Designing Semantic Publish/Subscribe Networks Using Super-Peers

Paul - Alexandru Chirita[1], Stratos Idreos[2], Manolis Koubarakis[2], and Wolfgang Nejdl[1]

[1] L3S and University of Hannover Deutscher Pavillon Expo Plaza 1
30539 Hannover, Germany
{chirita, nejdl}@learninglab.de

[2] Intelligent Systems Laboratory, Department of Electronic and Computer Engineering, Technical University of Crete, 73100 Chania, Crete, Greece
{sidraios, manolis}@intelligence.tuc.gr

Summary. Publish/subscribe systems are an alternative to query-based systems in cases where the same information is asked for over and over, and where clients want to get updated answers for the same query over a period of time. Recent publish/subscribe systems such as P2P-DIET have introduced this paradigm in the P2P context. In this chapter we built on the experience gained with P2P-DIET and the Edutella super-peer infrastructure and present a semantic publish/subscribe system supporting metadata and a query language based on RDF. We define formally the basic concepts of our system and present detailed protocols for its operation.

1 Introduction

Consider a P2P network which manages metadata about publications, and a user of this network, Bob, who is interested in the *new* publications of some specific authors, e.g., Koubarakis and Nejdl. With conventional P2P file sharing networks like Gnutella or Kazaa, this is really difficult, because sending out queries which either include "Koubarakis" or "Nejdl" in the search string will return all publications from these authors, and Bob has to filter out the new publications each time. With an RDF-based P2P network like Edutella [29], this is a bit easier, because Bob can formulate a query, which includes a disjunction for the attribute `dc:creator` (i.e., `dc:creator` includes "Nejdl" or `dc:creator` includes "Koubarakis"), as well as a constraint on the date attribute (i.e., `dc:date > 2003`), which includes all necessary constraints in one query and will only return answers containing publications from 2004 on. Still, this is not quite what Bob wants, because whenever he uses this query, he will get all 2004 publications including the ones he has already seen.

What Bob really needs from his P2P file sharing network are *publish/subscribe* capabilities [11]:

1. *Advertising*: Peers send information about the content they will publish, for example a Hannover peer announces that it will make available all L3S publications, including publications from Nejdl, a Crete peer announces that it would do the same for Koubarakis' group.
2. *Subscribing*: Peers send subscriptions to the network, defining the kind of documents they want to retrieve. Bob's profile would then express his subscription for Nejdl and Koubarakis papers. The network might store these subscriptions near the peers which will provide these resources, in our case near the Hannover and the Crete peer.
3. *Notifying*: Peers notify the network whenever new resources become available. These resources should be forwarded to all peers whose subscription profiles match them, so Bob should regularily receive all new publications from Nejdl and Koubarakis.

In this chapter we will describe how to provide publish/subscribe capabilities in an RDF-based P2P system, which manages arbitrary digital resources, identified by their URI and described by a set of RDF metadata. This functionality is useful in many application scenarios including distributed educational content repositories in the context of the EU/IST project ELENA [36, 1] whose participants include e-learning and e-training companies, learning technology providers, universities and research institutes. A second application scenario that interests us is information alert in distributed digital library environments [23].

The organization of this chapter is as follows. The next section specifies the formal framework for RDF-based pub/sub systems, including the languages used to express publications and subscriptions in our network. Section 3 presents our super-peer architecture and compares it briefly with other alternatives. Section 4 discusses the most important design aspects and optimizations necessary to handle large numbers of subscriptions and notifications, building upon the super-peer architecture and HyperCuP protocol implemented in the Edutella system [29], as well as on index optimizations recently explored in P2P-DIET [24]. Section 5 includes a short discussion of other important features of our system, and Section 6 includes a survey of related work. Section 7 concludes the chapter.

2 A Formalism for Pub/Sub Systems Based on RDF

In this section we formalize the basic concepts of pub/sub systems based on RDF: advertisements, subscriptions, and notifications. We will need a *typed first-order language* $\mathcal{L}$. $\mathcal{L}$ is equivalent to a subset of the Query Exchange Language (QEL) but has a slightly different syntax that makes our presentation more formal. QEL is a Datalog-inspired RDF query language that is used in the Edutella P2P network [28].

The logical symbols of $\mathcal{L}$ include parentheses, a countably infinite set of variables (denoted by capital letters), the equality symbol $=$ and the standard sentential connectives. The parameter (or non-logical) symbols of $\mathcal{L}$ include types, constants and predicates. $\mathcal{L}$ has four types: $\mathcal{U}$ (for *RDF resource identifiers* i.e., *URI references* or *URIrefs*), $\mathcal{S}$ (for RDF literals that are *strings*), $\mathcal{Z}$ (for RDF literals that are *integers*), and $\mathcal{UL}$ (for the union of RDF resource identifiers and RDF literals that are strings or integers). The predicates of our language are $<$ of type $(\mathcal{Z}, \mathcal{Z})$, $\sqsupseteq$ of type $(\mathcal{S}, \mathcal{S})$, and t of type $(\mathcal{U}, \mathcal{U}, \mathcal{UL})$. Predicate $<$ will be used to compare integers, predicate $\sqsupseteq$ (read "contains") will be used to compare strings and t (read "triple") will be used to represent *RDF triples*. Following the RDF jargon, in an expression $t(s, p, o)$, s will be called the *subject*, p the *predicate* and o the *object* of the triple.

The well-formed formulas of $\mathcal{L}$ (atomic or complex) can now be defined as usual. We can also define a semantics for $\mathcal{L}$ in the usual way. Due to space considerations, we omit the technical details.

The following definitions give the syntax of our subscription language.

Definition 1. *An* atomic *constraint is a formula of $\mathcal{L}$ in one of the following three forms: (a) $X = c$ where X is a variable and c is a constant of type $\mathcal{U}$, (b) $X \ r \ c$ where X is a variable of type $\mathcal{Z}$, c is a constant of type $\mathcal{Z}$ and r is one of the binary operators $=, <, \leq, >, \geq$, and (c) $X \sqsupseteq c$ where X is a variable and c is a constant, both of type $\mathcal{S}$. A* constraint *is a disjunction of conjunctions of atomic constraints (i.e., it is in DNF form).*

We can now define the notion of a *satisfiable* constraint as it is standard.

