Routing Algorithms for Content-Based Publish/Subscribe Systems

J. Legatheaux Martins, *Senior Member, IEEE,* Sérgio Duarte

***Abstract*—In content-based publish/subscribe systems, messages target a dynamic group of participants whose expressed interests match the contents of the messages. In this generalization of multicasting communication, also dubbed content-based networking, naming, binding and communication are intertwined in the same substrate. Optimal content-based routing uses dissemination trees dynamically pruned to only cover the matching subscribers. It is a complex problem that has motivated significant research efforts.**

This paper presents a compilation of the main algorithms for routing messages in distributed content- based publish-subscribe systems proposed and published in the last decade. Discussion is focused on the content-based routing problem in respect to optimality, complexity and applicability. Moreover, whenever it is appropriate, the algorithms covered are also matched to similar algorithms familiar to the networking community, setting this paper apart from other surveys on the broad topic of publish/subscribe systems.

***Index Terms*—Publish/Subscribe, Content-Based Networking, Overlay Networks, Multicasting, P2P.**

I. Introduction

PUBLISH/subscribe systems allow many parties to communicate by way of asynchronous, many-to-many message exchanges. Distributed publish/subscribe systems have been around for several years. However, due to the global impact of present-day networked systems, they have recently attracted a lot more attention due to their emergent applicability, from very large-scale scenarios with many thousand of users and spanning the whole Internet, up to the world of mobile systems and wireless sensor networks.

One of the main characteristics of distributed publish/subscribe systems is that they decouple subscribers and publishers in space and time [33] and do not require any previous binding. Participants only need to connect to *information spaces* associated with their *interests* and do not need any prior knowledge of each other.

Publish/subscribe systems can be topic-based or content-based. In *topic-based* systems, each information space is associated with a so called *topic, group or channel*. Publishers publish their notifications to a channel. Subscribers subscribe to the channel(s) they are interested in. For efficiency and convenience purposes, some systems allow subscribers to provide a semantic filter over the contents of the notifications. Systems with this extra functionality are designated as *content-based*.

Manuscript received 18 October 2007; revised 8 December 2008.

Authors are with CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal (e-mail: jose.legatheaux@gmail.com).

Digital Object Identifier 10.1109/SURV.2010.020110.00065

In the presence of large numbers of notifications, filters can ease the life of the subscribers and make the system overall more efficient by not delivering useless notifications. Since the number of different filters in the system may be very large, matching notifications and subscriptions needs to be done very efficiently. The centralized version of this problem has also been addressed by the database community as it is analogous to the problem of matching database updates with the triggers they fire [36] and to the problem of matching arriving events against a large number of persistent queries in data dissemination systems [20].

Content-based networking is a generalization of the content-based publish/subscribe model [18], [1]. In content-based networking, messages are no longer addressed to the communication end-points (e.g. network addresses). Instead, they are published to a distributed information space and routed by the networking substrate to the "interested" communication end-points. In most cases, the same substrate is responsible for realizing naming, binding and the actual content delivery.

Examples of Content-Based Publish/Subscribe Systems

Internet news feeds and similar bulletin boards are examples of early and well-known publish/subscribe systems. These are essentially topic-based systems offering a pull-based interface. On top of these, some Internet search providers have implemented more sophisticated content-based facilities, which can deliver alert messages to users, containing newly published messages matching some user defined filters, in general via e-mail or SMS. These alerts are generally sent by servers placed next to the news brokers.

The best known content-based alert systems are, for example, stock quote notifications and others like RSS feeds, bibliography publication notes, auction systems, etc. In general, they provide a formal filtering language, since notifications are often structured or semi-structured. For example, stock quote notifications allow filtering on "Symbol", "Volume", "Price" etc., i.e., on a set of typed attributes, of which most possess an order relation. Therefore, it is common for subscription filters to use expressions based on relational operators involving these attributes, such as:

$$\{\ \text{Symbol} = \text{“EDP”},\ \text{Price} < 4,\ \text{Volume} > 100000\ \}$$

A filter on the topic "cars for sale" in an auction system could be:

$$\{\ \text{Manufacturer} = \text{“BMW”},\ \text{Type} = \text{“SUV”},\ \text{Kilometers} \leq 20000,\ \text{Registered} \geq 2000\ \}$$

RSS feeds and bibliography notifications are semi-structured and encoded in XML. Thus, it is possible to express filters on the contents of certain fields (tags). For instance, filters over feeds of news concerning cooking receipts or scientific publications are exemplified by the following expressions:

$$\{\text{Country} = \text{“India”}, \text{Ingredients} \ni \text{“Fish”}\} \text{ and}$$
$$\{\text{Authors} \ni \text{“Duarte”}, \text{Keywords} \ni \text{“Event-Based Systems”}\}$$

Publish/subscribe systems can also be useful in monitoring and control scenarios. For example, air-traffic control systems require the diffusion of relevant events, such as periodic plane positions, or landing and take-offs. A continent-wide system of this kind has many different official agencies, as well as private companies, subscribing to these events for many different reasons. In this case, latency requirements are critical and a distributed solution seems the only suitable alternative [56]. This is an extreme example of a very common scenario where monitoring periodic and abnormal events is the common case. Other examples include, monitoring networks and all sorts of grids, wide area systems components state coordination, distributed games, and so on.

Another scenario where a content-based publish/subscribe model fits well is in wireless sensor networks [6]. In this scenario, sensor nodes are the publishers and sink nodes are the subscribers. Due to the strict power saving requirements of sensor nodes, they may stay mute as long as no subscriber seems interested in their sensed data. Directed diffusion [38] is a protocol allowing sink nodes (subscribers) to publish their filters (queries) which are flooded in the network. Sensor nodes (publishers) with readings that match the subscribers filters, start periodically publishing notifications that are propagated down to the interested subscribers. The language describing notifications and filters is also attribute based, as in the examples above. This is an emergent non-traditional scenario where the publishers can be orders of magnitude more numerous than the subscribers.

In more recent content-based systems, attributes in subscriptions can be used to convey contextual or quality of service indications. For example, a mobile subscriber of a monitoring system may be interested in events that took place up to a maximum distance of 4 Km from her. Or, a constrained mobile subscriber may be interested in at most 10 events per minute.

Finally, content-based networking is also closely related with attribute-based binding and naming [1]. In general, content-based communication [18] is an emergent paradigm ideally suited for a variety of application domains: publish/subscribe event notification, news and alert distribution, system monitoring, service discovery, data sharing, distributed electronic auctions, distributed games, mobile and context-dependent computing, etc.

Issues and Goals

Distributed publish/subscribe systems can be structured as client/server systems or as peer-to-peer systems (P2P). In the client/server model, publishers and subscribers are directly dependent of servers, known as notification brokers. Brokers route notifications among themselves and to their respective clients. These systems can implement a *push-based* delivery paradigm. In this case, notifications are immediately multicasted to the targeted subscribers. Alternatively, systems following a *pull-based* paradigm memorize notifications and allow their deferred retrieval by the intended recipients.

In a centralized client/server system, the critical scalability issue of the broker server is to determine efficiently which clients provided filters matching the contents of each of the incoming notifications. In a distributed system of brokers, a further requirement is that notifications must also be routed efficiently among brokers. Whereas in a P2P system there is yet another complication in that each participant can act as both a subscriber and as a publisher, in addition to all the participant nodes being required to route notifications among themselves.

When it is acceptable to have a large delay (in the order of tens of minutes) between the moment an event occurs and the reception of the notifications by the interested subscribers, then a straightforward centralized solution is a realistic option. To achieve this, a central server just needs to periodically evaluate the filters of its clients on the message flow and send matching notifications by e-mail, for instance. However, as stricter limits are imposed on the allowed latency of notifications, or the number of subscribers grows to very large numbers, a network of brokers will probably become a preferable alternative. In cases where there is no clear economic incentive to set up an expensive broker-based solution, a P2P system can become an interesting architectural alternative.

To summarize, the filtering or matching problem is the following: given a set of subscribers, with their associated filters, how to determine efficiently the subset of subscribers, or filters, that match the contents of a notification? This problem can be seen as dual to the following search problem: given a set of items (notifications) which ones have content matching a specific query (filter)? This is comparable to the problem of finding the triggers that an update to a database fires. On top of that, a distributed content-based publish/subscribe system has to solve the following problem: how to optimally route each message to the nodes with subscriptions matching its content? As such, in a distributed topic-based publish/subscribe system, notification routing is a problem similar to the multicasting routing problem and uses similar algorithms to determine and maintain the dissemination trees. In content-based systems, an extra level of complexity is introduced with the requirement that useless transmission of notifications be avoided. Usually, that mandates some way of dynamically pruning the dissemination trees to cover only the set of matching subscribers of each notification. As a general goal, an optimal routing solution must minimize routing costs and avoid message waste. Furthermore, well balanced solutions that do not incur an excessive burden on particular nodes are also favored.

This paper examines the most relevant work presented in the last decade for routing notifications in distributed content-based publish-subscribe systems. It differs from other surveys published on the broad topic of publish/subscribe systems by focusing the analysis of the content-based routing problem on optimality, complexity and applicability [33], [41], [9]. Moreover, when it is pertinent, the algorithms covered are also likened to similar algorithms developed and used by the networking community.

The next section introduces a few definitions needed to support the discussion. Section III describes an ideal optimal solution and discusses strategies for approximate solutions. Sections IV through VIII present and discuss these alternative solutions. Finally, section IX offers a qualitative comparison of the reviewed solutions and concludes with some final remarks regarding the evaluation steps presented in each of the proposals.

II. Terminology and Definitions

A *notification* is a tuple of equality attribute value pairs $(A_1 = v_1, ..., A_n = v_n)$ allowed by the language of the information space schema. It denotes a single point of the *notification space*, i.e., it denotes a member of the cartesian product of the sets of attribute values. A *subscription* is predicate made of a conjunction of *attribute constraints*, or simply *constraints*. A constraint can hold with an exact value (e.g., $A_i = v$) or a range of values in an ordered type (e.g., $A_i \in [v_{min}, v_{max}]$). A constraint over a set type attribute holds if the value belongs to the set (e.g., $v \in A_i$ or $A_i \ni v$).

If the constraint is of the form A_i ="any", which is equivalent to $True$, the value of this constraint is irrelevant and it can be omitted from the subscription. A subscription can have at most one constraint per attribute. Multiple exact values or multiple ranges over the same attribute can be modeled as a union of several distinct subscriptions.

In publish/subscribe systems involving mobile nodes, some constraints can specify context-dependent conditions. One such example is a $WithinRange < 2\ Km$ constraint, whose evaluation depends on the relative geographical positions of the publisher and the subscriber. Essentially, this can preclude some optimizations that are otherwise possible when predicate evaluation is invariant across all the nodes.

There are several other languages for expressing subscriptions. For example, values can be documents and subscriptions specify a list of keywords to be matched against the entire document. Other common used languages are SQL (Structured Query Language) or X/Path (for XML objects). However, the simple language sketched just above is popular and allows a more focused discussion of the specific problems of content-based routing without compromising generality.

A subscription S_i specifies a N-dimensional subspace of the notification space (due to restricting each attribute with a single constraint) and, depending on the context, the symbol S_i may denote the formula representing the subscription, or the subspace it specifies. If a notification n belongs to the subspace of a subscription S_i, i.e., $n \in S_i$, the notification is said to *match* the subscription. Matching requires each constraint of S_i to hold with the value of the corresponding attribute of n. Attributes of n not corresponding to constraint attributes in S_i are ignored since they are considered to always hold.

Some schemas do not require that notifications have values defined for all possible attributes. This can be modeled as if the type of each attribute included a special value known as Any. If a notification n includes $A_i = Any$, the attribute is not listed in n and it requires the constraint $A_i = Any$ to hold. Hence, a subscription with a defined constraint over attribute A_i never matches a notification where A_i is absent. Formally treating notifications not containing all attributes is more elaborate than what has been presented and is out of the scope of this paper. A schema is said *open* if it allows new attributes to be introduced dynamically.

