptions have to be diffused toward all possible publishers, and in the general case, when all nodes may act as publishers for any subscription, this means again flooding all subscriptions. However, differently from the subscription flooding approach, subscription diffusion can be limited by exploiting subscription identity or containment [88], as done in SIENA [29] and REBECA [122].

The natural architecture for this kind of solution are acyclic overlay of brokers. Actually, the presence of cycles requires duplicate detection while diffusing both event and subscriptions and thus is usually avoided in implemented systems. The addressing scheme of a structured overlay does not represent a useful feature in
this type of solution, except for the fact that it can keep the consistent association between a node and its position in the overlay, allowing easy overlay repairing upon failure. However, the consistency of information in the routing tables has still to be provided by specific event routing-level algorithms. This type of solution is considered in [122, 42]. The use of Filtering-based routing over unstructured overlays suffers mainly from the dynamic behavior of the network, that requires frequent updates of the routing information. Moreover, it is not possible to assume an acyclic topology.

The performance of filtering-based routing is obviously influenced by the topology of the overlay network. In particular, the diameter of the topology directly affects the length of the overlay paths traveled by events, thus influencing notifications latency. Obviously, increasing the number of neighbors of a node lowers the diameter of the network, but also the amount of routing information kept by nodes (memory overhead) increases.

RendezVous-based approaches

Rendezvous-based event routing is based on two functions, namely $SN$ and $EN$, used to associate respectively subscriptions and events to nodes in the pub/sub system. In particular, given a subscription $s$, $SN(s)$ returns a set of nodes, named rendezvous nodes of $s$, which are responsible for storing $s$ and forwarding events matching $s$ to all the subscribers of $s$. $EN(e)$ complements $SN$ by returning the rendezvous nodes of $e$, which are the nodes responsible for matching $e$ against subscriptions registered in the system. Upon issuing a subscription $s$, a subscriber sends $s$ to the nodes in $SN(s)$, which store $s$ and the subscribers’ identifier.

Then, rendezvous-based event routing is a two phases process: a publisher sends their events to nodes in $EN(e)$, which match $e$ against the subscriptions they host. For each subscription matched by $e$, $e$ is forwarded to the corresponding subscriber. In order for the matching scheme to work and forward $e$ to the consumers, it is necessary that the rendezvous nodes of $e$ collectively store all the subscriptions matched by $e$, i.e., if $e \ominus s$ for any subscription $s$, then $EN(e) \cap SN(s) \neq \emptyset$.

Rendezvous-based routing has been introduced in [132], and recently many systems appeared following such a scheme (Scribe [35], Bayeux [141], Hermes [99], Meghdoot [62] and [17]). This approach is motivated by the fact that a controlled subscription distribution allows to better load balance subscription storage and management: all subscriptions matching the same events will be hosted by the same node, avoiding a redundant matching to be performed in several different nodes. Also delivery of events is simplified, consisting in the creation of single-rooted diffusion trees starting from target brokers and spanning all subscribers.

However, it is clear that defining the couple of $EN(e)$ and $SN(s)$ functions so that they satisfy the mapping intersection rule is a non-trivial task. This implies defining a clustering of the subscription space, such that each cluster is assigned to a node that
becomes the rendezvous for the subscriptions and events that fall into that cluster.

The powerful abstraction realized by structured overlay networks greatly helps in the definition of the mapping functions, thanks to the fact that the fixed-size key space can be used as a target of the functions rather than the set of nodes. This allows the mapping to be independent from the actual system composition and not be influenced by changes in it.

Maybe the biggest drawback of rendezvous-based solutions is the restrictions it may impose to the subscription language. In general, mapping a multi-dimensional, multi-typed content-based subscription to the uni-dimensional or bi-dimensional numerical-only address space of structured overlays is not straightforward. While numerical range constraints can be intuitively handled, constraints over string attributes, like substrings, prefixes or suffixes, that are an important part of a content-based language, can be hardly reduced to numerical ranges.

Regarding performance, memory overhead depends on the mapping function used. In general, the mapping function should map a subscription to the lower number of nodes possible in order to satisfy the mapping intersection rule. It is natural though that “larger” subscriptions (i.e. matching more events) will be mapped to more nodes with respect to “smaller” ones. This allows also to share the load due to matching. Moreover, routing information should be preserved at a node to reach the rendezvous nodes.

1.6 Increasing the efficiency of event routing through event filtering and interest clustering

Ideally, in order to lower the cost of event dissemination, each broker should only process events that match subscriptions it hosts. Lowering this cost is a relevant objective [29, 86, 127] because it allows to reduce the overall processing load at each broker, due to event matching against subscriptions and event forwarding, thus enhancing the scalability of the whole system. In order to reach this ideal goal, spam messages should be avoided as long as it is possible.

In the previous section we introduced a technique, event filtering, that is widely adopted [29, 122] to reduce the amount of spam produced during event diffusion. Let us consider a simple event diffusion algorithm based on flooding. Such an algorithm, on the occurrence of the publish of an event, should first build on the overlay network a diffusion tree rooted on the broker where the event was issued and spanning all the remaining brokers, and then route the event throughout this tree. The filtering technique is applied between these two steps: on the basis of the information contained in issued subscriptions, it prunes those branches that do not contain matched subscriptions, thus avoiding the proliferation of spam on them.

Figure 1.4 details this mechanism through an example. Suppose we have a publish/subscribe system with two subscribers $S_1$ and $S_2$, a publisher $P_1$, and a dis-
The event filtering technique can greatly improve event flooding performance by pruning those branches of the diffusion tree that do not contain matched subscriptions.

Figure 1.4: The event filtering technique can greatly improve event flooding performance by pruning those branches of the diffusion tree that do not contain matched subscriptions.

The idea at the basis of event filtering is that only part of these spam messages are necessary: messages sent through branches that link the source broker \(B_9\) with the target brokers \(B_4\) and \(B_8\) are necessary, while all the other branches can be safely pruned without any impact on the correctness of the event diffusion operation. The application of event filtering to the spanning tree of figure 1.4(a) leads to the result depicted in figure 1.4(b): all the useless branches have been pruned greatly reducing the amount of spam produced. The spam messages that are still present are only those that cannot be avoided, as they are necessary to correctly deliver \(e\) to all the target brokers. Those brokers residing on the paths that lead from the event source broker to the target brokers are defined pure forwarder brokers as they act just as relaying machines for that specific event.

The example we used gives a clear idea about how far event filtering can go in improving flooding performance, but it pictures a nice scenario that could not happen in real applications. If the number of subscriptions matched by \(e\) is larger, and they
are scattered on various brokers, the filtering mechanism will probably prune just a few branches as a lot of spam messages will be necessary to bring the diffused event toward target brokers. [29] and [86], in fact, point out that the event filtering technique perform at its best only when the distribution of subscriptions inside system’s overlay exhibits a certain degree of regionalism, that is, subscriptions matching same events are hosted on brokers placed within a limited number of overlay hops. In scenarios where no regionalism is present the improvement obtained with event filtering with respect to simple flooding is often negligible [95] and does not compensate the overhead generated to spread subscription information.

These adverse scenarios can be pre-treated with a different technique: interest clustering. Interest clustering is based on the following idea: brokers hosting similar subscriptions have a good probability to receive similar events, thus it makes sense to cluster them as much as possible in the overlay network in order to reduce the number of distinct paths that must be traveled by an event to reach them. In this situation, an event can follow a single path toward the cluster rather than being diffused in different directions towards dispersed target brokers. Therefore, the lower is the number of such paths, the lower the number of pure forwarders, and the higher the number of useless paths that can be pruned during the filtering phase. In this sense, interest clustering aims at introducing some degree of regionalism in those scenarios where it is absent, thus creating a new environment where a more favorable spanning tree for event diffusion can be created.

Let us clarify the clustering mechanism through the example depicted in figure
The first picture (fig. 1.5(a)) represents the same overlay network used in the previous example. The interest clustering technique, applied to this network, would modify it adding a new application-level link connecting the two brokers \( B_4 \) and \( B_8 \) that share similar subscriptions. The resulting overlay (fig. 1.5(b)) can be then used to diffuse the event \( e \) with a clear advantage with respect to the previous example: it can lead to a new event diffusion tree (fig. 1.5(c) where event filtering has been also applied) where the number of spam messages is reduced to one.

It is important to note that the interest clustering technique works directly on the overlay network, modifying its topology. In some publish/subscribe systems the overlay network cannot be modified, and this makes it impossible to exclude pure forwarders by means of this technique. Therefore, a necessary condition for the application of interest clustering is that the overlay can dynamically change.

![Diagram of clustering and filtering techniques](image)

Figure 1.6: Clustering and filtering techniques can be seen as part of a workflow that starting from the overlay network’s topology returns a tree that can be used to efficiently diffuse the event.

The whole process involving interest clustering and event filtering is depicted in figure 1.6. Starting from an overlay represented by a general graph \( G \), the application of interest clustering leads to a new graph \( G' \) where subscribers with similar interests (the two black dots marked with \( S \) in the figure) are clustered. From \( G' \) a spanning tree \( T \) is obtained and passed to the event filtering mechanism, that prunes useless branches and returns an new tree \( T' \) that can be used to diffuse the event (published by node \( P \) in the figure) through a simple message flooding.
1.7 Research Contribution

In the previous section we introduced event filtering and interest clustering, two techniques that can be employed to increase the efficiency of event diffusion inside an event notification service.

This thesis focuses on the study, the design and the implementation of publish/subscribe systems based on these two techniques. More specifically, in the following chapters we will study how event filtering and interest clustering can be applied in different environments to develop efficient event notification services. The contribution is divided in three chapters:

A Self-organizing Infrastructure for Interest Clustering in Filter-based Systems. Chapter 2 investigates how interest clustering can be “plugged” in existing publish/subscribe systems based on event filtering, and how it influences their performance. The publish/subscribe systems we consider in this chapter are those designed to work in managed environments, were brokers are located on dedicated machines, and are managed by some central administration. Typical example of such setting is a large corporate internal network, where information needs to be exchanged among various locations interconnected through a WAN.

To implement interest clustering in such systems we propose a self-organizing algorithm executed at each broker, which dynamically reorganizes the underlying overlay network in order to maximize a similarity metric among neighbor brokers. This metric measures the degree of interest similarity between two brokers using an history of recently matched events maintained at each broker. Aim of the proposed algorithm is to continuously adapt the overlay network clustering similar interest, taking into account both active subscriptions and the current distribution of events published in the system.

We studied the effect of the introduction of the self-organizing algorithm in the context of a specific system implementing a tree-based routing strategy, namely SI-ENA [29], showing the actual performance benefits through an extensive simulation study. In particular performance results point out the capacity of the algorithm to converge to an overlay topology accommodating efficient event dissemination with respect to a specific scenario. Moreover, the algorithm shows a significant capacity to adapt the overlay network topology to continuously changing scenarios while keeping an efficient behavior with respect to event diffusion.

Efficient Topic-based Event Routing for Peer-to-peer Systems. In chapter 3 we show how dynamic unstructured overlay networks [2,129,81] can be used as a substrate to build a robust and efficient topic-based publish/subscribe system for peer-to-peer applications. This substrate can be then used to build a large spectrum of possible applications, like news feed distribution or update notifications, that lever-
age the decoupling characteristics of the publish/subscribe interaction paradigm to support the intrinsic dynamic behavior of the peer-to-peer environment.

The system we propose is designed as two-layer architecture where a probabilistic implementation of event-filtering works together with interest clustering to reliably deliver events while completely isolating their diffusion in specific interest groups. We show through both an analytical and a simulation-based study how our system is able to efficiently manage subscriber interests and reliably notify events reducing spam messages to a very low amount. Thanks to the use of simple, but effective, probabilistic algorithms our system is designed to scale up to very large settings, a characteristic that makes it well suited for today’s peer-to-peer applications.

**Proximity-driven Event Routing in Mobile Ad-hoc Networks.** The decoupling and asynchrony properties of the content-based publish-subscribe paradigm make it very appealing for dynamic wireless networks, like those that often occur in pervasive computing scenarios. Most of the currently available content-based publish/subscribe systems do not fit the requirements of such extreme scenarios, in which the network is subject to very frequent topological reconfigurations due to the nodes’ mobility.

In chapter 4 we propose a new protocol for content-based message dissemination tailored to Mobile Ad Hoc Networks (MANETs) showing frequent topological changes. Our proposal leverages the inherent characteristics of MANETs to implement a probabilistic variant of event filtering. The protocol we present implements the event filtering mechanism without the support of any network-wide logical infrastructure thus eliminating the need of maintaining such infrastructure on top of a continuously evolving physical network. Event filtering is realized maintaining on each node transient information about subscriptions held on neighbors and correlating this information to node movements. The resulting information is used to generate hints about the current position of subscriptions and then leveraged during event diffusion to probabilistically prune the diffusion tree. The suitability of the proposed approach is confirmed through an extensive simulation study along with a stochastic analysis of the central mechanism adopted by the protocol. The mechanism proposed lacks any form of interest clustering: this choice is motivated by the impossibility of adapting the structure of the brokers network in an environment where this aspect is indirectly decided by nodes movements.

Part of the content of this thesis has been published in the following papers: [12, 10, 11, 102, 15, 13, 5, 8, 9]. Let us finally remark that [5] has been selected as best paper of that conference and an extended version appeared in [6].
Chapter 2

A Self-organizing Infrastructure for Interest Clustering in Filter-based Systems

In chapter 1 we introduced event filtering as an advanced technique for increasing event diffusion performance in all those publish/subscribe systems whose diffusion algorithm directly descends from the event flooding approach. Event filtering reduces the cost of the diffusion of an event \( e \), that can be measured as the number of overlay hops necessary to reach all the interested subscribers. Lowering this cost is a relevant objective [29, 86, 127] because it allows to reduce the overall processing load at each broker, due to event matching against subscriptions and event forwarding, thus enhancing the scalability of the whole system. SIENA [29] and REBECA [122] are two examples of publish/subscribe systems employing this technique.

In chapter 1 we also pointed out that the ability of event filtering in reducing the diffusion cost, strictly depends on the level of subscription regionalism present in the system: the higher this level is, the better event filtering will perform.

In this chapter we show how systems like [29] and [122] can be enhanced with the addition of interest clustering. Our approach is based on a dynamic reorganization of the overlay network topology. More specifically, this reorganization is done through a self-organizing algorithm, executed by brokers, whose basic principle is to cluster brokers sharing similar interests within a limited number of overlay hops.

In order to design such a self-organizing algorithm, several points must be addressed:

- define a measure of the similarity of interests between brokers;
- discuss how to execute a convenient reorganization of the overlay topology exploiting only local information at each broker;
find a balance between optimizing overlay hops and network-level metrics, like latency or bandwidth, in order that a reconfiguration of the overlay network does not spoil an initial overlay topology favorable with respect to network-level metrics.

While interest similarity is a general principle that can be applied regardless of the considered system, the self-organizing algorithm and its relation with the network-level metrics are strictly dependent on the specific event routing strategy employed by the publish/subscribe system. Therefore, here we present a self-organizing algorithm specifically tailored to the tree-based event routing strategy employed in SIENA [29].

The validity of our approach has been tested through an extensive simulation study. Experimental results show (i) the ability of the algorithm to converge to a stable overlay topology (i.e. algorithm convergence), (ii) its ability to adapt to continuously changing scenarios while keeping an efficient behavior with respect to event dissemination (i.e. algorithm stability), (iii) the performance gain obtainable in various scenarios with respect to a plain event dissemination algorithm and (iv) how the self-organization algorithm can effectively cope with problems related to network-level metrics like latency or bandwidth.

The rest of this chapter is organized as follows: section 2.1 introduces the metric we use to measure interest similarity. In section 2.2 we detail the self-organization algorithm while section 2.3 we evaluate its performance through a simulation-based study. Finally, section 2.4 discusses related work.

2.1 Measuring similarity of interests among brokers

The first problem that must be solved, in order to implement interest clustering, is how interest similarity can be measured. This is a fundamental question that must be answered to understand which brokers must be clustered and which can be left apart.

The interest of a broker is completely defined by the set of subscriptions it holds. It would thus seem rather obvious that similarity of interest between two brokers should be measured comparing their subscriptions.

However, interest clustering aims at putting close in the overlay network brokers that, hopefully, will receive events matching subscriptions held on both of them. Consider, for example, the case of two brokers holding exactly the same set of subscriptions (i.e. maximum interest similarity): if a lot of events published in the system match the subscriptions maintained on these two brokers, then it it surely worth to cluster them together; however, if none of the events matches any of their subscriptions there is no point in clustering them. This means that interest similarity, in order to be effective, cannot be measured only on subscriptions’ basis, but also event distribution must be taken into account.

The metric we propose is thus based on an history of recently matched events that must be maintained by each broker. Let $m_i$ be the list of the last $Q_i$ events matched
2.2. SELF-ORGANIZING ALGORITHM

by the broker $B_i$ at a given time. We define the similarity of interests between $B_i$ and $B_j$ as the ratio

$$a_{i,j} = \frac{|\{e \in m_i : \exists s \in S_{B_j}, e \text{ matches } s\}|}{Q_i}.$$

where $S_{B_j}$ is the set of subscriptions stored at broker $B_j$.

$a_{i,j}$ is an estimation of the probability that a new event matched by one of the two brokers will also be matched by the other one. When $a_{i,j}$ is close to one then almost all the last $Q_i$ events were matched by both $B_i$ and $B_j$. By considering an obvious “locality” principle, we argue that further common matches can happen with a high probability in the future. If $a_{i,j}$ is zero then either subscriptions hosted by $B_i$ and $B_j$ are disjoined or none of the last matched events by $B_i$ have also been matched by $B_j$ and vice versa. In other words, let us consider the case that $B_i$ and $B_j$ are connected through a direct overlay link: if $a_{i,j}$ tends to one, then the probability that $B_i$ acts as a pure forwarder for an event $e$ matched by $B_j$ tends to zero (and vice versa).

Note that since $a_{i,j}$ is related to the actual number of common matched past events, this number dynamically adapts to the changes in the distributions of events and subscriptions, i.e. no a-priori knowledge is required about them.

The asymmetry of the proposed metric (i.e. in general $a_{i,j} \neq a_{j,i}$) accounts for the greedy nature of the proposed heuristic: each node tries to reorganize the overlay network for its own purposes. For this reason the algorithm implementing the heuristic and the overlay reorganization strategy must include techniques to let the overlay eventually converge to a single topology, avoiding oscillations among equivalent solutions.

Given the previous definition each link $l_{i,j}$ of the overlay network can then be labeled with a weight $w(l_{i,j})$ representing the similarity of interests between the brokers connected through it, i.e. $w(l_{i,j}) = a_{i,j}$.

2.2 Self-organizing Algorithm

In this section we present a self-organization algorithm specifically tailored to SIENA, a popular content-based pub/sub system introduced in [29]. However, our algorithm is more general and can be applied to all pub/sub systems relying on a tree-based event routing strategy, such as Rebeca [122], Jedi [44] and REDS [46].

SIENA [115] is a content-based pub/sub system which realizes event dissemination over a tree-shaped overlay network of brokers employing event filtering. Subscriptions and events are defined over a fixed event schema, constituted by a set of $n$ attributes each characterized by a name and a type, where the type can be a common

---

1The principle holds as long as subscriptions can be considered long-lived and the distribution of events in the event space changes slowly w.r.t. the event injection ratio. Both these assumptions are in fact true for a wide range of different publish/subscribe applications.
basic type such as integer, real or string. An event is therefore a set of \( n \) values, one for each attribute whose type is consistent with that attribute’s type. If all values are defined for every attribute, an event can be considered as a point in the \( n \)-dimensional event space. In the sequel we abstract the notion of subscription as a subset of events inside the event schema, defined by using a set of constraints over the attributes.