Definition 2. *A* query (subscription) *is a formula of the form*

$$X_1, \ldots, X_n : t(S, p_1, O_1) \wedge t(S, p_2, O_2) \wedge \cdots \wedge t(S, p_m, O_m) \wedge \phi$$

where S is a variable of type $\mathcal{U}$, $p_1, \ldots, p_m$ are constants of type $\mathcal{U}$, $O_1, \ldots, O_m$ are distinct *variables of type $\mathcal{UL}$, $\{X_1, \ldots, X_n\} \subseteq \{S, O_1, \ldots, O_m\}$, and ϕ is a constraint involving a subset of the variables $S, O_1, \ldots, O_m$.*

The above definition denotes the class of *single-resource multi-predicate* queries in QEL. This class of queries can be implemented efficiently (as we will show in Section 4) and contains many interesting queries for P2P file sharing systems based on RDF. It is easy to see that only *join* on variable S is allowed by the above class of queries (i.e., S is a subject *common to all* triples appearing in the subscription).

As it is standard in RDF literature, the triple notation utilizes *qualified names or QNames* to avoid having to write long formulas. A QName contains a prefix that has been assigned to a namespace URI, followed by a colon, and then a *local name*. In this chapter, we will use the following prefixes in QNames:

```
@prefix dc: <http://purl.org/dc/elements/1.1/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix isl: <http://www.intelligence.tuc.gr/publications/>
```

Example 1. The subscription “I am interested in articles authored by Nejdl or Koubarakis in 2004” can be expressed by the following subscription:[1]

```
X: t(X,<rdf:type>, <dc:article>) ∧ t(X,<dc:creator>,Y) ∧
   t(X,<dc:date>,D) ∧(Y ⊒ "Nejdl" ∨ Y ⊒ "Koubarakis") ∧ D=2004
```

Let q be a query. We will use the functions $schemas(q)$ and $properties(q)$ to refer to the sets of schemas (namespaces) and properties that appear in q. For instance, if q is the query of Example 1 then $schemas(q) = \{dc\}$ and $properties(q) = \{< dc : article >, < dc : creator >, < dc : date >\}$.

Queries (subscriptions) are evaluated over sets of RDF triples. If T is a set of RDF triples, then $ans(q, T)$ will denote the answer set of q when it is evaluated over T. This concept can be formally defined as for relational queries with constraints.

We can now define the concept of subscription subsumption that is heavily exploited in the architecture of Section 4.

Definition 3. *Let q_1, q_2 be subscriptions. We will say that q_1* subsumes *q_2 iff for all sets of RDF triples T, $ans(q_2, T) \subseteq ans(q_1, T)$.*

We now define the concept of *notification*: the meta-data clients send to super-peers whenever they make available new content. Notifications and subscriptions are matched at super-peers and appropriate subscribers are notified.

Definition 4. *A* notification *n is a pair (T, I) where T is a set of ground (i.e., with no variables) atomic formulas of $\mathcal{L}$ of the form $t(s, p, o)$ with the* same *constant s (i.e., a set of RDF triples with the* same *subject-URIref) and I is a client identifier. A notification $n = (T, I)$* matches *a subscription q if $ans(q, T) \neq \emptyset$.*

Notice that because URIrefs are assumed to be *unique*, and subscriptions and notifications obey Definitions 2 and 4, notification matching in the architecture of Section 4 takes place *locally* at each super-peer.

Example 2. The notification

```
({t(<isl:esws04.pdf>, <rdf:type>, <dc:article>),
  t(<isl:esws04.pdf>, <dc:creator>, "Koubarakis"),
  t(<isl:esws04.pdf>, <dc:date>, 2004)}, C3)
```

matches the subscription of Example 1.

We now define three progressively more comprehensive kinds of advertisement. Advertisements formalize the notion of what clients or super-peers send to other nodes of the network to describe their content in a *high-level intentional* manner. Super-peers will match client subscriptions with advertisements to determine the routes that subscriptions will follow in the architecture of Section 4. This is formalized by the notion of “covers” below.

[1] Sometimes we will abuse Definition 2 and write a constant o_i in the place of variable O_i to avoid an extra equality $O_i = o_i$ in ϕ.

Definition 5. *A* schema advertisement *d is a pair (S, I) where S is a set of schemas (constants of type $\mathcal{U}$ i.e., URIrefs) and I is a super-peer id. If $d = (S, I)$ then the expression $schemas(d)$ will also be used to denote S. A schema advertisement d* covers *a subscription q if $schemas(q) \subseteq schemas(d)$.*

Example 3. The schema advertisement `({dc, lom}, SP`$_1$`)` covers the subscription of Example 1.

Definition 6. *A* property advertisement *d is a pair (P, I) where P is a set of properties (constants of type $\mathcal{U}$ i.e., URIrefs) and I is a super-peer identifier. If $d = (P, I)$ then the expression $properties(d)$ will also be used to denote P. A property advertisement d* covers *a subscription q if $properties(q) \subseteq properties(d)$.*

Example 4. The property advertisement `({<dc:article>, <dc:creator>, <dc:date>, <dc:subject>, <lom:context>}, SP`$_6$`)` covers the subscription of Example 1.

Definition 7. *A* property/value advertisement *d is a pair $((P_1, V_1), \dots, (P_k, V_k)), I)$ where $P_1, \dots, P_k$ are* distinct *properties (constants of type $\mathcal{U}$ i.e., URIrefs), $V_1, \dots, V_k$ are sets of values for $P_1, \dots, P_k$ (constants of type $\mathcal{UL}$) and I is a super-peer identifier.*

Definition 8. *Let q be a subscription of the form of Definition 2 and d be a property/value advertisement of the form of Definition 7. Let $Y_1, \dots, Y_k$ $(1 \le k \le m)$ be the variables among the objects $o_1, \dots, o_m$ of the triples of q that correspond to the properties $P_1, \dots, P_k$ of d. We will say that d* covers *a subscription q if there exist values $v_1 \in V_1, \dots, v_k \in V_k$ such that the constraint $\phi[Y_1 \leftarrow v_1, \dots, Y_k \leftarrow v_k]$ resulting from substituting variables $Y_1, \dots, Y_k$ with constants $v_1, \dots, v_k$ in ϕ is satisfiable.*

Example 5. The property/value advertisement

```
( (<dc:creator>, { W. Nejdl, P. Chirita}),
  (<dc:title>, {"Algorithms", "Data Structures"}),
  (<dc:year>, [2002, ∞]), SP1 )
```

covers the subscription of Example 1.

