XL Peer-to-Peer Pub/Sub Systems

ANNE-MARIE KERMARREC, INRIA Rennes Bretagne-Atlantique, France
PETER TRIANTAFILLOU, University of Glasgow, UK

16

Increasingly, one of the most prominent ways to disseminate information on the Web is through "notifications" (also known as alerts), and as such they are at the core of many large-scale applications. For instance, users are notified of articles in which they are interested through RSS feeds, of posts from their friends through social networks, or of recommendation generated by various sites. Event notification usually relies on the so-called Publish-Subscribe (Pub/Sub) communication paradigm. In Pub/Sub systems, subscribers sign up for events or classes of events in order to be asynchronously notified afterward by the system. The size of such systems (with respect to events and subscriptions) keeps growing, and providing scalable implementations of Pub/Sub systems is extremely challenging. Although there exist popular examples of centralized Pub/Sub systems that currently support a large number of subscribers, such as online social networks, they periodically face formidable challenges due to peak loads and do not always offer a support for fine-grain subscriptions. In fact, providing scalability along with expressiveness in the subscription patterns calls for distributed implementations of Pub/Sub systems. In parallel, peer-to-peer (P2P) overlay networks have emerged, providing a sound and highly scalable network foundation upon which to build distributed applications including Pub/Sub systems.

In this article, we focus on fully decentralized (P2P), highly scalable, Pub/Sub systems. More specifically, we investigate how Pub/Sub and P2P research can be integrated. We define the design space and explore it in a systematic way. We expose an understanding of available design choices; provide a comprehensive classification and understanding of prominent P2P Pub/Sub systems, positioning them against the design dimensions; and highlight correlations between and implications, benefits, and shortcomings of design alternatives.

Categories and Subject Descriptors: D.2.10 [**Software Engineering**]: Design—*Methodologies*; D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*; H.2.4 [**Database Management**]: Systems—*Distributed databases*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Peer-to-peer, continuous queries, publish-subscribe systems

ACM Reference Format:
Kermarrec, A.-M. and Triantafillou, P. 2013. XL peer-to-peer Pub/Sub systems. ACM Comput. Surv. 46, 2, Article 16 (November 2013), 45 pages.
DOI: http://dx.doi.org/10.1145/2543581.2543583

1. INTRODUCTION

Alerts play a prominent role in information dissemination today. Although some statistics related to the Olympics reported more than 11 billion hits by the year 2000 [IBM News 2013], the exponential growth of social networks [Twitter] and RSS feeds [Livejournal] keeps increasing the scalability expectations of such systems. Typically, users today subscribe to RSS feeds, then receive the posts of their friends or people they

This work is supported by the European Research Council under Grant ERC SG GOSSPLE 204742.
Authors' addresses: A.-M. Kermarrec, INRIA, France, and Peter Triantafillou, University of Glasgow, UK.

DOI: http://dx.doi.org/10.1145/2543581.2543583

follow on Twitter or Facebook, and they also receive recommendations from various sites. Receiving notifications often triggers Web navigation. This now concerns hundreds of thousands of users. Such notification systems rely on the so-called Publish-Subscribe (PUB/SUB) communication paradigm, where subscribers register their interests to specific events or event patterns in order to be asynchronously notified whenever events matching their subscription is published. Therefore, scalable implementations of PUB/SUB systems are key to the development of a large number of modern large-scale applications. Exactly because subscribers and publishers are so loosely coupled, PUB/SUB facilitates the construction of scalable communication platforms.

Typically, a PUB/SUB system is responsible for (i) managing users' subscriptions, (ii) matching published events against the stored subscriptions, and (iii) disseminating the events to the matching subscribers. The differences between PUB/SUB systems reside in the degree of expressiveness they offer and their scalability capabilities. For instance, topic-based PUB/SUB systems allow subscriptions to only a given event topic, whereas content-based ones provide users with the ability to define their interests at a much finer granularity, based on the content of events. Obviously, the finer the granularity, the more complex the subscription-management and event-matching tasks become. Therefore, combining a high degree of expressiveness with a scalable implementation constitutes a highly desirable feature, albeit a very challenging task.

Many centralized PUB/SUB systems have been designed and are available on the market [Kafka 2013; Tibco 2013; Oki et al. 1993; Vitria 2013]. However, they typically fail to provide both scalability and high expressiveness. For example, RSS feeds or Twitter-like systems provide scalability but not high expressiveness. On the other hand, corporate systems (e.g., Apache ActiveMQ [ApacheMQ] or [RabbitMQ]) offer a wide range of features for medium-scale environments and provide high expressiveness but do not scale to million of subscribers. From the academic side, some decentralized PUB/SUB systems have also been proposed, where the PUB/SUB system is composed of a network of brokers [Carzaniga et al. 2001; Aguilera et al. 1999; Ramasubramanian et al. 2006; Rose et al. 2007]. Although these approaches improve upon scalability, they still experience issues associated with the costs of communication among distributed entities and high network maintenance overheads. In the rest of the article, we focus on academic approaches.[1]

Research from the domains of distributed systems and networks produced Peer-to-Peer (P2P) networks, which provide a sound and highly scalable network foundation upon which to build distributed applications. Several P2P platforms such as Gnutella [Gnutella], Kazaa [Kazaa], or Skype [Skype] have been used by millions of users, demonstrating the scalability of the P2P paradigm. Similarly, many file swarming systems rely on the Cassandra [http://cassandra.apache.org/] P2P system. The scalability of the P2P infrastructures stems from the fact that each peer in a P2P system can act both as a client and a server; thus, as the number of clients increases, the number of servers increases linearly, avoiding the creation of bottlenecks at the servers. They are self-organizing, coping naturally with node arrivals and departures, and are resilient to failures. In such systems, peers are organized into a logical structure, called an *overlay network*, on top of the physical network that offers a support to easily implement routing, search, and key-value store functionality. In a P2P system, each user both benefits from and contributes to the system. This may exhibit several advantages in the long run. First, this can provide a cheaper alternative than running a huge system on a datacenter, which can limit scalability due to the finite resources a company can dedicate. Second, and arguably more importantly, P2P solutions might become a healthy alternative in the near future, avoiding "big brother" concerns, en route to

[1]Technical details are not always provided in industrial products.

privacy-aware notification systems, should users become reluctant to publicize their preferences and interests. For all of these reasons, we believe that it is of great interest to consider implementing P2P-based notification systems.

Although P2P infrastructures have not been designed specifically for PUB/SUB systems, they can provide great support to implement fully decentralized and scalable PUB/SUB systems (with promises of a much greater scalability beyond that offered by the distributed brokers approach). This is precisely the focus of this article. We study the match between P2P and extra large (XL) PUB/SUB systems. By extra large, we refer to the size of the geographic areas that span the locations of the users of the system, the need for the system to process high volumes of events, and support extra large user populations. P2P PUB/SUB systems are an attempt to combine these two important research areas. The central aim is to combine the benefits of a communication paradigm, which facilitates scalability, with a scalable communication infrastructure. Delivering the promises of providing a scalable implementation of a distributed PUB/SUB system using a P2P network is, however, a formidable task. The coupling between those paradigms is not as straightforward as it might appear at first sight, although they do share similar goals, in particular with respect to scalability. Scalability in a PUB/SUB system refers to the fact that as (1) the number of subscribers and/or (2) the number of events and/or (3) the number of topics or subscriptions increase, the system keeps providing notifications to all interested subscribers within bounded delays.

We address precisely this issue: how to marry P2P and PUB/SUB research in an effort to deliver a highly scalable communication and system paradigm. We will endeavour to answer the following questions: Can the two be fitted together? Can the scalable flavor of PUB/SUB be fully leveraged in a P2P implementation and vice versa? Which P2P infrastructure fits best which PUB/SUB system? Are there any intrinsic contradictions between the two concepts? What are the natural resemblances? We will do so by offering a systematic study of the available design dimensions for P2P PUB/SUB systems and by studying existing solutions with respect to the available design dimensions and choices.

1.1. Pub/Sub: A Highly Scalable Communication Paradigm

PUB/SUB software systems [Tibco 2013; Oki et al. 1993; Vitria 2013] constitute a facility for asynchronous communication between entities and for filtering of information. Users are consumers of information. They submit to the system continuous queries, coined *subscriptions*. Sources of data generation (producers) feed the system with data-carrying *publication events*. The PUB/SUB system infrastructure is responsible for (asynchronously) matching the publication events to all relevant subscriptions.

Subscriptions are continuous queries, as the system must store them and keep evaluating every submitted publication event against them to determine their matching. In essence, this infrastructure can be seen as a communication paradigm between producers and consumers of information. Such communication is asynchronous in both time and space: subscribers and publishers are typically at distant locations, and the submission times of corresponding subscriptions and events can be arbitrarily distant. Further, PUB/SUB systems can be seen as information filters, which filter all available information for every user and present to each user only the information units (s)he has defined as relevant. As such, a PUB/SUB infrastructure can play a vital role in large-scale data systems, with huge volumes of data, shielding users from the burden of always actively searching for and retrieving relevant information units. Similarly, the asynchronous nature of communication in the paradigm places fewer requirements on the resources of the infrastructure. For these reasons, PUB/SUB systems are viewed as a potentially highly scalable communication paradigm.

Scalability is facilitated from the fact that PUB/SUB systems fully decouple *publishers* and *subscribers*. A fundamental feature is the decoupling between publishers and subscribers along several dimensions [Eugster et al. 2003a]: (i) time decoupling means that publishers and subscribers do not need to be online at the same time, implying the notion of rendezvous point between the two; (ii) space decoupling refers to location distance and implies the anonymous property—subscribers and publishers do not need to know each other; (iii) synchronization decoupling refers to the asynchronous nature of PUB/SUB systems, in the sense that any communicating party is not blocked until the other responds.

Hence, the PUB/SUB paradigm is a natural solution for a very large number of existing and emerging application classes, including multiplayer Internet games (to be notified, say, when adversaries enter into specific own territories) [Boulanger et al. 2006; Lehn et al. 2011; Bharambe et al. 2006, 2008], electronic commerce applications with online bidding and product recommendations (to be notified when recommended items are available at a specific price, or specific characteristics, etc.) [ApacheMQ; RabbitMQ], or information feeds (e.g., RSS and alerts used to be notified, say, when specific news articles of interest emerge, or when stock quotes and transactions take place in specific price ranges). Finally, one of the areas growing the most, in which PUB/SUB plays a crucial role, is online social networks, such as Twitter, Facebook, or Google+, where users subscribe to some information streams from the members of their social network. Although more and more users exhibit an intensive use of social networks, beyond notifications, alerts gain in importance as they provide a new way to navigate such systems (e.g., Twitter or Facebook wall). The attractiveness of the PUB/SUB paradigm is further exemplified considering that it can be used even for online system/network management. For instance, consider the task of maintaining consistency between data replicas in a distributed setting [Garrod et al. 2008]. This entails, when updates occur, to inform other replica sites so that they will not be obsolete. This is easily amenable with the PUB/SUB paradigm: replica sites simply issue subscriptions to the system identifying the data for which they wish to remain up-to-date. The same holds for cases where users wish to monitor the contents of, say, Web pages. For example, a participant in a wiki page may issue a subscription to be notified when (certain) other participants have uploaded new information to the wiki page. Or, blog readers may issue a subscription to be notified when new comments have been made. Network managers concerned with security attacks may for instance issue subscriptions to be notified when too many packets are directed to a given IP address, detecting thus denial-of-service attacks. System administrators within a cloud infrastructure may issue subscriptions to be notified when a machine's utilization has dropped below or exceeded a threshold, and take corrective action to ensure the proper elasticity of resources, and so forth.

Interestingly enough, all of those applications do not have the same requirements. For instance, information dissemination in online social networks are characterized by very simple subscription patterns (get all updates from a given friend) along with a huge number of subscribers (Facebook has now more than a billion users) and small- to medium-scale information dissemination patterns. On the other hand, some electronic commerce applications may rely on highly complex subscriptions patterns (e.g., a stock quote x be over v_1 and y under v_2). Although the communication paradigm is the same, clearly the requirements of the underlying platform to implement the system might greatly differ from one PUB/SUB system to another.

1.2. P2P Overlays: Highly Scalable Infrastructures

P2P systems are software systems built on top of *overlay* networks. Overlays are logical organizations of network nodes, utilizing specific topologies (rings, hypercubes, trees,

random graphs, etc.). These topologies define the neighbors for each node. Note, however, that overlays are built over physical networks, such as the TCP/IP Internet. That is, the connection between two overlay-network neighbors is in essence a path in the physical network that connects these nodes. The key discriminative feature of P2P networks is the complete decentralized algorithmic and system design as well as resource aggregation among peers. P2P systems are characterized by *self-organization*, being able to withstand high dynamics with respect to network nodes joining and leaving the system while continuing to offer efficient operation. The decentralized operation of P2P networks ensures that (i) no global knowledge is required at any stage in order for any node to perform its tasks and that (ii) there exist no nodes playing a central role. P2P overlay networks rely on the assumption that peers may act both as clients and servers, solving intrinsically the scalability requirement, as the number of potential servers increases linearly with the size of the system. No peer has a global knowledge of the system; based only on individual decisions, global properties on the system emerge. This is precisely what makes them both attractive and complex to maintain. P2P overlay networks naturally handle dynamics (a.k.a. churn), because nodes themselves ensure that joins and departures and failures/recoveries are gracefully handled, automatically maintaining the overlay network topology and connectivity.

These facts have established P2P networks as being able to scale to a large number of nodes, at large geographic distances, with a large number of user requests and data being accommodated. Furthermore, unlike other distributed system approaches, P2P substrates build/utilize specific overlay networks, which ensure that any network node can communicate with any other node in a logarithmic (with respect to the network size) number of overlay network messages (a.k.a. hops). This also goes a long way for ensuring the scalability of the system. For these reasons, in this work we consider P2P-based designs for the overlay network substrate.

P2P networks are broadly characterized as either *structured* [Rowstron and Druschel 2001a; Ratnasamy et al. 2001; Zhao et al. 2001] or *unstructured* [Gnutella 2013; Jelasity et al. 2007; Voulgaris et al. 2005]. Structured overlay networks require a joining node to follow a specific protocol that assigns it to a specific position within the overlay topology. On the other hand, unstructured overlay networks allow nodes to originally occupy any position within the overlay network. Specific communication protocols are then followed in order to form a random graph topology. Be they structured or unstructured, most P2P overlay networks ensure a topology with a logarithmic (to the size of the overlay network) overlay network diameter. This, in turn, guarantees scalable communication between any two nodes, which is a fundamental requirement for any P2P overlay network.

1.3. Integrating Pub/Sub and P2P

Early implementations of distributed PUB/SUB systems relied on a client/server model [Tibco 2013; Vitria 2013], which constitutes an inherent and intrinsic barrier to scalability. Subsequently, implementations moved towards broker-based approaches, where a set of networked servers collaborate to achieve the matching between subscriptions and publications [Chen et al. 2000; Hanson et al. 1999]. The networks connecting the brokers were fairly ad hoc. In addition, the increasing number of subscribers posed strict limits to scalability. In essence, each such distributed PUB/SUB system design was burdened with the additional details of constructing and maintaining an overlay network substrate, reliably and efficiently routing messages between brokers, and so forth. Moreover, in centralized or distributed broker-based approaches, it is extremely difficult to both achieve scalability and support complex subscriptions. Obviously, Facebook, Twitter, or Google+ are all examples of *(semi)centralized* approaches that are able to scale to a huge number of subscribers. Yet, they remain relatively small scale as far as

dissemination is concerned. For instance, Facebook limits the number of friends to 5,000. More importantly, Facebook only supports coarse-grained subscription expressiveness (e.g., the granularity is a user). More complex subscriptions, such as receiving the updates of a friend only when they relate to a specific theme or are written in a specific language, are not supported today, for they would greatly challenge the system. Conversely, systems supporting more complex subscription patterns do not scale to such large sizes of subscribers and events, and are usually used in corporate environments (Apache ActiveMQ).

In that respect, P2P infrastructures represent a promising infrastructure to implement large-scale and expressive PUB/SUB systems. Some recent implementations do leverage the scalability and efficiency features of P2P networks (e.g., Bianchi et al. [2007], Aekaterinidis and Triantafillou [2006], Anceaume et al. [2006], Baldoni et al. [2005], Gupta et al. [2004], Castro et al. [2002], Voulgaris et al. [2006], and Zhang et al. [2005]). All in all, implementing such systems at large scales has remained a challenge for a long time. P2P infrastructures are generic in the sense that they provide low-level communication abstractions, such as routing, so they can be tailored to various classes of applications. The central issues are whether and how the characteristics of the P2P overlay network infrastructure can be integrated with PUB/SUB systems. Studying related work, it becomes clear that there have been instances of such a good integration. Yet, there is no such thing as *one P2P overlay fits all*, as there is no clearly identified P2P overlay that would provide the ultimate solution to scalable PUB/SUB systems [Bender et al. 2006]. The central reason for this is that P2P overlay networks were designed to be generic in an attempt to match several kinds of applications. Typically, they have been applied to video streaming, storage systems, dissemination systems, and the like. In addition, there exists a wide range of possible design choices that may be considered when developing a PUB/SUB system upon a P2P overlay. This is precisely what we address in this article. With this work, we attempt to shed light on the integration of these two important research areas. The goals of this article are to explore the design space, to clearly identify the relevance of specific P2P overlays to implement specific PUB/SUB systems, and to highlight the key trade-offs, with respect to system complexity, ease of development and deployment, and performance when making specific design choices.

We are convinced that P2P overlays can be effectively leveraged to implement XXL P2P PUB/SUB systems, but this has to be achieved with care and awareness. This is what this article strives to contribute.

1.4. Roadmap

The rest of the article is organized as follows. Section 2 briefly provides the background on PUB/SUB systems, P2P overlays, and current distributed implementations. Section 3 presents the dimensions of the design space of decentralized PUB/SUB. The design space is explored in Section 4, where specific approaches are analyzed in greater detail and fitted with respect to the established design space dimensions. Section 5 discusses issues pertaining to synthesizing various designs utilizing the analysed design dimensions. We conclude in Section 6.

2. FOUNDATIONS

In this section, we provide an overview of the fundamentals of both components: PUB/SUB systems and P2P overlay networks.