In SIENA, subscriptions issued by each subscriber on a broker are inserted in the broker’s subscription table. We define the zone of interest of a broker \( B_i \), denoted \( Z_{B_i} \), as the union of all subscriptions contained in \( B_i \)’s subscription table.

Each Broker \( B_i \) is connected to a maximum of \( F_i \) neighbor brokers through overlay links (in the rest of this chapter, for the sake of simplicity, we will consider this value equal for all brokers, i.e. \( F_i = F \)). The routing table of \( B_i \) contains an entry for each of these links and each entry provides information about the subscriptions hosted on brokers reachable through the corresponding link. More specifically a covered zone \( Z_{l_{i,j}} \) is associated to each link \( l_{i,j} \) and it is defined as the union of all the zone of interests belonging to brokers that lies beyond \( l_{i,j} \). Routing tables are kept up-to-date by SIENA using a smart update mechanism whose details are here omitted (see [29] for such details).

Event dissemination with filtering is realized in SIENA through a simple mechanism: each broker \( B_i \) receiving an event \( e \) through a link \( l_{i,j} \), (i) matches \( e \) against subscriptions contained in its subscription table and then (ii) forwards \( e \) through all the links \( l_{i,k} \) (with \( k \neq j \)) such that \( e \) matches \( Z_{l_{i,k}} \). This behavior ensures that \( e \) is forwarded only through those links which can lead to interested subscribers. Note that a broker forwarding \( e \) to some neighbor in (ii) without any match in (i) is actually acting as a pure forwarder for \( e \).

2.2.1 Algorithm Details

Let us consider two brokers \( B_i \) and \( B_ℓ \) that are not directly connected by an overlay link. The goal of the self-organization algorithm is to connect directly \( B_i \) to \( B_ℓ \) only if there is an overlay link \( l_{p,q} \) in the path from \( B_i \) to \( B_ℓ \) such that \( a_{i,ℓ} \) is larger than the similarity of interests of the brokers connected by \( l_{p,q} \), i.e. \( a_{p,q} \). Moreover to keep the topology acyclic as required by SIENA’s CBR algorithm, \( l_{p,q} \) must be teared down. The overall self-organization algorithm can be split in four phases:

1. triggering of a broker discovery,
2. broker discovery,
3. tear-down link selection,
4. overlay topology update.

For the sake of clarity, along the description of the various algorithm phases we refer to a running example over the network of brokers depicted in figure 2.1(a). This
Figure 2.1: Broker discovery phase. Picture (a) shows the propagation of DREQ messages started at $B_9$. Picture (b) shows corresponding DREP messages while they converge toward $B_9$. Labels on links represent their weight.

<table>
<thead>
<tr>
<th>Broker</th>
<th>$Z_{B_i}$</th>
<th>$m_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>$S_1, S_{10}$</td>
<td>$e_1, e_{10}$</td>
</tr>
<tr>
<td>$B_2$</td>
<td>$S_6$</td>
<td>$e_6$</td>
</tr>
<tr>
<td>$B_3$</td>
<td>$S_7, S_9$</td>
<td>$e_7, e_8$</td>
</tr>
<tr>
<td>$B_4$</td>
<td>$S_4, S_5$</td>
<td>$e_2, e_9$</td>
</tr>
<tr>
<td>$B_5$</td>
<td>$S_7, S_{10}, S_{11}$</td>
<td>$e_1, e_7, e_{10}, e_{11}$</td>
</tr>
<tr>
<td>$B_6$</td>
<td>$S_6$</td>
<td>$e_6$</td>
</tr>
<tr>
<td>$B_7$</td>
<td>$S_7$</td>
<td>$e_7$</td>
</tr>
<tr>
<td>$B_8$</td>
<td>$S_5, S_6$</td>
<td>$e_6, e_8$</td>
</tr>
<tr>
<td>$B_9$</td>
<td>$S_1, S_3, S_{10}, S_{11}$</td>
<td>$e_1, e_3, e_{10}, e_{11}$</td>
</tr>
</tbody>
</table>

Table 2.1: Zone of interest and list of matched events for brokers in the example of figure 2.1(a)

The figure represents a snapshot of a system at a given time whose state, i.e., zone of interest and the last four events matched ($Q = 4$) by each broker, is reported in table 2.1. In figure 2.1(a) each link is labelled with its weight, i.e., the associativity between the brokers it connects; for example, the weight of link $l_{3,9}$ is given by $w(l_{3,9}) = a_{3,9} = |m_3 \cap m_9|/Q = |\{e_3, e_6\} \cap \{e_1, e_3, e_{10}, e_{11}\}|/Q = 1/4 = 0.25$. In this example we suppose that an event $e_i$ matches only subscription $S_i$.

---

2For simplicity, we consider in the following a same value $Q$ of $Q_i$ at each broker.

3For the sake of clarity we depict in figures a particular scenario in which each subscription $S_i$ is matched only by event $e_i$. 
**Triggering of a broker discovery**

The triggering phase, executed by a broker $B_i$, launches the execution of each run of the self-organization algorithm that could lead to an update of the overlay topology.

The triggering occurs every $\delta$ events received by the broker; the value of $\delta$ is updated at the end of each run according to its outcome and following a backoff policy: if the overlay topology has changed, then the self-organization succeeded and $\delta$ is set to a base value, otherwise $\delta = 2\delta$. The choice of employing a backoff mechanism to limit the frequency of runs is justified by the fact that, if a self-organization run does not lead to a change of the overlay topology, it will take time before the conditions in the overlay network become favorable for a successful execution of a new self-organization run started by the same broker.

Once a run of the self-organization algorithm has been launched, the following *Activation Predicate* (AP) is checked for each overlay link of $B_i$:

$$\text{AP} : \alpha_{i,j}(m_i) > a_{i,j}.$$  

where $\alpha_{i,j}(m_i)$ is the number of events in $m_i$ that match $Z_{l_{i,j}}$ divided by $Q$. The set $Z_{B_j}$, needed to calculate $a_{i,j}$, is a subset of $Z_{l_{i,j}}$ and can be thus derived from it\footnote{This can be realized marking subscriptions maintained at neighbor brokers when they are routed in the system to keep routing tables updated. These marks can be later used to reconstruct the set of subscriptions maintained at each neighbor. The implementation details of this mechanism are strictly related to the way SIENA updates routing tables on brokers.} without the need any message exchange between $B_i$ and $B_j$.

If the predicate is false, then the run terminates immediately. Otherwise, for each link $l_{i,j}$ such that the predicate holds, a broker discovery procedure is started. The rationale behind this action is that if AP holds for a link $l_{i,j}$, then there could be a broker behind $B_j$ with a similarity of interests with $B_i$, equal to $\alpha_{i,j}(m_i)$, that is larger than $a_{i,j}$.

At the end of the triggering phase, $k$ independent broker discovery procedures are started, where $k$ is the number of links for which AP is satisfied. For each activated procedure the broker $B_\ell$ with the highest similarity with $B_i$, located behind the link, will be returned, as we detail in the following.

Considering the example in figure 2.1(a), let us assume that the self-organization algorithm is triggered on broker $B_9$ by the arrival of an event. $B_9$ checks AP on its only link $l_{9,3}$ and finds out that $\alpha_{9,3}(m_9) = 1$ is greater than $a_{9,3} = 0.25$; given the fact that AP is true on $l_{9,3}$ a broker discovery phase is started on this link.

**Broker Discovery**

The broker $B_i$ starting the broker discovery on one of its links sends through it a request message\footnote{We consider messages exchanged by the algorithm sent through reliable (or perfect) point-to-point channels. An implementation example of such channels over fair-lossy links (e.g. TCP or UDP) can be} $\text{DREQ}$ containing $m_i$ and a hop sequence $\text{HS}$. A hop sequence $\text{HS}$
represents the overlay network path joining two brokers $B_x$ and $B_y$ and is defined as a list of pairs (broker_id, weight), $((B_x, 0), (B_{x+1}, w(l_{x,x+1})), \ldots, (B_y, w(l_{y-1,y})))$, such that any two adjacent brokers in the list are connected via an overlay link. We denote such a path as $HS(B_x, B_y)$. Note that the weight in the first pair is always 0. The hop sequence $HS$ contained in the DREQ message sent by $B_i$ is initialized to $((B_i, 0))$.

The forwarding of the DREQ message is driven by the list of events $m_i$, as explained in the following. When a broker $B_j$ receives DREQ on one of its links $l_{k,j}$, it (i) computes its own associativity w.r.t. $B_i$, $a_{i,j}$, (ii) updates $HS$ by adding $(B_j, w(l_{k,j}))$, i.e. $HS(B_i, B_j) = HS \cup (B_j, w(l_{k,j}))$ and, finally, (iii) computes the following Forwarding Predicate (FP):

$$FP : \exists l_{j,h} \neq l_{j,k} : \alpha_{j,h}(m_i) > a_{i,j}.$$ 

$FP$ is based on the same idea as AP: when FP is true for a link $l_{j,k}$ there are some possibilities to locate another node behind that link with a similarity w.r.t. $B_i$ larger than $a_{i,j}$. In this case a copy of the DREQ message is sent on that link$^6$. Note that the evaluation of FP is completely based on information that is either locally maintained on $B_j$ (i.e. $Z_{j,x}$ and $Z_{j,y}$) or contained in the DREQ message (i.e. $m_i$).

If no link exists such that FP is satisfied, then a Discovery Reply message, DREP, is sent back to $B_i$ along the path stored in $HS$. The DREP message contains $a_{i,j}$, $HS(B_i, B_j)$ and $al_{i,j}$, where $al_{i}$ is the number of overlay connections that can still be created on $B_j$ before the limit $F$ is reached.

Referring to our example (figure 2.1(a)), the DREQ message generated on $B_9$ is forwarded through various brokers toward $B_1$ and $B_5$ which are the nodes with the largest similarity of interests with $B_9$. Note that in $B_5$ the predicate FP is true for links $l_{3,1}$ and $l_{3,6}$, and a copy of DREQ is sent through them both. On the other hand, in $B_6$ FP is true only for link $l_{6,8}$ and not for $l_{6,2}$ as $\alpha_{6,2}(m_0) = 0$.

For each $l_{j,k}$ satisfying FP, $B_j$ forwards the DREQ and then waits for the corresponding reply message DREP($a_{i,e}, al_{i}, HS(B_i, B_n)$), where $B_n$ is the broker reachable through $l_{j,k}$ that can offer the highest similarity of interests w.r.t. $B_i$.

As soon as $B_j$ receives the reply from each link through which previously DREQ was forwarded, it computes the maximum among all the similarity values carried in the received replies and its own $a_{i,j}$. Let $a_{i,e}$ be the maximum and $B_{e}$ be the corresponding broker; then $B_j$ sends a DREP($a_{i,e}, al_{i}, HS(B_i, B_{e})$) through $l_{j,k}$ toward $B_i$.

In figure 2.1(b) broker $B_1$, after gathering DREP messages coming from $B_1$ and $B_5$ through $l_{3,1}$ and $l_{3,6}$ respectively, determines that the broker with the largest similarity with $B_9$ among itself, $B_1$ and $B_5$ is the latter; then it forwards toward $B_9$ (through

---

$^6$This forwarding mechanism can lead, in some rare case, to a simple flooding of the DREQ message on the whole network. To limit the depth of the search conducted through DREQ messages it is possible to add in FP a simple constraint based on a TTL counter.
The aim of this phase is to select the link that must be teared down during the overlay topology update phase. The procedure is activated for each DREP message received by the broker $B_i$ that started the self-organization run and returns a tear-down candidate link denoted as $l_{td}$. If no link exists that can be deleted, $l_{td} = \text{NULL}$.

A single selection works as follows: let DREP be the reply to the DREQ message sent along link $l_{i,j}$. The reply contains the identifier $B_\ell$ of the broker behind $l_{i,j}$ with the highest similarity with $B_i$, together with the path stored in $HS(B_i, B_\ell)$. Let us denote the link that should be created between $B_i$ and $B_\ell$ as $l_{new}$. Clearly, if $a_l = 0 \land a_i = 0$ the link $l_{new}$ cannot be created, as both $B_i$ and $B_\ell$ have no overlay connections still available, and thus $l_{td} = \text{NULL}$. Otherwise, we have two cases:

1. $a_l > 0 \land a_\ell > 0$: in this case both $B_i$ and $B_\ell$ have available overlay connections and thus they can establish the link $l_{new}$ between them. In this case, $l_{td}$ is the link with the minimum weight in $HS(B_i, B_\ell)$.

2. $a_\ell = 0, a_l > 0$ (resp. $a_l = 0, a_\ell > 0$): in this case $l_{td}$ is the link that connects $B_{\ell-1}$ to $B_\ell$ (resp. $B_i$ to $B_{i+1}$), i.e. $l_{td} = l_{\ell-1,\ell}$ (resp. $l_{td} = l_{i,i+1}$); in this way the constraint $F$ on the maximum number of overlay links remains satisfied for $B_\ell$ (resp. $B_i$).

The next phase of the algorithm (overlay topology update) takes place only if $l_{td} \neq \text{NULL}$ and $w(l_{new}) > w(l_{td})$. In other words a reconfiguration occurs only if $l_{new}$ is expected to be traversed by a number of events that are matched on both brokers directly connected by $l_{td}$.

Considering figure 2.1(b), after DREP message reaches $B_9$ through $l_{9,3}$, the algorithm selects $B_5$ as the candidate to establish a link (i.e., $l_{new} = l_{9,5}$) with $B_9$, and $l_{4,7}$ as $l_{td}$. The reorganization of the overlay network will occur as $w(l_{9,5}) = a_{9,5}$ is greater than $w(l_{4,7})$ (we suppose both $a_l$ and $a_\ell$ larger than 0).

The Overlay topology update

Let $B_i$ and $B_\ell$ be the two brokers that must be connected by $l_{new}$, and $B_p$ and $B_q$ be the brokers connected by $l_{td}$ in the path stored in $HS(B_i, B_\ell)$. To avoid network partitioning or the creation of cycles we must ensure that during the $l_{td}$ tear down, other links in $HS(B_i, B_\ell)$ are not teared down by concurrent runs of the self-organization algorithm. Therefore there is the need of a locking mechanism to ensure that only one tear down operation at a time can take place along the path from $B_i$ to $B_\ell$. Moreover the locking mechanism can be used to block and buffer events that should be diffused through the path in order to restart their diffusion at the end of the reorganization.
2.2. SELF-ORGANIZING ALGORITHM

A possible implementation of the locking mechanism is the following: $B_i$ sends a LOCK message along the path towards $B_\ell$. A generic broker $B$ in that path executes the following locking algorithm:

- **when receiving a LOCK message through link $l$:**
  1. if $l$ is not locked and $B \neq B_\ell$, locks $l$ and forwards the LOCK message to the next node towards $B_\ell$;
  2. if $l$ is not locked and $B = B_\ell$, locks $l$ and sends an ACK message to the next node towards $B_i$;
  3. if $l$ is locked or $B_\ell$ is no more reachable through the path $HS(B_i, B_\ell)$, sends a NACK towards $B_i$.

- **when receiving an ACK message:** $B$ forwards the ACK message to next node towards $B_i$.

- **when receiving a NACK message:**
  1. if $B \neq B_i$ removes the lock from the link and forwards the NACK message to next node towards $B_i$;
  2. if $B = B_i$ aborts the self-organization.

Locks on links must be associated to the corresponding reorganization in order to correctly handle concurrent instances of the lock algorithm. Locks are released after a timeout expires, thus avoiding that paths are indefinitely locked if a broker fails. This will cause $B_i$ to retrigger the algorithm, but the backoff mechanism described in

---

**Figure 2.2:** Overlay topology update phase. Picture (a) represents the locking algorithm in action while it locks links in the path connecting $B_0$ to $B_5$. Picture (b) shows the moment when link $l_{tu}$ ($l_{4,7}$ in this example) is teared down and the new link $l_{new}$ ($l_{9,5}$) is created.
section 2.2.1 ensures that this occurs when conditions in the network have changed, thus limiting the possibility of conflicting instances to fall into race conditions.

As an example figure 2.2(a) depicts an execution of the locking algorithm acting on the path from \( B_9 \) to \( B_5 \) assuming no concurrent instances. Numbers reported in the picture indicate the message order.

Once the path is locked, \( B_i \) establishes the new overlay link with \( B_\ell \), then both \( B_i \) and \( B_\ell \) send a CLOSE message to \( B_p \) and \( B_q \) respectively, along the locked path (no synchronization is needed between \( B_i \) and \( B_\ell \)). When these messages arrive at their destination, the link \( l_{p,q} \) is teared down and an UNLOCK message starts from \( B_p \) and \( B_q \) and follows the reverse path to \( B_i \) and \( B_\ell \) as shown in figure 2.2(b) with respect to the path from \( B_9 \) to \( B_5 \) (also in this case no synchronization is needed between \( B_p \) and \( B_q \)).

The UNLOCK message is also used to trigger an update of routing tables\(^7\) and to remove locks on the path. A lock can be removed only by the broker that established it (locks are associated with the identity of the locking broker). Once the UNLOCK message arrives at \( B_i \) and \( B_\ell \), the link \( l_{new} \) becomes operative. Let us also remark that if the path \( HS(B_i,B_\ell) \) changes (due to a concurrent reconfiguration) before the propagation of the LOCK message, but after the previous phases ended, eventually a NACK message is returned to \( B_i \) which aborts the self-organization run.

### 2.2.2 Addressing Network Awareness

The self-organization algorithm presented in the previous section follows a network-oblivious approach, trying to achieve only a reduction in the number of overlay hops.

Nevertheless, it is important to note that usually overlay networks are built trying to respect some constraint based on physical proximity between nodes, such that the resulting overlay topology closely matches the underlying network topology. With this technique it is possible to obtain good performance from the overlay network with respect to network-level metrics. For example, connections between brokers in a same LAN must be privileged, both for simplicity of management and for performance reasons. In this case, the intervention of the self-organization can disrupt this proximity, thus affecting network-level performance.

To avoid this problem, or at least limit its effects, our self-organization algorithm follows this simple principle: new links can be created only among those brokers whose network-level distance is less than a threshold value \( d \). Network distance can be measured indifferently with any metric, either IP hops or latency or bandwidth, depending on the specific application requirements. Anyway, the choice of the distance metric does not affect the algorithm specification.

When network awareness is considered each broker involved in a broker discovery phase checks if its network distance from the broker source of the self-organization

\(^7\)Various mechanisms can be used to fix routing tables (see [98] for an example), but it is important to note that only nodes on the path now connecting \( B_p \) to \( B_q \) are interested by this update.
is higher than $d$. If this is the case the broker will simply avoid to propose itself as a candidate, letting the discovery continue in its search for other candidates.

Performance obtained considering network awareness obviously depends from the choice of $d$ as this threshold value is actually used to prune the set of candidate brokers for a self-organization run; pruned candidate brokers can sometimes be best candidates, i.e. those brokers offering the largest similarity, thus leading to a less efficient overlay network. In particular if $d = \infty$ the network-level metric is not taken into account; if $d = 0$ no reorganization can happen.

2.3 Evaluation

To evaluate performance and applicability of our self-organization algorithm, we conducted an extensive simulation study. This study is based on results obtained from a prototype implementation of a SIENA-like pub/sub broker augmented with the self-organization algorithm. The prototype is deployed over J-Sim [60], a component-based real-time simulator that allows to simulate the whole TCP/IP protocol stack. The aim of this study is to compare the behavior of the algorithm on different scenarios in order to evaluate at which extent, on these scenarios, the self-organization can improve performance.

