Publish/Subscribe for RDF-based P2P Networks

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

Abstract. 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 paper we built on the experience gained with P2P-DIET and the Edutella P2P infrastructure and present the first implementation of a P2P 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. Our work utilizes the latest ideas in query processing for RDF data, P2P indexing and routing research.

1 Introduction

Consider a peer-to-peer 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 peer-to-peer 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 one these authors, and Bob has to filter out the new publications each time. With an RDF-based peer-to-peer network, 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 if he uses this query now and then during 2004, he will get all 2004 publications each time.

What Bob really needs from his peer-to-peer file sharing network are publish/subscribe capabilities:

1. *Advertising*: Peers sends 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 should 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 paper we will describe how to provide publish/subscribe capabilities in an RDF-based peer-to-peer system, which manages arbitrary digital resources, identified by their URL and described by a set of RDF metadata. Our current application scenarios are distributed educational content repositories in the context of the EU/IST project ELENA [16], whose participants include e-learning and e-training companies, learning technology providers, and universities and research institutes (http://www.elena-project.org/), our second application scenario being digital library environments.

Section 2 specifies the formal framework for RDF based pub/sub systems, including the query language used to express subscriptions in our network. Section 3 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 [10], as well as on index optimizations first explored in the RDF context in P2P-DIET [9]. Section 4 includes a short discussion of other important features of our system, and section 5 includes a survey of related work.

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 publications. 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 [11].

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 3) and contains many interesting queries for P2P file sharing systems based on RDF. It is easy to see that only *join* on 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 paper, 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)
```

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 3.

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, $ans(q_2, T) \subseteq ans(q_1, T)$.*

[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.

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

Definition 4. *A* publication b *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 publication* $b = (T, I)$ matches *a subscription* q *if* $ans(q, T) \neq \emptyset$.

Notice that because URIrefs are assumed to be *unique*, and subscriptions and publications obey Definitions 2 and 4, publication matching in the architecture of Section 3 takes place *locally* at each super-peer.

Example 2. The publication

```
({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 3. This is formalized by the notion of "covers" below.

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:subject>, <lom:context>}, SP_6) covers the subscription of Example 1.

Definition 7. *A* property/value advertisement d *is a pair* $((P_1, V_1), \ldots, (P_k, V_k)), I)$ *where* $P_1, \ldots, P_k$ *are* distinct *properties (constants of type* $\mathcal{U}$ *i.e., URIrefs),* $V_1, \ldots, V_k$ *are sets of values for* $P_1, \ldots, P_k$ *(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* $Y_1, \ldots, Y_k$ *(*$1 \leq k \leq m$*) be the variables among the objects* $o_1, \ldots, o_m$ *of the triples of* q *that correspond to the*

properties $P_1, \ldots, P_k$ of d. We will say that d covers *a subscription q if there exist values $v_1 \in V_1, \ldots, v_k \in V_k$ such that the constraint $\phi[Y_1 \leftarrow v_1, \ldots, Y_k \leftarrow v_k]$ resulting from substituting variables $Y_1, \ldots, Y_k$ with constants $v_1, \ldots, v_k$ 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. In the architecture of Section 3 this advertisement will be sent using the RDF file given in the appendix of this paper.

3 Processing Advertisements, Subscriptions and Notifications

Efficiently processing advertisements, subscriptions and notifications is crucial for publish/subscribe services. After discussing our basic peer-to-peer topology based on the super-peer architecture described in [10], we will describe the optimizations necessary for processing advertisements, subscriptions and notifications in an efficient manner.

3.1 Basic Network Topology: Super-Peers and HyperCuP

Our publish/subscribe algorithm is designed for working with super-peer networks, i.e. peer-to-peer networks, where peers are connected to super-peers who are responsible for peer aggregation, routing and mediation.

Super-peer based infrastructures are usually based on a two-phase routing architecture, which routes queries and subscriptions 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 [4, 10]. In the following sections we will also present indexing and routing mechanisms appropriate for publish/subscribe services. These will be based on two levels of indices, one storing information to route within the super-peer backbone, and the other handling the communication between a super-peer and the peers connected to it. These indices will draw upon our previous work for query routing, as discussed in [10], as well as further extensions and modifications necessary for publish/subscribe services.