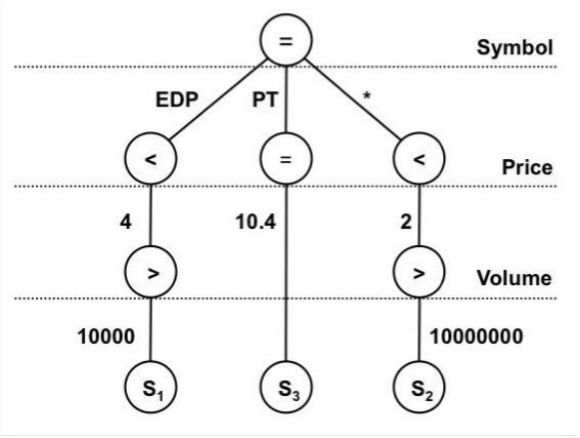

Fig. 1. The parallel search tree for subscriptions S_1, S_2 and S_3

A subscription S_i is said to *cover* a subscription S_j if $S_j \subseteq S_i$[1]. Two subscriptions S_i, S_j *intercept* if $S_i \cap S_j \neq \emptyset$.

The matching test is performed by a *filtering or matching algorithm*. The matching algorithm is a critical one in publish/subscribe systems. Chapter 3 of [45] presents a brief survey of the most relevant approximations developed by the database and publish/subscribe communities. Matching is outside of the scope of this paper. However, for illustration purposes we will briefly discuss one of the approaches, based on *Parallel Search Trees - PST* [5], [11].

In a PST, each leaf is associated with one subscription, an interior node represents a test over a single attribute and edges are labeled with constants or the value "*", meaning "Absent" or "Don't care". PSTs are rooted and organized in such a way that nodes of the same level concern the same attribute. Paths from the root to each leaf cross all the constraints that characterize the subscription associated with the leaf.

Figure 1 is an example of a PST representing the subscriptions S_1 = {Symbol = "EDP", Price $<$ 4, Volume $>$ 10000 }, S_2 = { Price $<$ 2, Volume $>$ 1000000 } and S_3 = { Symbol = "PT", Price = 10.4 }, where the attribute "Symbol" appears in the first level, "Price" in the second and "Volume" in the third. Given a notification n, it matches all subscriptions reached by a tree traversal that only follows an edge if n matches the constraint denoted by the attribute of the level, followed by the node and edge labels. Intuitively, the data structure factorizes tests common to several subscriptions and thus favors scalability, since it allows a sub-linear matching complexity in respect to the number of different subscriptions.

In a distributed publish/subscribe system, publisher and subscriber applications are represented by pure clients of a distributed client/server broker service, or by peers participating in a P2P system. In general, notifications are routed to subscribers by intermediate computing components of the pub-

[1]In a formal context, one should preferably say "if the corresponding specified subspaces verify the relation $\subseteq$".

TABLE I
EXAMPLE OF AN HYPOTHETICAL PUBLISH/SUBSCRIBE SYSTEM API.

Operation	Description
publish (n)	Publishes notification n
subscribe (S)	Subscribes with subscription S
advertise (S)	Advertises that future notifications will match subscription S
unsubscribe (S)	Unsubscribes a subscription S
feedback (m, n)	Sends message m to the publisher of notification n

lish/subscribe system (servers or peers) known as *notification routers*. In general, brokers behave as cooperating peers among themselves. In a pure P2P system without brokers, peers behave simultaneously as clients and routers, thus fulfilling the functions of brokers. In what concerns the algorithms for routing notifications, the distinction between brokers and peers is not generally relevant; reflecting that, participants will be called *router nodes* or simply *nodes*.

Nodes are associated or interested in subscriptions, either their own, or the ones of their clients. In both cases, without distinction, we will call them the *node's subscriptions*. Pure brokers are responsible for the subscriptions of their clients and for the routing of their notifications. When the broker serving a client is not important, the "closest" one is the usual choice. In this case, brokers are responsible for a set of arbitrary subscriptions. Alternatively, to cluster clients by a criterion other than network proximity, clients can be redirected to specific brokers. Subscribers in a pure P2P system are responsible for their own subscriptions and cooperate with other peers to route notifications.

Applications tap into the functionality provided by a publish/subscribe system through a wrapper interface that hides the concrete distributed architecture of the system. An hypothetical example of such API (Application Programming Interface) is presented in Table I.

The advertise and feedback operations are optional and often missing in concrete systems. The advertise operation is used to increase the efficiency of certain matching and routing algorithms. While, the feedback operation allows a subscriber to send an unicast message back to a publisher. Its main purpose is to avoid the need to fall back to an addressing mechanism outside the publish/subscribe system.

III. OPTIMAL AND APPROXIMATE NOTIFICATION ROUTING SOLUTIONS

A notification n must be routed to the group of all nodes with matching subscriptions, by a tree rooted at the publisher, preferably a shortest paths one. A correct routing solution must ensure that no *false negatives* exist, i.e., each subscriber receives all notifications that match its subscription(s). An optimal routing solution should also deliver no *false positives* or *spam*, i.e., a subscriber should only receive notifications matching its subscription(s).

A trivial solution of the routing problem consists in *broadcasting* notifications to all nodes. Broadcasting can be implemented by *flooding* which, in regard to routing, is stateless, adapts smoothly to configuration changes, increases fault tolerance and reliability and only requires a duplicate detection cache in each node if the network has cycles. Notification routing by broadcasting is the benchmarking algorithm regarding simplicity. Moreover, in a system based on a small number of brokers, each serving many clients with a dense distribution of matching subscriptions for most notifications, broadcasting is near optimal in what concerns false negatives.

However, when nodes are very numerous and a sparse distribution of matching subscriptions among different nodes is observed, network bandwidth waste and processing capacity usage (to discard false positives) can be significant. Therefore, in this scenario, the algorithm is far from being scalable and there is ample motivation to find an optimal one.

A. An Idealized Optimal Solution

Under certain idealized circumstances, namely, if nodes could rely on an optimal multicasting facility (IP Multicasting [25], [26] or an equivalent optimal multicasting overlay facility [55], [12], [21], [39]), an idealized optimal solution can be based on multiple multicasting trees, each suited for a different group of subscribers, as explained below.

Let $S = \cup\ S_i$ be the set of all subscriptions and let $\eta = \cup\ n_j$ be the set of all notifications subscribed and published by network nodes. There is a mapping $\psi : \eta \rightarrow S^*$, which maps each notification in the subset of subscriptions that match it, $S_i \in \psi(n) \equiv Match(S_i, n)$. Mapping ψ allows a set of different overlapping clusters of subscriptions $C = \{C_1, C_2, ..., C_M\}$, with $C_i \in S^*$, to be determined, such that each notification $n_j \in \eta$ matches all the subscriptions of exactly one of the clusters $C_k \in C$, or no subscription at all, and all subscriptions matching the same notification will belong to the same cluster.

As a subscription specifies a subspace of the notification space, the clustering above associates a specific subspace Sc_k with each cluster C_k. Each subspace Sc_k is contained in the subspaces of all the subscriptions belonging to cluster C_k, i.e.:

$$\forall\ 1 \leq k \leq M, \forall\ S_i \in S : [Sc_k \cap S_i \neq \emptyset] \Rightarrow [Sc_k \subseteq S_i] \quad (1)$$

Figure 2 introduces, in a two-dimensional attribute space, four subscriptions $S_1, \ldots, S_4$ and the corresponding notification subspaces. A subscription is represented by a rectangle with a label at the upper left corner. Subscriptions S_1 and S_3, as well as S_1 and S_2 intercept; S_2, S_3 and S_4 do not intercept, nor do S_1 and S_4.

Figure 3 introduces the 6 subspaces corresponding to the set of subscriptions clusters $C = \{C_1 = \{S_1\},\ C_2 = \{S_2\},\ C_3 = \{S_3\},\ C_4 = \{S_4\},\ C_{13} = \{S_1, S_3\},\ C_{12} = \{S_1, S_2\}\}$. Each C_i corresponds to the subspace of notifications that match *simultaneously* all the subscriptions belonging to C_i. Therefore, a notification will belong to exactly one of these subspaces or none at all, meaning that they are all disjoint.

Finally, a multicasting group $G_i = \textit{groupOfCluster}(C_i)$ (and a corresponding optimal dissemination tree) is associated with each of the M clusters of C, each node joins as many groups as clusters containing its subscriptions, and each publisher publishes a notification n via the group $\textit{groupOfCluster}(\psi(n))$. Since each notification is sent to only one group, and assuming an optimal multicast delivery path, this

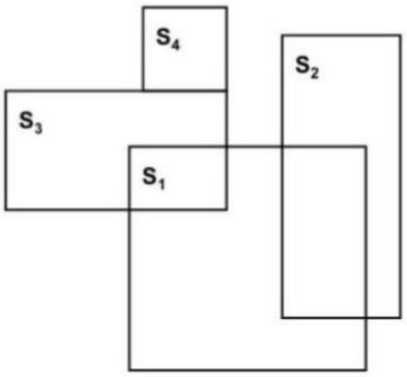

Fig. 2. Subscriptions $S_1 \ldots S_4$ and their matching notification subspaces

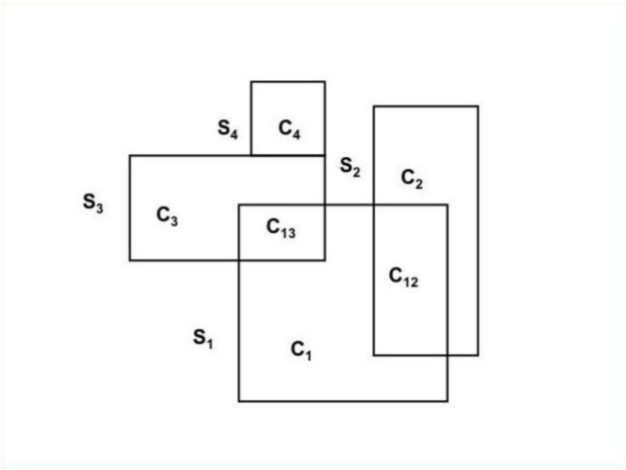

Fig. 3. Subscriptions clusters and the associated subspaces

solution is optimal in terms of network costs, it is correct since there are no false negatives, and it introduces no waste since there are no false positives. Figure 4 shows the algorithm in progress. It assumes that each node knows the subscriptions of all other nodes, computes the set of subscriptions matching the notification – $(A = 3)$ matches the subscription of nodes n_2 and n_3 – and selects the multicasting group optimally delivering the notification to the interested nodes.

In a N node system, this algorithm requires: $O(|S|)$ space for subscriptions in each node, computation of the clusters $C_1, C_2, C_3, ...$, an optimal multicasting facility providing a maximum of 2^N different groups and that each node subscribes the groups corresponding to its subscriptions. In a dynamic system, the information concerning subscriptions must be replicated and the clusters and groups dynamically calculated and maintained. As each multicast group has a non negligible cost, due to the potential huge number of required groups, the algorithm just presented is dubbed *Ideal Multicast* [46].

B. Optimal Solutions Not Requiring Multicasting Support

In order to find solutions for the content-based routing problem, researchers have tried to find practical variants of the

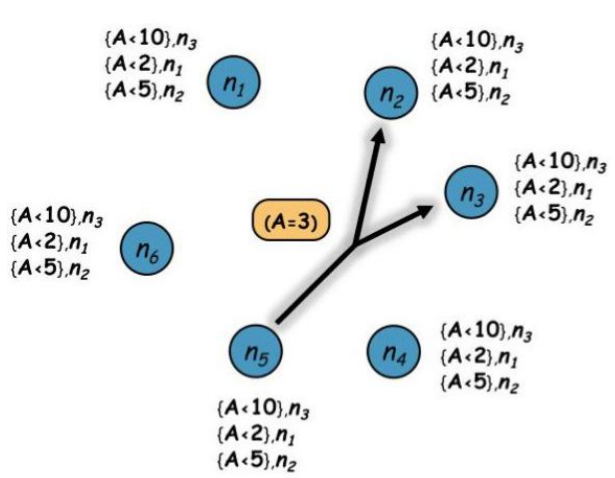

Fig. 4. Optimal routing of notification $(A = 3)$ via the multicasting group of $\{n_2, n_3\}$. Every node $(n_1, n_2, ..., n_5)$ knows every subscription in the system. This enables the computation of the groups that each node has to belong to. For publishers, it also allows them to select the multicast group implied by any given notification.

ideal solution. Some of these variants dynamically compute, for each notification, the required diffusion group.