This section is organized as follows: at first we introduce the simulation model used in our experiments and the scenarios on which the simulations were ran; then we study specific aspects of the algorithm like convergence and stability; the third subsection is devoted to performance results that show how our algorithm can effectively reduce the cost of event dissemination from an application-level point of view; at last we take network latency into account to show how our algorithm can cope with network-level metrics while trying to optimize an application-level metric.

2.3.1 Experimental model

TCP/IP and overlay network models

The experiments were run over a 100 nodes TCP/IP network. The IP network was first created using the GT-ITM tool [138, 93] configured to generate four subnetworks linked by backbone links. Then TCP/IP-based hosts were simulated on this network through the INET framework provided by J-Sim.

Brokers were installed on all hosts and connected to form an initial pub/sub network that closely matches the underlying IP network. The maximum number of allowed overlay connections per broker was set to $F = 10$. 
Given the absence of publicly available data traces of real pub/sub applications, we tested our algorithm on various synthetic scenarios. Following the same approach previously used in other simulation studies ([106, 26, 30]) we generated nine scenarios characterized as follows:

**Event space.** The proposed self-organization algorithm is independent from the actual event space used, as it only relies on the matching operations of the CBR algorithm. Then, for the purpose of simulations, we assumed a simple two-dimensional event space, i.e., subscriptions have two numerical range filters (defined on the domain of real numbers in the range \([-10, +10]\)). This simple space was chosen to limit the cost of matching operations in our simulations, but, given the independence of the algorithm from the actual structure of the space, we believe that the results presented in this chapter are of general validity.

**Distributions of subscriptions and events in the event space.** We generated various scenarios using two different events distributions: uniform (U scenarios) and Gaussian (G scenarios).

The Gaussian distribution simulates “hot spots” in the event space: each G-scenario presents four randomly chosen “spots” around which events are generated using a normal gaussian distribution with mean 0 and standard deviation 1. The gaussian distribution tries to mimic the load of those applications where publications are concentrated on some more relevant interest.

On the contrary, using a uniform distribution, an event can be generated in any point of the event space with the same probability, mimicking applications where published data is not concentrated on any specific interest but is rather uniformly spread among every possible interest.

### Table 2.2: Scenarios used for the experiments.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Distribution of events</th>
<th>Distribution of subscriptions</th>
<th>Popularity of events</th>
<th>Percentage of target brokers</th>
</tr>
</thead>
<tbody>
<tr>
<td>U0</td>
<td>uniform</td>
<td>uniform</td>
<td>1.7%</td>
<td>4.8%</td>
</tr>
<tr>
<td>U1</td>
<td>uniform</td>
<td>uniform</td>
<td>5.4%</td>
<td>14.6%</td>
</tr>
<tr>
<td>U2</td>
<td>uniform</td>
<td>uniform</td>
<td>10.1%</td>
<td>24.3%</td>
</tr>
<tr>
<td>U3</td>
<td>uniform</td>
<td>uniform</td>
<td>24.5%</td>
<td>48.9%</td>
</tr>
<tr>
<td>U4</td>
<td>uniform</td>
<td>uniform</td>
<td>50.9%</td>
<td>75.3%</td>
</tr>
<tr>
<td>G0</td>
<td>gaussian</td>
<td>uniform</td>
<td>1.4%</td>
<td>4.1%</td>
</tr>
<tr>
<td>G1</td>
<td>gaussian</td>
<td>uniform</td>
<td>5.2%</td>
<td>14.2%</td>
</tr>
<tr>
<td>G2</td>
<td>gaussian</td>
<td>uniform</td>
<td>8.8%</td>
<td>22.9%</td>
</tr>
<tr>
<td>G3</td>
<td>gaussian</td>
<td>uniform</td>
<td>24.6%</td>
<td>51.3%</td>
</tr>
</tbody>
</table>
2.3. EVALUATION

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>Maximum length of $m_i$.</td>
<td>50</td>
</tr>
<tr>
<td>$\delta_{\text{base}}$</td>
<td>Base value for the triggering backoff algorithm.</td>
<td>50</td>
</tr>
<tr>
<td>$d$</td>
<td>Maximum latency allowed for link created by the self-organization algorithm.</td>
<td>unbounded</td>
</tr>
</tbody>
</table>

Table 2.3: Parameters of the Self-organization algorithm.

As far as subscriptions are concerned a uniform distribution was always used to generate values for the constraints.

The different U and G scenarios are characterized by popularity of events, which is defined as the percentage of subscriptions matched on average by each event. Event popularity is an important parameter as it affects the average number of target brokers for each event thus affecting also the number of messages generated for their diffusion.

Table 2.2 reports characteristics of each scenario along with the corresponding values of average popularity of events and average number of target brokers. In this study we do not consider scenarios with extremely high popularity because, in such scenarios, simple event flooding has been proved to work quite well w.r.t. CBR [95].

**Distribution of subscriptions and events in the broker network.** Subscriptions and events are spread uniformly among the brokers. In our experiments we do not take into account the presence of regionalism (i.e., the presence of similar subscriptions and events clustered into the brokers’ network): the purpose of our algorithm is cluster subscriptions, so it would be pointless to apply it in a scenarios where clustering is already present because of regionalism.

2.3.2 Performance metrics

For each scenario the evolution of the system was observed under the same settings with and without the reconfiguration algorithm. The following statistics were collected:

- **Number of reorganizations**: the number of reorganization caused by the self-organization algorithm.
- **Notification cost**: the ratio between the number of total application messages generated by brokers and the number of subscriptions matched by diffused events (notifications). The number of application messages includes the control messages generated by self-organization algorithm, such as discovery, locks and route updates messages.

---

8Percentage of target brokers have been computed by injecting 300 subscriptions in the system.


- **Percentage of brokers involved**: the average number of brokers involved during the diffusion of an event. This metric is used to compare the number of pure forwarders in the different settings.

- **Event dissemination time**: the time spent for an event to travel through the overlay network from the injection point to all the target brokers. This metric is used for the evaluation of network-level performance as it reflects the network latency of the links traversed by events.

The average notification cost expresses the efficiency of event dissemination: the lower the cost, the higher the efficiency. The average percentage of brokers involved per event dissemination gives us the opportunity of comparing the algorithm against the ideal case, i.e., when this number corresponds to the percentage of target brokers.

For each scenario the above metrics have been estimated through several independent simulation runs with a confidence level of 95%. In each simulation run first 300 subscriptions are injected in the pub/sub system, then statistics are collected during the subsequent diffusion of 5000 events. The size of the event lists was set to \( Q = 50 \), and the base value of the backoff mechanism was set to \( \delta = 50 \). Unless differently specified in the text, the maximum latency allowed for links created by the self-organization algorithm was always set to \( d = \infty \). These parameters are reported in table 2.3 for the convenience of the reader.

Note that our simulation model does not explicitly model publishers and subscribers attached to brokers as all the network traffic generated by them can be considered as a fixed cost that occurs with or without the use of the self-organization algorithm; for this reason this cost is actually ignored in our results.

In the following, CBR indicates the results from the plain SIENA’s event dissemination algorithm and CBR+SO the ones obtained when self-organizations are allowed.

### 2.3.3 Algorithm convergence

The first aspect we analyze is the algorithm capacity to converge in a limited number of runs. With **convergence** we mean the ability to reach a stable overlay network topology, i.e., a topology where no more reorganizations happen, given that initial conditions of the scenario do not change.

To run these tests we first allocated 300 subscriptions on the brokers and then started injecting events. The number of successful self-organizations are collected every 50 events. The results obtained under scenarios U and G are reported in figure 2.3(a) and 2.3(b) respectively. The curves show how the number of reorganizations

---

*These values were chosen by basing on a suite of preliminary tests. These tests showed how, starting from a base value (i.e., 50 in our scenarios), the impact of \( Q \) on the performance results is negligible. This result probably stems from the fact that the minimum useful value for \( Q \) is a function of both the event injection ratio and the popularity of events. The value of \( \delta \) is instead proportional to \( Q \).*
2.3. EVALUATION

Figure 2.3: Reconfiguration distributions in U (graph (a)) and G (graph (b)) scenarios. The curves show how the number of reorganizations of the overlay network converges to a maximum within a limited number of events injected in the system.

The curves show how the number of reorganizations of the overlay network converges to a maximum within a limited number of events injected in the system. During the injection of the first 1000 events there is a huge growth of the number of reorganizations that converges to a maximum after the 2000th event. The initial growth is needed to transform the initial overlay topology into an efficient one and can be considered as a transient condition; then the system enters a steady-state condition, in which the number of successful self-organizations is negligible w.r.t. the number of injected events.

2.3.4 Algorithm stability

Results about the convergence of the algorithm previously shown were obtained under “static” systems in which the subscriptions inserted at start time remain unchanged for the whole observation period. Obviously, such a scenario does not give any hint on how the algorithm behaves in a dynamic setting where subscriptions (and then interest of brokers) can change during the lifetime of the system. This is an important aspect that must be analyzed to check how the self-organization algorithm reacts to changes in the state of the system and if it is able to eventually converge to a new efficient overlay topology.

For this purpose we ran specific experiments, in which the set of subscriptions is abruptly changed at once every 4000 events; although this situation is hardly going to happen in real applications, where subscriptions are expected to change continuously with a certain frequency during the whole lifetime of the system, it can be seen as a “worst case” scenario that puts the algorithm under the most stressing condition.

Figure 2.4(a) reports the notification cost during the evolution of the system as
Figure 2.4: Evolution of notification cost in scenario U1. Graph (a) shows how the algorithm is able to react to an abrupt change of the subscriptions present in the system (occurring after 4000 and 8000 events), and converge, in a limited number of runs, to a new stable overlay topology. Graph (b) shows the algorithm reaction to a continuous subscription change (occurring from event 4000 to 12000). The set of subscription updates is the same in both graphs.

more and more events are injected (the cost was measured every 50 events; the values shown are not cumulative); curves are shown only for scenario U1, as tests conducted for all the other scenarios showed the same behavior. The vertical dotted lines at 4000 and 8000 indicate the instants when subscriptions were substituted. The grey curve depicts the cost for CBR that remains mostly constant during all the test as it does not suffer from the overhead generated by the self-organization algorithm. The black curve depicts the notification cost for the content based routing algorithm augmented with self-organization (CBR+SO). As the curve clearly shows, a subscription update suddenly renders the overlay topology inefficient to diffuse events to new target brokers.

The self-organization algorithm quickly detects this new situation and starts re-adapting the overlay network by converging to a new stable topology. Moreover, the new overlay topology obtained after this transient condition provides almost the same increase in performance that was observed before changing subscriptions (this is true as long as the distribution of subscriptions does not change). The transient conditions showed in the plot are large both in amplitude (notification cost) and duration due to the fact that subscriptions were updated as a whole at the same time; a more graceful adaptation is expected when each subscription changes independently from each other.

Figure 2.4(b) shows the results for a similar scenario where subscriptions are not changed all at the same time, but their updates are uniformly spread in time: starting
2.3. EVALUATION

Figure 2.5: Performance analysis in U-scenarios. Graph (a) shows average notification cost for event dissemination (values are normalized with CBR set to 100%) where the cost of CBR+SO includes the reconfiguration overhead. Graph (b) shows average percentage of brokers involved in each event dissemination.

from event 4000, up to the end of the test, the same updates used in the previous experiment are periodically issued, one at a time. In this case the abrupt performance evolution observed in figure 2.4(a) is absent: when subscriptions are spread over time, the algorithm gracefully adapts the overlay topology, continuously pursuing for an efficient one.

2.3.5 Performance results

In this section we investigate the amount of performance improvement obtained through the adoption of our algorithm in the different scenarios described above. Let us note that while G-scenarios should favor the mechanism of self-organization given the concentration of events on specific zones of the event space they offer, U-scenarios represent more difficult settings for CBR+SO as the similarity of interests is uniformly spread among all the brokers. The results shown in this section are obtained from measures taken on stable overlay networks.

Figure 2.5(a) reports the notification cost in U-scenarios for both CBR and CBR+SO. The cost is normalized, so that the CBR’s cost is always represented as 100%. As the plot shows SO+CBR can obtain a gain on plain CBR that ranges from 20% to 26% on scenarios U0 through U3. Scenario U4 shows a more modest gain; this is due to the large number of target brokers induced by the large popularity of events which characterizes U4 (see table 2.2): each event in this scenario must be diffused, on average, to 3/4 of all the brokers, and this means that the gain of CBR itself w.r.t. a simple event flooding mechanism is negligible. High popularity scenarios are not
CHAPTER 2. INTEREST CLUSTERING FOR FILTER-BASED SYSTEMS

(a)

<table>
<thead>
<tr>
<th></th>
<th>CBR</th>
<th>CBR+SO</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>U0</td>
<td>0.097</td>
<td>0.182</td>
<td>88%</td>
</tr>
<tr>
<td>U1</td>
<td>0.235</td>
<td>0.342</td>
<td>46%</td>
</tr>
<tr>
<td>U2</td>
<td>0.325</td>
<td>0.448</td>
<td>38%</td>
</tr>
<tr>
<td>U3</td>
<td>0.521</td>
<td>0.659</td>
<td>26%</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th></th>
<th>CBR</th>
<th>CBR+SO</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0</td>
<td>0.02</td>
<td>0.115</td>
<td>478%</td>
</tr>
<tr>
<td>G1</td>
<td>0.09</td>
<td>0.237</td>
<td>161%</td>
</tr>
<tr>
<td>G2</td>
<td>0.182</td>
<td>0.356</td>
<td>96%</td>
</tr>
<tr>
<td>G3</td>
<td>0.461</td>
<td>0.642</td>
<td>39%</td>
</tr>
</tbody>
</table>

Table 2.4: Average link weights in U and G scenarios.

Figure 2.6: Performance analysis in G-scenarios. Graph (a) shows average notification cost for event dissemination (values are normalized with CBR set to 100%) where the cost of CBR+SO includes the reconfiguration overhead. Graph (b) shows average percentage of brokers involved in each event dissemination.

favorable workloads for our self-organization algorithm and for simple CBR as well [95]; for this reason we decided to concentrate our efforts only on scenarios with lower popularity.

Figure 2.5(b) shows the percentage of brokers involved on average in each event dissemination under various U-scenarios for CBR and CBR+SO. The ideal value, i.e., the average percentage of target brokers for each event, is also reported as a reference. It represents the minimum value that could be reached by an hypothetical “optimal” diffusion mechanism that delivers all the events in one hop per target broker.

The figure shows how the self-organization algorithm modifies the application network topology trying to reach this ideal value. In fact, the gap between the ideal value and the CBR value can be interpreted as due to pure forwarder brokers that are needed by the routing mechanism to forward each event toward all the target brokers: the self-organization algorithm reduces the average notification cost lowering the average number of pure forwarders.
2.3. EVALUATION

This result is obtained by CBR+SO reorganizing the network in order to increase the similarity between brokers connected by links. The relationship between the average number of pure forwarders for each event dissemination and the average similarity between brokers is demonstrated by the values reported in table 2.4(a). The table shows that the self-organization algorithm accommodates the principles introduced in section 2.1 and that an increase in the average link weight (i.e., similarity between brokers) obtained by CBR+SO corresponds to the reduction of pure forwarders shown in figure 2.5(b).

Figure 2.6 and table 2.4(b) report the results obtained under G-scenarios. As it appears from the plot, CBR+SO on these scenarios can obtain a gain w.r.t. plain CBR up to 33%.

2.3.6 Taking network latency into account

All the results previously shown were obtained without taking the effect of self-organization on network-level metrics into account. Here we choose link latency as the network proximity metric, and study the behaviour of our algorithm when the value of threshold \( d \) varies from 0 (no reorganizations are allowed) up to \( \infty \) (link latency is not taken into account). The initial broker topologies considered in these tests were optimized with respect to the proximity metric.

Figure 2.7(a) shows how the average event dissemination time\(^{10}\) varies with \( d \). The maximum and minimum values for event dissemination time obtained from the tests are also reported in the figure. Figure 2.7(b) shows the corresponding average

\(^{10}\)We do not take the time required to process matching operations on brokers into account.
notification cost. As the plots show, by varying $d$ it is possible to fine-tune the behaviour of the algorithm in order to reduce the notification cost while limiting the increase in network latency (15\% in our experiments by choosing $d = 0.2$).

Figure 2.7(a) has also another important interpretation: even though the average event dissemination time increases with $d$, vertical bars show that there can be cases where the diffusion time decreases with $d$. This means that the use of our algorithm can also transform the initial overlay topology in a new one that also has better network-level performance than the initial one. For this reason $d$ should be determined on a per-case basis and possibly by means of an automatic mechanism that can dynamically adjust it.

2.4 Related Work

Managed vs. self-managed overlay networks. Pub/sub systems based on networks of brokers ([29, 44, 86, 26, 124, 94]) rely on a managed overlay, that is brokers and links connecting them are setup manually by an administrator. In particular, [21] introduces algorithms and tools to simplify network administration, i.e. to make topology changes in the overlay network, but these tools are intended to be just an aid for a human administrator.

The notion of automatic self-organization of the overlay network has been introduced by Cugola et al. in [98] and then included in the REDS system [46]. In REDS, self-organization is driven by changes happening in the underlying network, often occurring in mobile systems and peer-to-peer networks. Differently from our approach, authors of [98] introduce techniques that “react” to topological changes, but does not induce such changes to improve performance. A similar approach was presented also in [134].

A completely self-organized publish/subscribe system was introduced in [131]. In this work the authors propose an implementation of the content-based publish/subscribe scheme on top of an unstructured peer-to-peer system. The proposed publish/subscribe system is able to self-organize in order to exploit similarity between client subscriptions. Differently from our approach the proposed similarity metric is only based on subscriptions, without taking into account event distributions.

Event dissemination through multicast trees and gossiping. In single source systems efficient event dissemination can be achieved by dynamically building single-source multicast trees ([103, 94]) over an overlay network. However the model addressed in this chapter, where every broker can be a source of events, is not suitable for such technique. This is due to the fact that the number of trees that should be built for each broker is given by the number of topics, in topic-based pub/sub systems, and by the number of possible events in content-based pub/sub systems. Moreover, these trees should be rebuilt/updated each change/removal of a subscription.
Gossip-based algorithms have been considered as a viable alternative to traditional deterministic multicast algorithms in large scale environments [54]. Using these algorithms, there is no need of building/maintaining any structure over the overlay network. This is at the cost of actually flooding any event through the overlay.

Distributed query processing systems. If we consider an event as a piece of data and a subscription as a persistent query, our CBR architecture becomes very similar to wide area distributed query processing systems (e.g. PIER [66], Astrolabe [128], etc.). In PIER, for example, a query dataflow engine is deployed over a DHT system, however queries are instantaneous (i.e. non persistent) in the sense that they do not last for a given period of time. The aim of PIER is indeed to build database related functions (selection, projection, join, grouping and aggregation) over the basic put/get primitives provided by DHTs.

Semantic overlay networks. Keeping close over a virtual network all pieces of information having similar semantics is a principle that has been also used in the self-organizing semantic overlay network named pSearch [121]. In this system the basic idea is to adapt classical information retrieval algorithms (like the one used in Google-like search engines) to work on DHTs. However the semantic space used in pSearch is based on document summarization, therefore queries can return only approximate results which is in contrast with the content-based pub/sub system which is able to deliver precise results with respect to persistent queries (i.e. subscriptions). Semantical proximity has also been successfully exploited for enhancing the search performance in peer-to-peer file sharing systems. In particular, [117, 130] present a mechanism for improving search by grouping peers sharing similar content. In both, semantic proximity is computed following on a principle similar to our similarity metric: taking into account previous query result, a node consider as 'semantically close' those nodes that in the past either have provided positive replies to its query or have issued queries to which it has replied positively.