Our super-peers are arranged in the HyperCuP topology, not only because this is the solution adapted in the Edutella infrastructure [11], but also because of its special characteristics regarding broadcasts and network partitioning. The HyperCuP algorithm described in [15] is capable of organizing super-peers of a P2P network into a recursive graph structure called hypercube that stems from the family of Cayley graphs. Super-peers join the network by asking 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 query broadcasts. For broadcasts, each node can be seen as the root of a specific spanning tree through the P2P network, as shown in figure 1. The topology allows for $log_2 N$ path length and $log_2 N$ number of neighbors, where N is the total number of nodes in the network (i.e., the number of

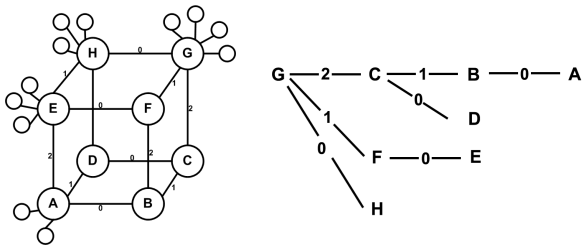

Fig. 1. HyperCuP Topology and Spanning Tree Example

super-peers in this case). Peers connect to the super-peers in a star-like fashion, providing content and content metadata. 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.

3.2 Processing Advertisements

The first step in a publish/subscribe scenario is done by a client c which sends an advertisement a to its access point AP, announcing what kind of resources it will offer in the future. Access points use advertisements to construct *advertisement routing indices* that will be utilized when processing subscriptions (see Section 3.3 below). Advertisements are then selectively broadcast from AP to reach other access points of the network. The advertisement indices are updated upon each advertisement arrival on three levels (we use three separate indices): schema level, property (attribute) level, and property/value level. Table 1 shows examples for these indices.

Schema Index. We assume that different peers will support different RDF schemas and that these schemas can be uniquely identified (e.g. by an URI). The routing index contains the schema identifier, as well as the peers supporting this schema. Subscriptions are forwarded only to peers which support the schemas used in the subscription.

Property Index. Peers might choose to use only part of (one or more) schemas, i.e. certain properties/attributes, to describe their content. While this is unusual in conventional database systems, it is more often used for data stores using semi-structured data, and very common for RDF-based systems. In this index, super-peers use the properties (uniquely identified by name space / schema ID plus property name) or sets of properties to describe their peers. Sets of properties can also be useful to characterize subscriptions.

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

Schema	RouteTo
{dc, lom}	SP_1

Property	RouteTo
{<dc:subject>, <lom:context>}	SP_6
{<dc:language>}	SP_7

Property	Set of Values	RouteTo
<dc:creator>	{ W. Nejdl, P. Chirita }	SP_1
<dc:title>	{"Algorithms", "Data Structures" }	SP_1
<dc:year>	[2002, ∞]	SP_1
<dc:year>	[1990, 2000]	SP_2

Table 1. Advertisement Routing Indices Example: a) Schema Index; b) Property Index; c) Property/Value Index

with the exception that the index entries do not refer to the resource, but the super-peer / peer providing it.

We use two kinds of indices: super-peer/super-peer indices (handling communication in the super-peer backbone) and super-peer/peer indices (handling communication between a super-peer and all peers connected to it). Except for the functionality they employ, both indices use the same data structures, have the same update process, etc.

Update of Advertisement Indices. Index updates are triggered when a new peer registers, a peer leaves the system permanently or migrates to another access point, or the metadata information of a registered peer changes. Peers connecting to a super-peer have to register their metadata information at this super-peer thus providing the necessary schema information for constructing the SP/P and SP/SP advertisement indices. During registration, an XML registration message encapsulates a metadata-based description of the peer properties. A peer is assigned at least one schema (e.g., the dc or the lom element set) with a set of properties (possibly with additional information) or with information about specific property values.

If a peer leaves a super-peer all references to this peer have to be removed from the SP/P indices of the respective super-peer. The same applies if a peer fails to re-register periodically. In the case of a peer joining the network or re-registering, its respective metadata/schema information are matched against the SP/P entries of the respective super-peer. If the SP/P advertisement indices 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 the indices. The following algorithm formalizes this procedure:

We define S as a set of schema elements: $S = \{s_i \mid i = 1...n\}$. The super-peer SP_x already stores a set S_x of schema elements in its SP/P indices. The SP/P indices of a super-peer SP_x can be considered as a mapping $s_i \mapsto \{P_j \mid j = 1...m\}$. A new peer P_y registers at the super peer SP_x with a set S_y of schema elements.