The first alternative, still optimal, computes shortest paths spanning trees (SPT) of the nodes with matching subscriptions, rooted at the publisher. This can be done by dynamically pruning to the matching subscribers, a SPT rooted at the publisher and spanning all nodes [11]. This solution requires all nodes to have complete knowledge of the network topology (nodes and links) and subscriptions. It uses some techniques of the same nature of the ones used by link-state unicast and multicast routing algorithms [42], [43]. Another alternative [15], also optimal, requires each publisher (but not all nodes) to know all the subscriptions. As the matching algorithm provides an implementation of the mapping ψ, each publisher can compute the set of matching nodes and then use explicit or stateless multicasting [24], [4], [40] to route the notification, as will be further discussed later.

C. The Channelization Problem

As the computation and pruning of the dissemination tree for each notification can be quite expensive, researchers, still inspired by the ideal multicasting algorithm, have tried to find approximate solutions based on a bounded number of multicasting groups, reused across notifications. If IP Multicasting or an equivalent overlay multicasting facility is available, one can try to use it with a bounded number of different groups. This is defined as the channelization problem in large scale data dissemination which consists in finding an optimal mapping of several information flows to a bounded number of multicast groups. This is a computationally hard problem and therefore only approximate solutions are known [2].

D. Optimal Routing Solutions Based on Subscription Propagation

Another path to the optimal solution, not requiring so much global state available in all nodes, is based on subscription

propagation among nodes. With this approach, each node only possesses information about the subscriptions "reachable" by each of its links [17], [45]. Propagation of subscription S_i is used to build a path leading to it, which is of the same nature as of "learning by the reverse path" [24], as is commonly used by several network routing protocols (e.g., IEEE 802 networks). In this approach, a node takes routing decisions by computing the set of its links leading to nodes with matching subscriptions for each notification. Assuming the subscriptions reachable by a link can be summarized, i.e., replaced by one or a small set of subscriptions covering them, one hopes that the number of locally known subscriptions be much smaller than the number of all subscriptions present in the network. In these algorithms, summarization plays the same role as IP address prefix aggregation in shrinking the routing tables size and therefore improving scalability.

Algorithms based on this strategy perform optimally in what concerns routing costs, but are required to propagate and summarize subscription announcements, as well as to repeat the execution of the matching algorithm in every node that each notification reaches.

E. Approximate Solutions

Besides the approaches presented above, there are approximate solutions concerning optimal routing, whose main motivations are scalability or exploiting alternative overlay network organizations. A common starting point for several approximate solutions consists in reducing the number of the clusters $C_1, C_2, C_3, ...$ used by the ideal solution by relaxing the constraint that a notification should be diffused via only one group and accepting that each group may deliver false positives, as in the channelization problem presented above. For example, semantically similar clusters, or clusters with low density of notifications can be grouped to reduce their number and therefore the number of required multicast groups.

A systematic way of reducing the number of needed clusters uses notification space partitioning [69]. It reduces the size of the problem by dividing the notification space in several cells. This technique introduces false positives. For example, figure 5 shows the same partitioning of the notification space shown in figure 2 using a regular grid. Assuming that each subscription in the figure represents a node, if we associate each node with the cells its subscription intercepts, we can determine the different subsets of nodes, i.e., the groups, needed. $G_1 = \{S_1, S_3, S_4\}, G_2 = \{S_1, S_2\}$ and $G_3 = \{S_1\}$ in the example. A notification must be diffused via the group corresponding to the cell it belongs to. This approximation replaces the set of all possible notifications by the set of all cells, and the mapping ψ by a mapping that maps each cell in the list of nodes with subscriptions matching it. Each cell is associated with exactly one group and each group is associated with at least one cell.

When a grid of contiguous and non overlapping cells is used to partition the notification space, each cell can be mapped to a key (seen as some sort of encoding of the cell coordinates). Subscriptions can also be mapped to one or more keys corresponding to the cells they intercept. Notifications can then be routed to keys where they should meet the potentially matching subscriptions. This suggests that nodes can be organized as a structured overlay network and that subscription and notification routing solutions can be expressed in terms of routing in that network. Some authors call this approximation rendezvous-based since notifications and subscriptions find each other by way of common keys.

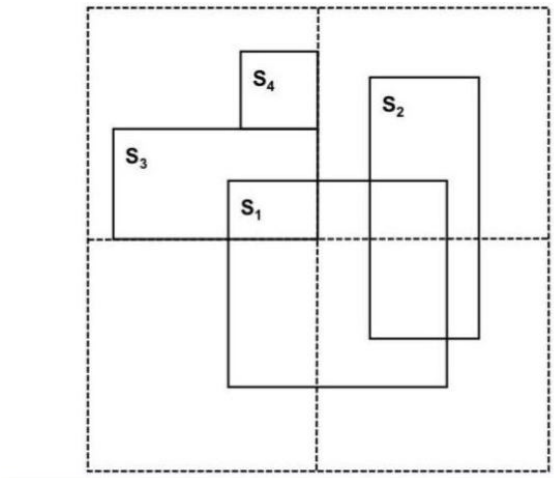

Fig. 5. Grid partitioning of the notification space

Another alternative way to mitigate the problem consists of using subscription partitioning instead of notification space partitioning [66]. This is the natural way to proceed in a network where it is possible to group subscriptions by semantic similarity, which is the case of a P2P overlay where each node represents at most one subscription, or when clients can be redirected to nodes responsible of subscriptions semantically close to their own. Organizing the network, driven by the semantic relations among nodes, can be a good starting point to achieve better notification diffusion paths.

F. A Taxonomy of Known Solutions

The surveyed solutions found in the literature can be grouped according to the strategy they use, in the following approximations:

1) **Centralized Matching and On-Demand Multicasting Groups**. All nodes, or at least the publishing ones, have full knowledge of the subscriptions present in the system. Therefore, if they also know the network topology and know how to route unicast messages, they can compute the links required to reach the nodes with matching subscriptions.
2) **Use of a Bounded Number of Multicasting Groups**. Subscriptions are clustered and a bounded number of multicasting groups is used.
3) **Learning by the Reverse Path**. Subscriptions are flooded using an underlying pre-existent overlay network, and subscription based routing tables are populated. Notifications are diffused to the neighbors in the routing table that have matching subscriptions.
4) **Mappings of Notification Subspaces to Keys Spaces**. Notifications and subscriptions are mapped to one or more keys from a large key space. Nodes are organized into a structured overlay in charge of this key space. Nodes with subscriptions are connected to the nodes in

charge of their keys, and notifications are routed in the overlay routing substrate in such a way that they reach the nodes with the matching subscriptions.

5) **Semantic Neighborhood**. Nodes, in a full-mesh network, can freely establish links among themselves based on some semantic neighborhood metric of their subscriptions. This semantically structured overlay is then used to support notification routing.

In the following sections these algorithms will be presented and discussed.

IV. Centralized Matching and On-demand Multicasting Groups

This section presents optimal algorithms based on global and replicated state. These algorithms dynamically compute the dissemination trees required to diffuse each notification based on the complete knowledge of the subscriptions, which may be replicated in all or in a subset of nodes of the network; and, in some cases, by also taking advantage of replicated information about the network topology.

To diffuse a notification, authors of [11] use a shortest paths spanning tree rooted at the publisher's node. However, to perform better than flooding in a shortest paths tree encompassing all nodes of the network, this tree is dynamically pruned to only reach nodes with subscriptions matching the notification. To compute the shortest paths tree, as well as its pruning in respect to each received notification, a complete description of the network and of the list of all subscriptions in the system is replicated in each node, see Fig. 6.

Authors of [11] do not detail the method they propose for state replication. However, the maintenance of this replicated information can be based on the flooding of "link-states", as performed by the protocol OSPF [42], and on the flooding of subscriptions and unsubscriptions, the same way the protocol MOSPF [43] floods group joins and leaves. Links of the local node belonging to the shortest paths spanning tree rooted at the publisher, can be lazily computed by the Dijkstra algorithm [29] whenever a notification published by a new node is received. This algorithm has computational complexity bounded by $O(L\ log\ K)$ in a network of K nodes and L links.

Every time a notification is received, the tree corresponding to its publisher must be pruned, that is, the set of local links used to flood n must be restricted to the ones leading to nodes with subscriptions matching n. In [11], an algorithm is presented that uses a parallel search tree (PST), see section II, that has been previously annotated with information concerning local links leading to the nodes corresponding to the subscriptions of each subtree. The algorithm is called *link matching* by their authors. Thus, the handling of a notification is based on the PST traversal in order to find local matching subscriptions, as well as to restrict the set of local links through which it must be flooded. This traversal takes place with sub-linear complexity on the number of subscriptions.

The algorithm is correct since it introduces no false negatives in a stable network, routes optimally and delivers no false positives. However, it requires each node to know the complete network and all the subscriptions to compute a PST for each subscription, to update this PST for every new subscription and unsubscription, and to perform a PST traversal for every notification received. There are as many diffusion trees as there are publishers, meaning that nodes must redo their computation whenever there is a change in the network configuration and a new notification is received from a publisher.

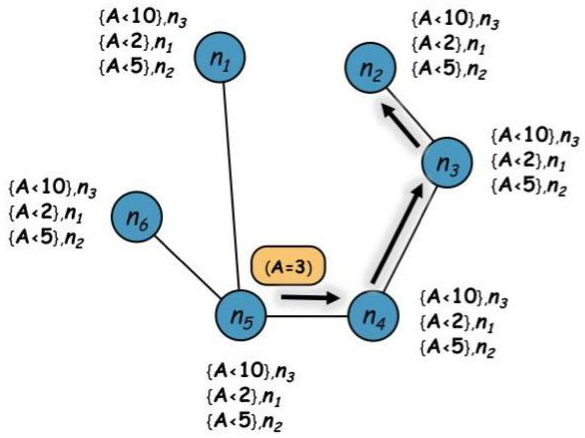

Fig. 6. An example of overlay on-demand multicast routing with matching in every node. A notification $(A = 3)$, published by node n_5 is propagated along a dynamically pruned SPT to reach the nodes with matching subscriptions, n_3 and n_2

MEDYM [15] is an algorithm with a similar starting point but using a different way of multicasting a notification to the list of matching nodes. In this algorithm, all publishers have a replicated copy of all the subscriptions and use a matching algorithm to compute the list of matching nodes of each published notification. With this list, the publisher then uses stateless multicast routing to diffuse the notification. Stateless or explicit multicast is a technique of implementing multicasting initially proposed in [24], that has been revisited in several contexts [4], [40]. It is usable in contexts where a node knows the list of members of a group in addition to some form of optimal unicast routing.

The algorithm requires the header of each multicast message to convey a list of receiving nodes. The initial publisher selects a set of neighbors that are the start of shortest paths to reach subsets of the receivers. Each of these neighbors will receive a copy of the notification annotated with the list of receivers for which the neighbor is the start of a shortest path to reach them all. Each neighbor node will proceed recursively with this process, until each destination list is empty.

The two algorithms just presented compute similar routes to diffuse notifications. The former requires subscription replication and the matching algorithm execution in all nodes, while the latter only requires the matching computation by the publishing nodes. However, it also demands each notification to carry an explicit list of destination nodes, which limits its scalability.

Both algorithms perform an on-demand computation of the dissemination tree for each notification. Alternatively, these trees can be setup and reused as needed. This is what can be done using some multicast groups, previously or dynamically

setup. As the required number of different groups is in general huge, an alternative is to limit their number to a practical value.

V. Use of Limited Number of Multicasting Groups

IP Multicasting, as an example of a facility providing an optimal path to deliver a notification to a group of nodes, can support an optimal solution, see section III. The number of different required groups can be very high (greater than 10^6 with just 20 nodes). This huge number poses problems in what concerns the IP Multicast address space administration and the required state in routers, since in IP Multicasting there is no way to aggregate routing entries. Thus, viable ways of using IP Multicasting must rely on a limited number of groups.

In [46], several alternatives to reduce the number of needed groups are discussed. All of them require the publisher of a notification to know all the subscriptions of all nodes, be able to compute the list of matching subscription nodes and select the IP Multicasting groups required to disseminate the notification. Therefore, these algorithms suppose that there is a consistent way of mapping groups to their identifiers and that nodes join the groups they belong to. Of the several algorithms presented, the following are based on node clustering.