Network-aware overlays. Many techniques for building network-aware overlay topologies have been described in the literature ([76, 133, 119, 33, 105]). In all these works the overlay is built only by taking into account the underlying network topology, without considering, as we do in our algorithm, also application-level information.

On the other hand our self-organization algorithm presents some resemblances with the network aware heuristic used in Pastry. Pastry addresses network proximity (as described in [34, 33]) through a heuristic that chooses the neighbors for any node in the network as the ones that are the closest at network-level to the node (according to a generic distance metric).
Chapter 3

Efficient Topic-based Event Routing for Peer-to-peer Systems

The filtering mechanism employed in SIENA [29] or REBECA [86] is based on the complete knowledge of the set of subscriptions issued in the system. Thanks to this information event filtering can safely prune diffusion tree branches without negatively affecting the correctness of the diffusion process, i.e. every target broker will receive the event regardless of the presence of event filtering.

This approach to event filtering is effective for static systems, mostly based on managed broker overlays, where the event notification service can be considered as immutable (apart from possible failures).

Peer-to-peer applications show a completely different behavior. They are made up of a very large number of participants that span several administrative domains; there is no fixed infrastructure: the system is composed only by its users, each sharing part of its resources and collaborating with others. A publish/subscribe system, implemented using a peer-to-peer approach, would be characterized by a highly mutable event notification service, where participants, playing at the same time the role of publishers/subscribers and brokers, can join or leave at any time, and the overlay network connecting them evolve accordingly. The cost of maintaining complete and updated subscription information for event filtering in such scenario can be overwhelming.

Maintaining incomplete information for event filtering can mitigate this problem, but it has an high impact on the correctness of the event diffusion process. In this chapter we propose a synergic approach based on a probabilistic event filtering mechanism and an interest isolation technique. Event filtering is used to probabilistically diffuse a published event towards only one of the target brokers. Interest isolation is a variant of the interest clustering technique: it guarantees that brokers sharing the same interest are all connected to form a specific subnetwork for the interest they share. In this way, an event that reaches one of the target brokers, can be
then flooded in the subnetwork relative to the interest it matches and then reaches all the remaining target brokers.

Figure 3.1: Two-layers architecture for complete interest isolation. Each subgroup at the upper layer is formed by brokers sharing same interests. Events are routed at the lower level through an access point toward for the subgroup whose interest is matched by the event, and then simply flooded inside it.

The approach previously outlined can be implemented through a two-layer architecture, as depicted in figure 3.1: at the lower layer an interconnection medium connecting all the brokers is used to build and maintain interest-based subgroups of brokers, while at the upper layer all these subgroups remain independent. An event published on a broker is routed, at the lower layer, toward one access point for the target subgroup, i.e. toward one of the target brokers pertaining to the subgroup. This broker can then just apply the event flooding mechanism, limiting its coverage only to brokers pertaining to the target subgroup.

This mechanism is able to actually reduce the production of spam messages only to the first part of the event routing, that is when it searches for an access point to the target subgroup. Once the event has reached an access point, all the further brokers that will receive it are part of the target subgroup, and thus, if the group is correctly built and maintained, they host one or more subscriptions matched by the event. In some sense, this two-layer architecture is able to completely isolate disjoint interests, and then limit flooding only to interested brokers.

The effectiveness of the described mechanism clearly depends on how interest isolation is realized: if interests are too fine grained, the system would be fragmented in a huge number of groups, each containing few brokers, thus increasing the effort needed to build and maintain them. From this point of view, interest isolation is perfectly suited for those applications where the set of possible interests is limited, like in topic-based event diffusion.

In the following of this chapter we apply the described mechanism to a specific
setting: large scale dynamic peer-to-peer systems. Section 3.1 gives the reader a brief introduction to overlay networks for large scale dynamic peer-to-peer systems; section 3.2 introduces TERRA, a topic-based publish/subscribe system built on top of such networks, while section 3.3 evaluates its properties. Finally, section 3.4 discusses related work.

### 3.1 Overlay networks for large scale dynamic peer-to-peer systems

In the last decade the advent of peer-to-peer computing introduced a new model of distributed computation where (i) the scale of the system can be very large, comprising up to million of users (peers), (ii) each peer acts independently from all the others, actually precluding any form of centralized network-wide administration or management, (iii) each peer acts as a client of the service and cooperates with other peers to enable services for other participants, and (iv) the system, due to its size and the autonomy of each peer, is intrinsically dynamic as peers can join in or leave at any time.

In this context the basic problem that must be solved in order to build distributed applications is how to guarantee connectivity among participants. Connectivity is, in fact, the basic building block to enable network communications among peers. Modern peer-to-peer systems use, to this aim, an overlay network. A distributed algorithm running on nodes, known as the overlay maintenance protocol, takes care of the overlay “healthiness” managing these logical links.

The common characteristic of all overlay maintenance protocols is that each node maintains links to other nodes in the system. This set of links is limited in its size in order to favor system scalability and it is usually known as the view of the node. Views construction and maintenance should be such that the graph, obtained by interpreting links in views as arcs and nodes as vertexes, is connected, as this is a necessary condition to enable communication from each node to all the others.

As we saw in section 1.4.2, overlay maintenance protocols can roughly be classified in two groups [32]:

**Structured.** Systems that exploit a highly-constrained overlay network topology and specific resource placement to provide applications with high performance resource-discovery primitives [110, 139, 119, 104, 79].

**Unstructured.** Systems that try to maintain a randomized network topology characterized by some properties inherited by random graphs, like high connectivity and low diameter. [39, 58, 55, 73, 129, 2, 70, 81, 96].

There is still no agreement both in the research community and in the industry on which group is better for which application. However a common thought is that
peer-to-peer systems based on unstructured overlay maintenance protocols, thanks to their loosely organized structures and simple protocols, can be better suited for very large-scale systems, where the dynamic behavior of nodes, continuously joining and leaving the system, can play a strong role, easily hampering the performance of structured systems.

Overlay maintenance protocols for unstructured peer-to-peer systems differentiate among themselves with respect to the technique they employ to build and maintain views. They can be divided in two broad groups basing on the strategy used to manage node leaves:

**Reactive Protocols** - require leaving nodes to execute some protocol before actually leaving the system (e.g. [73]) in order to ensure network connectivity.

**Proactive Protocols** - continuously adjust the network topology in order to allow nodes to leave without executing any specific algorithm (e.g. [129, 2, 70, 81, 96]).

Today there is common agreement, confirmed also by simulation studies [13, 14], in considering proactive protocols more suited than reactive ones to dynamic environments. All these systems are based on some variation of a technique known as view exchange (or view shuffle), that requires nodes to continuously exchange part of their views in order to keep the overlay topology as close as possible, along the time, to a random graph. Random graphs are characterized by strong connectivity, a property that is exploited to avoid network partitioning: node faults can, in fact, be simply ignored by the OMP as the random topology is supposed to remain connected despite node removals.

The small diameter proper of random graphs is exploited by these OMPs to offer developers effective broadcast primitives able to span the whole overlay network in few hops.

### 3.2 TERRA: Topic-based Event Routing for Randomized p2p Architectures

In this section we introduce TERRA, a topic-based event routing infrastructure for unstructured peer-to-peer systems based on view exchange.

#### 3.2.1 Overview

TERRA is a topic-based publish/subscribe system designed to offer a robust and efficient event diffusion service for very large scale systems, made up of autonomous and collaborative participants. In TERRA we do not make any distinction between clients of the publish/subscribe service and brokers implementing it: each client, either
publisher or subscriber or both, also acts as a broker. In the following we will refer to any TERRA participant simply as a node of the system.

In TERRA subscribers and publishers can subscribe or publish events tagged with a topic. The set of available topics does not need to be limited or predefined: applications using TERRA can define a new topic at any moment just subscribing to it or publishing an event tagged with it. TERRA will automatically take care of managing topic creation or deletion.

All nodes in TERRA are organized in a logical two-layers architecture. Each group of nodes, belonging to either the lower or upper layer, is kept connected using an instance of an overlay maintenance protocol. The overlay network representing the lower level group contains all TERRA’s nodes and is called global overlay. Overlay networks representing upper level groups are used to maintain connected all nodes sharing the same subscription (i.e. each overlay represents a topic), and are called topic overlays. All these overlay networks are completely independent, and work concurrently, thus every node maintains a separate view for each overlay it is part of. The overlay maintenance protocol employed to manage general and topic overlay must only provide a simple mechanism to allow neighbor nodes to communicate.

A node subscribing to a topic $t$ must join the corresponding topic overlay containing all the other nodes subscribed to $t$. A node publishing an event $e$ for topic $t$, in order to diffuse it toward all the target nodes, must forward $e$ to one of the nodes subscribed to $t$. Once this node receives $e$ it will first notify locally the event, and then broadcast $e$ in the topic overlay associated with topic $t$, where all other $t$’s subscribers are connected.

Various problems must be addressed in order to realize this form of event diffusion. The most important is how a node can find another node in the system that is subscribed to a specific topic $t$ (if such node exists); in other words: we need a service able to return a node that can be used as an access point for a specific topic in order to publish events or subscribe to that topic. TERRA solves this problem through a topic sampling service that is used by each node to maintain a cache of access points. This service requires from the overlay maintenance protocol employed to manage the global overlay, in order to work correctly, access to a set of neighbors that at each moment represents a uniform random sample of the whole system population. For this reason we consider, for the development of TERRA, only overlay maintenance protocols based on a view exchange mechanism, as this type of overlay maintenance protocols is able to maintain an overlay whose topology closely resembles a random graph [67]. Further details about this aspect will be introduced in the next section.

Another important problem in TERRA is related to the fact that the algorithm used to manage topic overlays, cannot ensure that a single overlay will not partition [13, 14], i.e. that there will always be a one-to-one correspondence between topics and topic overlays. For this reason TERRA employs a partition recovery mechanism that is used to merge partitioned topic overlays.

A last problem is caused by the fact that a node subscribed to many topics will be
also part of many different overlay networks (one for each subscribed topic, plus the
general overlay). All these instances periodically generate some overhead due to the
view exchange mechanism. To reduce this overhead, that can be cumbersome when
the number of topics and subscriptions grows, TERRA employs a shuffle avoidance
mechanism. This mechanism leverages the fact that all the nodes pertaining to a topic
overlay are also part of the general overlay to reduce the number of view exchange
executions at the upper layer.

3.2.2 Topic sampling

Topic sampling is realized in TERRA using this simple strategy:

- every node periodically sends the list of its active subscriptions to a subset of
  the nodes in its global overlay view (subscription advertisement);

- each time a node $n$ receives a subscription advertisement for topic $t$ from a node
  $m$, it updates the content of a local cache, named Access Point table (APT).

Let us assume, for the moment, that APTs can contain an infinite number of
entries. An update is performed adding the couple $<t, m>$ to the local APT or sub-
stituting an entry $<t, m'>$ previously present in the APT\(^1\). This simple mechanism
maintains APTs with the following properties:

1. as time passes by each APT tends to contain an entry for each active topic, i.e.
   for each topic subscribed by a node at least;

2. an APT entry for an active topic $t$ tends to contain a non-stale access point, i.e.
   a node that is still subscribed to $t$;

3. an APT entry for an active topic $t$ contains a node that is a uniform random
   sample from the set of nodes subscribed to $t$.

All these properties are guaranteed by the fact that each global overlay view is
a uniform random sample of the system population, thus, if subscription advertise-
ment is done with a larger period than the global overlay view shuffle, each list of
subscriptions will be received by a random set of nodes.

To better understand the first property, suppose that in the system there is only
one active topic $t$ that is subscribed by a node $n$. The global overlay view of $n$ changes
periodically due to the overlay maintenance protocol view shuffle mechanism, and its
content is a uniform random sample of other nodes. If the system population does
not change quickly, eventually all nodes will appear in $n$’s global overlay view the

\(^1\)With a small notation abuse we indicate with $m$ both a node or its network identifier (like its
IP address). When the distinction between these two interpretations is not implicit in the text, we
differentiate them explicitly.
same number of times. \( n \) chooses targets for its periodic subscription advertisement from this view, thus eventually it will send its only subscription to all nodes in the system. This means that, after a certain amount of time, all nodes in the system (but \( n \) itself) will contain an entry \( < t, n > \) in their APT. Generalizing this reasoning we can conclude that each node will eventually receive a subscription advertisement for each active topic in the system.

The second property is rather obvious. Suppose that in the system there is only one active topic \( t \) that is subscribed by two nodes, \( n_a \) and \( n_b \), and also suppose that, at a certain point in time, \( n_b \) unsubscribes topic \( t \), thus leaving \( n_a \) as the only active subscriber for \( t \). From that moment on, only \( n_a \) will continue to advertise \( t \). Therefore, eventually every node in the system will receive a subscription advertisement from \( n_a \) that will update its APT, thus, in the end, every APT will contain an entry \( < t, n_a > \).

The third property is a direct consequence of the reasoning done to justify the first one: if all nodes subscribed to a topic \( t \) advertise their subscriptions with the same period, a generic node has the same probability to receive an advertisement for \( t \) from any of them.

APTs built and maintained in this way can be straightly used as a simple way to find access points for topics: a node publishing an event for a topic \( t \) (or subscribing to that topic) should only look in its APT to find the address of a broker subscribed to that topic. Topics that are not listed in the APT (or whose entry points to a stale node) can be considered as inactive.

Despite these properties, the topic sampling service, as it was presented so far, presents some important drawbacks:

- APTs indefinitely maintain each entry, even if the corresponding topic is no more active;
- given that the set of admitted topics in the system is not predetermined, APTs can occupy an infinite amount of memory;
- the frequency at which a node receives advertisements for a topic is proportional to its popularity, i.e. to the number of active subscribers for that topic.

To address the first two points we simply limit the size of each APT to a fixed number of entries: when an APT exceeds this limit, enough randomly chosen entries are removed. In this way topics that are no more active will eventually disappear from APTs, as their entries will be removed without any update that can insert them back in.

If APTs’ size is limited, the uneven frequency at which topic advertisements are received tends to fill APTs only with entries related to more popular topics, excluding less popular ones whose advertisements are rarely received, as there are less nodes in the system sending them. To correct this problem we slightly modify the APT management policy: each time an advertisement for a topic \( t \) sent by a node \( m \) is
received on a node $n$, $n$ updates the access point for $t$ if an entry $<t,m'>$ was already present in its APT, otherwise it adds an entry $<t,m>$ with probability $1/P_t$, where $P_t$ is the popularity of topic $t$.

The popularity $P_t$ can be obtained through an estimation of the size of $t$’s topic overlay. This estimation can be realized by the advertising node using well known techniques (e.g. [77, 75, 69]), as a subscriber of $t$ it is part of the corresponding topic overlay, and then included in the advertisement message.

Thanks to these improvements, APTs also enjoy the following properties:

1. the size of each APT is limited;
2. topics that become inactive tend to disappear from APTs;
3. the content of each APT is a uniform random sample of the set of active topics in the system.

The second property derives from the fact that inactive topics are not advertised by any nodes, thus their entries in APTs eventually are replaced by other entries containing active topics (in the hypothesis that the set of active topics is larger than the maximum APT size).

The third property is a consequence of the policy employed to replace APT entries: given that the probability for a generic node to receive an advertisement for a topic $t$ is proportional to the number of nodes advertising it, i.e. to its popularity $P_t$, and that an entry for it is added to the APT with a probability that is inversely proportional to $P_t$, then we can conclude that each active topic has the same probability to be added to an APT.

Limiting the size of APTs actually brings in another problem: given that each node has a limited “knowledge” of the set of active topics, it cannot decide, only looking at its APT, if a specific topic $t$ is active or not. If an entry for $t$ is present in its APT, in fact, it can deduct that $t$ is an active topic and obtain directly an access point; but if its APT does not contain an entry for $t$, there is no way for the node to decide if $t$ is really not active, or simply its APT currently lacks an entry for it. To solve this problem each node can search for an access point on the APTs pertaining to other nodes. This search is conducted launching a random walk on the global overlay that travels through nodes in the system looking for an access point for the desired topic. The rationale behind this search mechanism is that, given the randomness of both the global overlay topology and the APTs’ content, the walk can visit a number of APT entries sufficient to ensure that either (i) an access point will be found or (ii) the topic can be safely considered as inactive. The APT size and the maximum length allowed for each random walk are two parameters that must be cautiously tuned in order to ensure a good reliability for the topic sampling service. An in-depth analysis of this aspect is reported in Section 3.3.3.
3.2. Implementation details

In this section we explain the details of TERRA’s implementation. The presentation is divided in various functional aspects for ease of comprehension.

Data structures and commodity functions

Before dwelling deep into the details of the various TERRA’s functionalities, we introduce some data structures and commodity functions that will be later referred to.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>(Subscription table) a set of couples with the form (&lt;\text{topic}, \text{id}&gt;) where \text{topic} is a topic identifier, and \text{id} is an overlay identifier.</td>
</tr>
<tr>
<td>OT</td>
<td>(Overlay table) a set of triples with the form (&lt;\text{id}, \text{view}, \text{updated} &gt;) where \text{id} is an overlay identifier, \text{view} is a set of pointers to other nodes, and \text{updated} is a number.</td>
</tr>
<tr>
<td>APT</td>
<td>(Access Point table) a set of couples with the form (&lt;\text{topic}, \text{node}&gt;) where \text{topic} is a topic identifier, and \text{node} is a pointer to a node.</td>
</tr>
</tbody>
</table>

Table 3.1: Data structures

TERRA maintains on each node three main data structures that are listed in table 3.1. The Subscription table (ST) is used to keep track of locally active subscriptions. Each time an application running on the node, and using the publish/subscribe infrastructure provided by TERRA, issues a subscribe or an unsubscribe operation, the subscription table is modified accordingly adding or removing an entry. Each entry in the table is a couple \(<\text{topic}, \text{id} >\), where \text{topic} is a subscribed topic, and \text{id} is the overlay id associated to that topic. Each overlay id identifies a single topic overlay in the system, and the node keeps track of information related to this topic overlay in a second structure named Overlay table (OT). Each entry \(<\text{id}, \text{view}, \text{updated} >\) in the overlay table represents an overlay network the node belongs to, and whose overlay id is \text{id}. The element \text{view} is the view the node maintains for that specific overlay network, thus it is a collection of pointers to other nodes belonging to the same overlay. A special entry in the overlay table is \(<\Phi, v, u >\) that represents the general overlay. Finally, \text{updated} is a simple counter whose use will be later clarified. The last data structure maintained on each node is the Access Point table (APT) we introduced in section 3.2.2.