2.1. Pub/Sub Systems

As mentioned previously, PUB/SUB is a communication paradigm, enabling a full decoupling between the communicating parties: the event producers (publishers) and

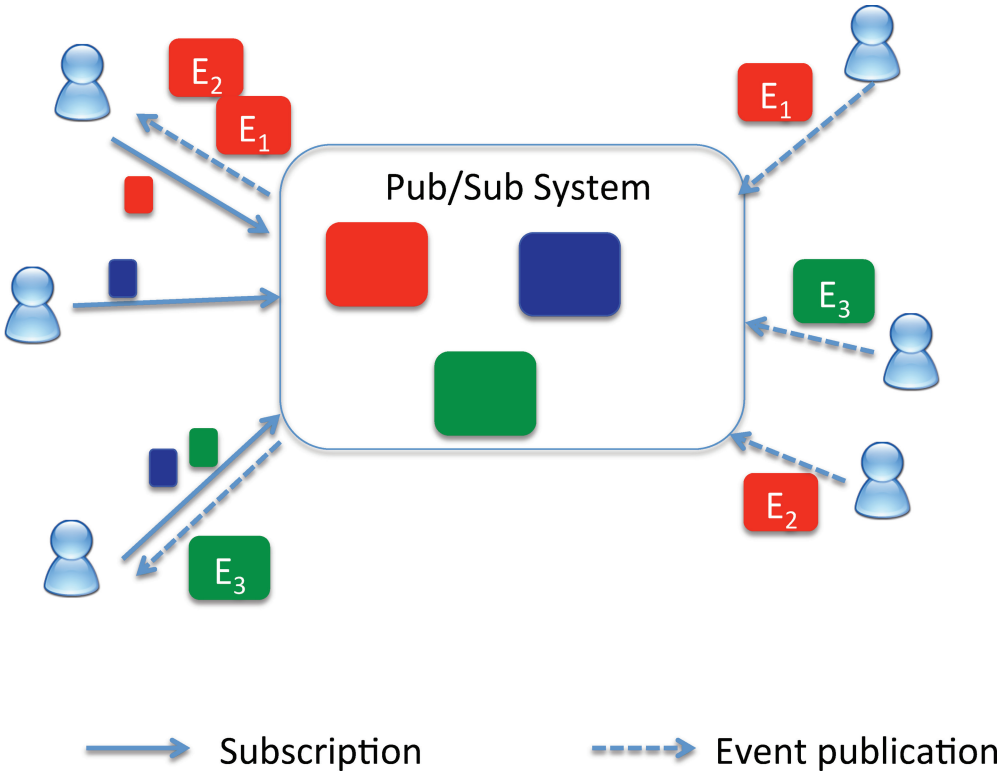

Fig. 1. Example of a Pub/Sub system. Subscribers subscribe to the system, which is in charge of storing the subscriptions, matching events against the subscription upon publication and disseminating events to interested subscribers.

consumers (subscribers). A Pub/Sub system is essentially a middleware implemented in the form of a *broker* (or a network of brokers) to which publishers send their events and with whom subscribers register their interests. The main task of the middleware is to record subscriptions and process events, matching them against recorded subscriptions and, in the case of a match, notifying subscribers of the event matching their interests (Figure 1).

The Pub/Sub paradigm has been acknowledged as a potentially highly scalable communication paradigm, for it ensures time, space, and synchronization decoupling between publishers and subscribers [Eugster et al. 2003a], unlike other synchronous message passing systems, in which typically one needs to know the address/ID of the entity being communicating with and the application sending a message is usually blocked until the recipient process performs the corresponding receive operation.

Several classes of Pub/Sub systems exist that are characterized by the achievable expressiveness of subscriptions. The first Pub/Sub systems were based on the concept of groups, and they were generally referred to as *topic-based systems* (a.k.a. channel-based systems). Topic-based systems [Object-Management Group 1998; Sun Microsystems 2000] categorize events into predefined groups. Users subscribe to the groups of interest and receive all events sent to these groups. Typically, following some Twitter users or inviting a friend on Facebook represents a form of topic-based subscription. *Subject-based systems* [Cugola et al. 2001; Object-Management Group 2000] are very similar to topic-based systems, where each event is enhanced with a tag describing

its subject. Subscribers can declare their interests about event subjects flexibly using string patterns. The topic-based approach may generate some *noise* on the subscriber side. This follows, since the granularity of interest may be too coarse and too static to match the subscribers' needs. For example, a statically predefined topic (say, the Greek Stock Exchange) may be too broad and too general for a subscriber, who may be only interested in the trading of a particular Greek stock and thus may be swamped with messages of little or no interest to him. Similarly, all updates of an Italian Facebook friend might not be relevant to an English-speaking user, who might be interested in updates when they are written in English. On the other hand, the simplicity of the topic-based model enables very rapid and efficient implementations.

In the *content-based model* [Carzaniga et al. 1998; Banavar et al. 1999], subscribers use flexible means to declare their interests with respect to the contents of the events. Thus, subscribers may be much more expressive with their subscriptions, specifying exactly the content of events in which they are interested. For example, a trader interested in the stock of the Greek Telecom Operator, OTE, may define his interest specifying "Stock Name = OTE." Further, he may wish to be notified of tradings that occur within a specific price range, and so he may further specify "price $\leq$ 10." In this way, this trader will receive only events of true interest, reducing noise. (A very similar model is the *content-based with patterns model* [Carzaniga et al. 2001; Oki et al. 1993], with extra functionality on expressing user interests.) For the sake of simplicity, we consider next that range queries are composed of set of discrete values.[2]

In general, the notion of content is typically implemented assuming a well-known schema for both subscriptions and events. The schema consists of a set of known a priori attributes with known types. The *event schema* of this model is thus an untyped set of typed attributes. Each event attribute consists of a $<type, name, value>$ triplet. The type of an attribute belongs to a predefined set of primitive data types commonly found in most programming languages. The attribute's name is a string, whereas the value can be any value in the domain defined by the type. A set of $<type, name, value>$ triplets for some schema attributes constitutes the event. The *subscription schema* is more general, allowing for expressing a rich set of interests, and it is also a set of triplets, containing all interesting subscription-attribute data types (integers,strings, etc.) and all interesting operators ($=, \leq, \geq, prefix, suffix, containment$, etc.).

An event is said to match a subscription *if and only if* all predicates defined by the subscription are satisfied by the values carried by the event. Although content-based systems offer a high degree of expressiveness and consequently limit the noise at the subscribers' side, they incur a significantly higher complexity.

PUB/SUB systems differ from traditional data systems, where first data is stored and indexed, and then queries can be served, accessing the data and computing the data items that are relevant for the query. In PUB/SUB, queries are represented by subscriptions. These queries are *continuous* in the sense that they remain in the system until explicitly deleted. Subscriptions, therefore, are stored within the system and indexed. Data in PUB/SUB systems are represented by events (and consist of the values of the attributes in the events). When an event arrives at the system, the system accesses its stored subscriptions to determine the set of subscriptions matching the event.

Early work in the area was (correctly) concerned with efficient matching algorithms and indexing data structures for subscriptions, as exemplified by the LeSubscribe and Gryphon approaches [Fabret et al. 2001; Aguilera et al. 1999]. For concreteness, we explain the Gryphon approach.

[2]Dealing with a continuous range of values usually requires a discretization step from the system designer; this task can potentially be complex, and we do not address this issue in this article.

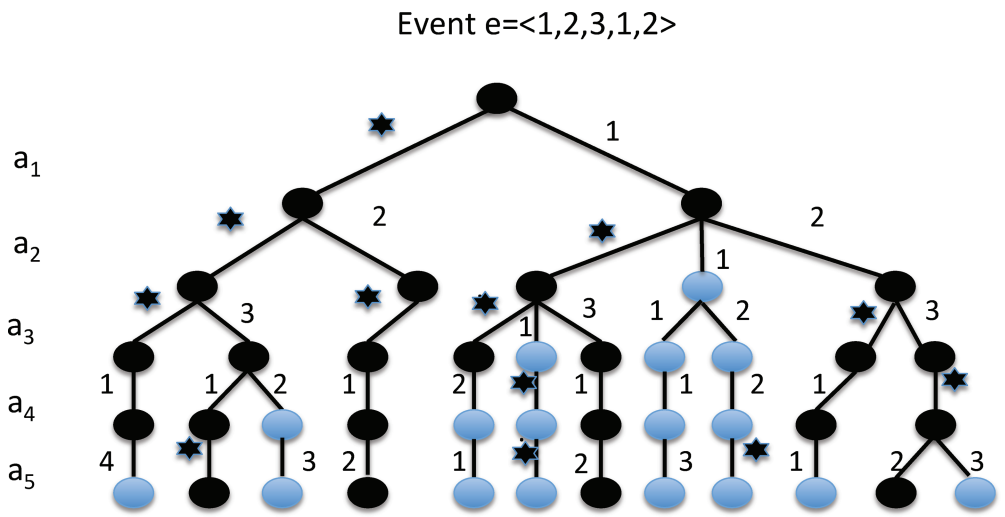

Fig. 2. Example of subscription/event matching in Gryphon. When the event $e = <1, 2, 3, 1, 2>$ is published, all dark circles (representing brokers) are visited. Each level of the broker tree represents an attribute.

Gryphon. In the Gryphon approach, subscriptions take the form of a schema consisting of attributes, such as $A_1, \ldots, A_n$, and are stored in the form of a tree. A parallel search tree structure is formed, with n levels (equal to the number of attributes). The i-th level corresponds to the i-th attribute, A_i. The tree is formed as subscriptions arrive. A subscription specifying a value V_1 to the first attribute of the schema, A_1, will follow from the tree root the edge labeled with V_1. If no such edge exists, one will be created. For $A_i = V_i$, the subscription will follow at level $i - 1$ the edge with label V_i. If a subscription does not name an attribute at level i, then it will follow the edge with label $*$. Label $*$ represents a *do-not-care* condition.

The Gryphon indexing structure is illustrated in Figure 2, where five attributes are represented, one per level in the tree. In this figure, each leaf is labelled with the identifier of the subscriber(s) whose subscription matches the predicate from the root to the leaf. For instance, consider the subscription $S = <A_1 = 1,\ A_2 = 2,\ A_3 = 3,\ A_5 = 3>$. This subscription is represented by the rightmost leaf on the figure. At the root, the algorithm examines the value of the first attribute A_1. Since it is "1," it will follow the child of the root, linked with label "1". At the next level, since $A_2 = 2$, it will again follow the right child, and so forth. Thus, S will follow the rightmost path and be deposited at the rightmost leaf node. The inner nodes in the figure represent the brokers. An event is processed similarly. Event $<A_1 = 1,\ A_2 = 2,\ A_3 = 3,\ A_4 = 1,\ A_5 = 2>$ will visit all darkened nodes of the figure (in parallel) to reach the leaves storing all matching subscriptions. Note that events must traverse $*$-edges to find matching subscriptions with "do-not-care" attributes. Therefore, Gryphon offers a support for complex subscriptions and provides an efficient event-matching algorithm.

With works such as Fabret et al. [2001] and Aguilera et al. [1999], the community gained a good understanding of efficient indexing techniques and fast matching algorithms. However, when implemented within a single-broker architecture, it obviously suffers from the same limitations of any centralized architecture when the number of users, their geographical dispersion, and the number of events/subscriptions keep increasing. (Gryphon supports up to 10,000 subscribers [Gryphon].) However, the works mentioned earlier largely facilitated the jump to distributed implementations,

where brokers could locally employ efficient indexing structures and apply efficient matching algorithms.

To address scalability, researchers at the same time turned to distributed approaches. Early works in distributed PUB/SUB systems relied on a network of brokers to which subscribers were attached and that were in charge of the distributed matching of subscriptions and events. These approaches, in addition to the PUB/SUB logic, were concerned with developing and maintaining an ad hoc broker-network architectures as the PUB/SUB infrastructures. Examples of such systems are Jedi [Cugola et al. 2001] and Siena [Carzaniga et al. 2001] detailed next.

Jedi and Siena. Jedi developed a tree topology of broker nodes, where users were associated with leaf brokers and users' subscriptions and events were first directed to their associated brokers. During subscription processing, subscriptions were directed up the leaf-to-root path, leaving at each intermediate node some state identifying it and the child it came from. During event processing, events also followed the leaf-to-root path. This guaranteed that any event and any subscription would rendezvous at (least at) the root. Assuming all broker nodes had a local matching algorithm, any incoming event would therefore be matched against all subscriptions that have passed through each broker. This, in turn, guaranteed that no subscription matching an event would go undetected. In addition, as events travelled towards the root, if at some intermediate node they found a matching subscription (based on the state left behind by subscriptions traveling to the root), the event was also sent down the reverse path of the matched subscription to the leaf where the subscription originated and then the event was handed to the associated user. This sped up the matching process. Note that this is one of the first instances of *content-based routing*; that is, messages (in this case events) were routed according to their contents (e.g., the values carried by the event) and the contents of the nodes being visited. Therefore, the paths being followed depend on the contents of events and subscriptions and is not predetermined. Figure 3 depicts this process. Dispatching Servers (DS) are in Jedi equivalent to brokers, and Active Objects (AOs) are equivalent to users issuing events and subscriptions. The figure shows the processing of a subscription (dotted arrow) and of an event (straight arrow) submitted by different users. In Jedi, each subscription (e.g., as the one generated by the middle AO in the figure) is distributed to all of the DS in the leaf-to-root path, and each DS registers itself to its parent and descendants in the tree, which in turn do the same. When an event is published (from the leftmost AO in the figure), the event is propagated up in the tree and down to the matching subscribers.

Siena, on the other hand, formed a graph broker topology. Each user was associated with one broker. A subscription was first handed to the user's broker and then was propagated from broker to broker throughout the graph. Again, at each node n visited by a subscription s, state was left behind identifying s and the neighbor of n that s came from. This subscription state, kept at brokers visited by s, collectively established the path that s followed from the originating broker to broker n. Events, similarly, were first handed to the user's broker. Then, the event was propagated throughout the graph based on the available subscription state at each visited broker. If at some broker a matching subscription was found, then the event followed the reverse path setup by the subscription as it was propagated. Thus, following these reverse paths, the event would reach the brokers where all matching subscriptions had originated and would be handed to associated users. This is also one of the first instances of content-based routing.

Siena [Carzaniga et al. 2001] proposed another influential idea, based on exploiting the notion of *subscription coverage*: a subscription S_2 is covered by S_1 (denoted $S_2 \subset S_1$) if whenever S_1 is matched by an event e, then S_2 is matched by e as well. This idea saved considerable messaging overhead during subscription processing, since covered

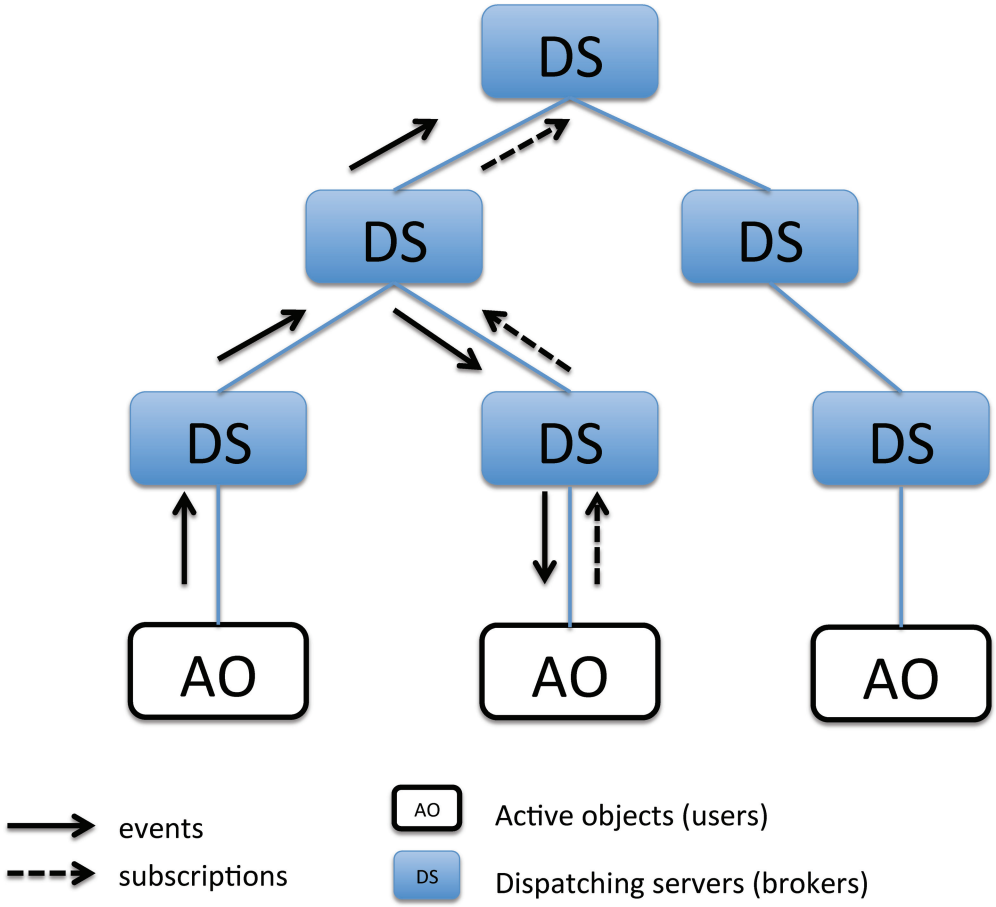

Fig. 3. Jedi topology and subscription/event processing.

subscriptions need not be forwarded everywhere. Figure 4 depicts the processes of event and subscription processing with coverage. For example, subscription S_1 is sent from a subscriber s_1 to broker B_5. Broker B_5 forwards S_1 to broker B_4, which in turn forwards it to broker B_3, and so forth. At each broker, say B_4, state is maintained so that B_4 remembers that S_1 came from B_5. Note that S_2 is covered by S_1 since any event matching S_2 will also match S_1. Therefore, S_2 is not forwarded further by B_4, because it has already forwarded a more general subscription. This saves messages upon subscription processing. Event e follows the reverse path. At B_4, it is determined that it only matches S_1 and is forwarded accordingly to subscriber s_1.

Please note that a number of key notions, structures, and algorithms were already defined by such prior works, including efficient algorithms for local matching, local-indexing structures for subscriptions, content-based routing, and subscription coverage.