3 The Super-Peer Architecture

The algorithms that we present in this chapter are designed for super-peer systems [45]. Thus, we assume two types of nodes: *super-peers* and *peers*. A peer is a typical network node that wants to advertise and publish its data and/or subscribe to data owned by others. A super-peer is a node with more capabilities than a peer (e.g., more cpu power and bandwidth). Staying on-line for long periods of time is another desirable property for super-peers. In

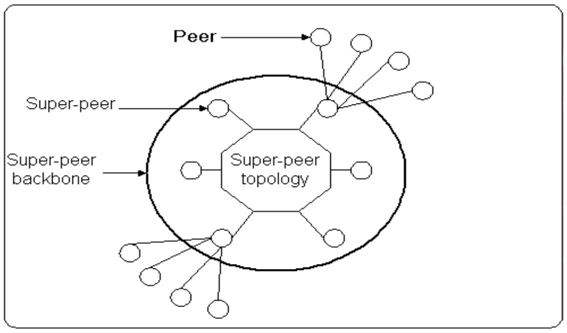


Fig. 1. An example of a super-peer architecture

our architecture, super-peers are organized in a separate network which we call the *super-peer backbone* and are responsible for processing notifications, advertisements and subscriptions. Peers connect to super-peers in a star-like fashion, providing content and content metadata. Each peer is connected to a single super-peer which is its *access point* to the rest of the network and its services. Once connected, a peer can disconnect, reconnect or even migrate to a different super-peer. A high level view of this architecture is shown in Figure 1.

Our super-peers are arranged in the HyperCuP topology. This is the solution adopted in the Edutella infrastructure [28] because of its special characteristics regarding broadcasts and network partitioning. An example of this architecture is shown in Figure 2. The HyperCuP algorithm, described in [27], is capable of organizing the super-peers of a P2P network into a binary hypercube, a member of the family of Cayley graphs. Super-peers join the network by contacting any of the already integrated super-peers which then carries out the super-peer integration protocol. No central maintenance is necessary. HyperCuP enables efficient and non-redundant broadcasts. For broadcasts, each node can be seen as the root of a specific spanning tree through the super-peer backbone, as shown in Figure 2. The topology allows for $log_2 N$ path length between any two peers and $log_2 N$ number of neighbors for each peer, where N is the total number of nodes in the network (i.e., the number of super-peers in this case).

Super-peer architectures are usually based on a two-phase routing protocol, which routes messages first in the super-peer backbone and then distributes them to the peers connected to the super-peers. Super-peer based routing can be based on different kinds of indexing and routing tables, as discussed in [14, 29]. In the following sections we present indexing and routing mechanisms appropriate for publish/subscribe services.

In this chapter we do not deal with the question of “Who becomes a super-peer?”. As an example, super-peers can be centrally managed by a company that owns and runs the overlay to offer a service (e.g., a content provider such

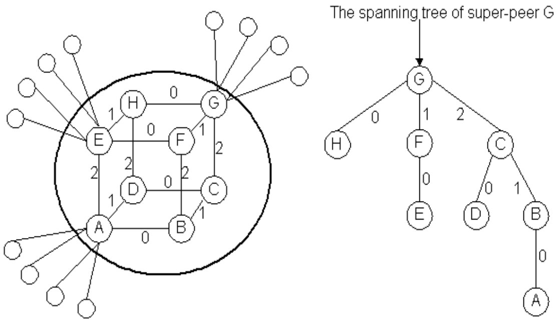
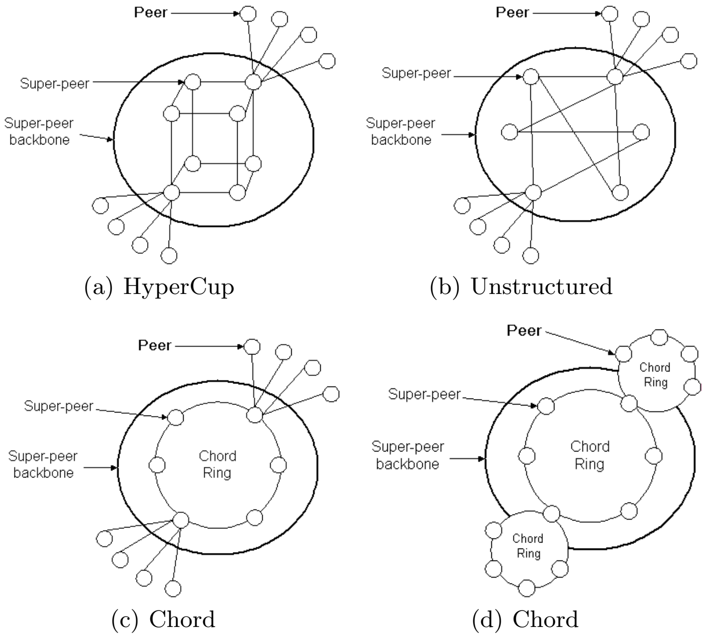


Fig. 2. The HyperCuP topology and an example of a spanning tree

as Akamai). A more challenging design is that super-peers are normal peers that either volunteer to play the role of a super-peer for a time window (i.e., because they will get a number of privileges as a return) or the system forces all peers to become super-peers periodically in order to be able to use the services of the overlay. This is an area where some interesting research has been carried out recently e.g., [26, 31]. The authors of [31] introduce the concept of altruistic peers, namely peers with the following characteristics, (a) they stay on line for long periods and (b) they are willing to offer a significant portion of their resources to speedup the performance of the network. Although [31] does not uses the term super-peer directly, the concepts of super-peers and altruistic peers are related: one can view altruistic peers as one kind of super-peers in a P2P network.

Alternatives to this topology are possible, provided that they guarantee the spanning tree characteristic of the super-peer backbone, which we exploit for maintaining our index structures. For example, the super-peers may form an unstructured overlay like in P2P-DIET [21]. P2P-DIET does not force any kind of structure between super-peers. Instead it lets super-peers choose their neighbour peers. An example is shown in Figure 3(b). A minimum weight spanning tree is formed for each super-peer based on the algorithm presented in [7]. Then brodcasting in the super-peer backbone takes place according to a well-known and widely used solution, *reverse path forwarding* [46]. This is a very simple technique with minimum storage requirements for the nodes of the network. According to reverse path forwarding, a node that receives a message will accept it only if the sender is part of the shortest path that connects this node with the node that generated and broadcasted the original message (the root of the message). Then, it will forward the message to all its neighbors except the sender.

A crucial difference between an unstructured and a structured super-peer topology is the depth of the spanning tree which is unbounded for the structured topologies but bounded for unstructured ones. The HyperCuP protocol can limit this depth to *logN* by forcing the HyperCuP structure as super-



(a) HyperCup (b) Unstructured

(c) Chord (d) Chord

Fig. 3. Various super-peer architectures

peers join or leave (see Figure 3(a)). Of course, this brings an extra cost to the join and leave operation for the super-peers but it is a cost that we are willing to pay given our wish for efficient query processing. The main advantage of limiting the depth of the spanning tree is the low latency achieved for broadcast operations. For example, consider the extreme case of an unstructured super-peer backbone where the super-peers form a chain. In this case, if a super-peer at the one end of the chain decides to broadcast a message, then the super-peer at the other end will see the message only after all other peers have received it (i.e., after $N-1$ steps). The advantage of HyperCup is that it limits the path between any two super-peers to $log_2 N$ which is much better than the $N-1$ path length of the unstructured design of the previous example.