1. If $S_y \subseteq S_x$, then add P_y to the list of peers at each $s_i \in S_y$

2. Else, if $S_y \setminus S_x = \{s_n, ..., s_m\} \neq \emptyset$, then update the SP/P indices by adding new rows $s_n \mapsto P_y, ..., s_m \mapsto P_y$.

Generally, upon receiving an advertisement, the access point (let's call it SP_a) will initiate a selective broadcasting process. After the advertisement has been received by another super-peer (say SP_i), it is matched against its advertisement indices and updated using the algorithm described above. When this operation does not result in any modification of the advertisement indices, no further broadcasting is necessary. So for example if a peer publishes something on Physics and the advertisement indices are already sending subscriptions on this topic towards this partition of the network, then there is no need to update these indices, nor any other indices further down the spanning tree of the super-peer network – they will also be pointing towards SP_a already.

3.3 Processing Subscriptions

When a client C posts a subscription q to its access point SP, which describes the resources C is interested in, SP introduces q into its *local subscription poset* and decides whether to further forward it in the network or not. A subscription poset is a hierarchical structure of subscriptions and captures the notion of subscription subsumption defined in Section 2. Figure 2 shows an example of a poset. The use of subscription posets in pub/sub systems was originally proposed in SIENA [2]. Like SIENA, our system utilizes the subscription poset to minimize network traffic: super-peers do not forward subscriptions which are subsumed by previous subscriptions.

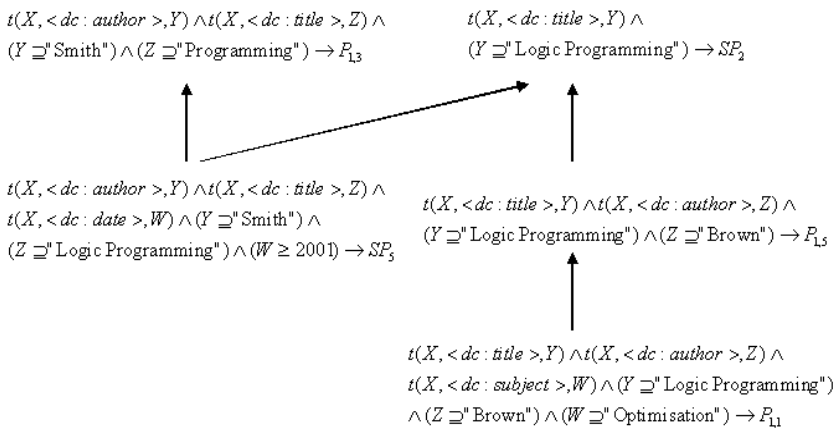

Fig. 2. Poset Example

As shown in the example, each super-peer will add 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.

Once a super-peer has decided to send the subscription further, it will initiate a selective broadcast procedure (based on the HyperCuP protocol). Upon subscription receival, a super-peer will have to use its advertisement routing indices in order to decide whether to send it to its neighboring super-peers along the spanning tree or not. There are two criteria which need checked:

1. If the indices contain a super-peer that supports the targeted schema (or properties), but there is no information about the values it covers, then we route the subscription to the respective super-peer, using HyperCuP.[2]
2. If there is also information about the values it covers, then we check if the values are consistent with the constraints of the subscription. If yes, we route the subscription forward, otherwise, we don't.

We give a more formal description of this routing process in the algorithm below.

Algorithm 1. Routing subscriptions.