2.2. P2P Infrastructures

Initially introduced in the context of file sharing (Gnutella, Kazaa, etc.), the applicability of P2P infrastructures goes far beyond. P2P infrastructures provide an efficient support for a wide variety of distributed applications, building a logical network on

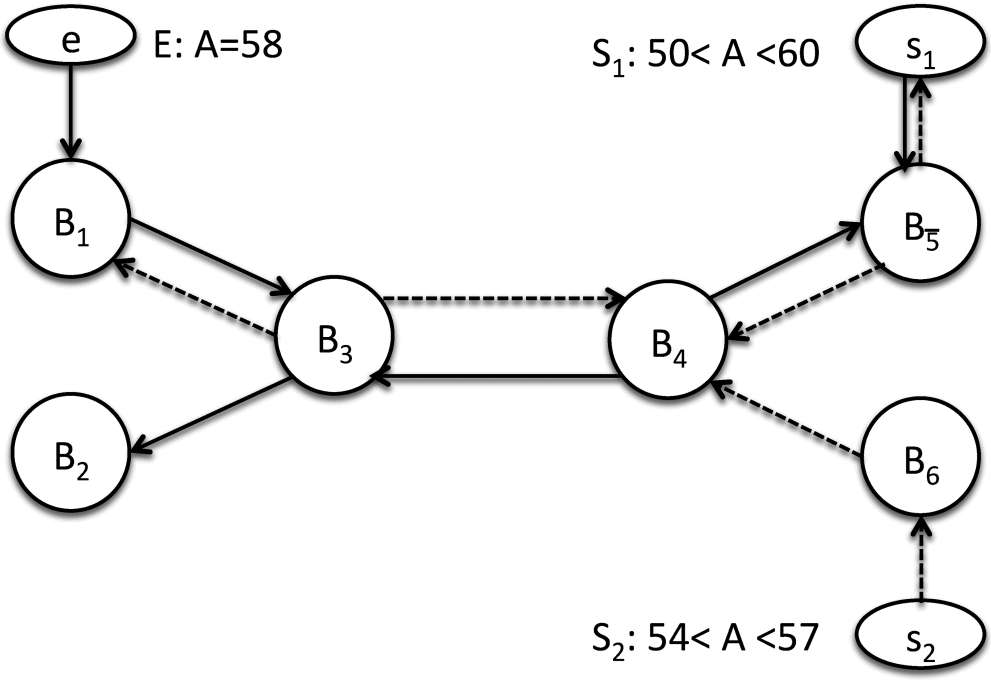

Fig. 4. Siena subscription/event processing with coverage.

top of a physical networking infrastructure. This is illustrated on Figure 5: Nodes form a logical network, where each node is connected to a few neighbors and a logical connection between A and B exists when A and B can communicate through the underlying networking infrastructure (typically know their respective IP addresses). Note that there is a priori no correlation between physical and logical neighborhood, even though this property is sometimes accounted for in network-aware P2P systems. Such overlay networks were a cornerstone development and introduced the capability of building and manipulating specific overlay network infrastructures at the application level—that is, without needing to change the Internet backbone.

Two main classes of P2P systems emerge as dominant from the plethora of works in this area. At one end of the spectrum, unstructured overlay networks form a mesh network, without strong requirements on the structure. In such networks, each node is connected to a subset of other nodes, irrespectively of their ID, their location, and so forth.

Several fully decentralized membership protocols have been proposed in this area to build unstructured networks: Araneola [Melamed and Keidar 2008] builds a basic overlay structure achieving three important mathematical properties of k-regular random graphs (i.e., random graphs in which each node has exactly k neighbors) with N, nodes: (1) its diameter grows logarithmically with N, (2) it is generally k-connected, and (3) it remains highly connected following random removal of linear-size subsets of edges or nodes. SCAMP [Ganesh et al. 2003] constructs an unstructured overlay network where each node is connected automatically to $log(N)$, nodes on average, without any node having the information about N, the size of the system. Gossip-based protocols [Jelasity et al. 2007] such as Lpbcast [Eugster et al. 2003b] or Cyclon [Voulgaris et al. 2005] create a dynamic unstructured overlay network, whose properties are close to those of random graphs with respect to degree distribution, clustering coefficient, and path length. In addition, such protocols create dynamic graphs, in the

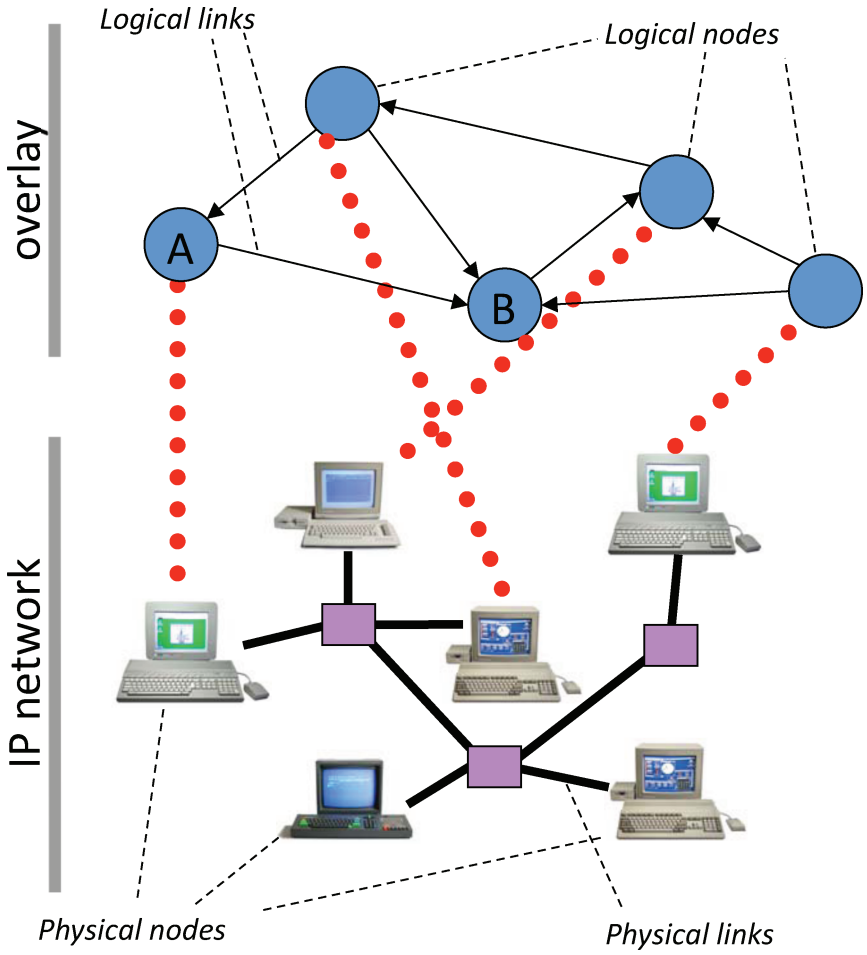

Fig. 5. Overlay network.

sense that the set of neighbors of a given node keeps changing, ensuring high resilience to churn (i.e., nodes joining/leaving the network at high rates). Unstructured overlay networks do not impose constraints on the graph topology, providing a highly flexible platform. However, such overlay networks do not provide advanced functionalities in terms of search querying, for example. We will see in the sequel how such topologies can be exploited in the context of PUB/SUB systems.

Structured P2P overlay networks, on the other hand, organize nodes in a specific logical structure, such as a tree, a ring, a hypercube, or a multi-dimensional space etc. Structured P2P overlays associate an identifier to nodes and impose constraints (identifier-wise) on nodes so that a given structure emerges. As an example, Figure 6 illustrates the Pastry [Rowstron and Druschel 2001a] overlay infrastructure. In Pastry, each node is associated with a 128-bit identifier within $[0, 2^{128} - 1]$. Pastry organizes nodes in a ring so that a message can be routed from any node to any key, reaching the node with the closest identifier to the key, in a logarithmic, in the size of the system, number of steps. Routing is achieved by reaching, at every step, a node whose identifier is closer to the key, i.e. the node's identifier has a longer common prefix with the key's identifier. In fact, under normal circumstances, at every step this routing

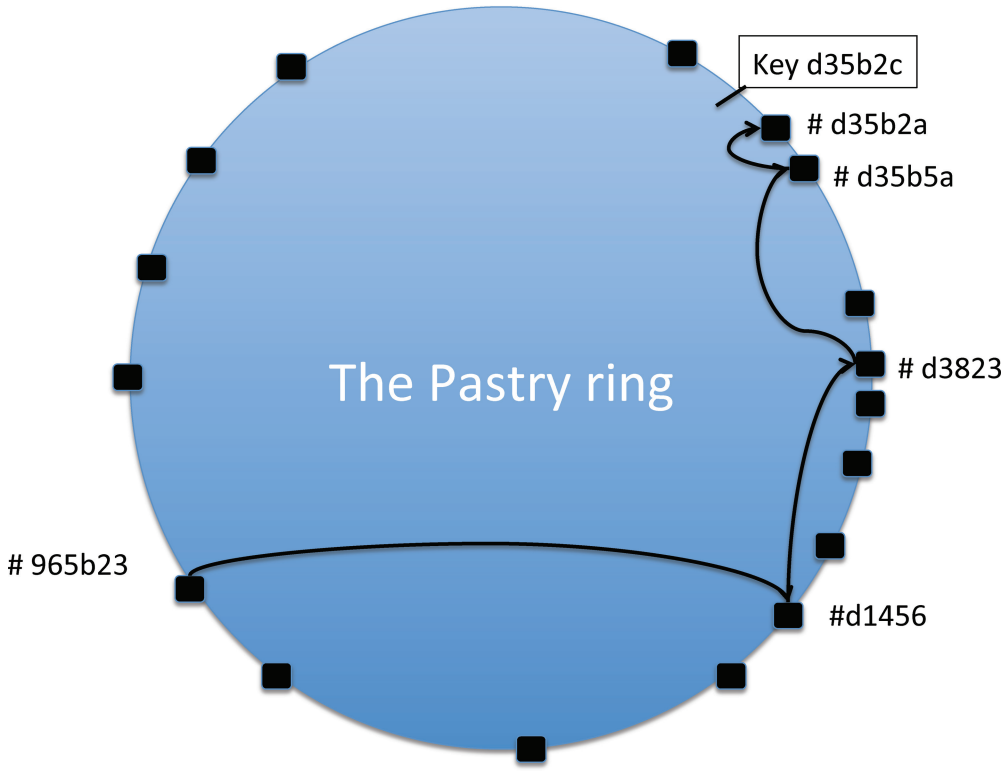

Fig. 6. Pastry routing example.

process is closer to the final destination by at least one digit. Given that identifiers use a logarithmic number of digits, it follows that routing takes $O(log_b(N))$ (for some base b) steps to complete. This so-called *prefix-based routing* was first proposed by Plaxton et al. [1997].

Consider the example depicted in Figure 6 pertaining to Pastry. Node 965b23 routes a message to the destination node d35b2c. The source node forwards it to node d1456, fixing the first digit, then to node d3823, reaching nodes with a prefix matching more and more the destination ID. To this end, each node stores a set of nodes in its routing table, having a number of rows equal to the size of the node IDs, and a number of columns equal to the different values each digit can take (typically numbering is base 16, so there are 16 columns). Pastry, in addition, distinguishes itself from the other systems by taking into account the network topology: the routing table is filled with the closest nodes, in a geographical sense, matching the routing table constraints. Experiments depicted in Rowstron and Druschel [2001a] have shown that on average, any message takes only 1.6 times the IP distance (in terms of latency).

As mentioned previously, many other systems exist (such as Chord, Kademlia, CAN, and Tapestry) that also provide an efficient routing substrate. They all provide a similar interface namely: `join/leave()` used by nodes to join and leave the overlay, `route(m,key)` to route a message to a key. An additional functionality is typically implemented on top of such networks: `lookup(key)` and `store(item,key)` can be added to store and retrieve items from the network. The fundamental high-impact characteristics of structured P2P overlays are (1) a good routing performance (which typically requires a logarithmic, in the size of the system, number of steps); (2) a limited-size

information maintained at each node (a.k.a. routing tables), which is also typically logarithmic in the size of the network; and (3) an efficient support for exact-match search queries, by associating keys to nodes. Pastry, as well as Chord, Tapestry, and CAN [Rowstron and Druschel 2001a; Stoica et al. 2001; Zhao et al. 2001; Ratnasamy et al. 2001] are the dominant examples of structured networks following the Distributed Hash Table (DHT) design. In DHTs, node IDs and object IDs are resolved (i.e., located) by hashing their name and obtaining an ID from a given name space. Instead of maintaining a single, centrally located hash table to then map such an ID to its location in the network (e.g., its IP address), a distributed hash table is utilized, with strategically determined portions of it being placed at strategically determined network nodes.

P2P structured overlay networks have been designed to ensure that the load is evenly balanced between the nodes of the systems. This property holds as long as the distribution of the keys is uniform in the system. In addition, like unstructured overlays, they have been designed to naturally cope with churn, namely node arrivals and departures. This is ensured by avoiding the network to be partitioned, implementing an efficient repair mechanism in the routing tables when nodes leave (voluntarily or as a result of a failure) and integrating new nodes in the system efficiently. Nevertheless, application-specific procedures have to be implemented by application designers to manage data structures and content associated with the applications. For example, in P2P file systems [Rowstron and Druschel 2001b; Kubiatowicz et al. 2000], file replication to ensure availability is up to the file sharing applications, not to the P2P infrastructure. Similarly, specific load balancing mechanism may need to be implemented depending on access patterns of the applications [Rowstron and Druschel 2001b]. Resilience to Byzantine failures has also generated a lot of work in the area of P2P networks but hardly in the area of ; therefore, we will not consider such failures in this article.

We will concentrate on such overlays, called by abuse of language DHT, whenever we address structured P2P overlays, as they are the most dominant ones.

A fundamental requirement for all P2P networks that we consider (be they structured or unstructured) is that scalable and efficient communication be ensured when any node communicates with any other node. In essence, this translates to the requirement that care be taken to ensure that the topologies that emerge either in a structured or unstructured network guarantee that communication requires a logarithmic (to the total number of network nodes) messages.

2.3. P2P Overlays and Pub/Sub

With the penetration of Internet into homes and the growing number of applications, the area of Pub/Sub has also acknowledged that there are significant idle resources at the edge of the Internet and gradually started considering P2P alternatives for the Pub/Sub system substrate.

In addition, researchers were concerned with the fact that the early works on distributed Pub/Sub were bundled with ad hoc overlay network infrastructures. Thus, they could not concentrate solely and simply on developing, implementing, testing, and deploying the Pub/Sub logic, but they also had to devote significant resources to developing, implementing, testing, and deploying the network substrate. The latter is a formidable challenge if one is to guarantee scalability and network connectivity in the face of high dynamics (a.k.a. churn, with nodes joining/leaving, and with node failures/recoveries occurring almost constantly).

Finally, note that the aforementioned distributed Pub/Sub systems, based on broker-network architectures such as Siena and Jedi would be hard-pressed (if at all capable) to deliver scalability and efficiency. This follows, since events and/or subscriptions may require $O(N)$ messaging overhead to propagate through the network (to facilitate matching), stressing available bandwidth and increasing event delivery times

significantly. Therefore, the inherent advantages of P2P overlay network infrastructures of high scalability, efficiency, and self-maintenance brought them naturally to the surface as the most promising substrate upon which to deploy XL P2P PUB/SUB systems.

3. SETTING UP THE SCENE: THE DESIGN SPECTRUM

We present and define next the fundamental dimensions of the design space. Our aim is to, on the one hand, understand the alternative design decisions that can be made and their trade-offs and, on the other, to facilitate a deeper understanding of the internals of available well-known decentralized PUB/SUB systems and their classification. The dimensions discussed are network (in)dependence, state, and expressiveness.

3.1. Network (In)dependence

Scalable decentralized PUB/SUB systems, as mentioned, largely depend on an underlying overlay network that efficiently maintains the broker network and reliably delivers messages (either for subscription or event propagation). How the PUB/SUB logic connects to the network logic is a fundamental design decision bearing significant implications relating to the overall efficiency, scalability, and applicability, as well as to the level of difficulty for developing, deploying, and maintaining the complete system. From an architectural point of view, the key question here is how closely coupled will be the P2P network and the PUB/SUB layers. We address this design criterion by referring to it as the *network (in)dependence dimension*. Note that in the following discussion, the network refers to the (logical) overlay network.

A *network independent* PUB/SUB system views the underlying overlay network layer largely as a black box with a well-defined, generic interface. The PUB/SUB logic works without any assumed knowledge of the internal algorithms running at the network level. It simply calls upon the network interface to deliver and receive messages and assumes that the network logic provides reasonable guarantees with respect to the reliability and connectedness of the network.

At the other end of the spectrum, a *network dependent* PUB/SUB system's functionality (and/or its key characteristics) depends on specific functionalities, properties, and/or guarantees provided by the overlay network. Between the two extremes exist a large number of designs, from those that work, say, over all DHT-based overlay networks, to those that can work over a subset of DHT-based networks having specific characteristics, and to those that work over a specific DHT.

The consequences of making a specific decision along the dimension of network (in)dependence center around the trade-offs of simplicity and ease of development, deployment, and maintenance (for the network independent choice) against the possibility of high performance for the network dependent choice, whereby the PUB/SUB logic can cater to, harness, and exploit network specifics. It is also worth keeping in mind that network independent solutions must typically deal with a form of a mismatch problem, as overlay networks were designed independently and were not designed to support complex queries of the type associated with PUB/SUB systems, such as range queries and/or prefix/suffix/substring queries for numerical and string attributes, respectively. For instance, relying on a generic *route()* primitive/interface provided by DHT networks while supporting the mentioned complex query types is a nontrivial undertaking. Solving these mismatch problems typically introduces performance penalties.

3.2. State

With the *state dimension*, we wish to characterize those PUB/SUB systems having the propensity to maintain *meta-information* on which the PUB/SUB logic relies in order to perform efficiently its *in-network* subscription and event propagation tasks. Such state is typically maintained at all network nodes and is used primarily for the *content-based*

routing of events and subscriptions appropriately in order to guarantee and expedite the subscription-event matching. It should be clear that the state to which we refer here concerns state maintained at the PUB/SUB layer and exists in addition to any routing state maintained at the underlying network layer.

At one end of the spectrum lie *stateful* solutions. One popular type of state in such solutions assumes the form of the state required to maintain specialized structures (trees, grids, etc.) and with which each node can know (and implement the role associated with) its position within the structure. Such structures are employed by the PUB/SUB logic in order to disseminate events and/or subscriptions appropriately within the network. Another such popular type of state in content-based PUB/SUB systems assumes the form of tags, which are special labels that the PUB/SUB logic at each node associates with each one of the network links of that node. Specifically, these tags typically represent *per-link filters*. This means that a link between two nodes in the overlay network carries the semantic information related to the PUB/SUB system. Typically, per-link filters are in essence predicates defining which subscriptions have been forwarded to a node and from which neighbors/links. Thus, a later incoming event is forwarded to only those neighbors whose link tags (predicates) match the values carried by the event. Referring back to Figure 4, broker B_3 maintains state that associates its link to broker B_4 with the filter $(50 \leq A \leq 60)$. Thus, any event arriving at B_3 that satisfies this predicate will be sent by B_3 to B_4; otherwise, it will be filtered out.

At the other end of the spectrum lie *stateless* solutions. Stateless decentralized PUB/SUB systems choose not to maintain and utilize any additional state during the in-network processing of events and subscriptions (other than that kept at the network layer). As a result, the route that events and subscriptions will follow during their dissemination in the network is predetermined when the events and subscriptions first arrive in the system, and next-hop decisions are not influenced by any extra state maintained by the PUB/SUB logic at the intermediate nodes visited en route.

It is worth drawing a parallel between the stateful versus stateless designs and the traditional so-called *end-to-end arguments* versus designs as in *active networks* for networked system architectures. Stateless decentralized PUB/SUB systems are akin to networked systems designed with the end-to-end principles in mind. On the other hand, their stateful counterparts resemble active network designs in principle and functionality, in that extra state is maintained at nodes and is utilized during the in-network routing of packets.

The consequences of making a choice along the state dimension can be far reaching. The general principle in system design that stateless solutions are easier to develop, deploy, and maintain holds here as well. Furthermore, the additional state kept at each node is classically associated with extra performance costs for maintaining it appropriately in the face of dynamic networks, with node failures and recoveries and subscription updates and deletions. On the other hand, extra per-node state for content-based routing can prove beneficial in a number of ways. For instance, when such state is in the form of per-link filters, it can be used to reduce the total number of messages needed for event-subscription matching, since events are propagated only down links from where matching subscriptions have been seen. As another example, maintaining state used to build structures like, say, trees can also prove beneficial because computer scientists find it much easier to visualize and develop solutions using such structures.