Another architectural choice for the super-peer backbone is to organize the super-peers according to a distributed hash table based protocol like Chord [37] as shown in Figure 3(c). An interesting approach is that even the peers attached to a super-peer can be organized according to a DHT based protocol. An example is shown in Figure 3(d). A good discussion of such possible architectural choices can be found in [31].

4 Processing Advertisements, Subscriptions and Notifications

In this section we present protocols for processing advertisements, subscriptions and notifications in a super-peer based network. In addition, we discuss the data structures and indices that a super-peer uses.

4.1 Processing Advertisements

Once a peer connects to the network, it is *mandatory* to advertise the kind of resources it will offer in the future. For example, an advertisement can include information on the schema that a peer supports. As we have already discussed, each peer is attached to one super-peer, which is its access point to the rest of the network. Thus, a peer constructs an *advertisement* and sends it to its access point. A peer will send an advertisement again if its information needs to be updated.

A super-peer receives advertisements from all peers that are attached to it. A super-peer uses these advertisements to construct *advertisement routing indices* that are utilized when processing subscriptions. There are three levels of indexing: the schema level, the property (attribute) level, and the property/value level. Each time an advertisement arrives from one of the peers, the super-peer updates those three indices. In the following paragraphs we give a description of each index.

Schema Index. The first level of indexing contains information on the schema that peers support. We assume that different peers will support different RDF schemas and that these schemas can be uniquely identified (e.g., by a URI). The schema index contains zero or more schema identifiers. Each schema identifier points to one or more peers that support this schema. As an example of the use of this index, we can say that subscriptions are forwarded only to peers which support the schemas used in the subscription. We discuss about this in more detail in the next section.

Property Index. The second level of indexing is the property index. This index is useful in cases where a peer might choose to use only part of one or more schemas, i.e., certain properties/attributes, to describe its content. While this is unusual in conventional database systems, it is more often used for data stores that use semi-structured data, and very common for RDF-based systems. In such a case, the schema index cannot be used and indexing is done at the property level. Thus, the property index contains properties, uniquely identified by name space/schema ID plus property name. Each property points to one or more peers that support them.

Property/Value Index. Finally, the third index is the property/value index. For many properties it will be advantageous to create a value index to reduce network traffic. This case is identical to a classical database index with the exception that the index entries do not refer to the resource, but the peer providing it.

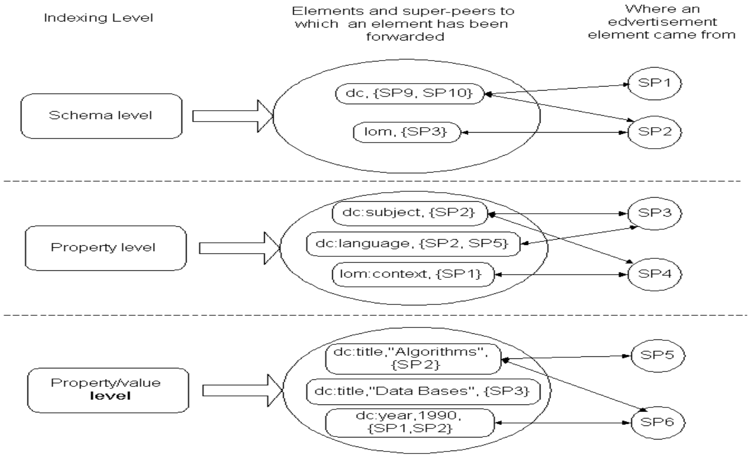


Fig. 4. An example of SP/SP advertisement routing indices

We use two kinds of indices, namely the super-peer/super-peer indices (*SP/SP indices*) that handle communication in the super-peer backbone and the super-peer/peer indices (*SP/P indices*) that handle communication between a super-peer and all peers connected to it. These indices draw upon our previous work for query routing, as discussed in [29], as well as further extensions and modifications necessary for publish/subscribe services based on [24, 21]. Except for the functionality they employ, both indices use the same data structures, have the same update process, etc. Figure 4 shows an example of super-peer/super-peer indices.

Let us now discuss how a super-peer reacts upon receiving an advertisement. Assume a super-peer SP_i that receives a new advertisement d from a peer p which is one of the peers that are directly connected to SP_i. First the new advertisement has to be inserted in the local indices as follows. An advertisement contains one or more elements. Each element is either a property value pair, for example, {`<dc:year>, 1900`}, or a property, for example, `<dc:year>` or a schema identifier, for example, `<dc>`. For each element e, SP_i does the following.

1. If e is a property value pair then e is inserted in the property/value index.
2. If e is just a property then it is inserted in the property index.
3. If e is just a schema identifier then it is inserted in the schema index.

After having updated the local indices, the advertisement is *selectively broadcasted* from SP_i to reach other super-peers. This is necessary so that subscriptions of peers that are attached to other access points are able to reach SP_i and the peer that advertised d. Thus, for each super-peer SP_x that is a child of SP_i in the spanning tree of SP_i, SP_i does the following. For each element e of d, SP_i checks if it has already forwarded an identical element of

e to SP_x, i.e., because of an advertisement of another peer. If this is not true then e is forwarded to SP_x.

When another super-peer, say SP_y, receives a forwarded element e, it inserts it into its local indices as described above. Then for each super-peer SP_x that is a child of SP_y in the spanning tree of SP_i (that originally initiated the selective broadcast procedure), SP_y checks if it has already forwarded an identical element of e to SP_x. If not, the element is forwarded to SP_x.

Updating Advertisement Indices. Index updates as discussed above are triggered when (a) a new peer connects, (b) a peer leaves the system permanently, (c) a peer migrates to another access point, or (d) the metadata information of a registered peer changes.

In the case of a peer joining the network, its respective metadata/schema information are matched against the SP/P entries of the respective super-peer. If the SP/P advertisement indices of the super-peer already contain the peers' metadata, only a reference to the peer is stored in them. Otherwise the respective metadata with references to the peer are added to SP/P indices and then are selectively broadcasted to the rest of the super-peer backbone. If a peer leaves from a super-peer AP permanently, then all references to this peer have to be removed from the SP/P indices of AP. If no other peer attached to AP supports the same metadata/schema information, then AP has to selectively broadcast a remove message to the rest of the super-peers so believes that AP supports this schema anymore. This is done in the same way as selectively broadcasting advertisements with the difference that now advertisements are removed from the SP/SP indices. In the case that a peer x migrates from an access point $AP1$ to an access point $AP2$, then $AP1$ acts as if x left permanently while $AP2$ acts as if x just joined the network.