The fist proposal is dubbed CGM or Clustered Group Multicast and requires the publisher of a notification to send more than one multicast. If k nodes are divided in c mutually exclusive clusters, each cluster only needs $2^{k/c}$ different groups, for a total of $c * 2^{k/c}$ groups. For example, 20 nodes in one cluster will require at most 2^{20} nodes, while the same 20 nodes divided in 5 clusters will only require $5 * 2^4 = 80$ groups. The publisher will send at most c multicasts instead of just one.

The second proposal complements the first one by reducing group precision. It is dubbed TCGM or Threshold Clustered Group Multicast. For each of the previously proposed clusters, a threshold T is established. If a notification matches more than T nodes in a cluster, it is sent instead to the group of all members of the cluster, in an attempt to further reduce the number of needed IP Multicasting groups. With this algorithm, a node receiving a notification must test if it matches its subscriptions since the algorithm delivers false positives.

A third alternative consists of using a limited total number of groups, for example g groups. Given the available set of subscriptions, the algorithm starts by computing the disjoint subspaces required by the ideal solution. Then, the probability of each subspace receiving notifications is established. If this probability is initially unknown, it can be estimated as proportional to the subspace volume. Finally, as long as there are more than g subspaces, the subspaces with lower probability are merged. In fact, if several groups have relative low probability of receiving a notification, mergers of such groups also have a relatively low probability of introducing wasted notifications.

These approximate solutions require a variable number of groups. At first sight, it seems that the gains obtained should grow as the number of available groups also grows. In fact, this is far from being exact in the frequent scenario where there is a small number of brokers, with a large number of subscriptions scattered among them. In this setting, there is a high probability that each broker has at least one subscription matching each notification and, therefore, ideal multicast would cost the same as flooding in terms of communication messages, but a lot more in computing complexity.

Paper [52] presents an extensive analysis of this matter. Their results show that in networks with one hundred brokers but thousands of subscriptions, ideal multicast does not lower significantly the communication costs. Higher gains are obtained when: the brokers are numerous (in the thousands), the number of subscriptions is at most one order over the number of brokers and there is a heterogenous distribution of subscriptions among brokers in terms of the targeted notification content. For scenarios like that, their work presents a set of clustering algorithms that obtain efficiency gains of 60% to 80% with multicast groups in the tens. They use a grid-based clustering framework that partitions the notification space in cells and associates a feature vector with each cell, to deduce the subscriptions interested in each one. The cells are then clustered to minimize the expected waste of traffic.

Authors of [69] present a solution that uses a fixed grid-based partition of the notification space to compute the required groups. Overlay diffusion trees are then established for each of these groups.

The algorithms discussed in this section are valid and deserve attention if a multicasting facility is available, regardless of the level it is provided: network or overlay. Mapping a cluster of nodes to an identifier and being able to optimally multicast a notification to the cluster, in a single operation, is a powerful building block for content-based routing.

In the next section, we will return to a class of solutions based on the dynamic pruning of dissemination trees. However, these solutions clearly separate the tree computation from its pruning and use a different approach to the pruning computation: sets of subscriptions are replicated in each node and associated with local links, instead of their issuing nodes.

VI. Learning by the Reverse Path

This section presents a group of algorithms based on the propagation of subscriptions announcements among nodes to populate routing tables using the principle of "learning by the reverse path" – if a node Y receives a message originated at node X, the receiving link of Y is the beginning of a (optimal) path leading to X. In the context of content-based routing, announcements represent subscribers *interests*. They allow each node to build routing tables associating interests (subscriptions) to the paths leading to them (links). Notifications are routed to the nodes in the routing tables [45] that have matching subscriptions.

Each node has a routing table with entries of the form (S_i, L_j), where S_i stands for a subscription and L_j for a link. Besides entries of this form, the nodes routing table also has entries for local application (in a P2P system) or client (in a broker) notification delivery. Without loss of generality, unless otherwise stated, we will assume that there is one only such local subscription, announced by link "Local".

With this class of algorithms, whenever a notification is received by a node (locally originated or received from a peer node) it is sent through links with an associated matching

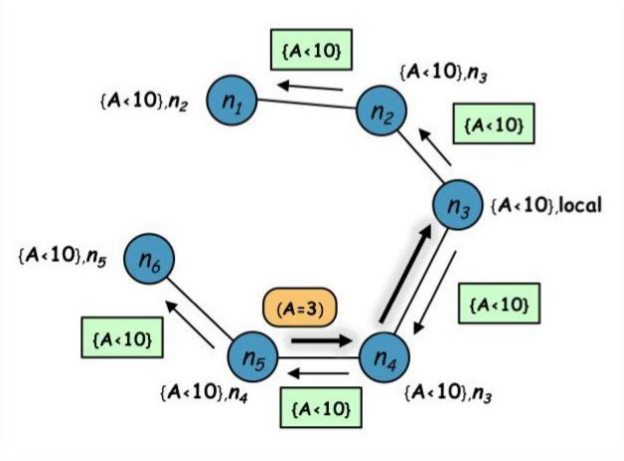

Fig. 7. The flooding of the subscription $\{A < 10\}$ initiated by node n_3 and the routing of the notification $(A = 3)$ issued by node n_5

subscription, except the one from which it was received. Accordingly, if the notification came from a peer and matches the local subscription, it is also locally delivered. When all routing table entries take the form $(True, L_j)$ for all links of each node, the algorithm behaves the same as broadcast.

These algorithms have similarities with those that bridges and ethernet switches, in tree-shaped networks, use to learn how to route frames to the address of their original sender [60]. Interest announcements propagation also bear some similarities with reachability announcements propagation in distance-vector routing protocols [60].

For clarity and simplicity, we will initially assume that nodes are organized in an acyclic graph. Later we will discuss other network configurations.

A. Maintenance of the Subscriptions Routing Tables

When a node receives a new local subscription S_i (from the application or a client), it inserts the entry $(S_i, Local)$ in its routing table and floods a message announcing S_i to all its neighbors. Each peer receiving, by link L_j, an announcement of S_i, inserts an entry of the form (S_i, L_j) in its routing table, if an equivalent entry does not exist already and continues the announcement flooding, or stops it otherwise. Figure 7 illustrates the flooding of the subscription $\{A < 10\}$ initiated by node n_3 and the routing of the notification $(A = 3)$ issued by node n_5.

If routing table entries are maintained by soft-state, unsubscriptions may be dealt with by timers, but spam will be introduced during some periods. Otherwise, unsubscriptions must also be propagated like subscriptions. Each peer receiving an unsubscription of S_i by link L_j, deletes a pre-existing entry of the form (S_i, L_j) of the routing table and, if there are no more entries of the form $(S_i, *)$ in the routing table, continues the unsubscription flooding.

Compared to flooding in a loop-free network, this algorithm avoids notification propagation waste at the expense of routing table maintenance and by executing the matching algorithm in each node every time a notification is received. In a network with N nodes, the routing table space complexity is $O(|N|)$, assuming each node only generates a single subscription. If nodes can supply multiple subscriptions, the worst-case complexity is $O(|S|.|L|)$, where S stands for the set of different subscriptions in the network and L stands for the set of links of the node. For each notification received, the worst-case execution times of the matching algorithm is $O(|S|)$.

In what concerns management of subscriptions and matching tests execution times, the algorithm has the same space and computing complexities of the algorithms presented in section IV. The main difference relates to the fact that there is no global isolated view of the network in each node, and routing is not based in minimum cost paths but in a spanning tree shared by all publishers.

In the following subsections different possible optimizations and variations of this base algorithm are presented.

B. Using advertisements

If publishers publish advertisements, cf. section II, these can be used to prevent subscriptions to propagate to regions of the network that will not produce relevant notifications. Their only usefulness is to make subscriptions routing tables smaller and thus speedup the execution of the matching algorithm.

Advertisements must be propagated in the network exactly by the same way subscriptions are propagated. In each node, advertisements stay associated with the links from where their announcements came. Moreover, advertisements can also be canceled.

The condition allowing subscription S_i to be propagated through link L_j, in a node with entries (Adv_1, L_j), $(Adv_2, L_j), \ldots (Adv_M, L_j)$ in its advertisement routing table, is

$$\exists\ 1 \leq m \leq M : [Adv_m \cap S_i \neq \emptyset] \tag{2}$$

This propagation takes place whenever new subscriptions or new advertisements are received. However, there is a race condition, since subscriptions not intercepting any known advertisement could have been forgotten. One way to deal with this issue is to record subscriptions not intercepting any known advertisement and reconsider them whenever new advertisements are received.

Advertisements introduce extra complexities that are only justified if the expected set of subscriptions not served by any publisher is significant.

C. Subscriptions Routing Table Compression or Summarization

In order to reduce the number of routing table entries, it is possible to aggregate or merge some of them. This can be achieved in two different ways. First, it is possible to replace entries (S_1, L_j) and (S_2, L_j) by (S_1, L_j) whenever $S_2 \subseteq S_1$. Second, it is possible to replace entries (S_1, L_j) and (S_2, L_j) by an entry with a new subscription (S_3, L_j) such that $S_3 \supseteq (S_1 \cup S_2)$, that is, notifications matching S_1 or S_2 also match S_3, but it may exist other notifications matching S_3. The first method is a precise compression, introducing no false positives or negatives, the second one is an imprecise one, enlarging the notification subspace associated with the link, since it introduces false positives. Figure 8 illustrates subscription summarization.

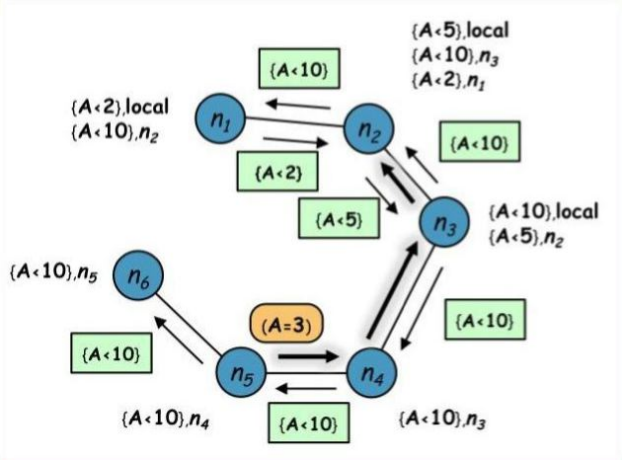

Fig. 8. An example of subscription summarization. Node n_1, n_2 and n_3 issued subscriptions $\{A < 2\}$, $\{A < 5\}$, $\{A < 10\}$, respectively. Then, nodes n_4, n_5 and n_6 only need to retain subscription $\{A < 10\}$ since it covers the other two

The main motivation to reduce the number of routing entries with imprecise compression is to trade computing complexity for network bandwidth. For example, $\{A_1 = v_1, A_2 = v_2\}$ and $\{A_1 = v_1, A_2 = v_3\}$ can be imprecisely merged in $\{A_1 = v_1\}$. Used in an aggressive manner, imprecise compression can degenerate in flooding when routing entries reduce to the form $(True, L_j)$.

Precise summarization is an optimization that plays the same role as IP prefix summarization with classless routing. There is no parallel technique in the IP routing world for imprecise summarization, but default routing.

Reasoning about routing table compression can be made more rigorous in terms of summaries [67]. Given a set of subscriptions S, a *summary* of S is a set of subscriptions, not necessarily belonging to S, that covers[2] S. A subscription $S_i \in S$ is *maximal* if there is no $S_j \in S$ such that $S_i \subseteq S_j$. The set of all maximal subscriptions of S covers S. Let S_l denote the set of subscriptions associated with a link l and Sum_l some summary of S_l. The summary Sum_l is *precise* if the subspaces denoted by Sum_l and S_l are the same, otherwise, $S_l \subset Sum_l$ and it is called an *imprecise summary*. The set of all maximal subscriptions of S_l forms a precise summary of S_l. The most imprecise summary is $True$ since it covers the full notification space.

A precise summary of S formed with all the maximal subscriptions of S is also called a *covering subscription set (CSS)* in [17], where this subject is discussed. To efficiently compute the CSS, one realizes that the cover relation is a partial order one and a *partially-ordered set* (poset) is used. A poset is a directed acyclic graph where each subscription is represented as a node and nodes can have parents and children nodes. Parent nodes have a subscription subspace that is a superset of its children nodes, while subscriptions that partially intercept or are disjoint will appear as siblings. The CSS is the set of immediate children of the imaginary root node.