3.3. Expressiveness

One of the central questions when designing a P2P PUB/SUB system is that given the targeted expressiveness of a PUB/SUB system, which infrastructure is required, or best suited? Although the expressiveness of PUB/SUB systems is potentially endless, the

two end points of the spectrum are topic-based systems and content-based systems, respectively.

The targeted expressiveness of PUB/SUB systems largely defines the requirements placed on the underlying structure. In the context of this article, this largely influences the choice of the underlying P2P overlay network. On the one hand, P2P overlays were designed to be the ultimate scalable infrastructures and be generic enough to accommodate a large number of applications. On the other hand, they were not specifically designed with the PUB/SUB paradigm in mind. An interesting question here is the extent to which the different overlay types and network structures fit (and how they impact) the targeted expressiveness of decentralized PUB/SUB systems. Typically, structured overlays provide an efficient exact-match interface through an efficient routing protocol, whereby a message sent to a key can be routed in a small number of steps to the node responsible for that key. However, in unstructured overlays, each peer is freely connected to a set of other peers, without imposing constraints on that set of peers. The lack of any restriction on the choice of neighbors makes them extremely flexible, which can be seen as an asset and will be explained later.

Topic-based systems require (i) to group subscribers that have subscribed to the same topic together and (ii) to provide publishers with a means to reach the members of the group related to the topic of their event. Content-based systems introduce further requirements and the need for more flexibility. Their requirements largely depend on the following factors: (i) the range of values included in subscription predicates, (ii) whether values are discrete or continuous, (iii) whether attributes have well-defined domains of values, and (iv) the dimensionality of the subscriptions and the publications.

Topic-based systems can be straightforwardly implemented with an application-level multicast system, by clustering subscribers of a topic together in some structure (e.g., a dissemination tree). Such systems may greatly benefit from the exact-match interface and the associated efficient routing protocol of DHT systems. In addition, the volatility of subscribers may be crucial. For instance, a large number of volatile subscribers would render most typical dissemination structures (like trees) more fragile.

Content-based systems usually define a multidimensional space (one dimension per attribute). From this point of view, the most natural match for a P2P overlay infrastructure may be one based on a multidimensional space. However, this is only true as long as the two spaces can be mapped naturally and efficiently (typically with respect to the number of dimensions). The discrepancies in the range of values in subscriptions, for instance, might be better exploited in a hierarchical tree-based systems, where subscription ranges of values are refined going down the tree, as is the case in many tree-based PUB/SUB systems. Another interesting alternative are loosely structured systems, which start from arbitrary graphs and employ appropriate clustering mechanisms to build the clusters where events must be delivered.

Choosing the best structure for a given expressiveness requirement might seem straightforward. However, deciding upon the right way of building that particular structure is a complex undertaking, as many parameters come into play. Finding out which (structure/overlay) is the best match for the targeted expressiveness is a challenging and core task. These are the key questions that pertain to the structure dimension of our design space. They are fundamental, as they require to accurately assess the matches and mismatches between PUB/SUB and P2P systems in order to design a system preserving the inherent promises of scalability and efficiency of the P2P infrastructure.

4. EXPLORING THE DESIGN SPACE

In this section, we delve into the details of how well-known systems support the key aforementioned dimensions of the design space. As a caveat, we emphasize that it is

not our intention to present all published solutions in full detail when describing each design dimension. Instead, our intent is to present the essential features of well-known approaches that is enough to shed light into the specific design space dimensions and help in understanding and classifying these approaches. For each design dimension, we will discuss the appropriate internals of the systems that exemplify the characteristics of the dimension. Although we account for complexity and resilience to failures in the following, we leave out the resilience to Byzantine failures, for there are few works in the PUB/SUB area accounting for such extreme settings.

4.1. The Network Independence Dimension

In this section, we study the correlation and the dependence between PUB/SUB systems and the underlying P2P infrastructure. We should first mention that to our knowledge, there exists no PUB/SUB solution that can work over *any* overlay network. Instead, solutions can be classified according to the type of underlying overlay they assume, be it a structured (DHT-based) overlay or an unstructured one. For the former case, available solutions form two distinct categories: *DHT-independent* solutions, which work over any DHT, and *DHT-dependent* solutions, which work over a specific DHT. One of the main advantages of the DHT-independent approach is its wide applicability, whereas DHT-dependent solutions are tied to a specific underlying P2P infrastructure. Nevertheless, as we will see, designing for a specific DHT, one can fully leverage its potential. Additionally, in between, there are approaches that are designed to work over a subset of DHTs, which share specific properties. For the latter case, as these overlays are typically based on arbitrary graphs without any specific additional structures and algorithms, available solutions are all network independent. Hence, hereafter, we shall focus on DHT-based approaches.

4.1.1. DHT-Independent Approaches.

Prefix Hash Tree. We start our exposition with the Prefix Hash Tree (PHT) [Ramabhadran et al. 2004], one of the first indexing structures, proposed in order to efficiently support *range queries* over DHTs in a DHT-independent manner. PHT indexes items whose IDs are binary strings of length D (i.e., the size of the domain being indexed). PHT itself is a binary trie, all nodes of which are associated with a specific label (binary string) with the property that the left and right children of an internal node with label l have labels l_0 and l_1, respectively. An item with a specific ID, K, is stored at the (unique) leaf node whose label is a prefix of K. PHT (leaf) nodes can store up to B items, and when this threshold is exceeded, two children are created and the parent's dataset is partitioned among its two children based on their labels.

A PHT node with label l is associated with the DHT node $hash(l)$, where $hash()$ is the DHT's hash function. A domain with IDs having maximum label length L yields a PHT of a maximum of $L+1$ levels. How many levels exist depends on how many items have been inserted. Looking for any item ID may require up to $L+1$ DHT lookups in order to locate the level of the leaf responsible for storing this item. Since this search can be performed using binary search, in general, looking up an ID requires $\log(L+1)$—that is, $loglog(D)$ lookups.

PHT is made to efficiently support range queries by having each PHT leaf node maintain pointers to the leaf nodes being in its left and right in the PHT. Thus, collecting all items with IDs in a given range is performed by locating the leaf node storing an item with an ID in this range and following the left and right leaf pointers until the queried range is covered.

Although PHT was not designed as a PUB/SUB solution per se, it can fairly easily be employed to support content-based PUB/SUB functionality, as we will now discuss. In order to support PUB/SUB functionality, each subscription defining a range predicate can

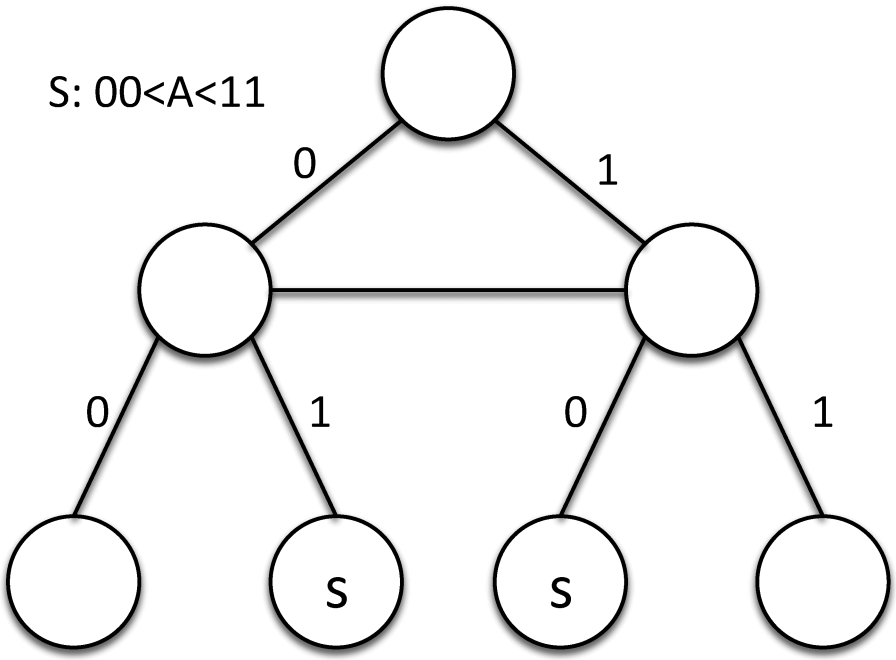

Fig. 7. Range subscriptions with PHTs.

be stored in PHT at all leaf nodes responsible for some key (value) in the given range. A matching event can be forwarded to the PHT leaf node storing the value carried by the event. The latter leaf node will by construction also hold the subscription with the matched range predicate, hence the event-subscription matching occurs. Consider Figure 7 exemplifying a simple PHT structure with three levels. IDs are binary strings from 00 to 11. A subscription expressing an interest for a single value may be stored at the leaf of the tree corresponding to the value (i.e., the leftmost leaf for value 00). A subscription s with a range predicate for values 01 and 10 may be stored at both of the two middle leaves. Any event carrying one of the two values will be directed to one of the two PHT nodes, where it will meet the matching subscription s.

As the PHT is traversed during event and subscription processing, each node communicates with the next node using the underlying DHT's *lookup*(). Hence, PHT depends solely on the *lookup*() function provided by the underlying DHT and is thus DHT independent.

As for all content-based PUB/SUB systems, PHT performance is difficult to compute, because it does not only depend on the performance of the underlying P2P network but also depends largely on the subscriptions patterns of the system. In PHT, subscriptions need to visit all nodes covering the given range of r values, which yields a cost of $O(log(N) \times r)$ hops. Interestingly, event processing in PHTs is faster since all that is required is to locate the PHT node holding the value in the event, which yields a cost of $O(loglog(D))$ lookups, where D is the domain size. Finally, there is no specific support in PHT for failures. The resilience to failures in PHT is therefore down to the DHT resilience to failures and its ability to repair the network and preserve the routing facility in the presence of churn.

Order-Preserving DHTs. A subtly different approach relies on introducing an additional hash function into the DHT realm with specific properties that cater to (and

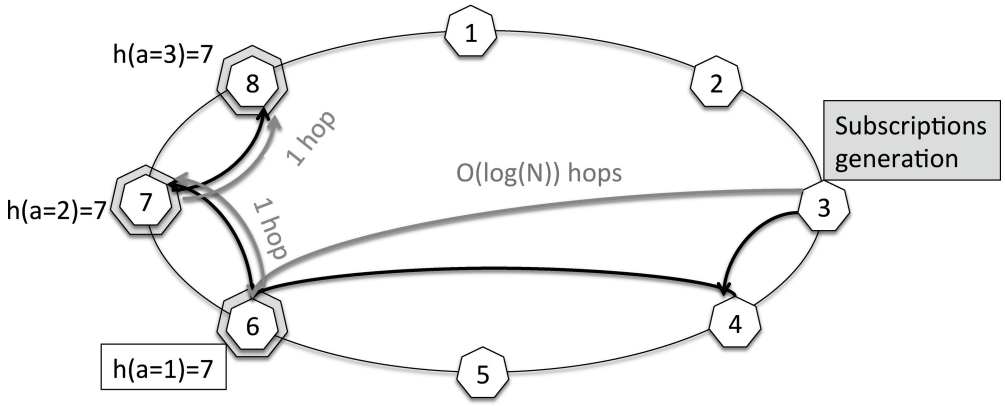

Fig. 8. Range subscriptions with order-preserving hashing.

facilitate the support of) the efficient processing of subscriptions with *range predicates* [Triantafillou and Pitoura 2003]. This new hash function can be any from a class of hash functions, which are coined *order preserving* (a.k.a. locality preserving): A hash function $h_{op}()$ is order preserving if for any values $v_1,\ v_2$ if $v_1 < v_2$, then $h_{op}(v_1) < h_{op}(v_2)$. Items are now placed in the DHT network by using the order-preserving hash function and the item's value. This is exactly as all DHTs do, with the exception of using this special function for producing the items' IDs, which determines at which DHT node each item/value will be stored. To the extent that DHTs have nodes maintain pointers to their immediate successors and predecessors in their name space (i.e., their alphanumeric ID neighbors), as they typically do, the algorithm for supporting range queries is evident: given any two items with values v and $v + 1$, it is possible by selecting the appropriate domain and range of h_{op} to associate them with IDs that also differ by one. Then, using the DHT lookup, these items/values will be placed at either the same or successive nodes in the DHT's name space. Processing a range query then involves finding a DHT node storing any value in the range and following successor/predecessor pointers until the complete range is covered. This is very similar, in the end, to the PHT approach, with the difference being that PHT utilizes a special tree structure to connect together the DHT nodes (PHT leaves) storing items, whereas Order-Preserving (O-P) DHTs rely on a special hash function to place successive values at successive DHT nodes, which are already connected by the DHT logic. It is also worth noting that with O-P DHTs, any node can be located with only one DHT lookup (in contrast to PHT, which requires $log(L + 1)$ lookups for identifiers of length L). The price to pay comes in the form of losing the nice storage load-balancing characteristics ensured by the uniform hashing function for data placement used by DHTs. However, any load-balancing algorithm may be used to address this, such as the one specifically developed for order-preserving hashing [Pitoura et al. 2006].

Keeping in mind this approach for range query support in O-P DHTs, the basic algorithms for supporting subscriptions with range predicates is as follows [Triantafillou and Aekaterinidis 2004]. A range subscription is stored at all DHT nodes (using $h_{op}()$ and the previous algorithm) responsible for a value in the range. Figure 8 illustrates this point for a subscription with range $1 \leq a \leq 3$. Any incoming event is sent to a DHT node (again using $h_{op}()$), responsible for the value in the event, where it will match any subscriptions that have been stored there. In the figure, an event carrying value $a = 2$ will be sent to node with ID $= 7$. As is easy to see, event processing requires just one DHT lookup, ($O(log(N))$ hops), whereas subscription processing requires a maximum

of $O(log(N) + r)$ hops, where r is the size of the range, making the O-P DHT-based approach significantly more efficient than the PHT-based approach for subscription and event processing.

DHTStrings. The common denominator of the previous approaches was the support for DHT-independent PUB/SUB, where subscriptions involved range predicates. Obviously, the same approaches can be employed for exact-match (point) predicates and predicates using $\geq$, $\leq$, $<$, and so forth. However, they cannot be useful in cases where subscriptions define predicates on string attribute types, and these predicates involve operations such as prefix (e.g., ABC*), suffix (e.g., *ABC), and containment (e.g., A*B). DHTStrings [Aekaterinidis and Triantafillou 2005] is an approach concerned with supporting subscriptions with such string-attribute complex predicates.

Given such a subscription s with, say, a prefix predicate ('ABC*'), DHTStrings computes the value h (*ABC*) (at the node where the subscription arrived), where $h()$ is the DHT's native hash function, thus determining the ID of the DHT node where subscription s will be stored, and stores it there. Later, when an event e appears with a matching value (e.g., 'ABCD'), DHTStrings computes all possible prefixes of the event value (i.e., 'A', 'AB', 'ABC', and 'ABCD') at the node where the event appeared and visits all DHT nodes with IDs $h(A)$, $h(AB)$, $h(ABC)$, and $h(ABCD)$. Obviously, in this way, event e will meet at some DHT node any stored subscription, like s, with a matching prefix predicate. The approach for supporting suffix operators is similar. For containment operators (e.g., A*C), the main idea is to view it both as prefix (i.e., 'A*') and suffix (i.e., '*C') predicates and process them using the algorithms shown earlier.

Clearly, DHTStrings is a DHT-independent approach, since it solely relies upon DHT lookups to perform its PUB/SUB functionality. For subscription processing, DHTStrings requires one DHT lookup operation per attribute (predicate) in a subscription. For event processing, it requires O(l) DHT lookups, where l is the average string length given in the string predicates (which is typically a small number, less than five). Subscription processing performance depends largely on the routing operation of the underlying DHT. In DHTStrings, only a single lookup is required (i.e., $O(log(N))$ hops). However, this advantage for DHTStrings becomes a disadvantage for event processing, since to resolve a prefix/suffix query $O(log(N) \times l)$ hops are required, with l being the length of the string. DHTStrings also accounts for load balancing, and as the number of attributes increases, the load balance improves, ensuring the scalability of the approach. As for the previous approaches, no specific support is provided for failure resilience. Although the approach is DHT independent, to the resilience failure is fully tied to the xfresilience to failure of the underlying DHT.

4.1.2. DHT-Dependent Approaches. DHT-dependent approaches include those that are designed to work over a DHT substrate with specific properties or those that are tightly coupled with a specific DHT.