4.2 Processing Subscriptions

Until now we have described how a super-peer reacts upon receiving an advertisement either from a peer or from a super-peer. In this section we present the protocol used by a peer for inserting a subscription in the network. Remember that the purpose of the subscription is that the subscriber peer will receive all future matching notifications from all peers of the network.

A peer always sends a new subscription to its access point. When a super-peer receives a subscription, it inserts it into the *local subscription poset.* A subscription poset is a partially ordered set (a hierarchical structure of subscriptions), and captures the notion of subscription subsumption defined in Section 2. Figure 5 shows an example of a poset. Each super-peer adds to its local subscription poset information about where the subscription came from (either from one of the peers connected to it or from another super-peer). The addition of super-peer information in the poset reduces the overall network traffic and is therefore very important. The use of subscription posets in publish/subscribe systems was originally proposed in SIENA [11].

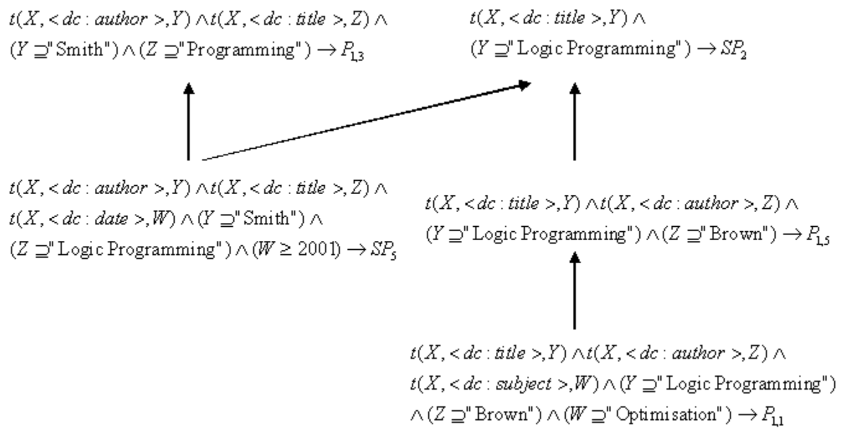

$t(X, <dc: author>, Y) \wedge t(X, <dc: title>, Z) \wedge$
$(Y \sqsupseteq \text{"Smith"}) \wedge (Z \sqsupseteq \text{"Programming"}) \rightarrow P_{1,3}$
$t(X, <dc: title>, Y) \wedge$
$(Y \sqsupseteq \text{"Logic Programming"}) \rightarrow SP_2$
$t(X, <dc: author>, Y) \wedge t(X, <dc: title>, Z) \wedge$
$t(X, <dc: date>, W) \wedge (Y \sqsupseteq \text{"Smith"}) \wedge$
$(Z \sqsupseteq \text{"Logic Programming"}) \wedge (W \geq 2001) \rightarrow SP_5$
$t(X, <dc: title>, Y) \wedge t(X, <dc: author>, Z) \wedge$
$(Y \sqsupseteq \text{"Logic Programming"}) \wedge (Z \sqsupseteq \text{"Brown"}) \rightarrow P_{1,5}$
$t(X, <dc: title>, Y) \wedge t(X, <dc: author>, Z) \wedge$
$t(X, <dc: subject>, W) \wedge (Y \sqsupseteq \text{"Logic Programming"})$
$\wedge (Z \sqsupseteq \text{"Brown"}) \wedge (W \sqsupseteq \text{"Optimisation"}) \rightarrow P_{1,1}$

Fig. 5. An example of a poset

Like SIENA, our system utilizes the subscription poset to minimize network traffic, i.e., super-peers do not forward subscriptions which are subsumed by previously forwarded subscriptions. In this way, when a super-peer receives a subscription, it inserts it into the local poset and decides whether to further forward it in the super-peer backbone or not. If this super-peer has already forwarded a subscription that subsumes the new one, then no forwarding takes place. If not, then the subscription has to be forwarded to all neighbour super-peers (according to the spanning tree of the super-peer that broadcasted the subscription) that may have peers that will create matching notifications. Once a super-peer has decided to send the subscription further, it will initiate a selective broadcast procedure. This procedure depends on the advertisement routing indices of the super-peer in the following way.

Assume a super-peer SP_i that receives a new subscription q from one of the peers that are directly connected to it. Then for each super-peer SP_x that is a child of SP_i in the spanning tree of SP_i, SP_i performs the following steps.

1. If the local index at the property/value level contains SP_x and one or more of its advertisements cover the subscription q, i.e., the values of the properties in an advertisement are consistent with the constraints of q, then q is forwarded to SP_x.
2. If the above is not true, then if the indices in the schema or property level contain SP_x and one or more of its advertisements cover the targeted schema (or properties) used in the subscription q, then q is forwarded to SP_x.

When another super-peer, say SP_y, receives a forwarded subscription q, it inserts it into its local subscription poset. Then, for each super-peer SP_x that is a child of SP_y in the spanning tree of SP_i (that originally initiated the selective broadcast procedure), SP_y checks if it has already forwarded a subscription q' to SP_x that subsumes q. If not, q is forwarded to SP_x.

Forwarding subscriptions when advertisement indices are updated. As we have already discussed, when a super-peer receives a subscription q, it forwards q to a portion of its neighbor super-peers or it does not forward it at all, according to the local subscription poset and advertisement indices. However, advertisement indices may change in the future so the following process takes place each time a new advertisement arrives to a super-peer AP from another super-peer $AP1$: AP has to check if there are one or more subscriptions in its local subscription poset that (a) cover the new advertisement and (b) have not been forwarded towards $AP1$. Any found subscriptions are sent to $AP1$. This is necessary to happen in order not to ignore future matching notifications that are generated by $AP1$ or by other super-peers that forward those notifications through $AP1$.

4.3 Processing Notifications

Let us now discuss how subscriptions are triggered by notifications. A peer creates a notification for a new resource that it wants to make available to other peers, and publishes the notification to the network. The goal is that all appropriate subscriptions are triggered and their subscribers receive the notification.

A peer always forwards a new notification to its access point. A notification is a message that contains metadata about the new item and additional information on the peer that makes it available, i.e., its IP address and its identifier. The schema that is used to create a notification has to agree with the schema that this peer has previously advertised, otherwise the protocols cannot guarantee that all relevant subscribers will receive the notification.