Let q be the subscription. Then, q is of the form:
$t_1(x, < s_1 : p_1 >, a_1) \wedge ... \wedge t_m(x, < s_m : p_m >, a_m) \wedge C(a_{p1}) \wedge ... \wedge C(a_{pf})$
s_i are (possibly) different schemas,
p_i are (possibly) different attributes,
a_i are either constants or variables; for all a_i which are variables we have some additional constraints on them at the end of the subscription.
Let us denote the *Schema Index* SI,
Property Index PI,
Property/Value Index PVI.
Finally, let us consider the subscription came on dimension D of the HyperCuP spanning tree.
1: pvFound $\leftarrow$ false; pFound $\leftarrow$ false;
2: **for all** $s_i : p_i$ **do**
3: **if** $s_i : p_i \in PVI$ **then** pvFound $\leftarrow$ true;
4: **if** $s_i : p_i \in PI$ **then** pFound $\leftarrow$ true;
5: **if** pFound $\wedge$ pvFound **then** break;
6: **if** $\neg$pFound $\wedge$ $\neg$pvFound **then**
7: **for all** targets $t_i \in SI, dimension(t_i) \geq D$ **do**
8: **for all** s_j **do if** $s_j \notin SI$ **then** break;
9: **if** j=m **then** routeTo t_i; // All tuples have matched
10: exit;
11: **if** pFound $\wedge$ $\neg$pvFound **then**
12: **for all** targets $t_i \in SI \cup PI, dimension(t_i) \geq D$ **do**
13: **for all** $s_j : p_j$ **do if** $(s_j \notin SI) \wedge (s_j : p_j \notin PI)$ **then** break;
14: **if** j=m **then** routeTo t_i;
15: exit;

[2] The HyperCuP algorithm uses *dimensions* to avoid sending a message twice to the same peer/super-peer. Every broadcast message is sent only on higher dimensions than the dimension on which it was received. See [10] for more details.

16: **for all** targets $t_i \in PVI, dimension(t_i) \geq D$ **do**
17: **for all** $(s_j : p_j, a_j)$ **do**
18: **if** $(s_j : p_j \subset l \in PVI) \wedge (a_j \not\subseteq l \in PVI)$ **then** break;
19: **if** $(s_j : p_j \notin PVI) \wedge (s_j : p_j \notin PI \cup SI)$ **then** break;
20: **if** j=m **then** routeTo t_i;

3.4 Processing Notifications

When a new notification arrives at the super-peer, it is first matched against the root subscriptions of its local subscription poset (see also figure 2). 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. The complete algorithm is depicted in the following lines.

Algorithm 2. Notification Processing.

Let P be the poset and n the notification.
1: **function** match (posetEntry pe, notification n)
2: **if** $n \supseteq pe$ **then**
3: **for all** targets $t_i \in pe$ **do** routeTo t_i;
4: **for all** children $c_i \in pe$ **do** match (c_i, n);
5: **end function**;
6:
7: **for all** roots $r_i \in P$ **do** match (r_i, n);

4 Handling Dynamicity in a P2P Pub/Sub Network

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

4.1 Offline Notifications and Rendezvous at Super-Peers

Whenever a client *A* disconnects from the network, its access point *AP keeps* the client'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 he reconnects.