Meghdoot. Meghdoot [Gupta et al. 2004] is meant to be employed over the CAN DHT [Ratnasamy et al. 2001] and supports subscriptions with numerical (range) predicates. CAN, in general, defines a d-dimensional Cartesian space, where peers' IDs define a point in the space. Peers also own specific partitions (a.k.a. zones) of the space and store data items whose IDs fall in their zones. In Meghdoot, subscriptions are defined over a schema of n attributes $A_1, A_2, \ldots, A_n$, defining a2$n-$dimensional Cartesian space, implemented using CAN. Each attribute A_i is associated with a domain range $[L_i, H_i]$. A subscription defining a range predicate l_i, h_i over $A_i, 1 \leq i \leq n$, occupies a point in the 2n-dimensional Cartesian space, coined the *subscription point*. In schemas with

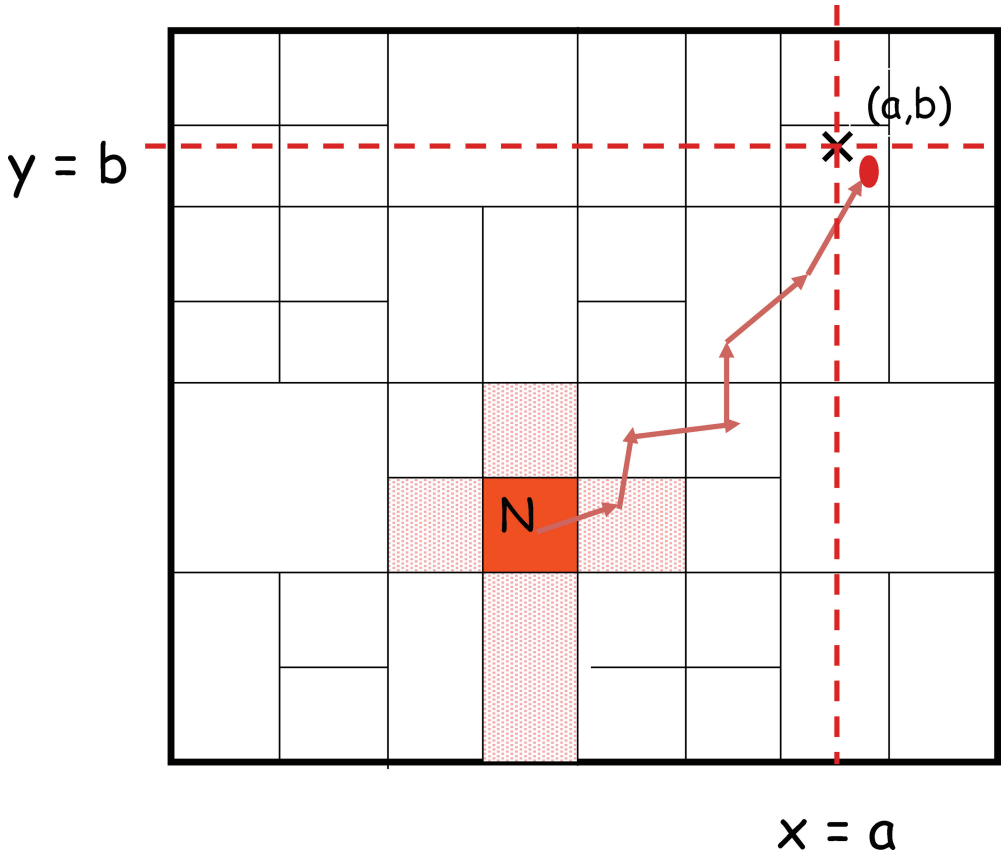

Fig. 9. The CAN overlay network.

only one attribute, a two-dimensional (2D) space is constructed and a subscription l_i, h_i is associated with point (coordinates) (l_i, h_i). To process an incoming subscription s at a CAN node, s is routed towards its subscription point and is stored at the peer owning the zone where the point lies. The routing of s uses CAN routing: at each CAN node n_i receiving s, n_i selects among its neighbors (maintained by CAN) node n_j whose Euclidean distance from the subscription point of s is the smallest and sends s to n_j. This is repeated until s reaches the target peer. Figure 9 shows an example of space division in a 2D space. Node N routes a message to point $(x = a; y = b)$. The message is routed to the node responsible for the zone containing that point. Events carry values v_i for each named attribute (A_i). Events are associated with event points, defined similarly to subscription points, with $l_i = h_i = v_i$. When an event e arrives at a node, the node computes the associated event point and routes e (using CAN) to its target peer, responsible for the zone that includes e's event point. Finally, the target peer sends e to all its neighbors falling in the region of the $2n-$dimensional Cartesian space that is affected by e. This region is defined to contain all subscription points of all possible subscriptions that have arrived in the system and are matched by the event. For example, in schemas with only one attribute, an event e with value v_1 is associated with the event point given by coordinates (v_1, v_1). The region (triangle) of the 2D space that contains subscription points of subscriptions that could match event e is given by coordinates $<(L_1, v_1), (v_1, v_1), (v_1, H_1), (L_1, H_1)>$. Figure 10 illustrates a

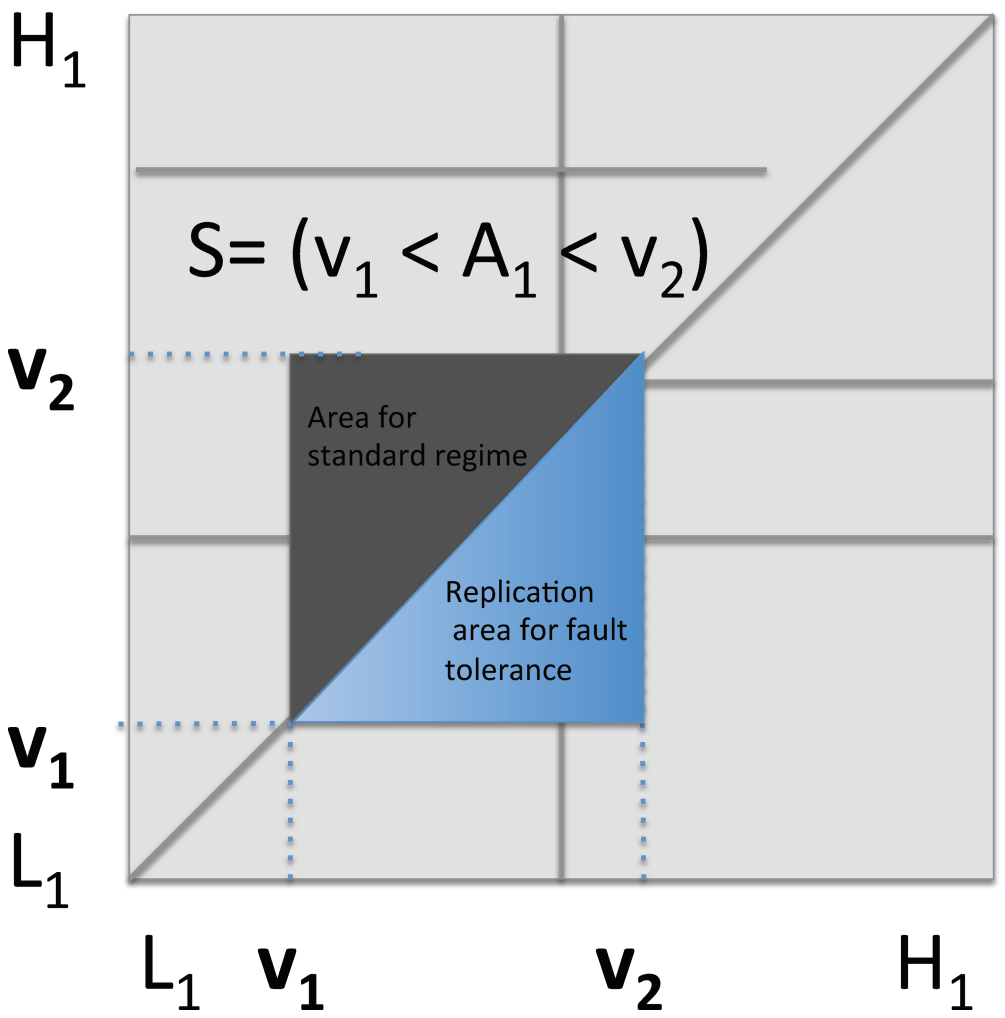

Fig. 10. Subscription in Meghdoot.

subscription in Meghdoot and the zone associated to that subscription in a 2D CAN (1 attribute). Attribute A_1 may take values between L_1 and H_1. A range subscription $S = v_1 \leq A_1 \leq v_2$ specifies a triangle, which represents the subscription (note that only the triangle above the diagonal is considered here). Whenever an event is routed to one of the point/peer in that triangle, therefore matching the subscription, a notification is delivered to the associated subscriber.

Meghdoot utilizes CAN to model the space to be indexed, as defined by subscriptions, and associates subscriptions and events (through subscription and event points) with specific peers where subscriptions will be routed and stored and where events will be routed. Meghdoot is also based on the CAN routing algorithm and provides extensions to the algorithms for joining and leaving the system in order to provide load balancing, given skewed data-value distributions and queries hence imbalances in the load for storing subscriptions and for routing events and subscriptions. Subscription processing requires $O(n\sqrt{(N)})$ messages in a CAN network with $2n$ dimensions and N nodes. This cost comes directly from routing to the subscription point. Note that this could be

improved with the (small-world-like) shortcuts offered by later version of CAN [Sun 2007]. Event processing also requires such a cost to route an event to its event point. Additionally, however, the event is then routed also to all peers falling in the region of the space storing subscriptions that may be affected by the event. Further, note that the performance of CAN worsens as the number of dimensions increases. Thus, for schemas with a large number n of attributes, a 2n-dimensional CAN needs to be built, yielding a routing process that takes a much longer time. However, note that this event cost in Meghdoot is independent of the number of attributes involved in the event. Meghdoot is one of the few PUB/SUB systems accounting for failures. Whereas the resilience of the routing operation relies on the resilience to failure of CAN, Meghdoot uses the unexploited part of the CAN space to ensure that the stored subscriptions are persistent even in the presence of failures. This is achieved by leveraging the triangle below the diagonal (on the figure). Whenever one of the nodes in the subscription triangle above the diagonal is faulty, the load is taken over by the node holding the symmetric point under the diagonal. To conclude, Meghdoot highly depends on the structure of CAN and could not be adapted in a straightforward way over other DHTs, such as Pastry.

Hermes. Hermes [Pietzuch and Bacon 2002] was developed to work over a Pastry-like DHT network. It is a typed system, where both events and subscriptions have types that in essence define which attributes they involve. In addition, it employs *advertisements* a la Siena in order to inform the networked brokers about which brokers publish which types of events. A defining characteristic of Hermes is the use of *rendezvous* nodes. When a subscription, event, or advertisement arrives at a broker (node), its type is determined and it is routed to the rendezvous node for that type, which is defined to be $h(type\_name)$, where $h()$ denotes the DHT's hash function. At the end, the subscription, event, or advertisement will be stored at the appropriate rendezvous node, where it is guaranteed that all events will meet all matched subscriptions. Figure 11 illustrates subscriptions and events being routed to the rendezvous node R.

Additionally, Hermes requires that the underlying overlay network exports at each node the set of overlay neighbors of that node. A distinguishing characteristic of Hermes is that at each node n_i, it associates a *filter* with each link to a neighbor node n_j. The dependence of Hermes upon this information, exported from the underlying network, makes Hermes a DHT-dependent system. This filter aggregates (using standard coverage and merging operators) the predicates of the subscriptions that n_j has forwarded to n_i. Similar state is associated for every advertisement en route to its rendezvous node at each intermediate node. Then, as a subscription s is being routed to its rendezvous node, whenever it come across a node n_i that has seen a matching advertisement, s is also routed towards node n_j, which is the neighbor of n_i that sent the matching advertisement to it. In this way, subscriptions are also stored close to the nodes that publish events of the same types. Similarly, as an event e travels to its rendezvous node, if at some intermediate node n_i finds a filter that matches the event, e is also forwarded to n_j, which is the neighbor of n_i that sent the matching filter to it. In this way, e follows the reverse paths followed by a matching subscription until e reaches the node where the subscription originated and where it will be matched with e. Figure 11 illustrates the routing algorithm in Hermes. Before an event is published, a rendezvous point (here R) is set for the specific type of event. Publishers then advertise themselves to that rendezvous point (dashed lines). The figure illustrates a system with two publishers, two subscribers, and five brokers. Subscribers Sub_1 and Sub_2 send their subscriptions to the rendezvous point R (grey arrow), and each broker on the path to R maintains that information. When an event is published from Pub_1 (black arrow), the message is forwarded through the brokers to the interested subscribers.

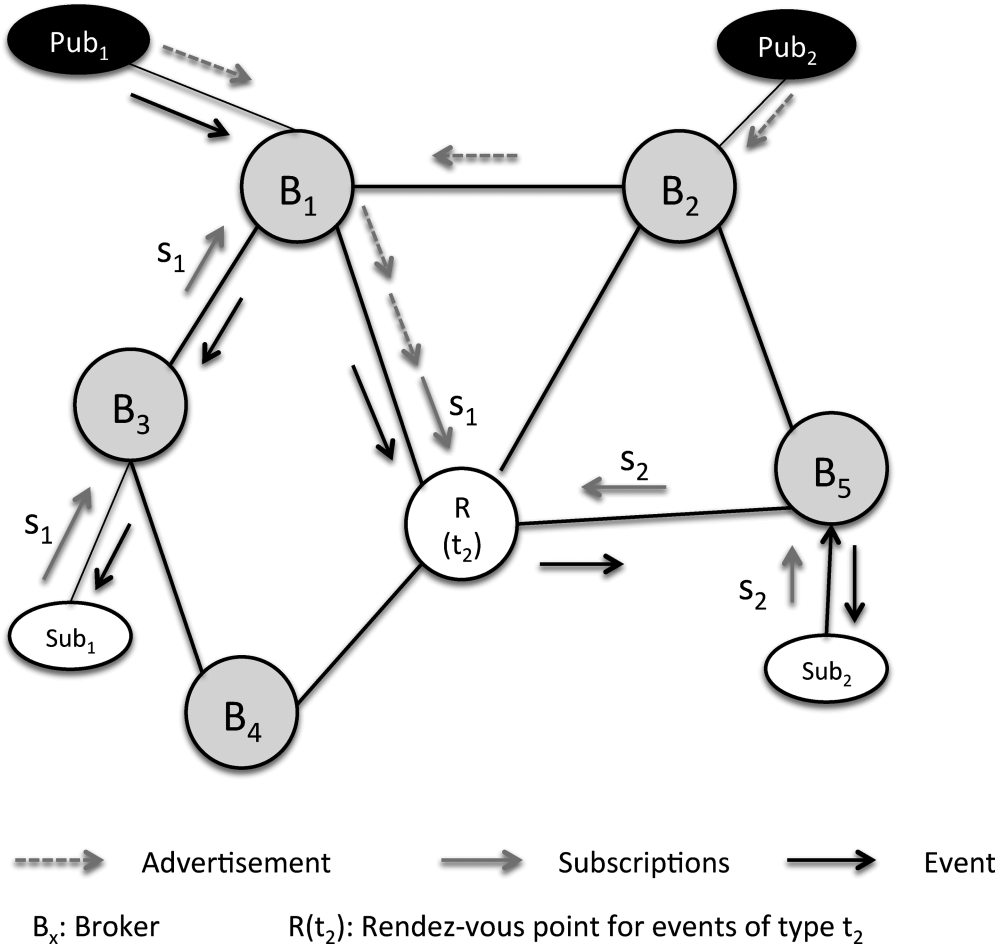

Fig. 11. Rendezvous in Hermes.

In Hermes, a subscription needs to be forwarded all the way to the rendezvous node, yielding a cost of $O(log(N))$ messages. Given that coverage techniques are used, the subscription propagation may stop before reaching the rendezvous node, if a node with a covering subscription is encountered.

Resilience to link failures in Hermes is ensured by the underlying P2P overlay routing resilience to failures. To cope with event broker failures, periodic heartbeat messages are sent on the broker network. When a failure is detected, the underlying routing infrastructure is used to reroute the subscriptions and advertisements to the rendezvous node. Event client failures are treated silently. In the absence of refresh messages of advertisements and subscriptions, the Hermes state expires. Rendezvous nodes are replicated (and identified by a specific hash function). In addition, those replicas are used to balance the load. Consistency between replicas has been left as an open problem in Hermes.

The dependence of Hermes on the underlying network basically stems from its requirement that the underlying overlay network exports at each node the set of its neighbors, with which particular state (e.g., advertisement and subscription types and filters) is associated.

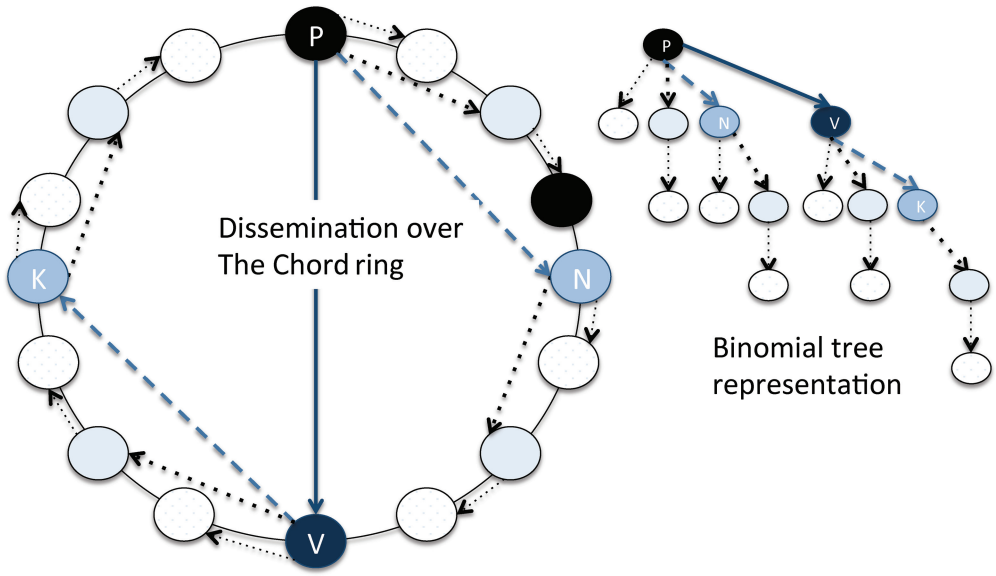

Fig. 12. Event dissemination in Rebecca (over the Chord ring and the binomial tree representation).

Rebecca. Rebecca [Terpstra et al. 2003] operates over the Chord DHT. Similar to Hermes, its dependence on the underlying overlay lies in the requirement of the PUB/SUB logic at each node to know which are its overlay neighbors and to associate particular state with each neighbor link. The per-link state maintained at each node is also similar to (actually a subset of) that in Hermes and mainly consists of subscription filters. A node u in Rebecca, thus, keeps along with each finger table (the finger table is the name of the routing table in Chord) entry pointing to node w a filter that represents the subscription predicates that u has received from w. Rebecca also requires that each node w knows which ($O(log(N))$) Chord nodes point to it (i.e., have it in their finger tables). Further, for each node w in u's finger table, node u assigns to node w a region of responsibility (actually, an arc of the Chord ring). Specifically, if w_i and w_{i+1} are the i-th and $(i+1)$-th entries in the finger table of u, then w_i is responsible for the Chord arc $[ID(w_i), ID(w_{i+1}))$ (where for the last node in the finger table, the arc of responsibility is defined to be the arc (half ring) between it and node u).

Event processing is very similar to broadcasting over Chord and employs the notion of regions of responsibilities to guarantee that an event will be delivered only once to each node. For an event e appearing at a node u, u is by default thought to be responsible for the whole Chord ring. Then, e is sent to each node in the finger table of u, along with a notification of which region each recipient node is responsible for. The process then is repeated recursively, with each node propagating e to its region of responsibility employing the subset of nodes in its finger table that point to its region of responsibility. Figure 12 illustrates event processing in Rebecca. The figure represents how the dissemination load is spread over the Chord ring and its representation as a binomial tree.

This process differs from broadcasting only in the following sense: a node u receiving event e will only send it to another node from its finger table w if the subscription filters associated with the link (u, w) is satisfied by the values in e. Subscription processing follows the reverse paths. A subscription s appearing at a node w will be sent by w to all the nodes pointing to w (i.e., all nodes that may wish to send to w events, using the

previous algorithm). This process is then repeated recursively, until the Chord ring is covered. When s is received by a node u from w, if the predicate filters in s are covered by the filter that u has associated with the link (u, w), then s is no longer propagated by u. Otherwise, the filter associated with the link (u, w) is updated to account for s.

In Rebecca, subscriptions and events are broadcast across Chord, yielding a worst-case cost of up to $O(N)$ messages. However, subscription filter state is used, and coverage can be used to reduce the high cost for subscription propagation. The same holds for event propagation as well: an event is further sent only along a link with a matching subscription filter, avoiding the cost of a complete broadcast.

Like Hermes, Rebecca on Chord exhibits a very tight coupling between the PUB/SUB logic and the network logic at each node.

Rebecca provides only best-effort guarantee in the face of node and edge failures, ensuring subscription resilience remains up to the application designer. Rebecca relies on simple retransmission mechanism to ensure reliable delivery and relies on Chord properties to re-establish links in the dissemination graph.