When a new notification n arrives at a super-peer with a database db of local subscriptions, the super-peer has to find all subscriptions $q \in db$ that satisfy n. This can be done as follows using ideas from SIENA [11]. The new notification is first matched against the root subscriptions of a super-peer's local subscription poset. In case of a match with the subscription stored in a root node R, the notification is further matched against the children of R, which contain subscriptions refining the subscription from R. For each match, the notification is sent to a group of peers/super-peers (those where the subscription came from), thus following backwards the exact path of the subscription. It is also possible to use more sophisticated filtering algorithms like those in P2P-DIET [42] and those in [6, 12, 16, 19].

5 Dynamics of P2P Pub/Sub Networks

As peers dynamically join and leave the network, they may be off-line when new resources arrive for them. These are lost if no special precautions are taken. In the following paragraphs, we discuss which measures are necessary to enable peers to receive notifications that are generated when those peers are off-line.

5.1 Offline Notifications and Rendezvous at Super-Peers

Whenever a peer A disconnects from the network, its access point AP *keeps* the peer's identification information and subscriptions for a specified period of time, and its indices will not reflect that A has left the network. This means that notifications for A will still arrive at AP, which has to store these and deliver them to A after it reconnects. A peer may request a resource at the time that it receives a notification n, or later on, using a saved notification n on his local *notifications directory*.

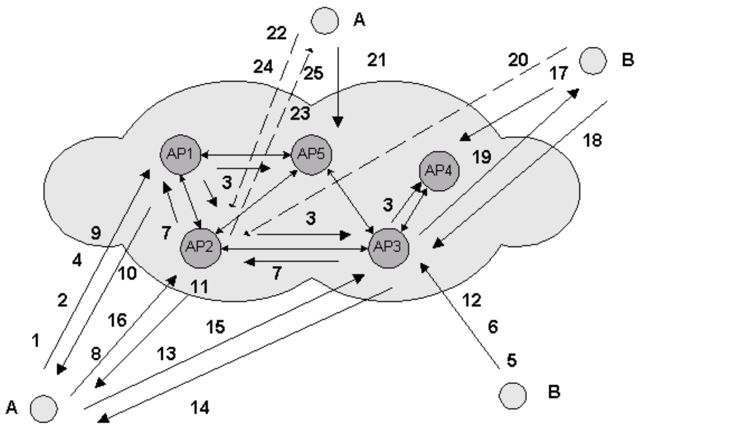
Let us now consider the case when a peer A requests a resource r, but the resource owner peer B is not on-line. Peer A requests the address of B directly from *AP2* (the access point of B). This is feasible since the address of $AP2$ is included in r. In such a case, peer A may request a *rendezvous* with resource r from *AP2* with a message that contains the identifier of A, the identifier of B, the address of AP and the location of r. When peer B reconnects, *AP2* informs B that it must upload resource r to AP as a *rendezvous file* for peer A. Then, B uploads r. AP checks if A is on-line and if it is, AP forwards r to A or else r is stored in the *rendezvous directory* of AP and when A reconnects, it receives a rendezvous notification from AP.

The features of off-line notifications and rendezvous take place even if peers migrate to different access points. For example, let us assume that peer A has migrated to *AP3*. The peer program understands that it is connected to a different access point *AP3*, so it requests from AP any rendezvous or off-line notifications and informs AP that it is connected to a different access point. A receives the rendezvous and off-line notifications and updates the variable's *previous access point* with the address of *AP3*. Then, AP updates its SP/P and SP/SP indices. Finally, A sends to *AP3* its subscriptions and *AP3* updates its SP/P and SP/SP indices. A complete example is shown in Figure 6.

5.2 Peer Authentication

Typically, authentication of peers in a P2P network is not crucial, and peers connecting to the network identify themselves by just using their IP-addresses. In a pub/sub environment, however, where we have to connect peers with their subscriptions and want to send them all notifications relevant for them, this leads to two problems:

- IP addresses of peers may change. Therefore the network will not be able to deliver any notifications, which might have been stored for a peer during its absence, after it reconnects with another IP address. Furthermore, all



1. *A* connects to *AP1*
2. *A* subscribes **to** *AP1* with a continuous query *q*
3. *AP1* broadcsats *q* to all super-peers
4. *A* disconnects from *AP1*
5. *B* connects to *AP3*
6. *B* publishes a resource *r*
7. *AP3* generates a notification *n* for *A* and unicasts *n* to *AP1*
8. *A* connects to ***AP2*** (*migration*)
9. *A* requests stored data from *AP1*
10. *AP1* sends the stored notification *n* to *A*
11. *AP2* request resource metadata from *A*
12. *B* disconnects from *AP3*
13. *A* requests address of *B* from *AP3*
14. *AP3* replies that *B* is disconnected
15. *A* requests from *AP3* to arrange a rendezvous wiith *r*
16. *A* disconnects from *AP2*
17. *B* connects to ***AP4*** (*migration*)
18. *B* requests stored data from *AP3*
19. *AP3* informs client *B* that it must upload *r* to *AP2*
20. *B* sends *r* to *AP2*
21. *A* connects to ***AP5*** (migration)
22. *A* requests stored data from *AP2*
23. *AP2* sends the rendezvous notification to *A*
24. *A* requests to download *r* from *AP2*
25. *AP2* **sends** *r* to *A*

Fig. 6. An off-line notification, rendezvous and migration example

subscriptions stored in the network for this peer lose their relationship to this peer.

- Malicious peers can masquerade as other peers by using the IP address of a peer currently offline. They get all notifications for this peer, which are then lost to the original peer. Moreover they can change the original peer's subscriptions maliciously.

We therefore have to use suitable cryptography algorithms to provide unique identifiers for the peers in our network (see also the discussion in [4]).

When a new peer x wants to register to the network, it generates a pair of keys (E_x, D_x) where E_x is the *public key* of x (or the *encryption key*) and D_x is the *private key* of x (or the *decryption key*) as in [34]. We assume that the peer x has already found the IP address and public key of one of the super-peers s, through some secure means e.g., a secure web site. Then, x

securely identifies the super-peer s and if this succeeds, it sends an encrypted message to s (secure identification and encryption are explained below). The message contains the public key, the IP address and port of x. The super-peer s decrypts the message and creates a *private unique identifier* and a *public unique identifier* for x by applying the cryptographically secure hash function SHA-1 to the concatenated values of current date and time, the IP address of s, the current IP address of x and a very large random number. The properties of the cryptographically secure hash function now guarantee that it is highly unlikely that a peer with exactly the same identifiers will enter the network. Then, s sends the identifiers to x with an encrypted message. From there on the private identifier is included to all messages from x to its access-point and in this way a super-peer knows who sends a message. The private identifier of a peer is never included in messages that other peers will receive; instead the public identifier is used. To clarify the reason why we need both public and private identifiers we give the following example. When a peer x receives a notification n, n contains the public identifier of the resource owner *x1*. When x is ready to download the resource, it communicates with the access-point of *x1* and uses this public identifier to request the address of *x1*. If a peer knows the private identifier of x then it can authenticate itself as x, but if it knows the public identifier of x then it can only use it to request the address of x or set up a rendezvous with a resource owned by x. All the messages that a peer x sends to a super-peer and contain the private identifier of x are encrypted. In this way, no other peer can read such a message and acquire the private identifier of x.