[2]In a more rigorous context we should say, a summary of S is a set of subscriptions denoting subspaces whose union contains the union of subspaces denoted by subscriptions of set S.

Which type of compression is the best, based on precise or imprecise summaries, is determined by subscription and notification statistics, as well as the relative costs of computing power and bandwidth. Article [67] presents an algorithm capable of dynamically changing the precision of the summaries to maintain bandwidth usage within a certain budget, at the expense of greater cpu usage. The more precise the summaries are the less bandwidth is consumed, but greater effort is required to compute them.

Whatever the type of summarization used, precise or imprecise, for each new subscription received, a summarization algorithm is executed. If it leads to the creation of a new summary, it must also be propagated. Otherwise, the propagation of the previously received subscription stops.

In general, this requires some modifications to the base algorithm presented in the previous subsections. A naive implementation, without taking into consideration unsubscription messages, only allows the precision of the installed summaries to decrease, by always enlarging their scope. To correctly deal with this, a node is required to maintain a set of unused routing entries describing all previously received subscriptions. Whenever a subscription (or unsubscription) is received, these entries are used to recompute the routing table entries as well as the new subscription (or unsubscription) to propagate. In fact, a node needs to maintain an history of all still valid subscriptions previously received, and propagate to its neighbors all modifications of the summaries it is using. These more complex data-structures and computations can be avoided by periodically flooding all current subscriptions from their originating nodes and by calculating new routing tables to replace the old ones, see [18] for a complete algorithm using this approach.

As advertisements are only tested when subscriptions are propagated, their compression is, in general, less interesting and would increase the complexity even more.

D. Using Rooted Trees

To further reduce the size of the routing table, it is possible to use rooted trees. Algorithms using this approach are often called hierarchical. If the network is organized as a rooted tree, subscriptions and unsubcriptions are only propagated, as explained in subsection VI-A, upwards to the root of the tree, where their propagation stops. Subscriptions will be only associated with the downward links leading to the subtree of the correspondent subscriber. This ensures that every subscription will be known from the issuing node up to the root of the tree. Notifications are propagated starting from the root downwards to every subtree with matching subscriptions.

If the tree is well balanced, this organization of the network and the modified algorithm permits a significant routing table reduction. Each node is only required to know the subscriptions of its subtree. Thus, leaf nodes have smaller routing tables, while the root node will know all the subscriptions of the network, as before.

In a system with n nodes, each with a different specific subscription, organized as a well balanced binary tree with $O(log\ n)$ levels, the root node will know n subscriptions, and

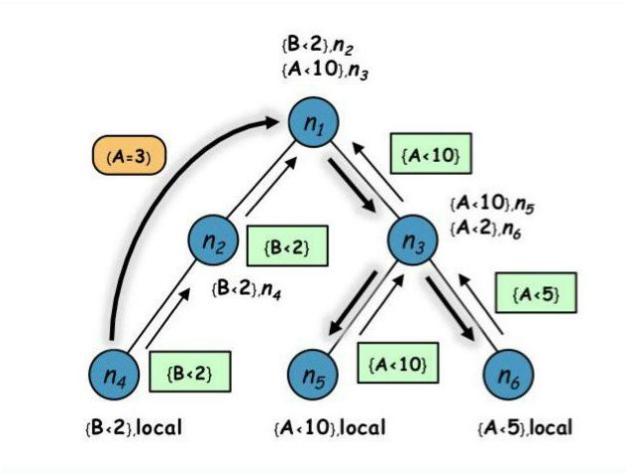

Fig. 9. Propagation of subscriptions and notifications using rooted trees

nodes of level k will only know $O(n/2^k)$ subscriptions, while leaf nodes, i.e., $O(n/2)$ nodes, will just have to know their own individual subscriptions. Each individual subscription is only present in at most $O(log\ n)$, instead of $O(n)$, routing tables.

As the space and computational complexities of the matching algorithm are proportional to the number of routing table entries, this alternative has a significant impact from this point of view. The price to pay stems from the particular way of routing notifications, which is only optimal if the tree is a shortest paths one and publishers are coincident with its root. This hierarchical organization also introduces a significant unbalance in the work load. In particular, the root node can become a critical congestion point.

Notifications can be directly sent from the publisher to the root node, and then from the root to every subtree with matching subscribers, as shown in figure 9. Alternatively, notifications can follow the tree upwards from the publisher to the root and, on each intermediate node visited, downwards on all other subtrees with matching subscribers. In the latter, the tree is used bidirectionally to diffuse notifications, while in the former the tree is used unidirectionally for the same purpose.

Subscription messages follow a path that resembles the path of join messages in multicast tree building algorithms, used by IP Multicasting routing protocols, based on a shared tree, and rooted at a rendezvous node [32]. Notifications are also treated in a way resembling the multicast packet diffusion methods used by the same protocols, or by their more recent variants (e.g., BIDIR-PIM).

Up to now, we have not discussed how nodes join the overlay and form the network. One possibility is to do it manually. The administrators of broker nodes can structure them in a tree by choosing a root broker and the set of suitable overlay links. However, this process can be automated. In general, the algorithms most often used have a root/rendezvous node as input, and then organize nodes in a spanning tree of shortest paths, as in PIM-SM [32].

If there is an easy way of partitioning the notifications space into disjoint spaces, one way to promote scalability is to spread the load of the dissemination of these different notification sets across several trees, one per set.

In the system Hermes [49], the notification universe is partitioned according to the language type of the notifications. A node subscribing several different notification type streams will join as many trees as different streams it subscribes. Hermes uses the P2P structured overlay routing substrate[3] Pastry [54], a variant of a Plaxton tree [50], to build diffusion trees as pioneered by the system Scribe [55]. The root node of each tree is the node in charge of the key associated with its type. Each tree supports the execution of a different instance of a version of the routing algorithm presented above to build routing tables, and to route the notifications of its type.

It is also possible to spread the load using a different routed spanning tree per publisher. In a network of n nodes, there will be n such trees, each rooted at a different node. Paper [61] proposes this method of routing notifications in an overlay network of publish/subscribe nodes organized around the DHT Chord [57]. Building several trees is also further discussed in the next subsection.

Methods required to deal with unsubscriptions and advertisements, as well as summarization of subscriptions, can also be adopted in algorithms using rooted trees.

E. Networks with cycles

If the network has cycles and subscriptions are flooded, a node will receive duplicates of the same subscription announcement, coming from different links. One way to avoid these duplicates is to only propagate subscriptions issued by node n through a SPT rooted at n. These SPTs can be previously built by a suitable broadcast algorithm, or may be dynamically built by the subscription dissemination process itself.

There are several ways of implicitly propagating the subscriptions issued by node n on a SPT rooted at n. For example, messages containing subscriptions announcements can carry the total cost of the path already traversed. Assuming that links have symmetrical costs, a receiving node can compute the total cost of the path up to the subscriber, allowing it to just retain the subscription that arrived by a shortest path. This technique of selecting shortest paths is used by standard 802.1D (*Spanning Tree Protocol*), which also has methods to deal with paths of equal cost. Alternatively, if the nodes of the network know the unicast shortest paths to each destination, the reverse path forwarding test [24] can be used to detect duplicates: a subscription originated at node B, arriving at node A, is not a duplicate if it came from a link of the shortest path from B to A.

Propagation of subscriptions by SPTs, explicitly or implicitly, builds paths linking potential publishers to the interested nodes. Routing of notifications using these paths promotes a more balanced use of the available links, than what is afforded by a single spanning tree shared by all publishers. However, notification routing based in the routing entries built by subscription propagation over these SPTs will introduce duplicate notifications.

[3]Often dubbed a *Distributed Hash Table* (or DHT).

A complete approach is the following. First, subscriptions are broadcasted by spanning trees rooted at each subscriber node. Second, each notification is also disseminated using a spanning tree rooted at the publishing node. Both trees are implicitly or explicitly build by a broadcasting algorithm. Third, dissemination of notifications by SPTs is restricted to the links leading to nodes with matching subscriptions, using the subscriptions associated with links to restrict the propagation. See [18] for a complete description of the algorithm. This algorithm will correctly route notifications, i.e., without introducing false negatives, neither false positives, if the network and the broadcasting algorithm ensures that all paths are symmetrical, which in [18] is called *all-pairs path symmetry*. Spanning trees with the previous property will guarantee correctness, while shortest paths ones will achieve optimal routing.

F. Network organization, dynamics and reliability

The several variants of the base algorithm, presented so far, use one or more trees to diffuse notifications and routing tables subscriptions updates. Algorithms used to build trees have been extensively studied in IP Multicasting protocols and in overlay networks used for multicasting [55], [12], [39], [21], [51], [48]. Two main methods were devised for overlay networks: mesh-first systems and tree-first systems.

With mesh first systems, the overlay is initially structured as a graph that provides nodes with the means to route unicast messages to a destination. In this scenario, algorithms used by IP multicasting routing protocols [32] to build diffusion trees rooted at a rendezvous node, can be used to build overlay diffusion trees. This method can be used independently of the overlay routing substrate, be it an unstructured mesh [21], [30] or a structured overlay [57], [54], [51]. For example, Hermes [49] uses the Pastry [54] structured overlay as the routing substrate. A structured overlay is well suited to build rooted trees because nodes joining the tree can route join messages to a previously agreed key denoting its root [55], [10]. Other ways of building trees on top of a unicast routing substrate have been suggested in the previous subsection.

Tree-first overlays [12], [39], [48] build the tree directly. They require special algorithms to avoid introducing cycles during tree reorganization. The system NaradaBrokering [47] is a publish/subscribe system that also adopts a tree-first approach.

Besides the problems related to tree building and maintenance that must be dealt with, it is also necessary to manage the subscription routing tables in response to nodes joining or leaving the network, or crashing.

When a node crashes or a link becomes unavailable, all routing entries pointing to a now unavailable link must be deleted. When a new node is connected to a tree, or a new link becomes available, the new neighbors are required to exchange subscription messages for all the subscriptions found in their routing tables. If explicit unsubscriptions or advertisements are used, these must also be dealt with. During these reconfigurations, notifications can be lost or duplicated. Recovering from these errors is the responsibility of a reliability protocol which is orthogonal to the routing and network maintenance ones, see [22] for an example.

In systems where a significant percentage of nodes or links exhibit an high rate of instability, the impact of reconfigurations can be devastating, preventing the system from ever stabilizing. This is more probable in large-scale P2P systems with high churn rate and in wireless mobile ad-hoc networks. System reconfigurations in these extreme scenarios is a challenging problem well behind the routing one. Nevertheless, it is worth noting that directed-diffusion [38], a protocol solving a problem resembling the content-based routing one in a wireless sensor network scenario, is based on periodic floods of subscriptions.

G. Conclusion

Routing using routing tables made of subscriptions summaries is one of the oldest methods for dealing with content-based routing. Examples of content-based research prototypes like Siena, Jedi, Rebeca and Hermes [17], [23], [44], [49] were sophisticated key testbeds for the proposal and evaluation of many of the algorithms based in the principle “learning by the reverse path”. In the previous subsections, where these algorithms have been presented, many references were given to several contributions allowing their deeper development and understanding.

All the algorithms presented in this section are based on a clear separation between the buildup of diffusion trees and their pruning. These algorithms are correct and with precise summaries introduce no spam. In terms of notification delivery, the algorithms will perform as optimally as the overlay routing trees used.

The variants based on rooted trees do not require every node to know all subscriptions and perform better than some variants of dynamic multicasting, see subsection IV, in what concerns space and computing complexity. Reduction of the size of the routing tables by summarization of subscriptions allows an extra speedup of the matching algorithm. However, its management introduces increasing complexity and therefore is only a clear win if the notification rate is high compared to the rate of subscription and unsubscription events. There is also room for load distribution and routing cost optimizations if several rooted trees are used, instead of a shared one.

VII. Mappings of Notification Subspaces into Keys Spaces

In this section we will consider systems organized around the use of mappings from notifications and subscriptions to key spaces. These mappings allow publishers and subscribers to ”meet” each other, whenever the key or keys associated with a notification belong to the set of keys associated with the matching subscriptions.

Routing in a system around the above idea achieves progress in several steps:

1) A node must be registered with the nodes in charge of the keys to which its subscription maps to.
2) A notification is routed from the publisher node to the node or nodes in charge of the key or keys to which the notification maps to.
3) While being routed in the overlay, the notification will cross or will be transmitted to all nodes possibly interested in it.