TERRA’s algorithms also uses some commodity functions that are reported in table 3.2. The table details for each function its behavior as well as the required parameters.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Broadcast(e,v)</code></td>
<td>implements a broadcast primitive on the overlay network reachable through view v.</td>
</tr>
<tr>
<td><code>EstimateSize(i)</code></td>
<td>returns a size estimation of the overlay network with overlay identifier i.</td>
</tr>
<tr>
<td><code>GenerateOverlayID(t)</code></td>
<td>returns a new unique overlay identifier based on topic identifier t.</td>
</tr>
<tr>
<td><code>GetAccessPoint(t)</code></td>
<td>returns a set of pointers to access points for the topic identifier t. If <code>&lt; t, n &gt; ∈ APT</code>, it returns n, otherwise it starts a number of concurrent random walks that search for access points stored in other nodes' APTs. Each random walk either returns an access point or a negative response. If the function is unable to find any access point it returns an empty set.</td>
</tr>
<tr>
<td><code>JoinTopicOverlay(t,n)</code></td>
<td>starts the standard overlay maintenance protocol join procedure for an overlay network associated to topic t, using n as a bootstrap node. The function returns a couple <code>&lt; i, v &gt;</code> where i is the overlay identifier of the joined overlay network, and v is the startup view produced by the overlay maintenance protocol join procedure.</td>
</tr>
<tr>
<td><code>Notify(e)</code></td>
<td>notifies applications, that are using the publish/subscribe infrastructure, about a published event matching one of the locally issued subscriptions.</td>
</tr>
<tr>
<td><code>Shuffle(t,i,v,n)</code></td>
<td>starts the overlay maintenance protocol view exchange procedure on view v representing the overlay with identifier i associated to topic t. If an argument n different from – is provided, then the view exchange procedure is forced between the current node and node n; if – is specified the view exchange procedure is executed following the standard overlay maintenance protocol strategy.</td>
</tr>
<tr>
<td><code>Truncate(APT)</code></td>
<td>if the APT’s size is larger than a predefined parameter, then removes the correct number of entries to meet the size constraint. Entries to be removed are randomly selected.</td>
</tr>
<tr>
<td><code>UpdateView(v,n)</code></td>
<td>insert the pointer to node n in view v.</td>
</tr>
</tbody>
</table>

Table 3.2: Functions.
### 3.2. TERRA

#### APT management

One of the main mechanisms on which TERRA relies for its correct functioning is the topic sampling service introduced in section 3.2.2. This service is responsible for maintaining continuously updated an access point table (APT) on each node. The topics sampling service is constituted by a procedure periodically executed on each node, and a message handler.

**Algorithm: 1 - Periodic subscription advertisement**

1: Every $T_1$ time units do
2: $L \leftarrow D$ nodes randomly chosen in $v$, where $<\Phi,v,u> \in OT$
3: for all $n \in L$ do
4: for all $< t, i > \in ST$ do
5: send SUBUPDATE[$t,i,\text{EstimateSize}(i)$] to $n$

The procedure reported in algorithm 1, executed every $T_1$ time units, is used by a node to advertise its subscriptions on some of its neighbors. The node executing it first selects in the view associated to the general overlay $D$ random neighbors ($D$ is a parameter of the algorithm whose size must be less or equal to the view size). Then it sends to these nodes a SUBUPDATE message with the content of its subscription table. For each entry it also includes its current estimation of the size of the corresponding topic overlay obtained through a call to the EstimateSize function; this estimation reflects the popularity of the topic, i.e. how many nodes have subscribed it.

**Algorithm: 2 - SUBUPDATE message handler**

1: On receive SUBUPDATE[$t,i,s$] from $n$ do
2: if $< t, j > \in ST$ and $< j, v, u > \in OT$ then
3: if not $j = i$ then
4: Shuffle($t, j, v, n$)
5: else
6: UpdateView($v, n$)
7: $u = u + 1$
8: if $< t, x > \in APT$ then
9: $x \leftarrow n$
10: else
11: $APT \leftarrow < t, n >$ with probability $1/s$
12: Truncate($APT$)

Each time a node receives a SUBUPDATE message it executes the handler reported in algorithm 2. The first operations executed in the handler (lines 2-7) are used for overlay management and will be detailed later. After these operations the node
updates its APT following the criteria explained in Section 3.2.2: if an entry for the topic is already present in the APT then the corresponding access point is simply updated (line 9); if an entry for the topic is not present in the APT, then it is added with a probability that is the inverse of the estimated size provided in the message (line 11). In the latter case the the APT is truncated to respect the constraint on its maximum size.

Subscription management

**Algorithm: 3 - SUBSCRIBE event handler**

```plaintext
1: On receive SUBSCRIBE[t] do
2:   if ∄ i : < t, i > ∈ ST then
3:     L ← GetAccessPoint(t)
4:     if L = ∅ then
5:       < i, v >←<GenerateOverlayID(t), ∅ >
6:     else
7:       < i, v >← JoinTopicOverlay(t, n), where n ∈ L
8:     OT ←< i, v, 0 >
9:     ST ←< t, i >
```

Each time an application running on the node and using TERRA issues a new subscription the SUBSCRIBE event handler, reported as algorithm 3 is executed.

The handler first checks if the node is already subscribed to the same topic (line 2). Then it calls the GetAccessPoint function to obtain an access point for topic t. The GetAccessPoint function first check if an access point for the topic is contained in the local APT. If this is the case it returns the node pointer contained in the APT entry, otherwise it starts various concurrent random walks whose aim is to search for an access point for topic t in the APTs managed by the visited nodes. Each walk can return either a pointer to an access point for t, or a negative result. The GetAccessPoint function returns the first access point found or an empty set if no access points were found.

In the latter case the node can suppose that no subscriber exists in the system for t, therefore it can generate a new overlay network instance to represent the topic overlay associated to t. This is done simply generating a new overlay identifier through the GenerateOverlayID function (line 5), and then adding a new entry to both the overlay table and the subscription table (lines 8-9). The GenerateOverlayID function can be used to generate unique overlay identifiers.

In the former case, that is if an access point for t is found, the node must join an existing topic overlay. This is done through the JoinTopicOverlay function (line 7) whose purpose is to start the overlay maintenance protocol join procedure for the topic overlay, using the node returned by GetAccessPoint as a bootstrap node. As a
result of this join procedure, JoinTopicOverlay returns the overlay identifier of the
joined topic overlay, and an initial view, and both are added as a new entry to the
overlay table (line 8). Even in this case a new entry for \( t \) is added to the subscription
table (line 9).

Algorithm: 4 - UNSUBSCRIBE event handler

1: On receive UNSUBSCRIBE[\( t \)] do
2: \( \forall i, ST \rightarrow < t, i > \)
3: \( \forall i, OV \rightarrow < i, v, u > \)

The management of an UNSUBSCRIBE event is rather simple, and is reported as
algorithm 4. The algorithm simply removes the entry related to topic \( t \) from both the
subscription table and the overlay table. Note that this approach to topic unsubscrip-
tions is perfectly in-line with the proactive approach used in most view exchange-
based overlay maintenance protocols: topic unsubscription does not require any spe-
cific leave operation to be executed on the corresponding topic overlay.

Overlay management

The topic overlays created by the SUBSCRIBE handler, as well as the general overlay,
are managed by the same overlay maintenance protocol. As we stated in section
3.2.1 the employed protocol must satisfy some requirements, needed by TERRA
to work correctly, thus we restrict the choice to the class of overlay maintenance
protocols based on the view exchange (or view shuffle) technique. TERRA wraps
the view exchange mechanism in two procedures, reported as algorithm 5, that are
periodically executed.

Algorithm: 5 - Periodic overlay management

1: Every \( T_2 \) time units do
2: Shuffle(-, \( \Phi, v, \cdot \), where \( < \Phi, v, u > \in OT \)
3: Every \( T_3 \) time units do
4: for all \( < i, v, u > \in OT \) \( / < \Phi, v, u > \) do
5: if \( u < U \) then
6: Shuffle(\( t, i, v, \cdot \), where \( < t, i > \in ST \)
7: \( u \leftarrow 0 \)

The first procedure (lines 1-2), executed every \( T_2 \) time units, simply calls the
Shuffle function that triggers the standard overlay maintenance protocol view ex-
change mechanism on the general overlay.

The second procedure (lines 3-7), executed every \( T_3 \) time units, do the same thing
on all the entries of the overlay table related to topic overlays. Note however that in
this case a condition (line 5) is used to limit the number of view exchanges: if the
value of the counter \( u \) for some topic overlay is greater or equal to a parameter \( U \) then the corresponding view shuffle is skipped. This condition is part of a shuffle avoidance mechanism TERRA employs to limit the overhead generated for shuffling topic overlays. The counter \( u \) is updated in the SUBUPDATE message handler (see algorithm 2) on lines 6-7: if a node receives a SUBUPDATE message from a node \( n \) that is subscribed to the same topic it can update the corresponding view, adding a pointer to \( n \), and increment its counter \( u \) in the overlay table. Through this mechanism (shuffle avoidance) TERRA tries to feed topic overlay views with fresh nodes: if the period \( T_3 \) is large enough with respect to the period \( T_1 \), the number of fresh nodes put in the view between two successive shuffles can be large enough to allow TERRA to avoid a view exchange (see line 5 of algorithm 5). Note that the threshold \( U \) is defined as the number of nodes usually exchanged during an overlay maintenance protocol shuffle operation.

The view update done by the SUBUPDATE message handler can be actually executed only if the corresponding topic overlay is not partitioned. This condition can be easily checked by a node just comparing the overlay identifier \( i \) contained in the SUBUPDATE message, with the overlay identifier \( j \) stored in its overlay table (line 3 of algorithm 2). If the two identifiers are different it can conclude that the topic overlay is partitioned, and the node \( n \), source of the received message, is not in the same partition as itself. In this case TERRA’s partition recovery mechanism is triggered: to merge the two partitions the node calls the Shuffle function (line 4) forcing a view exchange to take place between its own view for the partitioned topic overlay, and the view for the same overlay belonging to node \( n \). Aim of this forced shuffle is to exchange some links between the two partitions, thus creating the conditions for the standard overlay maintenance protocol’s shuffle mechanism to completely and strongly merge the two partitions.

To complete the merge there is a detail that is still missing: nodes in the two merged partitions should, at the end, share a single overlay identifier for the topic overlay. This can be solved by the Shuffle function: each time two nodes shuffle their views of a topic overlay they also exchange the corresponding overlay identifier and use a deterministic rule to sync different IDs. The rule used depends on the type of overlay identifier used: it is always possible to generate through GenerateOverlayID totally ordered identifiers, and thus employ a simple min/max operator as a deterministic decision rule.

**Event diffusion**

Event diffusion is started as soon as an application generates a PUBLISH event on a node. The handler for the PUBLISH event, reported as algorithm 6, is executed in this case, or when a node receives a PUBLISH message from another node. In either case, the node first checks (line 2) if it hosts a subscription matched by that event; if this is the case it notifies the event to the application that issued the matched
Algorithm: 6 - PUBLISH event handler

1: On receive PUBLISH[e,t] do
2: if < t, i > ∈ ST then
3: Notify(e)
4: Broadcast(e,v), where < i, v, u > ∈ OT
5: else
6: L ← GetAccessPoint(t)
7: if not L = ∅ then
8: send PUBLISH[e,t] on n, where n ∈ L

subscription through the Notify function (line 3), and then forward the event to all the other nodes in the topic overlay associated to the topic the event was published in through the Broadcast function (line 4). This last function implements an application level broadcast primitive. The way the broadcast is implemented is an argument not treated here; however it is important to note that the reliability of the broadcast implementation can affect the reliability of the whole publish/subscribe system in terms of message delivery.

If the node does not hold a subscription matched by the published event, it must find an access point for the topic the event was published in, in order to forward the event and start from there the diffusion in the correct topic overlay. This operation is similar to the one we saw for the SUBSCRIBE handler; the node looks for an access point (line 6) and forward toward it the event (line 8). If an access point for the topic is not found, then the node discards the event as it can suppose that no subscription exists in the system for that topic.

3.3 Evaluation

In this section we evaluate various TERRA’s characteristics as well as its global behaviour. The evaluation is based on both an analytical study of the protocol, and an experimental study conducted through simulations.

3.3.1 Experimental setup

A prototype of TERRA publish/subscribe system was implemented on Peersim [68], an open source Java simulation framework for peer-to-peer protocols. Through peersim we were able to test TERRA on very large simulated network, modeling with sufficient precision the environment where TERRA is supposed to work. In this subsection we detail the model used to generate simulated scenarios.
Overlay network model

TERRA was tested on networks with various sizes ranging between 1000 and 10000 nodes. Each node, runs TERRA’s protocol and acts at the same time as a publisher, a subscriber, and a simple event broker. As we saw in section 3.2.1 TERRA employs an overlay maintenance protocol to keep nodes connected at both layers. For our testbed we chose Cyclon [129] as the reference overlay maintenance protocol as it satisfies all the requirements expressed in section 3.2.1.

Cyclon is an overlay maintenance protocol based on the view exchange technique, that requires nodes to perform a continuous periodic activity with their neighbors in the overlay. The view exchange phase (named in this case “shuffle cycle”) aims at randomly mixing views between neighbor nodes. Joins are managed in a reactive manner, through a join procedure, while voluntary departures of nodes are handled like failures (no leave algorithm is provided). A simple failure detection mechanism is used in order to clean views from failed nodes.

Each node maintains only a view of nodes it can exchange data with. The size of the view is fixed and can be set arbitrarily. Each node in the view is associated to a local age, indicating the number of shuffle cycles during which the node was present in the view.

A node joins the overlay network by choosing one node (bootstrap node) among those already present in the system. The protocol starts then a set of independent random walks from the bootstrap node. The number of random walks is equal to the view size, while the number of steps per each random walk is a parameter of the algorithm. When each random walk terminates, the last visited node adds the joining node to its view by replacing one entry which is added to joining node’s view using an empty slot.

The shuffle algorithm (view exchange) is executed periodically at each node. A shuffle cycle is made up of three phases. In the first phase a node A, after increasing the age of all the nodes in its view, chooses its shuffle target, B, as the node with higher age among those in its view. Then, A sends to B a shuffle message containing \( l - 1 \) nodes (where \( l \) is a parameter of the algorithm) randomly chosen in A’s view, plus A itself. In the second phase, B, once received the shuffle message from A, replaces \( l - 1 \) nodes in its view (chosen at random) with the \( l \) nodes received from A and sends them back to A. In the final phase A replaces the nodes previously sent to B with those received from it. Overall, the result of one shuffle cycle is an exchange of \( l \) links between A and B. The link previously connecting A to B is also reversed after the shuffle.

Overlay networks built by the cyclon protocol closely resemble random graphs, and show similar characteristics [129, 67]. Moreover, the continuously evolving topology updated by the view shuffle mechanism shows another important property: at runtime each view contains each node in the system with the same probability. This means that views represent uniform random samples of the system population and
can be then used to build a peer sampling service able to provide uniform samples.

To build our TERRA's prototype we used the original Cyclon implementation for Peersim, slightly modified to take into account concurrency among nodes.

Data model

Given the absence of publicly available data traces of real pub/sub applications, we tested our algorithm on various synthetic scenarios. Following the same approach previously used in other simulation studies [106, 26, 30] we generated various scenarios characterized as follows:

Subscriptions: The set of subscriptions issued in a topic-based publish/subscribe system can be generally fully characterized by the following properties:

- number of topics;
- number of subscriptions;
- topic popularity distribution;
- subscriptions distribution on nodes.

Once the total number of topics and subscriptions is decided, topic popularity defines how many subscriptions are issued for each topic. In our tests we used two different distributions for topic popularity, to simulate workloads for different applications:

- Uniform: each subscription can be issued with the same probability on any topic; with this distribution all topics have almost the same number of subscribers (same popularity).

- Power-law: topic popularity distribution follows a zipf curve, leading to a system where few topics have a lot of subscribers (high popularity) while a lot of topics have just a few or none subscribers (low popularity).

Subscriptions distribution on nodes was always considered as uniform in our tests, i.e. each subscription is issued with the same probability on any nodes in the system.

Events

The set of events generated during the simulations was characterized by the following properties:

- number of topics;
- number of events;
distribution of events on topics;

- event distribution on nodes.

Obviously the number of topics considered is the same used to generate subscriptions. In our tests we always considered uniform distributions for both event distribution on topics and nodes.

### 3.3.2 Topic distribution in APTs

![Distribution of subscriptions on APTs (uniform)](image)

Figure 3.2: The plot shows how topics are distributed among APTs (black dots) when topic popularity distribution is uniform (grey dots).

The method introduced in Section 3.2.2 to update APTs content is designed to ensure a uniform distribution of topics in every APT, regardless of APT maximum size and single topic popularity. This is a fundamental property for APTs as it allows TERRA to use their content as a uniform random sample of the active topic population and build on it an effective access point lookup mechanism.

To test uniformness of ATP content, we ran various tests with different subscription distributions. Tests were run over a system with 10000 nodes each advertising its subscriptions every 5 cycles to 5 neighbors out of 20 (the cache size). APT size was limited to 10 entries. We issued at the beginning 5000 subscriptions distributed in various ways on 1000 distinct topics. We used both uniform and zipf distributions to test the ability of the APT update mechanism to correctly handle scenarios were topic popularity are skewed.

Figure 3.2 shows the results for an initial uniform distribution of topic popularity. In this case we expect the equality among all topic popularity to reflect in a uniform
distribution of topics in APTs, i.e. we expect the APT update mechanism to “maintain” the uniformness of topic popularity. The x axis in the graph is used to represent the topic population (each topic is mapped to a number). Each black dot represents how many times that specific topic appears in APTs while each grey dot represents its popularity. The plot confirms our intuition: each topic is present, on average, in 100 APTs (100 was the expected value, i.e. the number of topics divided for the size of APTs), with a very small error that is randomly distributed around the mean.

Figure 3.3: The plot shows how topics are distributed among APTs (black dots) when the topic popularity distribution is skewed (zipf with parameter $a$ - grey dots).

Figure 3.3 shows the results for an initial zipf distribution of topic popularity. The various graphs report the results for the distribution parameter $a$ varying among...
Note how, in this case, topic popularity are highly skewed, thanks to the zipf distribution. Nevertheless, TERRA is able to balance APT updates, and deliver an almost uniform distribution. Even in extreme cases, like with $a = 2.0$, the APT update mechanism is able to balance the updates coming from the small number of active topics (in this scenario only 79 topics share the whole 5000 subscriptions), maintaining their presence in APTs around the calculated average (1265.8 in this case) with a small standard deviation (always below 5%). In the following of this chapter we report results only for zipf popularity distribution with $a = 0.7$ as results for other values of $a$ did not show significant differences.

### 3.3.3 Topic lookup success rate

The fact that the content of APTs is correct and quickly updated is a necessary condition for the topic lookup mechanism to work correctly, but it does not suffice. The limited size of each APT, in fact, forces each TERRA’s node to look in other nodes’ APTs for a searched topic. If we suppose that the number of active topics is fixed, the probability for a node to find a topic in some APT is affected by both APTs’ size and the number of APTs visited through random walks: the smaller is each APT content, the longer each walk must be in order to find an access point for the desired topic.

The probability $p$ to find an access point for a specific topic in an APT is:

$$p = \frac{|APT|}{|T|}$$

where $|APT|$ is the APT’s size, and $|T|$ is the number of active topics. If we assume that at runtime each APT is full, i.e. it contains the maximum number of active topics, and denote with $K$ the number of nodes visited with the random walks, the probability that the topic cannot be found is:

$$Pr\{\text{fail}\} = (1 - p)^K$$

The probability to find the access point visiting $K$ nodes is thus:

$$Pr\{\text{success}\} = 1 - (1 - p)^K$$

$$= 1 - \left(1 - \frac{|APT|}{|T|}\right)^K$$

Therefore, if we want to ensure a certain probability $P$ to find an access point for a topic, we can correctly size $K$ or $|APT|$ using the following formulas:

$$K = \frac{ln(1 - P)}{ln \left(1 - \frac{|APT|}{|T|}\right)}$$

$$|APT| = |T| \left(1 - \sqrt[1/P]{1 - P}\right)$$
3.3. EVALUATION

Note that, given a certain $K$ and $P$, APTs size depends linearly on the number of active topics. In order to maintain APTs compact it could be thus necessary to let random walks visit a lot of nodes, using a large value for $K$. Long random walks however negatively affect the latency experienced by each access point lookup. To mitigate this problem it is advisable to launch $m$ multiple concurrent random walks, each with length $k$, in order to maintain invariate the value of $K = m \cdot k$.