Secure identification of peers is carried out as in [4]. A peer A can securely identify another peer B by generating a random number r and send $E_B(r)$ to B. Peer B sends a reply message that contains the number $D_B(E_B(r))$. Then, peer A checks if $D_B(E_B(r)) = r$ in which case peer B is correctly identified. For example, in our system super-peers securely identify peers as described above before delivering a notification. In this case, the super-peer starts a communication session with a peer so it cannot be sure that the peer listens on the specific IP address.

When a peer disconnects, its access point does not erase the public key or identifiers of; it only erases the private identifier from the active peer list. Later on, when the peer reconnects, it will identify itself using its private identifier and it will send to its access point, its new IP address. In case that the peer migrates to a different access point, it will notify the previous one, so that it erases all information about the peer. Then, the peer securely identifies the new access point and sends a message to it that contains the public key, the public and the private identifiers and the new IP address of the peer. All the above messages are encrypted since they contain the private identifier of the peer.

6 Related Work

In this section we review related research on pub/sub systems in the areas of distributed systems, networks and databases.

Most of the work on pub/sub in the database literature has its origins in the paper [17] by Franklin and Zdonik who coined the term *selective dissemination of information (SDI).* Their preliminary work on the system DBIS appears in [6]. Another influential system is SIFT [43, 44] where publications are documents in free text form and queries are conjunctions of keywords. SIFT was the first system to emphasize query indexing as a means to achieve scalability in pub/sub systems [43]. Later on, similar work concentrated on pub/sub systems with data models based on attribute-value pairs and query languages based on attributes with comparison operators (e.g., Le Subscribe [16], the monitoring subsystem of Xyleme [30] and others). [10] is also notable because it considers a data model based on attribute-value pairs but goes beyond conjunctive queries – the standard class of queries considered by other systems [16]. More recent work has concentrated on publications that are XML documents and queries that are subsets of XPath or XQuery (e.g., XFilter [25], YFilter [15], Xtrie [12] and *xmltk* [19]). All these papers discuss sophisticated filtering algorithms based on indexing queries.

In the area of distributed systems and networks various pub/sub systems have been developed over the years. Researchers have utilized here various data models based on channels, topics and attribute-value pairs (exactly like the models of the database papers discussed above) [11]. The latter systems are usually called *content-based* because attribute-value data models are flexible enough to express the content of messages in various applications. Work in this area has concentrated not only on filtering algorithms as in the database papers surveyed above, but also on distributed pub/sub architectures [5, 11]. SIENA [11] is probably the most well-known example of system to be developed in this area. SIENA uses a data model and language based on attribute-value pairs and demonstrates how to express notifications, subscriptions and advertisements in this language. From the point of view of this paper, a very important contribution of SIENA is the adoption of a *P2P* model of interaction among servers (super-peers in our terminology) and the exploitation of traditional network algorithms based on shortest paths and minimum-weight spanning trees for routing messages. SIENA servers additionally utilize partially ordered sets encoding subscription and advertisement subsumption to minimize network traffic. The core ideas of SIENA have recently been used in the pub/sub systems DIAS [23] and P2P-DIET [2, 24, 21] but now the data models utilized were inspired from Information Retrieval. DIAS and P2P-DIET have also emphasized the use of sophisticated subscription indexing at each server to facilitate efficient forwarding of notifications [42]. In some sense, the approach of DIAS and P2P-DIET puts together the best ideas from the database and distributed systems tradition in a single unifying framework. Another important contribution of P2P-DIET is that it demonstrates how to support by very similar protocols the traditional *ad-hoc* or *one-time* query scenarios of standard super-peer systems [45] and the pub/sub features of SIENA [11].

With the advent of distributed hash-tables (DHTs) such as CAN [33], CHORD [37] and Pastry [3], a new wave of pub/sub systems based on DHTs has appeared. Scribe [35] is a topic-based publish/subscribe system based on Pastry [3]. Hermes [32] is similar to Scribe because it uses the same underlying DHT (Pastry) but it allows more expressive subscriptions by supporting the notion of an event type with attributes. Each event type in Hermes is managed by an event broker which is a rendezvous node for subscriptions and publications related to this event. Related ideas appear in [38] and [39]. PeerCQ [18] is another notable pub/sub system implemented on top of a DHT infrastructure. The most important contribution of PeerCQ is that it takes into account peer heterogeneity and extends consistent hashing [22] with simple load balancing techniques based on appropriate assignment of peer identifiers to network nodes.

Meghdoot [20] is a very recent pub/sub system implemented on top of a CAN-like DHT [33]. Meghdoot supports an attribute-value data model and offers new ideas for the processing of subscriptions with range predicates (e.g., the price is between 20 and 40 Euros) and load balancing. A P2P system with a similar attribute-value data model that has been utilized in the implementation of a publish-subscribe system for network games is Mercury [8]. Two other recent proposals on publish/subscribe using DHTs is DHTrie [41] and LibraRing [40]. These works use the same data models with P2P-DIET and concentrate on publish/subscribe functionality for information retrieval and digital library applications.

In the area of RDF-based P2P systems, Min Cai et. al. have also recently studied publish/subscribe systems for RDF [9] using essentially the same query language as this chapter and our original paper [13]. The approach of [9] is complementary to ours since their design builds on Chord. It would be interesting to compare experimentally the algorithms of this paper and the algorithms of [9] in more detail.

7 Conclusions

Publish/subscribe capabilities are a necessary extension of the usual query answering capabilities of P2P networks, and enable us to efficiently receive answers to long-standing queries over a given period of time, even if peers connect to and disconnect from the network during this period.

In this chapter we have discussed how to incorporate publish/subscribe capabilities in an RDF-based P2P network, specified a formal framework for this integration, including appropriate subscription and advertisement languages, and described how to optimize the processing of subscriptions and notifications handling in this network.

Further work will include the full integration of these capabilities into our existing P2P prototypes Edutella and P2P-DIET, as well as further investigations for extending the query language in this chapter with more expressive relational algebra and IR operators, while still maintaining efficient subscription/notification processing.

8 Acknowledgements

The work of Stratos Idreos and Manolis Koubarakis is supported by projects Evergrow `http://www.evergrow.org/` and OntoGrid `http://www.ontogrid.net/` funded by the European Commission under the 6th Framework Programme.