PastryStrings. PastryStrings [Aekaterinidis and Triantafillou 2006] is a PUB/SUB system intended to operate over all DHTs supporting prefix-based routing, such as Pastry, Bamboo, and Tapestry [Rowstron and Druschel 2001a; Zhao et al. 2001; Karp et al. 2004]. PastryStrings is comprehensive in the sense that it supports complex predicates on both numerical and string attributes, such as range predicates and prefix/suffix/containment, respectively. The basic idea rests on building so-called string trees, the nodes of which (coined *Tnodes*) are hosted by the DHT (e.g., Pastry) nodes. For each letter of the alphabet, there exists a different string tree; for instance, all predicate strings starting with a will be stored in the string tree rooted at $Tnode_a$.

There are two key ideas: First, each *Tnode* maintains an additional (to the DHT) routing table that connects a Tnode to each of its children. A Tnode has β children, where β is the number of different characters in the alphabet, building essentially a trie structure. To process a predicate with value abc, for example, first $Tnode_a$ will be accessed (by hashing the value a with Pastry's hash function). At $Tnode_a$, the routing table entry for its child b will be accessed, leading to $Tnode_{ab}$; from there, the c child will be accessed leading to $Tnode_{abc}$, where the subscription will be stored. An event carrying value abc will follow the same steps, arriving at $Tnode_{abc}$ and matching the stored subscription.

Subscriptions with prefix predicates (e.g., $s_1 = abc*$) follow the preceding algorithm and will be stored at $Tnode_{abc}$. Similarly, subscriptions $s_2 = ab*$ and $s_3 = a*$ will be stored at $Tnode_{ab}$ and $Tnode_a$, respectively. Figure 13 shows how string trees are constructed and how subscriptions are processed.

An event with value abc will follow the preceding algorithms, visiting in turn $Tnode_a$, $Tnode_{ab}$, and $Tnode_{abc}$. At each of these nodes, it will be collecting all relevant subscriptions having defined a matching prefix, such as s_3, s_2, s_1. Suffix predicates can be treated as prefix predicates by reversing the string values and applying the algorithms shown. Similarly, containment predicates can be treated as a combination of prefix and suffix predicates.

The second key idea is to exploit the underlying DHT to optimize the contents of the routing tables of *Tnodes*. Pastry, for example, can be built using the same alphabet base β (that contains all expected characters). Then, each Pastry node already has an ID that is a string value over this alphabet. Moreover, each Pastry node has a routing table aimed to support prefix-based routing: for example, the node with ID a has in the second row of its routing table nodes whose first two characters of their IDs are ai, with i being a character of the alphabet. In this way, Pastry routing table entries

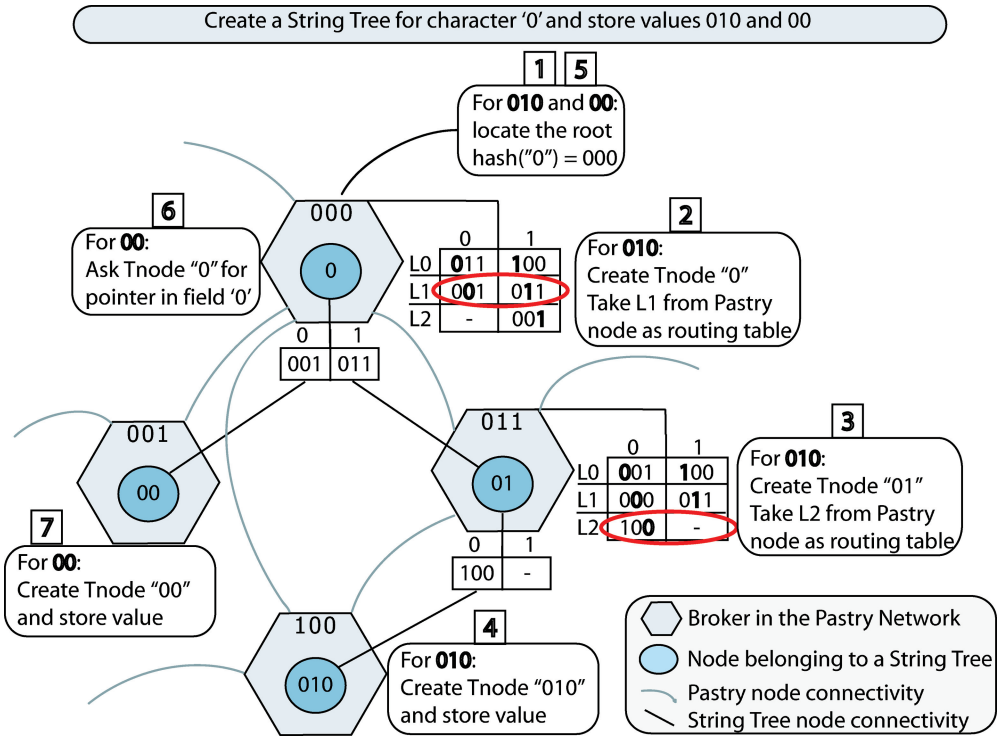

Fig. 13. Subscriptions in PastryStrings.

can be served directly to the Pub/Sub layer when the routing tables of *Tnodes* and the string trees are created.

As mentioned, PastryStrings can also support range predicates over numerical attributes. This is supported by employing *Range Search Trees* [Gao and Steenkiste 2004] that are associated with string trees and then employing the algorithms.

In PastryStrings, for string-typed attributes, we need $O(log(N))$ messages in order to reach the root of the appropriate string tree (i.e., one Pastry lookup) and then at most $O(log(N))$ messages in order to locate the Tnode inside the string tree that will accommodate the subscription ID (i.e., one message per string character), yielding a total of $O(log(N))$ messages. For numerical-typed attributes and range subscriptions, if the mean size of ranges is r, the range decomposition and string translations used in the embodied Range Search Tree results in $O(log(r))$ string values. Thus, $O(log(r) \times log(N))$ messages are required for storing a numerical range. $O(log(r))$ is expected to be small compared to $O(log(N))$ in real-life Pub/Sub applications, with range sizes being very small compared to N. Thus, we could view $O(log(r))$ as a relatively small constant. Event processing in PastryStrings requires a lookup to reach the root of the tree and then to follow a path from the root to node storing the string value in the event. In general, for each attribute of the event, $O(log(N))$ messages are required in order to collect the matching subscriptions.

4.2. The Expressiveness Dimension

The expressiveness of Pub/Sub systems specifies the granularity of the subscriptions patterns. We identified two main classes of Pub/Sub systems: topic based and content

based. Although the structure of the underlying P2P overlay has some impact on the two previous dimensions of the design space, it has a crucial role when expressiveness is considered. We now turn our attention to the following question: given the expressiveness requirement for a PUB/SUB system, which structure can best achieve it? It is clear that the two notions of structure and expressiveness are highly interrelated and are both concerned with analyzing to what extent the overlay structure and the PUB/SUB structure (guided by the targeted expressiveness) fit.

Surveying PUB/SUB systems depending on the degree of expressiveness they offer in their subscription patterns, one can observe that a few structures naturally emerge, typically trees, rings, and clustered graphs. Interestingly enough, such structures are almost similar to the ones provided by P2P overlays. In the sequel, we analyze the expressiveness—that is, the requirements of topic-based and content-based PUB/SUB systems—and put it in perspective with P2P overlay structures.

4.2.1. Topic-Based Pub/Sub Systems. In topic-based systems, subscribers register for a *topic* and expect to be asynchronously notified of any event published for that topic [Sandler et al. 2005; Patel et al. 2009]. This represents a fairly simple paradigm where every event of a topic should be disseminated within a group of interested subscribers. The first interesting note from this observation is that an application-level multicast system fulfills these requirements [Yeo et al. 2004]. The main difference between a mere application-level multicast system and a topic-based PUB/SUB system is the fact that the PUB/SUB system should support many topics. This in turn adds a dimension to scalability and flexibility: not only should a scalable topic-based PUB/SUB system be scalable in the number of subscribers per topic, but it should also be scalable in the number of topics. In addition, the PUB/SUB system should be able to efficiently support topics/groups of various sizes. Nonetheless, most application-level multicast systems can be used as a basic building block to build a fully fledged scalable PUB/SUB infrastructure [Hua Chu et al. 2002; Banerjee et al. 2006; Castro et al. 2003; Gupta et al. 2006].

Specifically, implementing a topic-based system requires (1) from the subscriber's point of view to locate and join a given topic's group and (2) from the publisher's point of view to locate and disseminate events to a given topic's group. When considering the dissemination of events in a large group of nodes with both low delay and low overhead requirements, trees come naturally to mind. In addition, locating efficiently an identifier (say a topic name) fits the publishers and subscribers requirements. In this space, structured overlay networks appear as an adequate candidate, provided that there is an efficient mechanism to repair the trees in case of failures, since (1) trees can easily be built and maintained over DHTs and (2) the exact-match interface of DHTs enable the easy and efficient location of any ID (including that of a group) based on a key identifier.

Tree-Based Pub/Sub Systems over DHTs. Scribe [Castro et al. 2002] and Bayeux [Zhuang et al. 2001] fall in the tree-based category and are implemented over a structured overlay network (respectively, Pastry and Tapestry). We chose to detail Scribe in this section, but Bayeux follows a similar process. In Scribe, a tree is built for each topic on top of Pastry as the union of the Pastry routes from all subscribers to the root of the topic tree. The root of the topic tree is the node responsible for hosting the topicID in the DHT. Typically, a subscribe procedure is processed as follows: a subscribe message is sent from the subscriber node to the node responsible for $key = topicID$. On each Pastry node encountered en route, if the node already belongs to the topic tree, then the message sender (the immediate previous hop) is added to the forwarding table for that topic at this intermediate node. If, on the other hand, the intermediate node is not part of the tree, its forwarding table for that topic is created (if this is the first time that a node is traversed by an operation regarding that topic) and the message is forwarded

by that node to the next hop along the Pastry route. This reverse path forwarding process saves messages as the subscription request stops as soon as a node belonging to the topic tree is encountered. When an event is published on a given topic, a Pastry message is sent to the root of the tree, and the event is disseminated along the branches of the tree from the root, following the nodes' forwarding tables. Scribe fully leverages the scalability properties of Pastry as well as its efficiency coming from its network awareness. In Scribe, there is no extra overhead, as the Scribe structure is directly mapped onto the Pastry infrastructure: a subscribing operation has the performance of the Pastry routing operation—that is, $O(log(N))$ messages—in a network of size N. Further, this cost might be lower than a routing operation, as the subscription consists of a message to be routed from the subscriber to the topic root, but the subscription stops as soon as a node already belonging to the topic is encountered. In terms of event propagation, Scribe requires $O(log(N) + r)$ hops, where r is the size of the group/topic. Bayeux has a similar cost. Scribe achieves (1) low publication delay, as the branches of the trees are Pastry routes between subscribers and the root; therefore, not only is an event disseminated in a logarithmic number of steps (in the size of the system) but it is also with a good ratio of overlay hops over IP hops(1.6 measured on average in Rowstron and Druschel [2001a]); (2) low overhead as the natural load-balancing properties of Pastry are automatically embedded in the tree; and (3) low network overhead as the network-awareness properties of Pastry ensure that events are replicated low in the tree, reducing the number of duplicates. Scribe is able to balance the load such that a large number of subscribers and topics can be supported. In Scribe, the trees are exactly mapped onto the Pastry structure. Scribe could be implemented with any DHT providing a routing operation. Yet, since Scribe is able to fully leverage the network awareness of Pastry, Scribe trees when implemented on top of Pastry benefit from low latency and low network traffic.

In order to cope with node failures and dynamics (i.e., node arrivals and departures, also referred to as churn), each node in Scribe is in charge of monitoring its parent in each tree. To this end, each node periodically sends heartbeat messages to each of its descendants. When a node fails to receive the heartbeat from its parent in the tree, it suspects a failure and triggers a routing operation through Pastry to fix the tree. In that respect, Scribe relies on the resilience properties of Pastry. Indeed, the Pastry routing tables are used to route around failures and find new internal nodes to repair the trees. In the case where several descendants detect a failure of a node, one only will trigger the repair, which will be leveraged by the other descendants thanks to location awareness. Effectively, if two nodes were connected to the same parent for a given topic, this was due to their network proximity; therefore, the new parent, found through Pastry, for one child is very likely to be the one that the second child would have found as well. Thus, repairing the tree is needed only once for the first node triggering the repair [Castro et al. 2002]. When the root of a tree fails, a new root is chosen through the routing operation of Pastry, which will pick the node whose identifier is the numerically closest to the topic identifier. Note that Scribe provides only best effort reliability: ensuring event persistence is left to the application developer. No default mechanism is implemented in Scribe to ensure that a new joining node will receive the past events on that topic if the root has failed, nor that the events published during the repair are not lost.

Topic dynamics, namely the fact that topics are created and deleted, does not harm the system. Topic creation is as efficient as a Pastry routing operation. When a topic is explicitly deleted, the root is notified and floods the deletion message along the branches of the tree to garbage collect the associated data structures.

A Clustering-Based Approach: Tera. Tera [Baldoni et al. 2007] also supports a large number of topics on the same infrastructure, and topics may be of various sizes. Tera

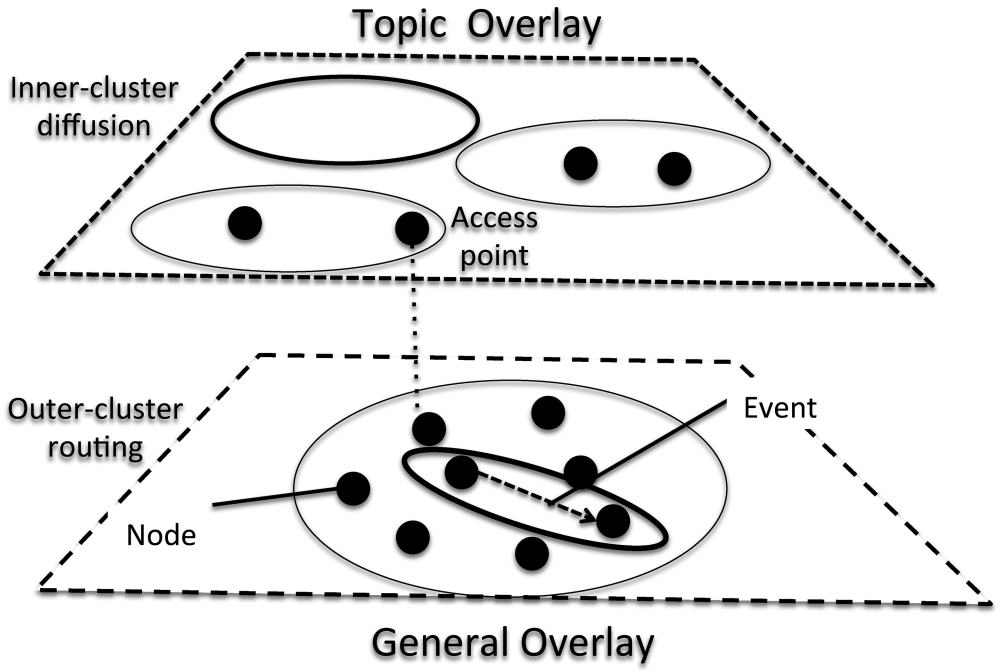

Fig. 14. Tera.

relies on a clustering protocol to connect nodes (subscribers and publishers) interested in the same topic. The dissemination of events is ensured in an epidemic manner within each cluster [Demers et al. 1987]. Basically, a gossip protocol where each node picks k of its neighbors to forward the event is implemented, providing a robust dissemination protocol. The epidemic approach introduces some overhead in stable systems when compared to trees, generating some redundancy. However, it provides a natural resilience to failures and dynamics. Note that within clusters, trees could also be easily implemented by sending a message using the gossip protocol from a source in the cluster. Each peer could then take as a parent in the tree the first node through which it received that message. In this sense, the specific structure within each cluster is left open. Tera is illustrated in Figure 14. Clusters in Tera emerge over time, starting from an unstructured overlay network of arbitrary topology. In order to locate a specific cluster, each node is equipped with a per-topic Access Point Table (APT). The cluster for a topic can be accessed directly from this table. Gossip protocols are employed to maintain the APTs and the overlay network and to cope with the dynamics of topics and nodes. In each node, the APT represents a uniform sample of the set of active topics. In addition, each access point is a uniform sample of the subscribers to that topic (ensuring load balancing). The use of a gossip protocol ensures the freshness of the entries in APTs. Topics are located (by publishers and subscribers) through a random walk, until they discover a node whose APT contains an entry for the desired topic. The fact that access points to any topic are uniformly replicated within the network limits the length of the random walk.

In contrast to Scribe, the scalability of which is directly inherited from Pastry, Tera leverages the flexibility of the underlying unstructured overlay to design a scalable PUB/SUB system while balancing the load among the nodes. Tera provides only best-effort guarantees with respect to subscriptions, with the underlying infrastructure

being made resilient to failures through the random peer sampling protocol [Jelasity et al. 2007].

A Coverage Approach: SpiderCast. Spidercast [Chockler et al. 2007] builds an overlay network so that the subgraph of nodes interested in a topic is a connected component and focuses on limiting the degree of each node. The diameter of a topic graph remains low to ensure a low latency during events' dissemination. The intuition behind SpiderCast is that each node should be connected, for each topic it is interested in, to K neighbors, with K being a parameter of the system. SpiderCast combines two heuristics to achieve this goal: the *greedy coverage* heuristic selects a neighbor minimizing the number of topics, K_g, which are not yet covered, whereas the *random coverage* randomly selects a node whose addition would reduce the number of topics not yet K_r covered. Setting K_r to 3 results in ensuring that nodes of the same topic form a connected component, whereas the greedy coverage leverages the correlation in interest common in some PUB/SUB workloads (e.g., RSS). SpiderCast, like Tera, does not depend on any specific P2P structure.

4.2.2. Content-Based Pub/Sub Systems. Obviously, content-based PUB/SUB systems are far more expressive than those that are topic based. This in turn imposes more constraints on the structure of the PUB/SUB system. Increasing the expressiveness of a PUB/SUB system introduces the need to view the underlying overlay structure at a finer grain. As shown earlier, DHTs have been often used to implement topic-based PUB/SUB systems that could fully leverage their scalability and efficiency properties [Sandler et al. 2005; Castro et al. 2002]. This follows directly from the exact-match functionality supported explicitly by DHTs, since essentially the DHT's lookup functionality is enough to support topic-based systems. However, the issues are much more complex when called to provide support for content-based systems. The main reason is that for a content-based PUB/SUB system to be effective, it must rely on a structure matching the multiattribute event/subscription schemas and the complex subscription predicates. Typically, natural candidate structures are trees and multi-dimensional spaces.