Starting from these results, we ran various experiments to test how TERRA access point lookup mechanism behaves when varying $|APT|$, $m$ and $k$. Tests were run over a system with 1000 nodes with views holding 20 nodes. We issued at the beginning 5000 subscriptions uniformly distributed on 1000 distinct topics. Lookups were started after 1000 cycles left for the system to stabilize. Each lookup was conducted starting four concurrent random walks ($m = 4$).

![Random Walk success rate](image)

**Figure 3.4**: The plot shows how the success rate for topic lookups changes varying maximum APT size and random walk length. Solid lines represent results from the simulator, while dashed lines plots values from the formula.

Figure 3.4 shows how access point lookup success ratio changes when varying the length of each random walk ($k$) for different values of $|APT|$. For each line we plotted both simulation results (solid line) and values calculated with formula (dashed line). The plot confirms the intuition given by the analysis: there is a trade-off between APT size and random walk length that must be taken into account when dimensioning their size to let the access point lookup mechanism work at its best.

The evaluation we did was based on the assumption that the number of active topics is a fixed value known by every node. This assumption holds only in particular scenarios, where the set of topics, as well as subscription distributions are known in advance, but generally it is not applicable.

However, thanks to the fact that each APT can be considered a uniform random
sample of the current set of active topics, each node can, observing its evolution during time, estimate the size of this set. The estimation can be derived by the number cycles elapsed between the removal of a specific topic from the APT and a subsequent insertion of the same topic in the same APT [77]. The estimation of \(|T|\)'s value can be used by a node to dynamically adapt the APT size, or the random walk length, at runtime.

TERRA's lookup mechanism, as we have shown, is able to probabilistically guarantee, with properly configured parameters, that an access point for a topic will be found, as long as the topic is active. This aspect is fundamental for the reliability of TERRA's event diffusion mechanism. Event diffusion in TERRA is realized, in fact, in two phases: first the event is forwarded towards an access point for the relative topic, then the event is broadcasted in the corresponding topic overlay. Each phase has an impact on the reliability of the whole process. A study of reliable broadcast mechanisms for dynamic peer-to-peer overlay networks is out of the scope of this work, but we want to point out that TERRA's topic lookup mechanism can be configured to be highly reliable: employing reliable broadcast mechanisms, developers can build a system that can probabilistically guarantee an event delivery ratio close to 1.

3.3.4 Partition Recovery

The algorithm employed in TERRA to handle new subscriptions (refer to Algorithm 3 in Section 3.2.3) is based on the assumption that, if a topic overlay exists for the newly subscribed topic, an access point to it will be surely found by the access point lookup mechanism.

As we saw in Section 3.3.3 this is not necessarily true if the parameters regulating how the lookup mechanism works are not carefully chosen. If the lookup fails, even if a topic overlay for the searched topic exists, the system will end up with two distinct topic overlays for a same topic. This event negatively affects the reliability of the whole system as events published for that topic will be diffused each time only in one of the two topic overlays hampering global event hit ratio (i.e. the ratio between the number of notified events and the expected number of notifications).

As we saw in Section 3.2.3, to address this negative behaviour TERRA employs a partition recovery mechanism whose sole purpose is to detect the presence of partitioned topic overlays and merge them. Here we analyze how much time it takes for the mechanism to detect and merge a single partitioned node in a topic overlay.

Suppose that in the system there is a topic overlay made up of \(|G|\) nodes. All this nodes are subscribed to the same topic \(t\). Now suppose that a node \(n \notin G\) decides to subscribe \(t\), but the access point lookup mechanism fails, thus it generates for \(t\) a new topic overlay where it is the only participant. To calculate how many cycles are needed to detect the partition we must start calculating which is the probability \(p\) to detect it in a cycle:
3.3. EVALUATION

\[ p = 1 - (p_a \cdot p_b) \]

where \( p_a \) is the probability that none of the nodes in \( G \) advertises its subscriptions to \( n \), and \( p_b \) is the probability that \( n \) does not advertise its subscriptions to any of the nodes in \( G \). The product \( p_a \cdot p_b \) gives the probability that a merge will not happen. Let us analyze separately these two probabilities.

Probability \( p_a \) can be expressed as:

\[ p_a = (1 - Pr\{x advertises to n\})^{\|G\|} \]

A node \( x \in G \) advertises its subscription to \( n \) only if \( n \) is contained in its view for the general overlay, and it is one of the \( D \) nodes selected for the advertisement. Let us suppose, for the sake of simplicity, that \( D \) is equal to the view size, thus it suffices for \( n \) to be contained in \( x \)'s view in order to be a target for \( x \)'s advertisement. In this case \( Pr\{x advertises to n\} = \frac{|\text{View}|}{N - 1} \), and thus:

\[ p_a = \left( 1 - \frac{|\text{View}|}{N - 1} \right)^{\|G\|} \]

Let us now come back to \( p_b \) that can be expressed as:

\[ p_b = Pr\{\nexists x \in G : x \in \text{View of n}\} \]

\[ = \frac{C(N - 1 - |G|, |\text{View}|)}{C(N, |\text{View}|)} \]

This probability is, in fact, the fraction of all the possible views for the general overlay \( n \) can have, that do not contain nodes subscribed to \( t \), i.e. nodes in \( G \).

Therefore, the overall probability \( p \) is:

\[ p = 1 - \left( 1 - \frac{|\text{View}|}{N - 1} \right)^{\|G\|} \cdot \frac{C(N - 1 - |G|, |\text{View}|)}{C(N, |\text{View}|)} \]

From the expression of \( p \) we can derive the probability that a merge will happen in \( H \) cycles:

\[ Pr\{\text{merge in less than } H + 1 \text{ cycles}\} = 1 - (1 - p)^H \]

\[ = 1 - \left( 1 - \frac{|\text{View}|}{N - 1} \right)^{\|G\|} \cdot \frac{C(N - 1 - |G|, |\text{View}|)}{C(N, |\text{View}|)}^H \]
Figure 3.5: The plot shows how the probability to detect a topic overlay partition increases with time (cycles). Solid lines represent results from the simulator, while dashed lines plots values from the formula. The test were run varying the number $|G|$ of nodes subscribed to the topic.

These results are also confirmed by the experiments conducted with our simulator. We tested the partition recovery mechanism in networks made up of 1000 nodes, with a single topic. In these test $G$ subscriptions for the topic are initially issued on various nodes, that quickly form a topic overlay. Then a new subscription is issued on a node not yet subscribed, and a failed lookup is simulated, in order to create a second topic overlay, constituted by this last single node, for the same topic. We then observed the time it took to the partition recovery mechanism to detect the partition.

Figure 3.5 reports the results for test conducted varying the number $|G|$ of nodes initially subscribed to the topic. For each value of $|G|$ we plotted both simulation results (solid line) and expected values calculated with the formula (dashed line). The results show how much harder is to detect partitions for less popular topics (i.e. lower values for $|G|$), with respect to highly popular topics. However, the merge probability always tends to 1 as time passes by, regardless of topic popularity. This means that TERRA is able to recover, in a finite amount of time, any topic partition.

3.3.5 Node stress distribution

TERRA is a topic-based publish/subscribe system designed to work in a completely peer-to-peer environment where participants share their resources to be part of the system and use it. One of the most important aspects that must be taken into account in such a scenario is node stress distribution. TERRA in fact uses nodes’ resources,
partly to fulfill their specific needs (for example to notify events to a node that subscribed some topics), and partly to let the system globally works (for example to maintain information updated in APTs or answer to access point lookups). The burden imposed on nodes for this second aspect must be equally subdivided among all participants to avoid the appearance of hot spots.

To test node stress under various possible workloads, we ran tests with different subscription distributions. Tests were run over a system with 10000 nodes. We issued at the beginning 200000 subscriptions distributed in various ways on 10000 distinct topics, and then diffused one event for cycle during the whole simulation duration. Events were distributed uniformly over topics, while subscriptions were distributed using both uniform and zipf distributions to test the ability of TERRA to evenly distribute node stress even when topic popularity distribution is skewed.

We measured the fraction of messages handled by each node, excluding messages generated at the upper level of the architecture, i.e. messages generated within topic overlays. This measure should represent the fraction of overhead imposed on each node only for TERRA’s management tasks.

Figure 3.6(a) shows the results for an initial uniform distribution of subscriptions on topics, while figure 3.6(a) show the same results for an initial zipf distribution with parameter $a = 0.7$. The plots show the complete absence of hot spots in the system, as management is evenly distributed among all nodes.
3.3.6 Shuffle avoidance

![Shuffle avoidance plot]

Figure 3.7: The plot shows how the probability to avoid a shuffle for a topic overlay increases with time (cycles). The probability is strongly affected by the number $|G|$ of nodes subscribed to the topic.

In section 3.2.3 we introduced the shuffle avoidance mechanism whose purpose is to reduce overhead generated at the topic overlay level to exchange views. The mechanism leverages information on nodes obtained at the general overlay level to feed topic overlay views.

We experimentally evaluated the probability to save a shuffle for a topic overlay varying the popularity of the relative topic. The shuffle avoidance mechanism, in fact, uses information received through subscription advertisements to feed topic overlay views. The amount of information received about a topic is directly proportional to the popularity of the topic: the higher is its popularity, the higher is the amount of subscriptions related to it that are advertised in the general overlay. Therefore we expect a higher probability to save shuffles for more popular topics.

The experiments were conducted on networks made up of 10000 nodes where a variable amount of subscriptions were issued on a single topic. We measured how much cycles it takes to a node to receive at least $U$ advertisements for the topic from other nodes. $U$ is the number of nodes usually exchanged during a view exchange operation.

Figure 3.7 reports the result of this test. As expected, the probability to avoid a shuffle in a topic overlay strongly depends on the popularity $|G|$ of the topic. $|G|$ in our experiment is expressed as the percentage of nodes in the system subscribed to the topic. When $|G|$ is large a few cycles are sufficient for a node to receive $U$ advertisements for the topic. Nodes subscribed to less popular topics have, instead, a
lower probability to save some overhead through this mechanism.

### 3.3.7 Scalability assessment

The maintenance of TERRA’s two-level infrastructure induces some overhead. In order to assess the global impact of this overhead, we evaluated the average cost incurred by TERRA to notify a single event to a subscriber, namely the total number of generated messages divided by the number of notifications. This cost includes both messages generated to diffuse the event, and messages generated for TERRA’s maintenance. To offer a reference figure, we also evaluated the cost incurred by a simple event flooding-based approach\(^2\) in the same settings.

Figure 3.8(a) reports the results when the total number of subscriptions varies between 10\(^2\) and 10\(^6\). The number of topics is fixed and equal to 100. The network considered in this test was constituted by 10\(^4\) nodes, while the event publication rate was maintained constant at 1 event per topic in each cycle. For the evaluation to be meaningful, we required each topic to be subscribed by at least one subscriber; therefore, each curve is limited on its left end by the number of available topics. Moreover, we required each node to subscribe each topic at most once; therefore, each curve is limited on its right end by the number of nodes in the system times the number of available topics (e.g. the curves start from 100 subscriptions and end at 10\(^2\) \cdot 10\(^4\) = 10\(^6\) subscriptions).

The reference cost expressed by the simple event flooding algorithm decreases as the number of subscriptions increases. This behaviour is justified by the fact that the total cost incurred by the algorithm for each event diffusion is constant, regardless of the number of subscriptions (as it only depends on the popularity of each topic). Consequently, increasing the number of subscriptions has a positive impact on the algorithm efficiency: each event broadcast in the overlay network will generate a higher number of notifications.

TERRA’s behaviour is more complicated, as various factors have an impact on its global cost. This global cost is the sum of two contributions: a constant amount and a variable one. The former does not depend on the total number of subscriptions: it corresponds (i) to the cost induced by the overlay management protocol’s view exchange mechanism for the general overlay, and (ii) to the cost induced by the access point lookup mechanism. The latter is proportional to the total number of subscriptions per topic issued in the system, and includes the cost (i) of subscription advertisements, (ii) of the view exchange mechanism for topic overlays, and (iii) of the broadcast service used to diffuse events in topic overlays.

When the number of subscriptions per topic is close to one (on the left end of the curve), the constant part of the total cost is dominant. Therefore, the average notifica-

\(^2\)Each event is broadcast in an overlay network containing all participants. The overlay is built and maintained through the same overlay maintenance protocol employed by TERRA (Cyclon). Also the broadcast mechanism is the same considered in TERRA.
Figure 3.8: The plots show the average number of messages needed by TERRA to notify an event when the number of subscriptions (a), of topics (b), the event publication rate (c) and the total number of nodes in the system (d) varies. For each figure, results from a simple event flooding algorithm are reported for comparison.
tion cost decreases as for the simple event flooding algorithm. On the contrary, when the number of subscriptions per topic increases, the variable part of the cost becomes dominant. Consequently, the average notification cost quickly reaches a lower bound that is defined by the out degree used in the broadcast algorithm (in our experiments we considered an out degree equal to the view size, i.e. 20).

As expected, TERRA and the event flooding protocol have a comparable behavior when the number of subscribers per topic is close to the total number of nodes. Indeed, in such case, each node is subscribed to every topic; therefore, it is interested in every event published in the system, making differences between the two approaches negligible.

Figure 3.8(b) reports the same test, ran varying the amount of topics and maintaining a fixed number of subscriptions ($10^4$). In this case, the algorithm’s behavior is dual with respect to the previous figure: a higher number of topics increases the load for simple event-flooding (because it causes each generated event to be matched by a smaller number of subscribers), while TERRA’s performance remain almost unchanged.

Figure 3.8(c) reports the same test when the number of subscriptions and topics is kept constant (100 topics and $10^4$ subscriptions), while the event publication rate per topic varies between $10^{-3}$ and $10^3$. The plots show a clear tradeoff: when the event publication rate is very low, the higher overhead caused by TERRA is not compensated by the advantages induced by traffic confinement. Nevertheless, these advantages comes into play as soon as the event publication rate raises. This result confirms TERRA’s ability to better scale in high load settings.

Finally, figure 3.8(d) reports how TERRA scales with respect to the number of nodes in the system. This test has been run in a scenario where $10^4$ subscriptions are uniformly distributed over 100 topics, and events are published with a rate of 1 event per topic at each cycle. The number of nodes varies between 100 and $10^9$. The curves show that TERRA gracefully scales as the number of nodes increases, up to a point after which the overhead due to view exchanges in the general overlay becomes dominant and is no longer compensated by event notifications (that only depends from the constant amount of subscriptions).

3.4 Related Work

Publish/subscribe systems based on peer-to-peer architectures have been introduced a few years ago with the development of topic-based event notification services built on top of Distributed Hash Tables (DHTs). SCRIBE [35] and Bayeux [141] are two pub/sub systems built on top of two DHT overlays (namely Pastry [110] and Tapestry [139]), which leverage their scalability, efficiency and self-organization capabilities. Systems like SCRIBE use the decoupled key/node mapping provided by the DHT to efficiently designate a rendez-vous node for each topic (see section 1.5.2 for more de-
tails on this technique). This node is responsible for collecting each event published for that topic and diffuse it toward the subscribers.

An interesting variant of this technique was proposed in [104]: members of the system subscribed to the same topic form a separate overlay where events belonging to the corresponding topic are simply flooded. From this point of view the two-layer architecture proposed with TERRA is very similar to the architecture of [104]. However, in [104] a single access point exists for each topic overlay and simple DHT routing is used to reach it. TERRA does not impose a single access point for each topic overlay thus avoiding issues related to traffic hot spots and single point of failures, but rather makes every node subscribed to a topic a possible access point.

Unstructured peer-to-peer systems were introduced as a substrate for topic-based event dissemination in [4]. The system proposed in that work maintains, through the widespread use of probabilistic algorithms, a hierarchy of groups that directly maps a topic hierarchy. Each group contains nodes subscribed to a specific topic and is maintained through a probabilistic membership protocol [73]. The lack in [4] of a general overlay network, not related to any specific topic, means that every publisher, before publishing an event, must became part of the group corresponding to the topic it wants to publish in. This also means that nodes playing the role of simple publishers receive events they are not subscribed to. Publishers in TERRA are not required to join any topic overlay before publishing events; they are part of the general overlay, and leverage it to diffuse events they produce. In this way the amount of spam messages can be greatly reduced.

Recently an interesting work appeared where an unstructured overlay network is used to implement a content-based publish/subscribe system: Sub-2-Sub [131]. In Sub-2-Sub subscribers sharing the same interests are clustered in ring-shaped overlay networks through a self-organizing algorithm that continuously analyzes overlapping intervals of range constraints. Nodes publishing events try to reach one of the target subscribers leveraging overlapping-interest links maintained by a proximity-based epidemic protocol that keeps connected nodes sharing intersecting subscriptions. When an event reaches a target subscriber it is diffused efficiently in the correct ring overlay. Information about subscriptions is periodically exchanged using a node sampling service realized with [129]. Sub-2-Sub is a system designed for content-based event diffusion, therefore it is in some sense more general than TERRA, as topic-based event diffusion can be seen as a specific case of content-based event diffusion. However, this comes at the cost of an higher complexity. Moreover, Sub-2-Sub clustering mechanism can quickly lead to a large number of ring overlays when range intervals defined in subscriptions are uniformly distributed in the attribute space (increasing the number of reciprocal interval intersections); a direct consequence of this problem is an higher overhead imposed on nodes: the number of rings a node participates to is not necessarily proportional to the number of subscriptions it manages, but it is rather linked to the number of intersections its subscriptions share with subscriptions hosted on other nodes. From this point of view it is also possible that
a node managing a subscription that spans a large part of the attribute space will maintain a lot more links than a node managing a "very selective" subscription.
Chapter 4

Proximity-driven Event Routing in Mobile Ad-hoc Networks

A Mobile Ad Hoc Network (MANET) is a collection of wireless mobile devices that are able to communicate and move at the same time through dynamic wireless links [125]. Neither pre-existing infrastructures nor centralized administration functions are required so that self-organization and adaptiveness are important properties. MANETs represent a concrete example of support for pervasive computing.

One of the main issues in MANETs is how to provide the application layer with communication abstractions suitable for the very dynamic nature of the underlying communication network. Content-based publish/subscribe, thanks to its decoupling properties, is a very appealing candidate for such dynamic contexts.

Well known techniques for event diffusion cannot be easily adapted to MANETs due to the highly dynamic nature of these networks. Solutions that implement event filtering relying on complete subscription information stored over a static overlay network would require an enormous effort to maintain this information up-to-date despite frequent topological changes due to node movements.

In this chapter we explore a different strategy, whose key aspect is the lack of any predefined logical infrastructure as a support to event filtering. We realize a distributed implementation of the event notification service by running a broker on each mobile node of the MANET but we do not try to maintain a fixed overlay dispatching network connecting them. Conversely, we leverage the broadcast primitives available in a MANET to forward events, letting each receiving broker to autonomously decide if and when re-forwarding an event on the basis of an estimation of its proximity to potential subscribers for that specific event. In particular, we use the time elapsed since two nodes went out from each other’s transmission range as an estimate of their distance. As we will formally show in section 4.3, within a given time interval this gives a good approximation of the distance metric between two nodes.

This fact, together with the efficiency of the implicit and indirect forwarding
method adopted, results in an effective and scalable event filtering mechanism in the scenarios we consider, as proven by the results of the simulations we ran.

The rest of this chapter is organized as follows: section 4.1 briefly motivates our work and gives a general description of the routing protocol we propose, while section 4.2 provides the details of the protocol. In section 4.3 we formally analyze the relationship between the elapsed time and the distance between the brokers, while section 4.4 presents the results of an extensive simulation study, which validates our approach. Finally, section 4.5 discusses related work.

4.1 Proximity-Driven Routing: An Overview

Event routing based on a distributed set of brokers interconnected in an overlay dispatching network is hard to implement efficiently in a MANET due to the cost required to cope with the frequent changes in the topology of the physical network.