References

1. Elena project home page. http://www.elena-project.org/.
2. P2P-DIET home page. http://www.intelligence.tuc.gr/p2pdiet/.
3. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale- peer-to-peer storage utility. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Paltforms (Middleware 2001)*, November 2001.
4. K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in Peer-to-Peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), May 2004.
5. M. K. Aguilera, R. E. Strom, D.C. Sturman, M. Astley, and T.D. Chandra. Matching Events in a Content-based Subscription System. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 53–62, New York, May 1999. Association for Computing Machinery.
6. M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. DBIS-toolkit: Adaptable Middleware for Large-scale Data Delivery. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, Philadelphia, USA*, 1999.
7. D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 1987.
8. A. Bharambe, S. Rao, and S. Seshan. Mercury: A Scalable Publish-Subscribe System for Internet Games, booktitle = Proceedings of the First International Workshop on Network and System Support for Games (Netgames). Braunchweig, Germany, 2002.
9. M. Cai, M. Frank, B. Yan, and R.MacGregor. A Subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 2(2), 2005.
10. A. Campialla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient Filtering in Publish Subscribe Systems Using Binary Decision Diagrams. In *Proceedings of the 23rd International Conference on Software Engeneering (ICSE-01)*, pages 443–452, Los Alamitos, California, May12–19 2001. IEEE Computer Society.

11. A. Carzaniga, D.-S. Rosenblum, and A.L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.
12. C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *Proceedings of the 18th International Conference on Data Engineering*, pages 235–244, February 2002.
13. P.-A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/Subscribe for RDF-based P2P Networks. In *Proceedings of the 1st European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 182–197, May 10-12 2004.
14. A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pages 23–34, Vienna, Austria, July 2002. IEEE.
15. Y. Diao, M. Altinel, M.J. Franklin, H. Zhang, and P. Fischer. Path Sharing and Predicate Evaluation for High-performance XML Filtering. *ACM Transactions on Database Systems*, 28(4):467–516, December 2003.
16. F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of ACM SIGMOD-2001*, 2001.
17. M. Franklin and S. Zdonik. "Data in Your Face": Push Technology in Perspective. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(2):516–519, June 1998.
18. B. Gedik and L. Liu. PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In *Proceedings of the the 23rd International Conference on Distributed Computing Systems*, May 2003.
19. T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML Streams with Deterministic Automata. In *Proceedings of the 9th International Conference on Database Theory (ICDT)*, pages 173–189, Siena, Italy, January 2003.
20. A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In *Proceedings of ACM/IFIP/USENIX 5th International Middleware Conference*, Toronto, Ontario, Canada, October 18-22.
21. S. Idreos, C. Tryfonopoulos, M. Koubarakis, and Y. Drougas. Query Processing in Super-Peer Networks with Languages Based on Information Retrieval: the P2P-DIET Approach. In *Proceedings of the 1st International Workshop on Peer-to-Peer Computing and DataBases (P2P&DB 2004)*, volume 3268 of *Lecture Notes in Computer Science*, pages 496–505, March 2004.
22. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, 4–6 May 1997.
23. M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proceedings of the 6th European Conference on Digital Libraries (ECDL2002)*, volume 2458 of *Lecture Notes in Computer Science*, pages 527–542, September 2002.
24. M. Koubarakis, C. Tryfonopoulos, S. Idreos, and Y. Drougas. Selective Information Dissemination in P2P Networks: Problems and Solutions. *ACM SIGMOD*

Record, Special issue on Peer-to-Peer Data Management, K. Aberer (editor), 32(3):71–76, September 2003.

25. M. Altinel and M.J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proceedings of the 26th VLDB Conference*, 2000.
26. A. Montresor. A Robust Protocol for Building Superpeer Overlay Topologies. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, August 2004.
27. M.Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP – Hypercubes, Ontologies and Efficient Search on Peer-to-peer Networks. In *Proceedings of the 1st Workshop on Agents and P2P Computing, Bologna*, 2002.
28. W. Nejdl, B. Wolf, Ch. Qu, S.Decker, M. Sintek, A.Naeve, M. Nilsson, M. Palmer, and T.Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the 11th International World Wide Web Conference*, 2002.
29. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer Based Routing and Clustering Strategies for RDF-based Peer-to-peer Networks. In *Proceedings of the 12th International World Wide Web Conference*, 2003.
30. B. Nguyen, S. Abiteboul, G.Cobena, and M. Preda. Monitoring XML Data on the Web. In *Proceedings of the ACM SIGMOD Conference 2001*, Santa Barbara, CA, USA, 2001.
31. N. Ntarmos and P. Triantafillou. AESOP: Altruism-Endowed Self Organizing Peers. In *Proceedings of the 2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, August 2004.
32. P.R. Pietzuch and J.M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.
33. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-addressable Network. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.
34. R.L. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *CACM*, 21(2):120–126, February 1978.
35. A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The Design of a Large-scale Event Notification Infrastructure. In J. Crowcroft and M. Hofmann, editors, *3rd International COST264 Workshop*, 2001.
36. B. Simon, Z. Mikls, W. Nejdl, M. Sintek, and J. Salvachua. Smart Space for Learning: A Mediation Infrastructure for Learning Services. In *Proceedings of the Twelfth International Conference on World Wide Web*, Budapest, Hungary, May 2003.
37. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.
38. D. Tam, R. Azimi, and H.-Arno Jacobsen. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In *Proceedings of the 1st International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, September 2003.
39. W.W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A.P. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *Proceedings of the*

2nd International Workshop on Distributed Event-Based Systems (DEBS'03), June 2003.

40. C. Tryfonopoulos, S. Idreos, and M. Koubarakis. LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. In *Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, Vienna, Austria, September 18-23 2005.
41. C. Tryfonopoulos, S. Idreos, and M. Koubarakis. Publish/Subscribe Functionality in IR Environments using Structured Overlay Networks. In *Proceedings of the 28th Annual ACM SIGIR Conference*, Salvador, Brazil, August 15-19 2005.
42. C. Tryfonopoulos, M. Koubarakis, and Y. Drougas. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In *Proceedings of the 27th Annual ACM SIGIR Conference*, Sheffield, United Kingdom, July 25-July 29 2004.
43. T.W. Yan and H. Garcia-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems*, 19(2):332–364, 1994.
44. T.W. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.
45. B. Yang and H. Garcia-Molina. Designing a Super-peer Network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, March 5–8 2003.
46. Y.K. Dalal and R.M. Metcalfe. Reverse Path Forwarding of Broadcast Packets. *Communications of the ACM*, 21(12):1040–1048, December 1978.