Tree-Based Systems. Let us concentrate on a popular example system. Earlier, we highlighted PHTs, a tree-based approach to support range subscriptions. Recall that the structure of the DHT and the structure of the PHT-based PUB/SUB system do not match: PHTs solely rely on the lookup functionality of the DHT to construct trees completely independent of the underlying DHT structure. This implies that PHTs do not exploit the actual structural properties of the underlying P2P system. As a consequence, the PUB/SUB logic must develop and maintain extra structures and functionality beyond what the P2P overlay has been designed to offer. This includes, of course, the tree structure itself but in addition includes additional pointers to link together nodes holding queried items (such as leaf nodes in PHTs) and complex algorithms for traversing these links to reach all values.

The same holds for other tree-based approaches as well, from Range Search Trees [Gao and Steenkiste 2004] to PastryStrings [Aekaterinidis and Triantafillou 2006]. The central conclusion is the need to build extra structures (on top of the overlay structure) and additional complex functionality to support complex subscription predicates. Yet, it should be noted that PastryStrings provides a much higher expressiveness than other approaches, accounting for both numerical and string complex predicates.

Multidimensional Spaces. Unlike tree-based approaches, multi-dimensional spaces are natural candidates to implement content-based systems, as the dimensions of the space can be mapped to the attributes of the PUB/SUB schema. This provides a natural support, for example, for range queries.

It is worth first focusing on Mercury [Bharambe et al. 2004], which supports range queries using several ring-based DHTs, such as Pastry or Chord. Such an approach can be used to implement a content-based PUB/SUB system [Bharambe et al. 2004]. In Mercury, there is one DHT per attribute. Load balancing is ensured by a monitoring system, tracking the popularity of attributes and values. Although Mercury is efficient and scalable to some extent, it is clear that to implement Mercury using DHTs requires additional complexity. This is due to the fact that there is an intrinsic mismatch between the PUB/SUB system structure and the overlay structure. Specifically, the ring structure is one-dimensional, used to accommodate values of a single attribute; on the contrary, the PUB/SUB schema is multidimensional. Mercury-like systems attempt to implement a d-dimensional space using d ring-based DHTs. This approach shows that this is possible, but the cost grows linearly with the number of attributes.

Although multiple ring-based DHTs are expensive to accommodate, there exist DHTs that rely on and are structured using multidimensional spaces. Prime examples of these DHTs are CAN [Ratnasamy et al. 2001] and VoroNet [Beaumont et al. 2007], which partition the space among the nodes so that each node is in charge of storing the subscriptions falling into that space. Several divisions of the space can be considered. CAN implements a d-dimensional Euclidean space, whereas VoroNet peers are organized in an attribute space according to a Voronoi diagram. A Voronoi diagram splits the space into cells so that any point in the cell is closer to the centre of that cell than to any other cell centre. The main difference between CAN and VoroNet is that space partitioning in CAN must be explicitly handled, whereas in VoroNet it is achieved naturally while maintaining the properties of a Voronoi diagram. Although this may lead to costlier node-join operations, it ensures better load balancing in the system. Both CAN and VoroNet can be used as a basis for content-based PUB/SUB systems. In these DHTs, the mapping between the PUB/SUB structure and the the overlay structure is more natural and a better fit for supporting range predicates, even though both approaches might have scalability issues in case of churn. For example, computing the exact Voronoi tessellation in VoroNet might be computationally intensive.

As mentioned previously, Meghdoot [Gupta et al. 2004] relies on CAN. In Meghdoot, the PUB/SUB structure is directly mapped onto the CAN structure. Indeed, Meghdoot relies on a $2n$-dimensional Cartesian space, implemented using CAN, where n is the number of attributes. This is an example, where the PUB/SUB structure exactly matches the DHT structure. Yet, note that only half of the structure (below the diagonal in Figure 10) is used during normal execution, with the other half of the space managed only for robustness. Please note that unlike other approaches such as PHTs and Mercury, no additional structures (i.e., additional rings and trees) need to be built and maintained in order to support complex queries and that the functionality for processing subscriptions and events is much simpler given the multidimensional DHT's functionality.

Unlike the systems we have examined so far, Sub-2-Sub [Voulgaris et al. 2006] implements a content-based PUB/SUB system on top of an unstructured overlay network and heavily relies on gossip protocols. The expressiveness required by the PUB/SUB system and user subscriptions collectively (which defines the 'structure' of the PUB/SUB system) fully guide the characteristics of the overlay structure. As such, Sub-2-Sub is an example of a system where the structure of the PUB/SUB system is the dominant factor and the structure of the overlay is adapted to fit these requirements. Sub-2-Sub supports range predicates as well as exact-match operators on attributes. In Sub-2-Sub, peers are subscriptions. The basic idea is that subscribers to the same events are automatically clustered, leveraging the overlapping intervals of range subscriptions. Sub-2-Sub is illustrated in Figure 15, showing the three types of links in

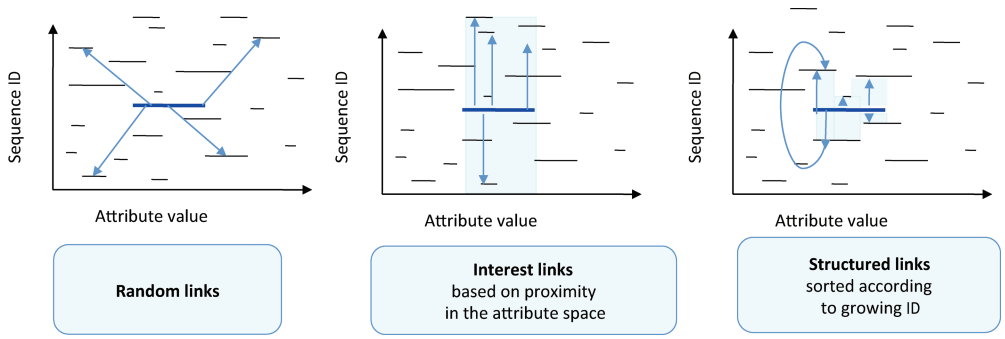

Fig. 15. Sub-2-Sub structure.

Sub-2-Sub. A random peer sampling protocol [Jelasity et al. 2007] assigns each peer a set of random neighbors (a dynamic sample of the network). A second clustering gossip protocol clusters semantically close subscriptions so that subscribers are connected according to their similarity in their subscription (central part of the figure). A greedy routing algorithm ensures that events reach the relevant cluster containing the subscribers to which the event should be delivered. Finally, to enable a fast dissemination within a cluster, a ring structure is created between subscribers (structured links).

4.3. The State Dimension

As explained previously, stateful approaches utilize one or two different types of state. First, state can refer to the meta-information needed to build specialized structures, such as trees and grids, which are used for routing event/subscription messages among the members of the structure. Second, state can also refer to the meta-information, in the form of per-link filters, employed by each peer in order to perform in-network, content-based routing. Finally, a related issue is whether state maintained by peers at the network layer is exported or made available for manipulation at the PUB/SUB layer.

The DHT-independent approaches we outlined, namely those based on PHTs, DHT-Strings, and O-P DHTs, are obviously stateless, since they require neither specialized structures, nor per-link state for event and subscription processing.

Among the DHT-dependent approaches, Hermes and Rebecca depend on special per-link filters being established and maintained for proper content-based routing of events. Thus, during event dissemination towards the rendezvous node, the event follows the reverse subscription paths (setup by subscription filters) if it discovers a matching filter at some node. This is also true for subscriptions and hence both obviously stateful. PastryStrings is also classified as stateful, but this is because it maintains explicitly a forest of trie structures and the relevant parent-child relationships. On the contrary, Meghdoot is an example of a DHT-dependent system that we classify as stateless. This is because its fundamental functionality for subscription and event processing does not depend on any per-link state at the PUB/SUB layer and on any additional structures for routing purposes.

We note that many stateful approaches require that the state maintained at the underlying routing layer (such as which are the neighbors of a peer) be exported to the PUB/SUB logic. All content-based PUB/SUB systems associating subscription-predicate filters per each neighbor link at a peer fall in this category. Typical examples of this are Rebecca and Hermes.

Tree-based, topic-based systems, like Scribe and Bayeux, are obviously stateful, given their reliance upon the tree structure to reach topic subscribers. Similarly,

Table I. Topic-Based Systems Summary

Topic-based system	Network independence	Structure	State	Locality awareness	Support for fault tolerance
Scribe	DHT dependent	Tree	Stateful	Yes	Best effort reliability (no support for subscription persistence)
Tera, Spidercast	Independent (unstructured)	Clusters	Stateful	No	Best effort reliability (no support for subscription persistence)
Rappel	Independent (unstructured)	Clusters	Stateful	Yes	Best effort reliability (no support for subscription persistence)

clustering-based, topic-based systems, like Tera, and coverage-based, topic-based systems, like Spidercast, are also classified as being stateful. In Tera, peers explicitly maintain information about other peers that are known to belong to specific topics. This can be viewed as maintaining links with topic labels. Similarly, Spidercast builds overlays where any node that has subscribed to a topic has a minimum number of links to other nodes belonging to the same topic. Since a node in general belongs to several topics, this is functionally equivalent to having each maintain links to other nodes with a per-link label that describes a topic to which that node belongs.

Sub-2-Sub, a content-based system, is also classified as stateful for the same fundamental reasons that Tera (a topic-based system) is classified as stateful: the need to maintain and route to specific clusters of nodes. Clustering in Sub-2-Sub attempts to group together peers with similar expressed subscriptions. For this, a notion of distance similarity is employed and peers maintain links labeled with their distance, which, in turn, is utilized when routing an event to the clusters of interest. Further, additional links connect peers that belong in the same cluster, which can be viewed as links with the cluster's ID as labels.

With respect to our previous discussion on structure and expressiveness, it is worth noting that the two types of state we have identified relate to it as follows. Specialized structures maintained at the PUB/SUB level typically define this state, such as trees, and grids. Note that this type of state is necessarily developed and employed in order to support increased expressiveness—that is, subscriptions with complex predicates—over overlay structures, such as DHTs, which provide support for exact-match operations only. The second type of state in stateful approaches—that is, the per-link subscription filters maintained at overlay nodes—has as its primary purpose the improvement of performance during subscription and event dissemination in two respects: first, by reducing the length of paths to be traversed by subscriptions and events and, second, by alleviating bottlenecks at popular DHT nodes.

5. DESIGN SYNTHESES

In this section, we focus on the interrelationships among the different design dimensions, as they have been adopted in the PUB/SUB systems we have covered so far. Tables I and II summarize the characteristics of the main topic-based and content-based PUB/SUB systems, respectively, that we surveyed in this article.

Stateless and DHTIndependent Approaches

At first, we emphasize that there is a strong correlation between DHT-independent approaches, as published, and stateless approaches. This is as expected given that

Table II. Content-Based Systems Summary

Content-based system	Expressiveness	Network independence	Structure	State	Locality-awareness	Support for fault tolerance
PHT, O-P DHT, DHT Strings	DHT independent	Range numerical predicates	Tree	Stateless	No	Best effort
PastryStrings	DHT dependent (Pastry, Tapestry, Bamboo)	String and numerical predicates	Tree	Stateful	Yes (DHT)	Best effort
Meghdoot	DHT dependent (CAN)	Range attribute based (numerical)	Multidimensional space	Stateless	No	Ensure subscription persistence
Sub-2-Sub	Independent (unstructured)	Range attribute based (numerical)	Multidimensional space	Stateful	No	Best effort
Hermes	DHT dependent (Pastry)	Attribute based	Tree	Stateful: per-link filters	No	Subscription persistence
Rebecca	DHT dependent (Chord)	Attribute based	Tree	Stateful: per-link filters	No	Best effort

DHT-independent PUB/SUB approaches, such as those based on PHTs, DHTStrings, and O-P DHTs, do not perform specialized in-network routing and therefore do not need state associated per each link at each intermediate peer. However, we stress that DHT independence is neither a necessary nor a sufficient condition for statelessness. To explain this, we first make an interesting but subtle point. Some of the DHT-independent approaches presented earlier, notably PHTs and DHTStrings, build an *implicit* tree structure: at any peer (tree node), the children IDs can be derived automatically. In PHT, based on the target, a parent peer knows the label of the appropriate child to contact (which is equal to the parent's label augmented with a 1 or a 0, depending on the target). In DHTStrings, when an event e with value ABCD appears at a node, the node will send e to nodes h(A), h(AB), h(ABC), h(ABCD)—in essence, these nodes form a character-string trie. The interesting point to note is that these approaches chose not to maintain explicit state for each node's children. However, they could have chosen to maintain an explicit tree structure, whereby each node explicitly maintains the IP address of each of its children. This would have resulted in faster routing. Instead, a DHT route request is issued per intermediate peer en route to the target node, making routing more expensive but avoiding the overheads and complexity associated with the maintenance of extra state, as outlined in Section 3.

Stateless and DHT-Dependent Approaches

In addition, note that there exist DHT-dependent PUB/SUB systems that are stateless. Meghdoot is a prime example, since its fundamental functionality does not depend on any per-link state at the PUB/SUB layer and on any additional structures for routing purposes. Here, we wish to point out that for improved query and storage load-balancing and fault-tolerance purposes, Meghdoot does introduce extra state at the nodes, in the form of the IDs of heavily hit nodes and additional routing alternatives, for overwhelmed links, for example. We choose to disregard such state for the reason that it is not fundamental to the PUB/SUB functionality, and as such it can be offered at a higher application layer concerned with fault tolerance and load balancing.

A further interesting point here is to recall the fact that Meghdoot is built over a DHT offering an m-d space (i.e., CAN) that facilitates a perfect mapping between the m-d PUB/SUB schemas and the overlay structure. In essence, this is what precludes the need for the PUB/SUB layer to maintain additional state.

Mercury, on the other hand, occupies a unique point in this respect: it does not maintain at the PUB/SUB layer additional state. However, it creates several overlays (one ring per attribute). In essence, it replicates overlay state in order to avoid PUB/SUB state.

Stateful and DHT-Dependent Approaches

In general, a strong correlation exists between DHT-dependent approaches, as published, and stateful approaches. Hermes and Rebecca depend on special per-link filters being established and maintained for proper content-based routing of events and subscriptions. PastryStrings also maintains explicitly a forest of trie structures. However, note that, in principle, PastryStrings could maintain implicit trie structures: for instance, when handling an event with value *abc*, the event processing algorithm will visit in turn $Tnode_a$, $Tnode_{ab}$, and $Tnode_{abc}$. The IDs of these nodes are easily identified using the DHT hash function, and hence no explicit state is required to perform the routing. However, this would have resulted in losing the benefits associated with exploiting the underlying Pastry's state and prefix-based routing, since instead of sending a message to an IP address directly, an overlay route message would be required. Similar to PastryStrings, Scribe maintains a set of explicit tree structures (one per topic). Like PastryStrings, Scribe maintains explicit state, coupling it tightly with that of Pastry, so as to enjoy the benefits from Pastry's self-organization and efficient routing.

We also emphasize that many stateful approaches require that the state maintained at the underlying routing layer (such as which are the neighbors of a peer) be exported to the PUB/SUB logic. All content-based PUB/SUB systems associating subscription-predicate filters per each neighbor link at a peer fall in this category. Typical examples of this are Rebecca and Hermes. In principle, though, the state required for such systems at the PUB/SUB layer does not force the design to become DHT dependent. That is, these PUB/SUB systems just rely on the underlying network layer to export to them neighbor information; how these neighbors are defined and how messages are routed to them is of no consequence to the state being maintained at the PUB/SUB layer.

Structure and EExpressiveness

The exact-match interface of DHTs naturally fits PUB/SUB systems with exact-match subscriptions and topic-based systems and enables the inheritance in the PUB/SUB infrastructure of the scalability and efficiency of the underlying P2P overlay. However, preserving these properties for content-based PUB/SUB is much more challenging. The main reason is the general purpose nature of DHTs: they have been introduced for efficient lookup operation. However, the graph structure that they achieve has been optimized for such an operation with load-balancing and self-organization concerns in mind. Therefore, most DHTs are generally incompatible with content-based PUB/SUB systems. To avoid the related problems, typically DHT-based approaches introduce additional state (in the form of trees, grids, or replicate overlay state) and additional complex functionality to deal with complex subscription predicates. This is typically the case of Mercury, the design of which becomes extremely complicated to account for load balancing. There is one exception to this—Medghoot—in which the PUB/SUB space perfectly matches the P2P CAN space.

On the other hand, the flexibility offered by unstructured graphs and the responsiveness of gossip protocols may be key to resolving some of the above difficulties. This follows, since the underlying overlay infrastructure imposes fewer constraints. Effectively, unstructured networks do not provide any functionalities more than flooding and an unstructured overlay. This offers more flexibility to PUB/SUB systems. In fact, as we have seen with the Sub-2-Sub system, the actual overlay structure can be adapted to fit the expressiveness requirements, as defined by user subscriptions. Nonetheless, complex functionalities are embedded in the PUB/SUB system, which needs to provide scalable solutions for routing for example.

One of the main lessons from this study is that the structure of the underlying P2P infrastructure is crucial for PUB/SUB scalability and tightly coupled to the expressiveness dimension. It should be noted that the appropriateness of any overlay may also largely depend on the type of PUB/SUB system considered: some overlays, for example, may be a better match with respect to topic-based or content-based systems, stateful or stateless, or network (in)dependence. Further, the primary performance metrics and emphasis that the designers are placing on them may influence the appropriateness of specific overlay structures. For example, by employing as the underlying P2P overlay network a DHT structure, a design could fully leverage the DHT's routing scalability and efficiency. On the other hand, insisting on employing a specific overlay structure may impose a use of the underlying interface in ways that were not anticipated when the overlay was designed, increasing complexity and resulting in an artificial binding of the PUB/SUB logic and the network logic, which in turn may hamper efficiency and scalability. Finally, a core question in this space is to what extent a PUB/SUB system can leverage the flexibility of unstructured overlays to implement scalable and efficient solutions—that is, to what extent the good properties of the P2P overlay (respectively, of a PUB/SUB system) are preserved upon such a mapping.

Conclusions. PUB/SUB systems' descriptions, as published, typically fail to explain the implications of combinations of their design decisions. It should be clear from the discussion that analyzing systems along the design dimensions we have identified can, on the one hand, bring to the surface interesting system internals and, on the other, expose alternative designs and versions of these systems catering to specific desiderata.

A new design, for instance, may opt for being completely stateful/stateless or just maintain a subset of the different state types we have identified, reconciling the trade-offs between maintenance overheads and event/subscription processing performance. Or, even further, a new design may opt for a completely stateless solution, delegating the responsibility and state necessary for achieving additional performance goals (such as load balancing and fault tolerance) to another layer above the PUB/SUB logic.

As another example, the relationships between the state and the DHT-(in)dependence dimensions and the effect of related design decisions made in popular systems should now be clear. We have already pointed out which stateful DHT-dependent approaches can be transformed to stateless, and vice versa, along with the related trade-offs. Similarly, we have identified stateful, DHT-dependent approaches that could readily be transformed to stateful, DHT-independent approaches.

Finally, the interrelationships between PUB/SUB expressiveness requirements and overlay structure have been shown to lead to stateful designs in many cases and have identified designs where this does not hold, such as the Meghdoot system and (partly) the Mercury system.