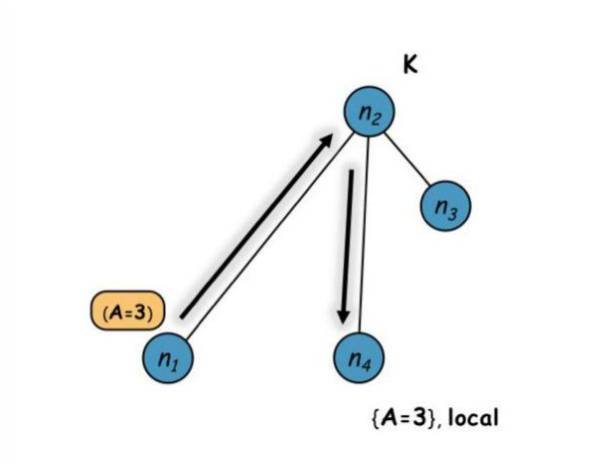

Fig. 10. An example of routing notifications with keys and intervals. Node n_2 being in charge of key K acts as the rendezvous point for the subscription $\{A = 3\}$, issued by n_1 and notification $(A = 3)$, published by n_4, as result of both being mapped to K

This approach is often called *rendezvous based*, since notifications and subscriptions "meet" in the node in charge of the keys that both are mapped to.

Using keys to represent notifications and subscriptions has been pioneered by the system EDN [66]. For every event and subscription, EDN uses a *subset of the attributes* to generate a *signature string*. The signature string must be formed with constrains only containing equality operators, which poses some limitations on EDN's applicability. This signature is hashed, by a function H, to obtain a key. So, if notification n matches a subscription S, then $match(n, S) \Rightarrow H(n) = H(S)$, where H denotes the hash of the signature. Therefore, a supplementary matching test is still needed to verify if $match(n, S)$ since the hash function acts as an imprecise summary of an already imprecise summary (the signature string).

EDN is mainly intended for load balancing. Broker nodes are in charge of different intervals of the key space. A node with subscription S must be connected to the node in charge of the interval to which $H(S)$ belongs to. A notification n is sent to the node in charge of the interval to which $H(n)$ belongs to and from there to all nodes interested in that interval, as illustrated in figure 10. When a hot spot appears, the system globally redefines the intervals and their assignment to nodes to redistribute the load more evenly.

With this solution, there is some control over the way the notification subspace is partitioned, since the greater the number of attributes used to generate keys, the better the partitioning. However, as this number grows, less subscriptions are legal because legal subscriptions (as well as legal notifications) must only have equality constraints over all the attributes used to generate the signature string. The same idea cannot be directly applied when range subscriptions are determinant to spread the load. Moreover, the diffusion of notifications to all nodes linked to an interval is based on a point-to-point communication with each of theses nodes. This can also be optimized.

A. Using Structured Overlays to Deal with Range Subscriptions

Structured overlay networks have emerged as routing substrates which are scalable, decentralized and self-organizing (i.e., they adapt, automatically, to node arrival, departure and crash when the churn rate is below a certain threshold). In a structured overlay, nodes are in charge of key sets, and the main functionality that is provided is the ability to route a message to the node in charge of a key in a bounded number of steps. On top of this basic functionality, structured overlays have been used to implement distributed hash tables (DHT), file-systems, content networks, multicasting, broadcasting, topic-based publish/subcribe systems, etc.

Choosing an Index: Paper [59] describes a proposal, based on the structured overlay Pastry [54], that supports range queries over a predefined set of attributes, dubbed an *index* by their authors. The system is able to deal with subscriptions adhering to an index, i.e., subscriptions having valid constraints over all the attributes of the index.

A valid notification has an *index digest* which is, by definition, the subset of the attributes of the notification concerning the set of attributes of the index. As exemplified in [59], if a notification schema over an information space related with desktop computers has the attributes Price, CPU, Clock, RAM, HDD and Monitor and the index selects the attributes CPU, RAM and HDD, the notification $[USD : Price = 1000,\ String : CPU = PIII,\ MHz : Clock = 650,\ Mbyte : RAM = 512,\ Inch : Monitor : 15\ ]$, will have the index digest $[\ String : CPU : PIII : MHz : Clock : 650 : Mbyte : RAM : 512\ ]$. Given an index digest, it is possible to define a function to compute a *key* of this string. The node in charge of that key in the overlay is its *home node*.

Partitioning of the Index: In order to deal with range queries, the range of values of each attribute of the index is divided into intervals. The different attributes and intervals determine a partitioning of the notification space in several subspaces or parts, see figure 5 in section III for an example. The key of the index digest denoting an agreed point of each part is used as the key of that part.

Partitioning is easy for attributes with a limited number of different values (e.g. CPU), since each different value induces a different interval. For attributes with many different values (e.g. Price, RAM), designers of the index must use a small number of interval indicators as values for partitioning and computing keys. As can be seen, an index with its intervals is a way of reducing the number of attributes and values relevant for the notification space partitioning and clearly determines the shape of this partitioning.

Computing Keys: Given a notification n, it is then possible to compute the part containing n, which will associate a key with n, denoted $K(n)$. Using the same approach, a subscription can be associated with a sequence of keys, the keys of the parts needed to completely cover the subspace denoted by the subscription. If a subscription S_i only has equality constraints over the attributes of the index (in addition, S_i can have any constraints over other attributes), it intersects a single part whose key ($K(S_i)$) can be computed from the equivalent of an index digest [59]. This is quite similar to the

approach used by the system EDN with its signature strings. If S_i contains range subscriptions, it is possible to compute the index digests and the keys of parts of the notification space that completely cover S_i. Therefore, given a subscription, it is possible to compute $K(S_i)$, which denotes one or more keys identifying the same number of subspaces or parts of the notification space.

If we associate a multicasting group with each key, and each node joins all the groups associated with the keys covering its subscriptions, we have just built one approximate solution of the content-based routing problem along the lines introduced in section III. Recall that this solution introduces false positives since the index partitioning method and the key generating function are imprecise summarization methods, as in EDN.

Paper [8] is an attempt to generalize the above approach to compare the performance of alternative methods of generating keys. The paper introduces the mapping N_s which maps each subscription to a set of keys and nodes where it should be registered, and a mapping N_n which maps each notification to a set of keys and nodes willing to deliver it to the set of locally registered matching subscriptions. These two mappings will allow the correct delivery of a notification, if and only if, $N_s(S) \cap N_n(n) \neq \emptyset \ \forall S, n : Match(S, n)$.

Multicasting the Notifications: There are several ways of using a structured overlay to build the multicasting groups allowing the final delivery of notifications to the nodes interested in them. An initial approach consists of connecting directly a node to all the home nodes of the keys of its subscriptions. This may be a poor solution for the home nodes of the keys corresponding to very popular subspaces. Several alternative solutions have been proposed. Authors of [59] use the Scribe protocol [55] over the Pastry structured overlay. Authors of [8] introduce their own multicasting protocol on top of the Chord [58] structured overlay.

The overall quality of the proposals discussed above depends of:

- The characteristics of the partitioning induced by the index and the key mapping function.
- The quality of the dissemination tree provided by the multicasting protocol on top of the structured overlay.

In what concerns the first aspect, the evaluations presented in paper [59] show that a poor index induces a poor partitioning and a significant increase in false positives, thus wasting many messages. It can also increase the probability of hot spots, leading to some nodes overloading. Ideally, more keys should be allocated to parts of the notification space with a higher load, but devising a good index seems to be a challenge that is very dependent of the schema used. Moreover, these evaluations are based on uniform synthetic distributions of notifications and subscriptions, while real world data sets tend to be skewed and hence will cause a non-uniform distribution of node's load, thus aggravating the problems observed.

In respect to the second issue, evaluations only show that the system is scalable when a good index is used, since the number of exchanged messages grows sub-linearly with the number of nodes. However, this is far from an evaluation comparing the dissemination solution to the performance of an optimal multicasting routing substrate.

Improving Load Balancing Issues: Meghdoot [35] is an alternative design that deals with the load balancing issue. This system uses the CAN overlay [51], which maps keys into points or volumes in a cartesian space of N dimensions. Meghdoot uses a CAN with as many dimensions as twice the number of attributes of the schema. It introduces two mappings: K_n, mapping a notifications into one key, and K_s, mapping a subscription into a key. A node with subscription S_i is connected to an unique node, the node $Home(K_s(S_i))$. A notification n is routed through the CAN to the node $Home(K_n(n))$.

The two mappings have the property that all nodes with subscriptions matching n stay in a well defined volume that includes node $Home(K_n(n))$. As the CAN overlay allows for a simple form of multicasting a message to all nodes of a volume in the key space, a notification n is routed to the node $Home(K_n(n))$, and from there to the volume where all nodes with matching subscriptions should have been linked.

To promote load balancing, newly arrived nodes are assigned keys such that they will become responsible for the most loaded regions of the key space. This technique may also be used in the previous approaches, but face greater difficulties there due to their dependance of the indexes used.

The scheme used by Meghdoot to multicast a notification to all nodes with matching subscriptions, based on the CAN, seems less interesting than using a multicast dissemination tree as in the previous methods. However, it is difficult, in general, to compare these approaches in terms of their cost and performance. That comparison is dependent of the characteristics of the different overlays, of their actual performance in terms of real physical network costs and also depends on the use of common data sets and common test suites. To the best of our knowledge, these studies remain unavailable in the literature.

Optimizing for Special Cases: The works presented in [62] and [3] are attempts of leveraging the Chord and Pastry structured overlays routing properties to deal more efficiently with range and substring subscriptions. In the second proposal, a subscription is registered in at least as many nodes of the network as constraints it has, since it requires that each attribute of a subscription be separately processed and registered. A constraint over a string type is, in general, registered in one only node. However, a range constraint will be registered in a number of nodes depending of the range. Thus, a subscription will be registered in as many nodes as the sum of nodes required to register each of its constraints.

Both proposals, given a constant value (a string or a number), allow efficient ways of locating in the overlay all subscriptions registrations matching that value in one of its constraints. This is done using order preserving hash functions with Chord [62], or specific string prefix based routing tables over Pastry [3]. Finding all subscriptions matching a notification n is then easy if n has one only attribute value. However, if the notification has several attribute values, a list of subscriptions must be found for each one of them. Subscriptions found in as many attribute lists as constraints they have, are the subscriptions that match n. This means that these proposals can only be competitive for situations where

most subscriptions and notifications only concern a single attribute. Otherwise, the simpler notification space partition approaches presented above seem preferable.

B. Conclusions

The use of keys spaces and structured overlays for content-based routing tries to solve a multifaceted problem with two tools: mappings from some semantic space to a space of keys and a structured routing substrate built around keys. The systems presented above exhibit simple solutions or very elegant algorithms for some specific contexts. However, it is hard to figure out a winning algorithm, valid for a broad class of content-based routing contexts.

First, no solution is clearly orthogonal to the schema used and all referred solutions introduce spam. Second, unicast key based routing and load balancing are too intertwined and generally also dependent of the schema. This makes the systems fragile to deal with certain contexts and data sets. Third, final notification delivery depends on a multicasting framework that is, in general, a by-product of the structured overlay. Large-scale systems where many facets are too interdependent, are likely to only have success in very specific scenarios, being harder to adapt and evolve. Knowing precisely the winning contexts, the advantages, the drawbacks, limits and more precise costs, would be very important.

Furthermore, significant and reproducible evaluations are missing. In fact, evaluation of the content-based routing problem is lacking well-accepted standard workloads and settings [16]. Evaluation of systems based on structured overlays is, in general, also quite difficult. Together, both difficulties make it really hard to have a reasonable judgement on the usage of keys mappings and structured overlays to support content-based routing in very large-scale scenarios.

The ideas behind structured overlays and their use for content-based routing are appealing from the algorithmic point of view. However, it is difficult to accept their real relevance without more exhaustive evaluations than those presented in any of the referred papers.

VIII. Routing Solutions Based on Semantic Overlays

In previous sections, nodes of the network were organized in an overlay network whose structure was driven by criteria like network or key proximity, geographic distribution, distribution of functions among clients and servers and so on. In this section, we will consider solutions based on network organization driven by semantic similarity.