To succeed in a MANET, and particularly in those including fast moving nodes, a publish/subscribe protocol should not require any predefined network-wide structures. Starting from this observation we developed a diffusion protocol, dubbed proximity-driven routing protocol, whose general concepts are described in this section. Details are given in the next section.

4.1.1 Assumptions

In our description we assume that the MANET is composed by $N$ mobile nodes, each running a broker. When necessary, to stress the difference between nodes and brokers, we will use the notation $n_i$ to indicate the $i$-th mobile node of the network, and $B_i$ to refer to the broker running on that node.

Broker $B_i$ acts as an entry point to the event notification service for every application running on node $n_i$. When an application running on a node $n_i$ wants to receive some events, it issues a subscription on $B_i$, which then stores the predicate associated with the subscription into its subscription table. Similarly, to publish an event, an application running on a node $n_i$ sends it to the broker $B_i$.

The protocol does not rely on any network layer protocol; rather it only assumes the availability of a local broadcast communication primitive, which allows a node to unreliable send a message to all its one-hop neighbors via a single transmission, a fundamental and always satisfied assumption in MANETs.

Finally, we assume that the interests of all the application components connected with a broker $B_i$ can be condensed into a single predicate, which reflects the content of $B_i$’s subscription table.$^1$

$^1$Note that this assumption is realistic for content-based publish-subscribe systems whose subscription language is usually powerful enough to allow it.
4.1.2 Some General Considerations

To develop our protocol we started by observing that if each broker knew the Euclidean distance of its neighbors from the recipients of an event (i.e., the target subscribers) as well as its own distance from them, then the event could be forwarded by letting a broker $B_i$ send it only to the neighbors closer than itself to the subscribers.

To put such principle in practice we must solve three problems:

1. how to calculate the list of target brokers for an event;
2. how a broker determines its distance from the others;
3. how a broker determines the set of neighbors that should process the event.

To calculate the list of target brokers for an event, a broker should know the subscriptions issued by all other brokers in the network, a solution that is hardly reasonable. Consequently, we decided to relax this requirement by collecting information about subscribers as the event is being forwarded and appending them as control information in the event.

To determine the distance between two brokers, we excluded the use of a location service support, e.g., based on positioning devices like GPS, and decided to measure it by exploiting the time elapsed since two brokers were most recently adjacent (i.e., in direct communication range). This estimation technique is very simple (a beacon signal is sufficient for this purpose) and reasonably accurate, provided that the elapsed time is not too long. Section 4.3 further elaborates on this issue.

Finally, since we want to keep any routing decisions as simple and “distributed” as possible, we decided to abandon the idea of each broker collecting the distance information from its neighbors and using it to explicitly choose the set of neighbors that have to further forward an event. Conversely, we adopted an implicit routing mechanism, which uses broadcast communication to reach every neighbor, letting them autonomously decide if re-sending the event or not, based also on the observed behavior of other neighbors. In particular, this realizes a store, delay, and cancel-or-forward approach which represents an efficient technique for routing: (i) it exploits the broadcast nature of the wireless transmissions medium to send multiple copies of the same message via a single transmission, (ii) it eliminates the need of collecting and maintaining information (i.e., distance data) about neighboring brokers, (iii) it avoids the burden of link breakage detection and, finally, (iv) it provides an intrinsic resilience to the topological changes caused by nodes mobility.

4.1.3 The Protocol

Let now consider how the basic event forwarding scheme works. Each broker $B_i$ periodically broadcasts in its transmission range a beacon message containing the predicate that summarizes its own subscription table. A broker $B_j$, adjacent with $B_i$,
receives this message and stores the predicate together with the time it received the beacon into a *proximity table*. This mechanism allows each broker to determine the time elapsed since it lost contact with any other broker. This time value is then used to calculate the *proximity value* (or simply “proximity”) $p_{ji}$ of $B_j$ with respect to $B_i$.

The proximity value, which can be viewed as an estimate of the distance of $B_j$ from $B_i$, is the basis of our probabilistic event filtering method.

Each event $e$ carries a *destination list*: the (estimated) list of brokers interested in receiving the event, each coupled with the lowest proximity computed by the brokers that forwarded the event so far. As an example, the destination list of an event $e$ includes a couple $<i, p>$ if broker $B_i$ is known to be interested in receiving the event (i.e. $e$ matches a subscription issued by some subscriber attached to $B_i$) and $p$ is the lowest proximity from $B_i$ calculated by all the brokers that forwarded $e$. The event has also a unique network-wide identifier provided by the source broker, we will refer to it with the notation $e.id$.

Suppose now that at time $t$ broker $B_i$ receives the event $e$ for the first time. It will resend the event if (i) it is aware of some new broker not mentioned in the destination list carried by $e$ or (ii) its proximity table holds for some broker $B_k$ a proximity lower than that associated to the same broker $B_k$ into $e$’s destination list.

Such a condition is in general not sufficient to trigger the actual transmission of the event. The broker $B_i$, in fact, schedules the transmission of the event after a delay proportional to $p_{ik}$ (the lowest value is considered if such a condition holds for more than one broker, see later). If during such a time interval it doesn’t hear the same event (i.e. an event with the same identifier) again, then the transmission will take place. Otherwise $B_i$ silently drops the event. The rational behind this decision is to

\[ \text{Note that, if we assume all transmission ranges to be equal, then } p_{ij} = p_{ji}, \text{ because when } B_i \text{ is in } B_j \text{’s transmission range, the opposite is also true.} \]
avoid that two adjacent brokers will forward the same event and also to let brokers closest to some destination to “suppress” transmission of adjacent brokers that are farther from the destination.

In order to clarify this mechanism, let us consider figure 4.1, which shows a set of nodes together with their transmission ranges (the dotted circle surrounding each node). Suppose $B_0$ publishes an event matching $B_4$’s subscriptions. The event is sent via broadcast and received both by $B_1$ and $B_2$. Assume that $B_0$ and $B_4$ have never came in contact so that the destination table carried by $e$ is initially empty. Assume that $B_2$ missed $p_{24} = 5$ beacons from $B_4$. The broker $B_2$ schedules the transmission with some delay proportional to 5. However, $B_1$ is adjacent to $B_4$ (i.e., $p_{14} = 0$) and immediately sends $e$. Broker $B_2$, on receiving the event from $B_1$ aborts the scheduled transmission and silently drops $e$. Moreover, since the proximity carried by the $e$ sent by $B_1$ is zero, the broker $B_3$ ignores the event (by definition zero is the lowest possible proximity).

### 4.2 Proximity-Driven Routing: Protocol Details

The pseudo-code of our Proximity-Driven Routing protocol, is reported as algorithm 7.

Each broker maintains the following data structures:

- A subscription table that holds information about the subscriptions issued by applications running on the same node of the broker. This table is organized as an array $st$ of pairs $⟨pred, id⟩$, where $pred$ is the predicate carried by a subscription and $id$ is the identifier of the component that issued the subscription.

- A proximity table organized as an array $pt$ of triples $⟨id, pred, time⟩$, where $id$ is the identifier of a broker, $pred$ is the predicate received from that broker, which summarizes its subscription table, and $time$ is the time when the predicate was received.

Every $\Delta T$ seconds each broker $B_i$ beacons a summary of the predicates stored into its subscription table using a broadcast packet (lines 1-4). The size of this summary depends on the number of subscription stored on the node and on the language used to express them. Various techniques can be used to reduce its size, like subscription covering [86] or bloom filters [24], at the cost of more complex algorithms or approximated message routing. In the rest of this chapter we assume for simplicity that the whole subscription set stored at a broker is included in each of its beacons.

A broker $B_j$ that is within the transmission range of $B_i$ receives such a beacon and executes the corresponding handler of algorithm 7 (lines 5-10) to update its proximity table. If the same predicate was already received from the same broker, then the entry is refreshed, i.e. the time associated to the entry is set to the current time. Otherwise
Algorithm: 7 - Hint-Driven Routing Protocol

1. Every $\Delta t$ time units do
2. $p \leftarrow \text{summarize}(st)$
3. broadcast($\text{PREDICATE}[p]$)
4. cleanup($pt$)
5. On receive $\text{PREDICATE}[p]$ from $n$ do
6. \hspace{1em} if $\exists k$ s.t. $pt[k].id = n$ then
7. \hspace{2em} $pt[k].pred \leftarrow p$
8. \hspace{2em} $pt[k].last \leftarrow \text{currentTime}$
9. \hspace{1em} else
10. \hspace{2em} append($pt, (n, p, \text{currentTime})$)
6. On receive $\text{PUBLISH}[e]$ from $n$ do
12. \hspace{1em} if event $e'$ s.t. $e'.id = e.id$ was already received then
13. \hspace{2em} de-schedule transmission of $e'$
14. \hspace{1em} else
15. \hspace{2em} for all $(pred, id) \in st$ do
16. \hspace{3em} if $e \odot pred$ then
17. \hspace{4em} $e.setProximity(myId, 0)$
18. \hspace{4em} notify($e$)
19. \hspace{2em} minHint $\leftarrow 1$
20. \hspace{2em} matched $\leftarrow$ false
21. \hspace{2em} for all $(id, pred, last) \in pt$ do
22. \hspace{3em} $h \leftarrow \text{hintFor}(id)$
23. \hspace{3em} if $e \odot pred$ and $(id \notin \text{destinationOf}(e)$ or $h < e.getHint(id))$ then
24. \hspace{4em} matched $\leftarrow$ true
25. \hspace{4em} $e.setHint(id, h)$
26. \hspace{4em} if $\text{minHint} > h$ then
27. \hspace{5em} $\text{minHint} \leftarrow h$
28. \hspace{4em} if matched = false and $e.credit > 0$ then
29. \hspace{5em} $e.credit \leftarrow e.credit - 1$
30. \hspace{5em} matched $\leftarrow$ true
31. \hspace{2em} if matched = true then
32. \hspace{3em} schedule $e$ for transmission with a delay proportional to $\text{minHint}$
4.2. PROXIMITY-DRIVEN ROUTING: PROTOCOL DETAILS

Figure 4.2: An example of event routing.

a new element is appended to the table. After a timeout, experimentally set to $10\Delta T$ seconds, entries are deleted from the table (procedure cleanup in algorithm 7 line 4). In other words, information about brokers for which more than 10 beacons have been missed, are dropped. This reflects the general intuition, also confirmed by the model provided in the next section, that too large proximity values are not correlated with the effective distance between brokers.

The information stored in the proximity table, together with the fact that the beacon interval $\Delta T$ is globally known, allow each broker $B_i$ to calculate the proximity value $p_{ij}$ at time $t$ with respect to any other broker $B_j$ as follows: $p_{ij}$ is infinite if $B_j$ is not present into $B_i$’s proximity table; otherwise it is a value in the range $[0..1]$ calculated as the number of $B_j$’s beacons missed by $B_i$ divided by 10.

Remembering from previous section that each event carries a unique identifier and a destination list composed of couples $\langle id, proximity \rangle$, we can describe how event forwarding proceeds (lines 11-22). On receiving an event $e$ a broker checks if the same event, i.e., an event with the same identifier, has been received before. If this is the case, the event is removed from the list of events scheduled for transmission (if present) and it is dropped without any further processing.

If $e$ was never received before then the broker checks if it matches some predicate into its subscription table. If this is the case, the broker notifies $e$ to the corresponding subscriber and set the proximity for itself into $m$’s destination list to 0. This step will avoid to trigger further transmissions aiming at hitting the broker, as clarified next. Moreover, the broker determines if it has to re-forward the event. This happens when $e$ matches at least a predicate advised by a broker $B_i$ such that: (i) $B_i$ doesn’t belong to the destination list of the event or (ii) the proximity value for $B_i$ computed by the receiving broker according to its proximity table is strictly lower than the one carried into the event.

In both cases the retransmission of $e$ is scheduled after a delay proportional to the

---

3This check can be easily accomplished by remembering the identifiers of the events received so far.
proximity for \( B_i \) owned by the receiving broker. When more than one broker exits that satisfies the conditions above, the delay is determined by the lowest proximity.

If none of the above cases holds, \( e \) should be dropped, but in order to increase delivery at the price of an increased reliability, a new chance is given to the event of being forwarded. To this end, an event also carries an integer value, called the \textit{credit} of the event, which represents the number of times a broker can force the retransmission of the event despite the fact that the conditions stated above about proximity do not hold. As shown in algorithm 7 (lines 28-30), if such a case occurs, the event is scheduled for transmission with the delay associated to the maximum possible proximity value, i.e. one. This way, forwarding due to credit tends to be cancelled by forwarding due to proximity.

Figure 4.2 portraits an example of event forwarding. The proximity table of a node is reported close to the node itself, while events show the destination list they carry. For the sake of simplicity instead of storing the absolute time when the node received a beacon message, the last column of the proximity table stores the proximity value computed as explained above.

Suppose broker \( B_0 \) generates an event matching subscriptions on brokers \( B_3, B_5, \) and \( B_7 \). The source node \( B_0 \) is only aware of the subscriptions \( S_3 \) at node \( B_3 \), for which it holds a proximity of 0.9. It then forwards the event with destination list \( B_3: 0.9 \). On receiving the event, broker \( B_1 \) decides to forward it. Indeed, it knows another broker, broker \( B_5 \), whose subscriptions are matched by the event. Moreover, the proximity for \( B_3 \) calculated by \( B_1 \) is lower than 0.9. Brokers \( B_2 \) and \( B_4 \) both receive the event sent by \( B_1 \). Broker \( B_2 \) re-forwards the event since it calculates a proximity 0.0 for \( B_3 \). Similarly, broker \( B_4 \) re-forwards the event because it is aware of new broker \( B_7 \) and also has a proximity for \( B_5 \) (0.4) lower than that included into the destination list of the received event (0.6). Finally, broker \( B_6 \) re-forwards the event since it calculates two proximity values for \( B_7 \) and \( B_5 \) lower than those included in the received event (0.0 against 0.5 and 0.4, respectively).

### 4.3 Elapsed time distance correlation

The proximity-driven protocol assumes that the time elapsed since two nodes were most recently adjacent to each other is a measure of the chances of the two nodes being still close to each other, i.e., the lower the time the closer the nodes. In this section we provide a stochastic model that supports this claim.

In particular, we developed a mobility model discrete in time and space, which captures the main short term characteristic of a physical movement and calculates the conditional expected distance between two nodes, given the time elapsed since they lost connection. A similar analysis has been carried out in [9] for a Manhattan-like topology. The model presented here is more general as it removes the constraints on movements, i.e. a node can move in any direction, including diagonals.
4.3. ELAPSED TIME DISTANCE CORRELATION

Figure 4.3: A $14 \times 14$ grid ($A = 7$) with an example of relative position and transmission area for $L = 4$.

The field we consider (see figure 4.3) is a $2A \times 2A$ two-dimensional grid, wrapped along both directions, i.e., forming a torus. Two nodes move in this field by jumping from one point of the grid to an adjacent one. Fixing the coordinates system on one of the two nodes allows to express the position of the other as $P = (x, y)$, where $x, y \in [0, 2A)$ are the coordinates of node measured along the two axes. Analogously, we define the distance between the two nodes as the norm: $||x, y|| = \max\{\min|x, 2A - x|, \min|y, 2A - y|\}$. We assume that a wireless link exists between the two nodes when $||x, y|| < L$. This means that the transmission area is a square with edge large $2(L - 1) + 1$ points. In the previous figure $L = 4$.

Movements change the relative position of the two nodes along the two axes independently from each other. The change can be at most one grid point, in each direction, so that the relative movement of the two nodes can be described by a pair $M = (m_x, m_y)$, where $m_x, m_y \in \{-1, 0, 1\}$. Movements occur at regular discrete time ticks. Hence, if $P$ is the current relative position of the two nodes and $P'$ was the previous one, we can write $P = P' \otimes M = (x' \otimes m_x, y' \otimes m_y)$, where we define $a \otimes b = (a + b) \mod 2A$ if $a + b \geq 0$ and $2A + (a + b)$ otherwise. According to this formula, movement at time $t$ determines the position of the node at time $t$ from its previous position at time $t - 1$.

To model realistic patterns of mobility while keeping the analysis tractable, we assume that the current movement, $M$, only depends on the previous one $M'$, according to the following rules:
a similar relationship also holds between $m_y$ and $m'_y$.

These are the transition probabilities of the Markov chain portrayed in figure 4.4; $\alpha$ can be interpreted as the probability that the relative position of the two nodes along one axis doesn’t change given that it was also unchanged in the previous time tick, while $\beta$ represents the probability that the movement along one direction repeats again. Clearly $\alpha$ and $\beta$ have to be less than 1. Also note that to take into consideration only smooth changes in the movement direction we do not allow $m_x$ and $m_y$ to change directly from 1 to $-1$ and vice-versa.

The average value of the module of the relative speed of the two nodes along an axis, denoted by $\xi$, is given by the stationary probability $\pi_1$ ($= \pi_{-1}$) of the chain:

$$
\xi = \frac{1 - \alpha}{1 - \alpha + (1 - \beta)} = \frac{1 - \alpha}{2 - (\alpha + \beta)}
$$

Let $D(k)$ be the distance at time $k$ between the two nodes. Its expected value depends on the initial “state” $s_0 = (x_0, y_0, m_{x0}, m_{y0})$ observed just after the wireless link between the nodes broke. Specifically, if $Pr(s_0)$ is the probability of observing the state $s_0$, we can write:

$$
E[D(k)] = \sum_{s_0} E[D(k)|s_0] Pr(s_0)
$$

From our model of connectivity we have that the position after a breakage must belong to the square with edge $L$. Similarly, given our definition of movement, $M$, we
4.3. **ELAPSED TIME DISTANCE CORRELATION**

have that $m_{x_0}$ and $m_{y_0}$ cannot be arbitrary numbers but must be compatible (see later) with current position of the two nodes and the event of link breakage. In particular, due to the symmetry of our model, we can divide the square with edge $L$ into eight equivalent segments, $A_1, .., A_8$, as done in figure 4.5. They are equivalent in the sense that the conditional expected distance given the node exits from segment $A_i$, say $E[D(k)|s_0 \in S_i]$ is the same for any segment ($S_i$ being the set of possible states just after the link is broken due to the node exiting from segment $A_i$). Moreover, a node has the same probability of exiting from any segment. Hence:

$$E[D(k)] = \sum_{i=1}^{8} E[D(k)|s_0 \in S_i]Pr\{s_0 \in S_i\}$$

$$= \frac{1}{8} \sum_{i=1}^{8} E[D(k)|s_0 \in S_i]$$

$$= E[D(k)|s_0 \in S_1]$$

To compute $E[D(k)|s_0 \in S_1]$ we first observe that $(x, y, m_x, m_y)$ can be an initial state, i.e. it belongs to $S_1$, only if the following conditions hold: (i) $x = L, y = 0..L$ and (ii) $m_x = m_y = 1$ if $y = L, m_x = 1, m_y = \in [0, 1]$ if $y = L - 1; m_x, m_y = \in [-1, 0, 1]$ if $y < L - 1$. We say that a movement $(m_x, m_y)$ is compatible with the position $(x, y)$ if it satisfies the above conditions; they are depicted as small arrows near the possible exit positions in figure 4.5.