Delivering guidelines to a P2P PUB/SUB developer is extremely challenging, even after having explored the design space. We now provide some insights that could help define the right P2P infrastructure and the right PUB/SUB design. Note that this represents only our vision of the landscape. The first distinguishing characteristic is whether the considered system is a content-based or a topic-based system.

If the goal is to implement a topic-based system, it is clear that the exact-match interface of DHT-based systems, along with its efficient routing, fits large-scale topic-based systems very well. In that spectrum, a tree-based approach, such as Scribe or Bayeux, is probably the best choice. This is even more true if the underlying P2P infrastructure can be leveraged for other functionalities, such as search engines or storage systems. The only reason an unstructured solution could be preferred is an extreme unbalance in the system. For instance, if only a few topics are extremely popular, the nodes involved in those popular trees would be overloaded. There is no simple algorithm to solve this issue.

If the target is a content-based system, again, the choice of the right PUB/SUB system and P2P infrastructure really depends on the goal. One of the main lessons that can be drawn is that if the P2P structure matches the PUB/SUB space exactly, then this can be leveraged such as in Medghoot. Yet, this is a particularly nonflexible approach. For instance, increasing the number of attributes requires to rebuild the system from scratch. If the goal is great applicability, the stateless and network-independent solution should be preferred, such as PHT, OP, or DHTStrings. As for topic-based systems, if a DHT is up and running and is already leveraged for other purposes, an approach such as PastryString could leverage the good properties of the underlying DHT. The gossip-based solutions are very promising because they leverage the high flexibility of unstructured networks. They are much simpler to implement and usually provide a more balanced system at the price of a higher overhead.

6. CONCLUSION

The PUB/SUB paradigm has emerged as the ultimate scalable communication paradigm. Concomitantly, P2P networks have been recognized as the ultimate scalable

infrastructure. In this article, we have delved into the issues associated with constructing XLPUB/SUB systems, leveraging the scalability promises of P2P networks.

In the past decade, there has been a large amount of research effort in this area. Specific approaches dealt with key problems and provided interesting solutions. Studying related literature in depth, one can notice drastic differences in assumptions made and in selecting the desirable systemic features. With this article, we attempted to put things in perspective and offer a global understanding of the available fundamental design features, their interrelationships, related trade-offs, and an understanding of the position occupied by each solution in the design space. We did not mean to be exhaustive.

We first identified some fundamental contributions made by researchers prior to the P2P PUB/SUB research domain, who found themselves in several subsequent P2P PUB/SUB systems. We subsequently defined the design spectrum as consisting of four core design dimensions, namely network (in-)dependence, statefulness, and expressiveness, with all of them tightly coupled to the P2P overlay structure. We proceeded to explore the design space, studying each dimension by looking at well-known approaches and how they are positioned against each dimension. We further brought to the surface the correlations existing between various design choices, discussed their necessity and implications, and showed alternatives, where possible, in order to facilitate the reconciliation of trade-offs. Following that, we paid attention to performance and fault-tolerance issues, identifying how the performance characteristics of various approaches depend on their position in the design spectrum.

We hope that this work can contribute towards a global understanding of the issues in XLPUB/SUB systems, by allowing readers to understand the design alternatives and their implications, to see how they are implemented in well-known approaches, to understand in depth the relation of different works in the area and their relative advantages with respect to specific design desiderata, and to synthesize different variations of designs found in the literature.

There are several key issues that would also deserve further investigation. Firstly, PUB/SUB systems usually assume simple text messages. Dealing with bandwidth-consuming content such as images or video may have a great impact on the design. Secondly, we have not addressed resilience to dynamics (churn and failures). P2P overlays are extremely well known for their ability to cope well with dynamics. For most P2P overlays, the context of PUB/SUB significantly complicates the process. For instance, how to ensure that nodes receive events that were published during the time they were offline is an important issue. Addressing precisely this problem would require diving into the details of P2P overlays to understand precisely how they handle dynamics and the guarantees they provide to upper layers. On the other hand, each PUB/SUB approach should be studied in detail to understand the impact of dynamics on the systems. In addition, resilience to malicious (Byzantine) behaviour should also be considered. There is hardly any work in that area so far.

Finally, there is a clear lack of benchmarks in PUB/SUB systems [JMS Benchmark 2008]. It is striking to realize that each PUB/SUB system is evaluated based on different workloads, be they synthetic or real. Coming up with PUB/SUB benchmarks for large-scale systems is a real challenge that should be taken up in order to be in a position to compare PUB/SUB alternatives. This is a complex issue that has yet been addressed by the community.

REFERENCES

AEKATERINIDIS, I. AND TRIANTAFILLOU, P. 2005. Internet scale string attribute publish/subscribe data networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM'05)*. ACM, New York, 44–51.

AEKATERINIDIS, I. AND TRIANTAFILLOU, P. 2006. PastryStrings: A comprehensive content-based publish/subscribe DHT network. In *Proceedings of the IEEE International Conference on Distributed Computing and Systems (ICDCS'06)*. IEEE, Los Alamitos, CA, 23–23.

AGUILERA, M., STROM, R., STURMAN, D., ASTLEY, M., AND CHANDRA, T. 1999. Matching events in a content-based subscription system. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'99)*. ACM, New York, 53–61.

ANCEAUME, E., GRADINARIU, M., DATTA, A., SIMON, G., AND VIRGILLITO, A. 2006. A semantic overlay for self- peer-to-peer publish/subscribe. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. IEEE, Washington, DC, 22.

APACHEMQ. 2013. ActiveMQ. Retrieved November 11, 2013 from http://activemq.apache.org/.

BALDONI, R., BERALDI, R., QUÉMA, V., QUERZONI, L., AND TUCCI-PIERGIOVANNI, S. 2007. TERA: Topic-based event routing for peer-to-peer architectures. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS'07)*. ACM, New York, 2–13.

BALDONI, R., MARCHETTI, C., VIRGILLITO, A., AND VITENBERG, R. 2005. Content-Based Publish-Subscribe over Structured Overlay Networks. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Los Alamitos, CA, 437–446.

BANAVAR, G., TUSHAR, C., MUKHERJEE, B., NAGARAJARAO, J., STROM, R., AND STURMAN, D. 1999. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'99)*. IEEE, Los Alamitos, CA, 262.

BANERJEE, S., LEE, S., BHATTACHARJEE, B., AND SRINIVASAN, A. 2006. Resilient multicast using overlays. *IEEE/ACM Trans. Networking 14,* 2, 237–248.

BEAUMONT, O., KERMARREC, A.-M., MARCHAL, L., AND RIVIERE, E. 2007. VoroNet: A scalable object network based on Voronoi tessellations. In *Proceedings of the International Conference on Parallel and Distributed Systems (IPDPS'07)*. IEEE, Los Alamitos, CA, 1–10.

BENDER, M., MICHEL, S., PARKITNY, S., AND WEIKUM, G. 2006. A comparative study of pub/sub methods in structured P2P networks. In *Proceedings of the International Conference on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)*. Springer-Verlag, Berlin, 385–396.

BHARAMBE, A., AGRAWAL, M., AND SESHAN, S. 2004. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*. ACM, New York, 353–366.

BHARAMBE, A., DOUCEUR, J. R., LORCH, J. R., MOSCIBRODA, T., PANG, J., SESHAN, S., AND ZHUANG, X. 2008. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*. ACM, New York, 389–400.

BHARAMBE, A., PANG, J., AND SESHAN, S. 2006. Colyseus: A distributed architecture for online multiplayer games. In *Proceedings of the Conference on Networked Systems Design & Implementation (NSDI'06)*. USENIX, Berkeley, CA, 12.

BIANCHI, S., FELBER, P., AND GRADINARIU, M. 2007. Content-based publish/subscribe using distributed R-Trees. In *Euro-Par 2007 Parallel Processing.* Lecture Notes in Computer Science, Springer, Berlin, 537–548.

BOULANGER, J.-S., KIENZLE, J., AND VERBRUGGE, C. 2006. Comparing interest management algorithms for massively multiplayer games. In *Proceedings of the ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'06)*. ACM, New York.

CARZANIGA, A., DI NITTO, E., ROSENBLUM, D., AND WOLF, A. 1998. Issues in supporting event-based architectural styles. In *Proceedings of the International Software Architecture Workshop*. ACM, New York, 17–20.

CARZANIGA, A., ROSENBLUM, D. S., AND WOLF, AL. L. 2001. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst. 19*, 3, 332–383.

CASSANDRA. 2013. Welcome to Cassandra. Retrieved November 11, 2013 from http://cassandra.apache.org/.

CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., AND ROWSTRON, A. 2002. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE J. Sel. Areas Commun. 20*, 8, 1489–1499.

CASTRO, M., JONES, M., KERMARREC, A.-M., ROWSTRON, A., THEIMER, M., WANG, H., AND WOLMAN, A. 2003. An evaluation of scalable application-level multicast using peer-to-peer overlay networks. In *IEEE INFOCOM'03*. IEEE, San Francisco, CA, USA, 1510–1520.

CHEN, J., DEWITT, D. J., TIAN, F., AND WANG, Y. 2000. NiagaraCQ: A scalable continuous query system for internet databases. In *ACM SIGMOD International Conference on Management of Data*. ACM, Dallas, Texas, USA, 379–390.

CHOCKLER, G., MELAMED, R., TOCK, Y., AND VITENBERG, R. 2007. SpiderCast: A scalable interest-aware overlay for topic-based pub/sub communication. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS'07)*. ACM, New York, 14–25.

CUGOLA, G., DI NITTO, E., AND FUGGETTA, A. 2001. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. Software Eng. 27*, 9, 827–850.

DEMERS, A., GREENE, D., HAUSER, C., IRISH, W., LARSON, J., SHENKER, S., STURGIS, H., SWINEHART, D., AND TERRY, D. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'87)*. ACM, New York, 1–12.

EUGSTER, P., FELBER, P. A., GUERRAOUI, R., AND KERMARREC, A.-M. 2003a. The many faces of publish/subscribe. *Comput. Surv. 35*, 2, 114–131.

EUGSTER, P., GUERRAOUI, R., HANDURUKANDE, S., KOUZNETSOV, P., AND KERMARREC, A.-M. 2003b. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst. 21*, 4, 341–374.

FABRET, F., JACOBSEN, A., LLIRBAT, F., PEREIRA, J., ROSS, K., AND SHASHA, D. 2001. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*. ACM, New York, 115–126.

GANESH, A. J., KERMARREC, A.-M., AND MASSOULIÉ, L. 2003. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput. 52,* 2, 139–149.

GAO, J. AND STEENKISTE, P. 2004. An adaptive protocol for efficient support of range queries in DHT-based systems. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP'04)*. IEEE, 239–250.

GARROD, C., MANJHI, A., AILAMAKI, A., MAGGS, B. M., MOWRY, T. C., OLSTON, C., AND TOMASIC, A. 2008. Scalable query result caching for web applications. *Proc. VLDB Endowment 1*, 1, 550–561.

GNUTELLA. 2013. Wego. Retrieved November 11, 2013 from http://gnutella.wego.com/.

GRYPHON. 2005. The Gryphon Project. Retreived November 11, 2013 from http://www.research.ibm.com/distributedmessaging/gryphon.html.

GUPTA, A., SAHIN, O. D., AGRAWAL, D., AND EL ABBADI, A. 2004. Meghdoot: Content-based publish/subscribe over P2P networks. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware'04)*. Springer Verlag, Berlin, 254–273.

GUPTA, I., KERMARREC, A.-M., AND GANESH, A. 2006. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Trans. Parallel Distrib. Syst. 17*, 7, 593–605.

HANSON, E., CARNES, C., HUANG, L., KONYALA, M., NORONHA, L., PARTHASARATHY, S., PARK, J. B., AND VERNON, A. 1999. Scalable trigger processing. In *Proceedings of the International Conference on Data Engineering (ICDE'99)*. IEEE, New York, 266.

HUA CHU, Y., RAO, S., AND ZHANG, H. 2002. A case for end system multicast. *IEEE J. Selected Areas in Communications (JSAC) 20,* 8.

IBM NEWS. 2013. History of IBM: 2000s. Retrieved November 11, 2013 fromhttp://www-03.ibm.com/ibm/history/history/year_2000.html.

JELASITY, M., VOULGARIS, S., GUERRAOUI, R., KERMARREC, A.-M., AND VAN STEEN, M. 2007. Gossip-based peer sampling. *ACM Trans. Comput. Syst. 25*, 3, 8.

JMS BENCHMARK. 2008. JMS Performance Benchmarks. Retrieved November 11, 2013 from http://www.codeproject.com/Articles/26461/JMS-Performance-Benchmarks.

KAFKA. 2013. Kafka 0.8 Documentation. Retrieved November 11, 2013 from http://kafka.apache.org/design.html.

KARP, B., RATNASAMY, S., RHEA, S., AND SHENKER, S. 2004. Spurring adoption of DHTs with OpenHash, a public DHT service. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*. Springer, Berlin, 195–205.

KAZAA. 2006. http://www.kazaa.com/.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. 2000. OceanStore: An architecture for global-scale persistent storage. *SIGPLAN Not. 35*, 11, 190–201.

LEHN, M., LENG, C., REHNER, R., TRIEBEL, T., AND BUCHMANN, A. 2011. An online gaming testbed for peer-to-peer architectures. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*. ACM, New York, 474–475.

LIVEJOURNAL. 2013. Homepage. Retrieved November 11, 2013 from http://www.livejournal.com.

MELAMED, R. AND KEIDAR, I. 2008. Araneola: A scalable reliable multicast system for dynamic environments. *J. Parallel Distrib. Comput. 68*, 12, 1539–1560.

OBJECT-MANAGEMENT GROUP. 1998. *CORBAservices: Common Object Services Specification*.

OBJECT-MANAGEMENT GROUP. 2000. *CORBAservices Notification Service Specification*. Retrieved November 11, 2013 from http://www.omg.org/.

OKI, B., PFLUEGL, M., SIEGEL, A., AND SKEEN, D. 1993. The Information Bus: An architecture for extensible distributed systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'93)*. ACM, New York, 58–68.

PATEL, J., RIVIÈRE, É, GUPTA, I., AND KERMARREC, A.-M. 2009. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Comput. Networks 53*, 13, 2304–2320.

PIETZUCH, P. AND BACON, J. 2002. Hermes: A distributed event-based middleware architecture. In *Proceedings of the Workshop on Distributed Event Based Systems (DEBS'02)*. ACM, New York, 611–618.

PITOURA, T., NTARMOS, N., AND TRIANTAFILLOU, P. 2006. Replication, Load Balancing and Efficient Range Query Processing in DHTs. In *International conference on Advances in Database Technology (EDBT06)*. Springer-Verlag, Munich, Germany, 131–148.

PLAXTON, G., RAJARAMAN, R., AND RICHA, A. 1997. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM, Newport, Rhode Island, USA, 311–320.

RABBITMQ. http://www.rabbitmq.comcom/. (????).

RAMABHADRAN, S., RATNASAMY, S., HELLERSTEIN, J. M., AND SHENKER, S. 2004. *Prefix Hash Tree - An Indexing Data Structure over Distributed Hash Tables*. Technical Report. IRB Tech Repor.

RAMASUBRAMANIAN, V., PETERSON, R., AND GÜN SIRER, E. 2006. Corona: A high performance publish-subscribe system for the World Wide Web. In *Proceedings of the Conference on Networked Systems Design & Implementation (NSDI'06)*. USENIX, Berkeley, CA, 2–2.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R. M., AND SHENKER, S. 2001. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*. ACM, New York, 161–172.

ROSE, I., MURTY, R., PIETZUCH, P., LEDLIE, J., ROUSSOPOULOS, M., AND WELSH, M. 2007. Cobra: Content based filtering and aggregation of blogs and RSS feeds. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*. USENIX, Berkeley, CA, 3.

ROWSTRON, A. AND DRUSCHEL, P. 2001a. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*. Springer, Berlin, 329–350.

ROWSTRON, A. AND DRUSCHEL, P. 2001b. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'01)*. ACM, New York, 188–201.

SANDLER, D., MISLOVE, A., POST, A., AND DRUSCHEL, P. 2005. FeedTree: Sharing web micronews with peer-to-peer event notification. In *Proceedings of the International Conference on Peer-to-Peer Systems (IPTPS'05)*. Springer-Verlag, Berlin, 141–151.

SKYPE. Homepage. Retrieved November 11, 2013 from http://www.skype.com/.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*. ACM, New York, 149–160.

SUN, X. 2007. SCAN: A small-world structured P2P overlay for multi-dimensional queries. In *Proceedings of the International Conference on World Wide Web (WWW'07)*. ACM, BNew York, 1191–1192.

SUN MICROSYSTEMS, INC. 2000. *Jini™ Technology Core Platform Specification.* Retreived November 11, 2013 from http://www-csag.ucsd.edu/teaching/cse291s03/Readings/core1_2.pdf.

TERPSTRA, W., BEHNEL, S., FIEGE, L., ZEIDLER, A., AND BUCHMANN, A. 2003. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS'03).* ACM, New York, 1–8.

TIBCO. 2013. Homepage. Retrieved November 11, 2013 from http://www.tibco.com/.

TRIANTAFILLOU, P. AND AEKATERINIDIS, I. 2004. Publish-subscribe over structured P2P networks. In Proceedings of the International Conference on Distributed Event-Based Systems (DEBS'04). ACM, New York.

TRIANTAFILLOU, P. AND PITOURA, T. 2003. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In *Proceedings of the Workshop on Databases, Information Systems, and Peer-to-Peer Computing (VLDB'03).* Springer, Berlin.

TWITTER. 2013. Welcome to Twitter. Retrieved November 11, 2013 from http://www.twitter.com/.

VITRIA. 2013. Homepage. Retrieved November 11, 2013 from http://www.vitria.com/.

VOULGARIS, S., GAVIDIA, D., AND VAN STEEN, M. 2005. CYCLON: Inexpensive membership management for unstructured P2P overlays. *J. Netw. Syst. Manage. 13*, 2.

VOULGARIS, S., RIVIÈRE, E., KERMARREC, A.-M., AND VAN STEEN, M. 2006. Sub-2-Sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06).* USENIX, Berkeley, CA.

YEO, C. K., LEEA, B. S., AND ER., M. H. 2004. A survey of application level multicast techniques. *Comput. Commun. 27*, 15, 1547–1568.

ZHANG, C., KRISHNAMURTHY, A., WANG, R. Y., AND SINGH, J. P. 2005. Combining flexibility and scalability in a peer-to-peer publish/subscribe system. In *Proceedings of the ACM/IFIP/UseNix 6th International Middleware Conference (Middleware'05)*. Springer-Verlag, Berlin, 102–123.

ZHAO, B., JOSEPH, A., AND KUBIATOWICZ, J. 2001. *Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing*. Tech. rep. UCB/CSD-01-1141. University of California, Berkley.

ZHUANG, S. Q., ZHAO, B. Y., JOSEPH, A. D., KATZ, R. H., AND KUBIATOWICZ, J. D. 2001. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. ACM, New York, 11–20.

Received March 2010; revised June 2011; accepted January 2013