Establishing the overlay network in a way that facilitates content-based routing may be a way of incrementing performance. For example, with a rooted tree, when notifications are disseminated to subtrees, notifications paths are shorter if nodes are organized in such a way that the paths taken by a notification will flow through nodes associated with a sequence of subspaces $S_1, S_2, S_3, ...$ such that

$$\forall i : S_i \supseteq S_{i+1} \quad (3)$$

If this could be the case, each notification would only flow through matching nodes. This suggests that if we could group together equivalent subscriptions and establish an overlay based on semantic relations among these groups, there is room for improvements on the routing costs.

This approach ignores network costs in favor of flexibility and requires that nodes be able to freely establish links among themselves, what also implies that all nodes must be visible from each other. This system model is best suited for a pure P2P system oblivious of network proximity costs and using a full-mesh network, where subscribers represent a single subscription and play also the role of notification routers for other nodes.

One of the simplest attempts in this direction is best suited for cases where the schema has a single ordered attribute and all constraints are based on the operators $>$, $<$ and $=$. Paper [7] proposes the distribution of these subscriptions in a semantic tree, organized in such a way that most paths from the root to the leaves cross nodes associated with constraints satisfying equation 3. To use the proposal with schemas not restricted to one attribute, each subscription must be associated with the tree of just one of its constraints, arbitrarily chosen, and each notification should be disseminated through the set of trees associated with all its attributes values. Thus, computing nodes with subscriptions with more than one simple constraint will potentially receive waste and must run the matching algorithm on each received notification. This again strongly confines the applicability of the proposal to scenarios where most subscriptions and notifications have few attributes. Despite of that, this line of work is still noteworthy as an example of a semantically driven networking approach.

A. Examples of Semantic Proximity Functions

In a more general setting, different subscriptions, represent different notifications spaces that can be inter-related by relations, other then the *cover* and *intercept* ones. In an attempt to find a more general solution, we will now introduce the functions *affinity* and *distance*.

Given T notifications, of which M match a subscription S_i, the area of the subspace of subscription S_i in respect to T, or simply the area of S_i, is defined by $Area(S_i) = M/T$. In [31], the *affinity function (ϕ)* of subscriptions S_i and S_j is defined as

$$\phi(S_i, S_j) = Area(S_i \cap S_j)/min(Area(S_i), Area(S_j)) \quad (4)$$

If $S_i \supseteq S_j$, i.e., one subscription covers the other, their affinity is 100%. If $S_i \cap S_j = \emptyset$, i.e. subscriptions do not intercept, their affinity is 0%.

When the area of a subscription cannot be statically computed, function ϕ can be, for example, computed by a counting method using a matrix H: whenever a notification is received and matches S_i and S_j, $H[i,j]$, $H[i,i]$ and $H[j,j]$ are incremented. That is, $H[i,j]$ counts all notifications that match S_i and S_j and $H[i,i]$ counts all notifications that match S_i. This allows affinity to be computed even in the presence of opaque subscriptions, materialized by "binary" executable filtering programs.

When the semantic definition of subscriptions is available it is, under certain circumstances, possible to compute a distance

function between subscriptions, which can be interpreted as a kind of inverse of the affinity function. For example, given subscription S_i denoting the Euclidean subspace of N dimensions $[l_1^i, r_1^i] \times ... \times [l_N^i, r_N^i]$, the *distance (d)* of subscriptions S_i and S_j is defined as

$$d(S_i, S_j) = \sqrt{\sum_{k=1}^{N} (min(r_k^i, r_k^j) - max(l_k^i, l_k^j))^2} \quad (5)$$

Affinity and distance are useful functions that can be used to link subscriptions according to a criteria of semantic proximity in a more more flexible way than the presented in the previous subsection.

In [31], an algorithm is presented to generate a tree of subscriptions with the following properties:

1) the root of the tree is the source;
2) the subscription with largest area of each subtree is its root;
3) subscription subtrees are clusters of related subscriptions, and are built with the help of the affinity function.

This tree is suitable to disseminate notifications through paths crossing "close" subscriptions, from the highest to lowest matching probability. This optimizes the total cost of routing. However, waste is introduced since these paths can cross subscriptions not matching the notification. The algorithm is a centralized one and requires the knowledge of all subscriptions, as well as, their affinity. In the next paragraphs, we will illustrate other approaches based on the distance function as well as in other semantic relations.

B. Semantic Proximity and Gossip-based Algorithms

Epidemic or gossip-based algorithms have been initially proposed as a way of implementing data replication [27] and have been recently extensively studied as a scalable means to support reliable multicasting [13], [34], [28]. A node in a epidemic-based system has a dynamic *view* of part of the state of a number of other peer nodes. Peers periodically exchange their views to allow state convergence. Random selection of peers is essential for fault-tolerance and reliability, however, view and peer selection *biasing* has been used as a means of dealing with an heterogeneous network [53], [14], in addition to a means of clustering nodes with related interests [65].

System "Sub-2-Sub (S2S)" [64] is a gossip-based proposal for content-based networking. For network organization, S2S uses the protocol introduced in [63]. This protocol allows a node to have a dynamic view of a random number of other nodes, independently of their subscriptions. To dynamically cluster nodes according to the similarity of their subscriptions, the protocol presented in [65] is used, on top of the previous one, using the distance function introduced by equation 5 as a criterion for node preference. This second view, allows a node to know a dynamic subset of other nodes with overlapping or close subscriptions.

Finally, using all the nodes it knows, a node periodically recomputes a third set of links pointing to as many clusters as the optimal delivery clusters introduced in section III. These clusters are reachable by the links of bidirectional rings associated with each one. To build these rings, each node periodically exchanges all its views with their members and does a computation by which the following invariant is enforced: a node keeps a link to another node with an higher identifier and subscribing to a common part of the subscription space not yet covered by any other of its links.

A publisher must find an initial matching node using all its views, or gossips, as fast as it can, until it locates one or a given small number of gossip steps is exceeded, in which case the notification is dropped since with very high probability there are no matching subscribers. When a node with a matching subscription receives a new notification, it delivers it, locally, and propagates it to the next member of the relevant ring and, to speedup the propagation, propagates it as well to a number of other random matching subscribers. Due to the random nature of this propagation, nodes are required to detect and discard duplicate notifications. Except for the inconsistencies that arise in the rings maintenance (churn or node subscription changes), the method delivers notifications to all the matching subscribers without introducing spam (false positives).

[19] introduces an algorithm that partially resembles the two first protocols of S2S. Yet, it introduces a not negligible percentage of false positives but, more serious than that, it also introduces a percentage of false negatives (misses), i.e., it cannot guarantee the delivery of a notification to all its matching subscribers.

C. A Distributed Implementation of a Parallel Search Tree

Brushwood [68] is another proposal based on semantic similarity. It builds a parallel search tree (PST), cf. section II, and distributes it across a variable number of broker nodes. Each broker takes care of a set of similar subscriptions and is related to other brokers by a network based on a similarity relation among subscriptions. Client subscribers are redirected and connect to the best broker according to their subscriptions. The specific PST used is dynamically built and maintained in a way that promotes load and tree balancing.

Each broker node is responsible for a subtree (subPST) and manages the subscriptions associated with its subtree leafs. All brokers know enough of the PST, and of enough other broker nodes, to route notifications to all brokers in charge of subtrees with matching subscriptions. Client subscriptions are identically redirected to the adequate broker. In fact, all brokers are able to initiate the handling of notifications and subscriptions and route them to suitable nodes, without necessarily resorting to a root broker. For this purpose each subtree receives a specific key or *Tree ID* and a the SkipNet [37] structured overlay is used.

Brushwood demonstrates that it is possible to distribute a PST in a flexible way across a variable number of brokers. New broker nodes joining the system will receive new subtrees, relieving the previously most loaded ones. Paper [68] claims that this is independent of a specific or a closed subscription schema, and also claims that it can scale to a large number of brokers, since in a network with n brokers, each one has links to $O(log\ n)$ other ones, and each subtree can be reached in at most $O(log\ n)$ hops.

TABLE II
CLASSIFICATION OF ROUTING METHODS VIS-À-VIS THEIR USAGE CONTEXT

Method	Schema independent	Data set independent	Network requirements
Flooding	yes	yes	acyclic network (or duplicate detection)
Ideal Multicast	yes	yes	multicast support
Dynamic Multicast	yes	yes	none
Bounded Multicast	yes	no	multicast support
Learnin by Reverse Path	yes	yes	none
Key-based	no		structured overlay
Semantic Affinity		yes	full-mesh network

D. Conclusions

All presented algorithms in this section use a network system model oblivious to network proximity and communication constraints, where nodes can freely communicate with each other, i.e., a full-mesh network.

In a network of nodes with full knowledge of all subscriptions of each other, the publisher node can compute the group of nodes with matching subscriptions, cf. section IV. If nodes are interconnected by a full-mesh network, it is possible to compute a bounded degree tree to optimally deliver the notification. This optimal solution has scalability problems and requires the publisher to fully execute the matching algorithm.

The systems presented in this section present several possible ways of dealing with this scalability issue. We have seen a gossip-based approximation and a system based on a distributed implementation of a parallel search tree. Semantic proximity is an extra research direction added to the toolbox of methods and algorithms available to deal with the content-based routing problem.

IX. BRIEF COMPARISON OF THE APPROACHES AND CONCLUSIONS

Content-based routing algorithms must be evaluated by taking into consideration several aspects concerning their context of usage and their characteristics:

1) independence of the schema language;
2) independence of the notification and subscription distributions (content data sets);
3) independence of the network configuration;
4) nodes in charge of the execution of the matching algorithm;
5) nodes in charge of routing notifications;
6) state replication requirements;
7) routing with or without false positives (spam);
8) optimality of the dissemination in terms of network costs.

Some of these issues are necessarily qualitative and are only suitable to select the usable solutions in some contexts. This is mainly the case of aspects 1, 2 and 3, which expose the dependence or the orthogonality of a solution vis-à-vis some characteristics of its context of usage, namely the schema, notifications and subscriptions distributions (data sets) and network characteristics. The second aspect is very important since some proposals perform better if notifications or subscriptions are uniformly distributed, while others excel when these distributions are highly skewed. Table II presents the dependence/independence of each routing method in relation to these three aspects. The gossip-based algorithm has been selected as representative of the semantic affinity method.

The remaining aspects characterize the routing method in a context independent way. Table III presents a qualitative characterization of the presented routing schemes according to these remaining issues. The presence of a bold-face text-style highlights a case in which the method performs well. Italic signals the opposite and performance of the method is poorest. While, a regular text is used for aspects that are neutral and have little impact on performance of a method. The hierarchical version of the learning by reverse path method has been selected to highlight the methods characteristics. The semantic affinity method characteristics were again established with the gossip-based version.

Clearly, only the ideal multicast and dynamic multicast methods ensure optimal routing in terms of network costs, since these methods are backed by algorithms that compute or use shortest paths trees (SP trees), rooted at each publisher and only covering the nodes with matching subscribers. All other methods are not optimal in what concerns this criterion. The hierarchical version of the learning by reverse path method routes optimally when all notifications come from the root of the tree. Otherwise, it is dependent of the position of the publishers in the network. A tree per publisher with this method will also be optimal.

State replication and matching load seem to be the main sources of complexity of almost all methods. Ideal, bounded and a version of dynamic multicast only require the publisher to execute the matching algorithm. Hierarchical learning by the reverse path only executes matching in half of the nodes. Additionally, most methods, but flooding, require some degree of state replication in what concerns subscriptions. In this respect, hierarchical learning by the reverse path, key-based and semantic affinity are methods that do not require full subscription replication in every node, or in the publishing nodes.

All in all, methods routing by optimal paths require a great deal of state replication, which constitutes their main drawback and prevents their scalability. Ideal and bounded multicast also require the availability of a (network or overlay level) multicasting facility optimal and supporting many nodes. In addition to this aspect, which promotes separation of concerns between matching and routing, they have the same characteristics as the other optimal delivery methods. Key-based and the semantic affinity methods seem to require less state replication and appear to be more scalable. However, it is very difficult to characterize these methods in terms of the costs of routing and state replication. Hierarchical learning by the reverse path solutions are a compromise between state replication and bounded routing costs.