If we call $C(x, y)$ the set of movements that are compatible with the position $(x, y)$ we can write:
\[
Pr((x, y, m_x, m_y)) = Pr((x, y))Pr((m_x, m_y))Pr((m_x, m_y) \in C(x, y))
\]

\[
= \frac{1}{L + 1} \sum_{(m_x, m_y) \in C(x, y)} Pr(m_x)Pr(m_y)
\]

Let now \( P_{s_0}(x, y, m_x, m_y, k) \) be the probability that at time \( k \) the position is \((x, y)\) and the movement is \((m_x, m_y)\), given that at all previous times \( k' < k \) the position \((x', y')\) was such that \(||x', y'|| \geq L\) (i.e. the nodes never established the link again) and that at time \( k = 0 \) the state was \( s_0 \). We can write the following recurrent equation:

\[
P_{s_0}(x, y, m_x, m_y, k + 1) = \\
\frac{\sum_{(x', y', m'_x, m'_y) ||x', y'|| \geq L, |m'_x| \leq 1, |m'_y| \leq 1} P_{s_0}(x', y', m'_x, m'_y, k)Pr(m_x|m'_x)Pr(m_y|m'_y)}{
\sum_{(x', y', m'_x, m'_y) ||x', y'|| \geq L, |m'_x| \leq 1, |m'_y| \leq 1} P_{s_0}(x', y', m'_x, m'_y, k)}
\]

hence:

\[
E[D(k)] = E[D(k)|s_0 \in S_1] \\
= \sum_{s_0 \in S_1} Pr(s_0) \sum_{||x, y|| = \psi, |m_x| \leq 1, |m_y| \leq 1} \psi P_{s_0}(x, y, m_x, m_y, k)
\]

Figure 4.6: Expected distance as a function of elapsed time after a link breakage, \( A = 20, L = 10 \).

In figure 4.6 we report the expected distance \( D(k) \) obtained by numerically solving the previous equation, as a function of \( k \) and with the speed \( \xi \) given as a parameter.
4.4 Evaluation

To assess the performance of our proximity-driven routing protocol we simulated it using J-Sim [60], an open-source network simulator that provides a complete implementation of the 802.11 protocol stack as well as a fairly complete and detailed signal propagation model. Simulations allow us to test the performance of our protocol in very large and complex scenarios, involving hundreds of mobile nodes, something very hard to achieve using real devices.

The main figures we measured were delivery ratio and notification cost. In a publish/subscribe application, the former is defined as the average ratio between the number of subscribers that received an event and the total number subscribers interested in that event. As for the notification cost, it is defined as the average number of link-layer packets generated for each delivered event (including the beacon packets generated by our protocol). Note that, differently from the definition we used in sections 2.3.2 and 3.3.7, in this case we evaluate the notification cost at the link layer due to the fact that our event routing mechanism works at this level of the protocol stack.

As a baseline to evaluate the performance of our protocol, we used a pure gossip protocol, which represents the simplest structure-less approach for publish/subscribe event diffusion. In the gossip protocol we considered, brokers delivers events using the broadcast facility provided by the MAC layer (i.e., 802.11 in our simulations), adopting a forwarding probability \( p \in (0, 1] \). This means that the broker running on the same node of the publisher, i.e., the first one forwarding the event, delivers it using an 802.11 broadcast packet, while the receiving brokers, independently from the content of their subscription tables, re-forwards it with a probability \( p \in (0, 1] \).

4.4.1 Simulation Settings

The reference scenario we considered is that of a MANET composed by a number of nodes dispersed in a square field, which move around according to a random waypoint mobility model [71]. Each node randomly chooses a destination and starts moving toward it at a random speed. Once the destination has been reached, the node randomly determines another destination, and continues in that direction with a new randomly chosen speed.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>( N = 100 )</td>
</tr>
<tr>
<td>Field area</td>
<td>( A = 1000 \times 1000 \text{ m}^2 )</td>
</tr>
<tr>
<td>Minimum speed</td>
<td>( S_{\text{min}} = 10 \text{ m/s} )</td>
</tr>
<tr>
<td>Maximum speed</td>
<td>( S_{\text{max}} = 20 \text{ m/s} )</td>
</tr>
<tr>
<td>Number of publishers</td>
<td>( N_p = 2 )</td>
</tr>
<tr>
<td>Publishing rate (per publisher)</td>
<td>( Pr = 0.5 \text{ msg/sec} )</td>
</tr>
<tr>
<td>Number of subscribers</td>
<td>( N_s = 10 )</td>
</tr>
<tr>
<td>Beacon interval</td>
<td>( \Delta t = 5 \text{ sec} )</td>
</tr>
<tr>
<td>Credits</td>
<td>( Cr = 0 )</td>
</tr>
<tr>
<td>Forwarding probability</td>
<td>( p = 0.5 )</td>
</tr>
</tbody>
</table>

Table 4.1: Default simulation parameters.

The total number \( N \) of nodes, the area \( A \) of the field, and the minimum \( S_{\text{min}} \) and maximum \( S_{\text{max}} \) speed nodes can move at are the main physical parameters that characterize the simulated scenario.

As for the wireless protocol, we used the 802.11 network model provided by J-Sim. To reflect a realistic open field scenario, we choose a two rays ground propagation model with a random transmission range varying between 100 and 200 meters.

A broker runs on each node and it has either a single publisher, or a single subscriber attached to it, or it acts as a pure forwarder. We assume that \( N_p \) publishers produce events of interest for \( N_s \) subscribers at a publishing rate of \( Pr \text{ msg/s} \). These parameters characterize the publish/subscribe application model.

Finally, the main parameters that characterize our protocol are the beaconing interval \( \Delta t \) and the number of credits \( Cr \) initially assigned to each event. Similarly, the gossip protocol we use for comparison is characterized by the probability of re-forwarding \( p \).

Unless otherwise stated all the simulation parameters above assume the default values listed in Table 4.1.

### 4.4.2 Simulation Results

We first tested the gossip protocol in our reference scenario (see table 4.1) varying the forwarding probability \( p \). Results are reported in figure 4.7. It is worth observing how the delivery exhibits the typical bimodal behavior of gossip protocols [137]. We also note how 100% delivery is never reached due to collisions at the MAC layer and network partitioning, while a reasonable percentage of delivery, say more than half of the number of interested subscribers, can be achieved at the cost of at least 5 packets per delivered event.

Figure 4.8 shows the performance of our protocol as a function of the number of credits under the same reference scenario. Although the maximum delivery is slightly lower than the one obtained by gossip, reasonable high values can be reached at a much lower cost. As an example, without using any credit, our protocol delivers 70%
of the events published using less than a half packets with respect to gossip (respectively 2.5 and 5.5 packets per delivered event). Gossip shows the same notification cost of our protocol with zero credits (i.e., 2.5 packets per delivered event) while delivering only 30% of the published events. By increasing the number of credits the delivery can be increased while still keeping a high convenience with respect to gossip.

The next point to evaluate is how the number of subscribers affects the protocol’s performance. Figure 4.9 shows the delivery and notification cost as a function of the number of subscribers, measured with various credits. The performance of the gossip protocol are also reported varying the gossip probability. It is interesting to note the effectiveness of the credits mechanism as a way to increase the delivery, especially when the number of subscribers is low. Our protocol is always able to assure a high delivery ratio (more than 85%) independently of the number of subscribers and at
progressively decreasing cost. As expected, the efficiency of the gossip protocol increases with the number of subscribers.

Figure 4.9: Effect of increasing the number of subscribers on delivery (a) and notification cost (b).

Another parameter that could impact performance is the rate of published events. As shown in figure 4.10, our protocol is only very marginally influenced by this parameter, while gossip is much more sensible, especially when \( p \) increases. This can be explained by remembering, from previous simulations, that gossip loads the network much more than our protocol. As a consequence, when the publishing rate increases, gossip suffers from a relevant number of collisions, which do not occur when our protocol is used. It is worth noticing that an increase in the publishing rate also decreases the notification cost of our protocol. To understand this behavior we have to remember that as part of the notification cost we also count the beacon packets produced by our protocol. The number of such packets is fixed (it depends on the beaconing inter-

Figure 4.10: Effect of increasing the publishing rate on delivery (a) and notification cost (b).
val, only) and it represents a large fraction of the notification cost when a few events are published. When the number of events published (and consequently received) grows, this contribute to the notification cost becomes negligible.

Figure 4.11: Delivery (a) and notification cost (b) versus speed at different beaconing intervals with our proximity-driven protocol.

The next step is the study of how mobility affects the performance of our protocol. Figure 4.11 shows how delivery and notification cost change when the speed of nodes increases, showing also the impact of adopting different beaconing intervals (i.e., 2.0, 4.0, and 6.0 seconds) at different speeds. At first one could think that a shorter beaconing interval would provide the best performance (not considering the impact on the notification cost). The number of missed beacons, in fact, is the key parameter we use to estimate the distance between brokers, and ultimately to guide our protocol. A shorter beaconing interval should provide the best accuracy and consequently the best performance. On the other hand, we have to remember that in order to make sure that the “missed-beacons to distance” correlation is valid for the entries stored in the proximity table, we delete them after 10 missed beacons. As a consequence, under low mobility a short beaconing interval results in removing valid entries from the table (i.e., those for which the correlation is still valid). Conversely, under a high mobility degree a long beaconing interval does not provide enough accuracy. This explains the results in figure 4.11. The same beaconing interval cannot account for every situation. Each range of speeds has an “ideal” beaconing interval, while other choices reduce performance.

To analyze how our protocol performs when the size of the network grows, and to compare it against gossip, as before, we varied the number of nodes in the network, while keeping their density constant, i.e., by also increasing the size of the simulated area. Given the high impact on protocols’ performances when the density of subscribers changes, as shown in figure 4.9, we fixed the percentage of subscribers with respect to the total number of nodes $N$ to 10%. For the same reason we fixed the percentage of publisher to 2% of $N$ and we kept the publishing rate (per publisher)
constant. As shown in figure 4.12 both our protocol and gossip scale very well, with gossip decreasing slightly its delivery as the network size grows, and our protocol marginally increasing it.

Figure 4.12: Delivery (a) and notification cost (b) as network size grows.

The second scalability test we ran, see figure 4.13, consists in increasing the number of nodes \( N \) while keeping the area \( A \) constant, hence producing an increase in the node density. Here, we observe an interesting phenomenon, which is due to the increasing number of collisions: a low gossiping probability provides better performance as the density increases, while using a higher probability performance starts decreasing after a given number of nodes. Our protocol is much more resilient to collisions because of the suppression mechanism it uses, which can be considered a form of auto-adaptation to the density of the network. Here, as usual, the notification cost of our protocol is far better than gossip and is rather constant with respect to the increasing density of nodes.

Figure 4.13: Delivery (a) and notification cost (b) as node density increases.
4.5 Related Work

The last decade saw the development of several publish/subscribe middleware platforms, which embody different routing algorithms to implement content-based publish/subscribe facilities for fixed networks (for a detailed comparison see [52, 29, 44, 109]).

At the same time, in the last few years we have seen a continuous growth in the dynamism of networks, motivated by peer-to-peer overlays and wireless networking, which originated a number of challenging issues for routing in general and for content-based routing in particular. Some work addressed the problem of failures in tree-based publish/subscribe systems, optimal from a pure traffic standpoint in stable networks, with the addition of advanced tree maintenance mechanisms for both fixed peer-to-peer networks [98], and mobile ad-hoc networks [85]. The results obtained in these works convinced us that tree-based routing techniques cannot be applied in very dynamic scenarios like those we consider in this work.

To overcome the intrinsic fragility of trees, other researchers experimented mesh based solutions. Among them, Yoneki and Bacon [136] proposed using the On-Demand Multicast Routing Protocol (ODMRP) for building an optimized dissemination mesh by applying techniques developed for multicast routing in MANETs to the context of a publish/subscribe system. In particular, they used bloom filters to summarize subscriptions. As a result, the publish/subscribe scheme is actually approximated to a topic based one, and the cost of this approximation is clearly an intrinsic limitation to such solution.

However, as mobility and size of the network grow, overlay state maintenance, whichever form it takes, tends to pose severe scalability problems. As a way to alleviate these, Content Based Multicast [140] and STEAM [80] introduce spatial scopes as a way to limit the diffusion of messages to a restricted geographical zone, thus avoiding the burden of maintaining a global routing topology. In Content Based Multicast messages are generated and spread in a given direction up to a given distance. Nodes interested in information about a given area send pull requests (i.e., subscriptions) in that area, where the matching actually occurs. Similarly, STEAM offers a content-based service to applications and implements it using a “proximity-group” communication mechanism. Proximity groups are defined by a proximity area, which defines the validity of messages and the type (the subject) of the filter, while the full content based filters are not used for routing but kept by subscribers. Moreover, the group communication middleware used by STEAM [74] provides strong guarantees about message delivery and ordering, but of course it will scale poorly if scopes grew beyond a given scale.

Besides spatial scoping, which intrinsically provides content-based routing only in a limited area around the publisher, other mechanisms were recently proposed to limit the amount of state brokers needed to keep while still providing content-based routing network-wide. In [43], a TTL is used to limit the propagation of subscrip-
tions. Published events “follow” the routes defined by subscription propagation when they are available, otherwise they are propagated using pure probabilistic gossip techniques. Similarly, the authors of Autonomous Gossip [48] propose a completely stateless, bio-inspired, self organizing mechanism to disseminate informations in a content-based fashion. Each mobile node of a MANET has a profile, which can be thought of as the node’s subscriptions, and a destination, which is the place where the node is going, which is assumed to be known to the routing algorithm. Messages are also labeled with a profile, a destination, or both. Messages migrate from node to node according to their “similarity” with the node’s profile and destination. Depending on the hospitality (based on similarity) received at the present node, messages decide to either continue to reside, migrate, or replicate to another node with a more suitable profile and/or goal destination. While similar to our algorithm in the idea of a totally structure-less dissemination based only on local information, Autonomous Gossip has several peculiarities. Indeed, with our algorithm messages follow the “traces” left by mobile nodes, which spread their interests through beacons that populate the proximity tables used for routing. Conversely, Autonomous Gossip messages reside in the routing nodes indefinitely, waiting to find interested targets.

Location-aided routing, also called geographical routing, or position-based routing, is a different approach, which also holds the promise of providing publish/subscribe in a structure-less fashion. Currently, this kind of routing has been used for addressing unicast messages in MANETs in a way that avoids keeping and maintaining routing tables by exploiting the knowledge about the geographical position of the destinations. This allows to keep the routing decision entirely local to the forwarding node, by propagating messages along a path of decreasing distance to the destination, in a similar way our own routing algorithm does. Recently, geographical routing have been extended to provide multicast services, e.g., see [78] and [126], while, to the best of our knowledge, it has never been applied to content-based dissemination. Clearly, our algorithm can be viewed as a special form of geographical routing, even if it is characterized by several peculiarities. First of all the domain of application, content-based routing, which is new in the area. Moreover, all the previous geographical routing approaches we are aware of, rely on some sort of self-localization mechanism to compute the current position of nodes (e.g., through a GPS) and on a location service, used to determine the position of message recipients. Our approach relaxes both these requirements, increasing the applicability of the approach.

Finally, the idea of exploiting the elapsed time since two nodes were most recently neighbors as a proximity metric has been originally proposed in [50] in the framework of on-demand path discovery. This idea has been further elaborated in [9].
Chapter 5

Conclusions

The publish/subscribe interaction paradigm is nowadays recognized as a good candidate to develop the communication infrastructures of forthcoming distributed applications. Thanks to its communication model, where the content of the exchanged information plays a prominent role, it can provide designers with those decoupling properties that are necessary for the development of modern, robust, large-scale distributed systems.

The research world produced in the last years a vast amount of works on publish/subscribe, spanning on various topics. One of the most studied issues is surely how efficient event dissemination can be realized.

5.1 Contribution and Future Directions

In this thesis we outlined how two techniques for efficient event routing in publish/subscribe system, namely event filtering and interest clustering, can be successfully employed to develop efficient publish/subscribe systems in widely different environments.

A Self-organizing Infrastructure for Interest Clustering in Filter-based Systems. Event filtering has been recognized as an effective method to increase the efficiency of event diffusion in many publish/subscribe systems. However, its effectiveness is largely reduced in all those scenarios where subscribers’ interests do not show a form of regionalism. We studied how interest clustering techniques can be added to existing publish/subscribe systems based on event filtering, to induce a sort of regionalism where this is naturally absent.

The solution we proposed is based on a dynamic reorganization of the underlying overlay network in order to place close in the overlay (in terms of overlay hops) brokers sharing similar interests. To measure similarity between two brokers we introduced a metric based on the comparison between recently matched events.
To test the effectiveness of this approach we designed a self-organizing algorithm for SIENA [115], a tree-based publish/subscribe system. An extensive simulation study showed (i) the ability of the algorithm to converge to a stable overlay topology (i.e. algorithm convergence), (ii) its ability to adapt to continuously changing scenarios while keeping an efficient behavior with respect to event dissemination (i.e. algorithm stability), (iii) the performance gain obtainable in various scenarios with respect to the plain SIENA event dissemination algorithm and (iv) how the self-organization algorithm can effectively take into account problems related to network-level metrics like latency or bandwidth.

The proposed solution currently exploit a deterministic mechanism to discover brokers with similar interest. This approach can be problematic on large settings, where “distant” nodes can be hardly discovered. An interesting approach to address this problem could be the employment of gossip-based techniques.

**Efficient Topic-based Event Routing for Peer-to-peer Systems.** Well known techniques for event diffusion cannot be straightly applied to peer-to-peer systems. These systems are, in fact, characterized by (i) very large scale, (ii) users’ autonomy that precludes any form of centralized network-wide administration or management, (iii) a dynamic behavior caused by nodes independently joining/leaving the system and, finally (iv) an intrinsic unreliability. The maintenance of fixed routing structures in such environments can be very expensive (in terms of used resources), or even impossible.

Therefore, we proposed a solution, namely TERRA, that leverages the characteristics of view exchange based overlay maintenance protocols to implement a probabilistic event diffusion algorithm. TERRA is a topic-based publish/subscribe system designed as a two-layer architecture: at the lower layer an interconnection medium connecting all the brokers is used to build and maintain interest-based subgroups of brokers, while at the upper layer all these subgroups remain independent. An event published on a broker is routed, at the lower layer, toward one access point for the target subgroup, i.e. toward one of the target brokers pertaining to the subgroup. This broker can then just apply the event flooding mechanism, limiting its coverage only to brokers pertaining to the target subgroup. In TERRA a probabilistic implementation of event-filtering works together with interest clustering to reliably deliver events while completely isolating their diffusion in specific interest subgroups. TERRA’s effectiveness has been proved through both an analytical study and simulations.

In the near future our plan is to further improve TERRA studying how reliable broadcasting services can be implemented in an dynamic peer-to-peer setting; in this way it would be possible to complete current TERRA’s characteristics and thus implement an highly reliable publish/subscribe system. Moreover, we are currently working in the implementation of a complete and working prototype that would al-
low us to test TERRA in more accurate testbeds, like PlanetLab [101].

**Proximity-driven Event Routing in Mobile Ad-hoc Networks.** The implementation of an event notification service for Mobile Ad-Hoc Networks (MANETs) is a challenging problem. Solutions that rely on complete subscription information stored over a static overlay network (often employed in static settings) cannot be adopted in an environment where nodes’ movements continuously change the overlay topology: the huge effort needed to maintain this information updated would probably hamper system’s performance.

The solution we proposed goes in a completely different direction: we do not try to run after node movements trying to quickly recover inconsistencies in event routing tables, but rather leverage reciprocal movements to track node movements. This information coupled with data on stored subscriptions is used to implement a form of event filtering that probabilistically prunes diffusion tree branches. The suitability of the proposed approach was confirmed by an extensive simulation study along with a stochastic analysis of the central mechanism adopted by the protocol.

The proximity-driven approach we presented is effective, but it can be further improved. We plan to include other aspects in the proximity evaluation to increase the quality of the estimation. For example, the amount of time some neighbor remains in a node transmission range can be taken into account to better estimate its future movements.
Bibliography