TABLE III
QUALITATIVE CHARACTERIZATION OF THE IDENTIFIED ROUTING METHODS

Method	Matching in nodes	Routing	Subscription replication	Spam / duplicates	Optimal
Flooding	*all*	*flooding*	**none**	*spam and duplicates*	*no*
Ideal Multicast	**publisher**	**network SP tree**	*in publishers*	**none**	**yes**
Dynamic Multicast	path nodes / **publisher**	**overlay SP tree**	*in all nodes*	**none**	**yes**
Bounded Multicast	**publisher**	**network SP tree**	*in publishers*	*spam*	no
Hierarchical Learning by Rev. Path	path nodes	overlay tree	in path nodes	**none**	tree root dependent
Key-based	path nodes	overlay key-based	in home nodes	*spam*	*no*
Semantic Affinity	path nodes	gossip--based	in nodes views	*duplicates*	*no*

This kind of analysis suggests that a successful and scalable system needs to be built by combining the strengths of several approaches. For example, one could try to use a hierarchical learning by the reverse path algorithm, with aggressive summarization, to meet large scale requirements, and resort to a full state replication approach for smaller clusters of nodes. Paper [15] presents an attempt in that direction. It is quite clear that the selection of a specific algorithm for a specific context must be done taking into consideration a deep understanding of the way the presented algorithms perform. Unfortunately, at present, this choice can hardly be based on rigorous analytical comparisons.

Many of the presented algorithms have been mostly evaluated by their authors using custom based simulators, and in several cases, using only synthetic data sets. The real value of these evaluations depends on the following factors: good network and overlay models, number of participants and their dynamism (number of servers and clients, churn and subscription evolution in general), and relevant notification and subscription data sets, models and distributions. In general, these issues are treated very differently from evaluation to evaluation. In certain cases, they are addressed in an incomplete manner. For example, claims of scalability are sometimes backed by evaluations where the number of nodes used is very small, even in P2P settings. Another common example of possibly incomplete evaluation pertains to the use of data sets with uniform distributions when this distribution favors the behavior of the proposed algorithm. With the exception of some quality evaluations based on real and representative notifications data sets (mostly stock quote or alert feeds), most data sets are synthetically generated. The same is true for most of the network and overlays models and configurations used. Only a few systems have been evaluated in a real network setting or considering real system logs as input for the model of churn and network configurations.

This state of affairs of the content-based communication evaluation is a consequence of the difficulties of the problem at hand. It is very multi-faceted problem, having a broad spectrum of configurations, contexts and applications. But, also, because we are still lacking well-accepted standard workloads and settings to drive evaluations and comparisons. In situations like these, the number of under evaluated and not comparable approaches tends to explode. That seems to be happening already, if the diversity of key-based and the semantic affinity proposals appearing in the literature is an indication. This is a real challenge for the community working in the content-based networking, as it poses a major obstacle to any serious intent to judge the relevance and the progress of the field.

ACKNOWLEDGEMENTS

The authors want to thank the anonymous reviewers by their helpful comments. This work has been partially supported by the Portuguese Ministry of Science, Technology and University Teaching under grant PTDC/EIA/76114/2006, Project *LiveFeeds – P2P Dissemination of Web Syndication Content*.

REFERENCES

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *SOSP '99: Proc. seventeenth ACM symposium on Operating systems principles*, pages 186–201, New York, NY, USA, 1999. ACM Press.

[2] M. Adler, Z. Ge, J. Kurose, D. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. *Network Protocols, 2001. Ninth International Conference on*, pages 100–109, 2001.

[3] I. Aekaterinidis and P. Triantafillou. PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network. *ICDCS'2006: Proc. 26th IEEE International Conference on Distributed Computing Systems*, 2006.

[4] L. Aguilar. Datagram routing for internet multicasting. *SIGCOMM Comput. Commun. Rev.*, 14(2):58–63, 1984.

[5] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. *Proc. eighteenth annual ACM symposium on Principles of distributed computing*, pages 53–61, 1999.

[6] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[7] E. Anceaume, M. Gradinariu, A. Datta, G. Simon, and A. Virgillito. A Semantic Overlay for Self-Peer-to-Peer Publish/Subscribe. *Proc. 26th IEEE International Conference on Distributed Computing Systems*, 2006.

[8] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. *ICDCS'2005: Proc. 25th IEEE International Conference on Distributed Computing Systems*, 5:437–446, 2005.

[9] R. Baldoni and A. Virgillito. Distributed event routing in publish/subscribe communication systems: a survey. Technical report, Technical Report TR-1/06. Dipartimento di Informatica e Sistemistica, Universitá di Roma 'La Sapienza'. http://www. dis. uniroma1. it/midlab/publications. php, 2006.

[10] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. *Proc. ACM SIGCOMM*, 93:85–95, 1993.

[11] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribesystems. *Distributed Computing Systems, 1999. Proc. 19th IEEE International Conference on*, pages 262–272, 1999.

[12] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2002)*, pages 205–217, 2002.

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

[14] M. Brahami, P. T. Eugster, R. Guerraoui, and S. B. Handurukande. Bgp-based clustering for scalable and reliable gossip broadcast. In C. Priami and P. Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 273–290. Springer, 2004.

[15] F. Cao and J. Singh. Medym: Match-early with dynamic multicast for content-based publish-subscribe networks. *Proc. ACM/IFIP/USENIX 6th International Middleware Conference (Middleware 2005)*, 2005.

[16] A. Carzaniga and C. P. Hall. Content-based communication: a research agenda. In *SEM '06: Proc. 6th international workshop on Software engineering and middleware*, pages 2–8, New York, NY, USA, 2006. ACM Press.

[17] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Computer Systems*, 19(3):332–383, 2001.

[18] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *INFOCOM*, 2004.

[19] R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. *Parallel Processing, 11th International Euro-Par Conference (Euro-par 2005)*, pages 1194–1204, 2005.

[20] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. *Proc. 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–390, 2000.

[21] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[22] P. Costa, M. Migliavacca, G. Picco, and G. Cugola. Introducing reliability in content-based publish-subscribe through epidemic algorithms. *Proc. 2nd international workshop on Distributed event-based systems*, pages 1–8, 2003.

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

[24] Y. K. Dalal and R. M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of ACM*, 21(12):1040–1048, 1978.

[25] S. Deering. Host extensions for IP multicasting. RFC 1112 (Standard), Aug. 1989. Updated by RFC 2236.

[26] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. Computer Systems (TOCS)*, 8(2):85–110, 1990.

[27] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proc. sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM Press.

[28] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra. Crew: A gossip-based flash-dissemination system. In *ICDCS '06: Proc. 26th IEEE International Conference on Distributed Computing Systems*, page 45, Washington, DC, USA, 2006. IEEE Computer Society.

[29] E. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1(269-270):6, 1959.

[30] S. Duarte, J. Martins, H. Domingos, and N. Preguiça. A case study on event dissemination in an active overlay network environment. *Proc. 2nd international workshop on Distributed event-based systems*, pages 1–8, 2003.

[31] S. M. Duarte. *DEEDS - A Distributed and Extensible Event Dissemination Service*. PhD thesis, New University of Lisbon, Feb 2006.

[32] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC 2362 (Experimental), June 1998.

[33] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[34] P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *The International Conference on Dependable Systems and Networks (DNS 2001)*, 2001.

[35] A. Gupta, O. Sahin, D. Agrawal, and A. Abbadi. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. *Proc. 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273, 2004.

[36] E. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. Park, and A. Vernon. Scalable Trigger Processing. *Proc. 15th International Conference on Data Engineering*, pages 266–275, 1999.

[37] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.

[38] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *Proc. ACM/IEEE International Conference on Mobile Computing and Networking*, pages 56–67, 2000.

[39] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *The Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, 2000.

[40] L. Ji and M. S. Corson. Explicit multicasting for mobile ad hoc networks. *Mob. Netw. Appl.*, 8(5):535–549, 2003.

[41] Y. Liu and B. Plale. Survey of Publish Subscribe Event Systems. *Indiana University Computer Science Technical Report TR-574*, 2003.

[42] J. Moy. OSPF Protocol Analysis. RFC 1245 (Informational), July 1991.

[43] J. Moy. MOSPF: Analysis and Experience. RFC 1585 (Informational), Mar. 1994.

[44] G. Mühl, L. Fiege, F. C. Gärtner, and A. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. In *MASCOTS '02: Proc. 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*, page 167, Washington, DC, USA, 2002. IEEE Computer Society.

[45] G. Muhl, L. Fiege, and P. R. Pietzuch. *Distributed Event-Based Systems*. Springer Verlag, Berlin, Germany, 2006.

[46] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, pages 185–207, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[47] S. Pallickara and G. Fox. NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. *Proc. International Middleware Conference*, 2003.

[48] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01)*, pages 49–60, San Francisco, CA, USA, Mar. 2001.

[49] P. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.

[50] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[51] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proc. 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[52] A. Riabov, Z. Liu, J. Wolf, P. Yu, and L. Zhang. Clustering algorithms for content-based publication-subscription systems. *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 133–142, 2002.

[53] L. Rodrigues, S. B. Handurukande, J. O. Pereira, R. Guerraoui, and A.-M. Kermarrec. Adaptive gossip-based broadcast. In *DSN*, pages 47–56. IEEE Computer Society, 2003.

[54] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proc. IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, November 2001. Springer-Verlag.

[55] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[56] A. Snoeren, K. Conley, and D. Gifford. Mesh-based content routing using xml. *Proc. eighteenth ACM symposium on Operating systems principles*, pages 160–173, 2001.

[57] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'01)*, pages 149–160. ACM Press, 2001.

[58] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Trans. Netw.*, 11, 2003.

[59] D. Tam, R. Azimi, and H. Jacobsen. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. *Proc. Databases, Information Systems, and Peer-to-Peer Computing*, 3:138–152, 2003.

[60] A. Tanenbaum. Computer Networks — 4th edition, 2004.

[61] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS '03: Proc. 2nd international workshop on Distributed event-based systems*, pages 1–8, New York, NY, USA, 2003. ACM.

[62] P. Triantafillou and I. Aekaterinidis. Content-based Publish-Subscribe Over Structured P2P Networks. *1st International Workshop on Discrete Event-Based Systems*, 2004.

[63] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *J. Network and Systems Management*, 13(2):197–217, 2005.

[64] S. Voulgaris, E. Riviere, A. Kermarrec, and M. van Steen. Sub-2-sub: Self-organizing content-based publish and subscribe for dynamic and

large scale collaborative networks. In *IPTPS'06: the fifth International Workshop on Peer-to-Peer Systems*, 2006.

[65] S. Voulgaris and M. van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Int'l Conf. on Parallel and Distributed Computing (Euro-Par 2005)*, pages 1143–1152. Springer, 2005.

[66] Y. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. Wang. Subscription Partitioning and Routing in Content-based Publish/Subscribe Networks. *16th International Symposium on DIStributed Computing (DISC'02)*, 2002.

[67] Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, and P. Larson. Summary-based routing for content-based event distribution networks. *SIGCOMM Comput. Commun. Rev.*, 34(5):59–74, 2004.

[68] C. Zhang, A. Krishnamurthy, R. Y. Wang, and J. P. Singh. Combining flexibility and scalability in a peer-to-peer publish/subscribe system. In G. Alonso, editor, *Middleware*, volume 3790 of *Lecture Notes in Computer Science*, pages 102–123. Springer, 2005.

[69] R. Zhang and Y. Hu. HYPER: A Hybrid Approach to Efficient Content-Based Publish/Subscribe. *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 427–436, 2005.

J. Legatheaux Martins is a professor of distributed systems and computer networks and head of the Department of Informatics of FCT/UNL, Universidade Nova de Lisboa, Portugal. In 1986 he received a PhD in distributed systems from Université Rennes II, France. From 1983 to 1987 he was with the Chorus Operating System team at INRIA, Paris, France. From 1988 to 1992 he was an assistant professor at Universidade de Lisboa, Portugal. In 1993 he joined Universidade Nova de Lisboa as an associate professor. In 1994 he was the founder of one of the first Internet Service Providers operating in Portugal, that has been acquired in 2000 by the US based Qwest Communications Company. His research interests include data management for mobile systems, and many-to-many communication systems based on network-level, overlay and peer-to-peer architectures. He is a member of IEEE and the ACM.

Sérgio Duarte is an assistant professor at DI/FCT/UNL, Universidade Nova de Lisboa, Portugal, where he received his PhD, in 2006. His main research interests concern content-based routing and publish/subscribe architectures, namely solutions based on structured overlays and peer-to-peer networking principles.