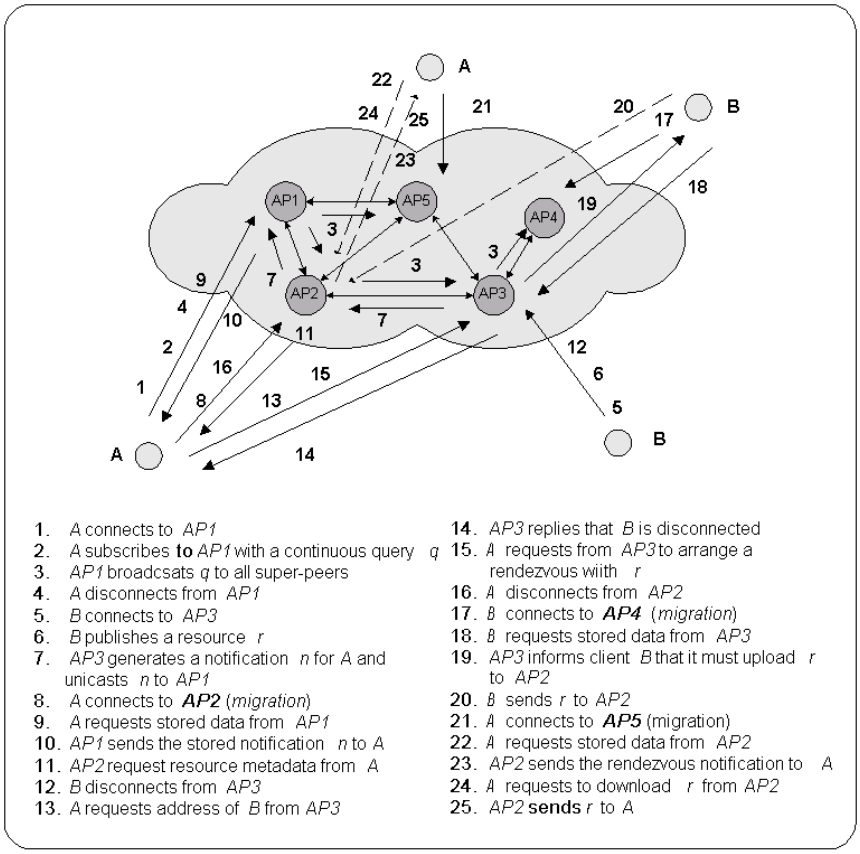

Fig. 3. An off-line notification, rendezvous and migration example

A client 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 client *A* requests a resource *r*, but the resource owner client *B* is not on-line. Client *A* requests the address of *B* from *AP2* (the access point of *B*). In such a case, client *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 client *B* reconnects, *AP2* informs *B* that it must upload resource *r* to *AP* as a *rendezvous file* for client *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 clients migrate to different access points. For example, let us assume that client *A* has migrated to *AP3*. The client 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 3.

4.2 Peer Authentication

Typically, authentication of peers in a peer-to-peer network is not crucial, and peers connecting to the network identify themselves just using their IP-adresses. 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 his absence, after he reconnects with another IP address. Furthermore, all 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 [7]).

When a new client *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 [13]. We assume that the client *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 client is never included in messages that other clients 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 client *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 client-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 client-peer *x* sends to a super-peer and contain the private identifier of *x* are encrypted. In this way, no other client can read such a message and acquire the private identifier of *x*.

Secure identification of peers is carried out as in [7]. 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 client-peers as described above before delivering a notification. In this case, the super-peer starts a communication session with a client-peer so it cannot be sure that the client-peer listens on the specific IP address.

When a client disconnects, its access point does not erase the public key or identifiers of; it only erases the private identifier from the active client list. Later on, when the client 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 client migrates to a different access point, it will notify the previous one, so that it erases all information about the client. Then, the client 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 client. All the above messages are encrypted since they contain the private identifier of the client.

5 Analysis of Other Publish/Subscribe Systems

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 [5] by Franklin and Zdonik who coined the term *selective dissemination of information*. Other influential work was done in the context of SIFT [18] 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 scalable filtering in pub/sub systems [17]. Since then work concentrated on query indexing for data models based on attribute-value pairs and query languages based on attributes with comparison operators. A most notable effort among these works is [1] because it goes beyond conjunctive queries – the standard class of queries considered by all other systems. More recent work has concentrated on publications that are XML documents and queries that are subsets of XPath or XQuery (e.g., Xtrie [3] and others).

In the area of distributed systems and networks various pub/sub systems with data models based on *channels*, *topics* and *attribute-value pairs* (exactly like the models of the database papers discussed above) have been developed over the years [2]. Systems based on attribute-value pairs are usually called *content-based* because their 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. SIENA [2] 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 *peer-to-peer* 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 by some of us in the pub/sub systems DIAS [8] and P2P-DIET (`http://www.intelligence.tuc.gr/p2pdiet`) [9]. DIAS and P2P-DIET offer data models inspired from Information Retrieval and, in contrast with SIENA, have also emphasized the use of sophisticated subscription indexing at each server to facilitate efficient forwarding of notifications. In summary, 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 *one-time* query scenarios of standard super-peer systems [19] and the pub/sub features of SIENA [9].

With the advent of distributed hash-tables (DHTs) such as CAN, CHORD and Pastry, a new wave of pub/sub systems based on DHTs has appeared. Scribe [14] is a topic-based publish/subscribe system based on Pastry. Hermes [12] 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. PeerCQ [6] 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 with simple load balancing techniques based on appropriate assignment of peer identifiers to network nodes.

6 Conclusions

Publish/subscribe capabilities are a necessary extension of the usual query answering capabilities in peer-to-peer networks, and enable us to efficiently handle the retrieval of answer 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 paper we have discussed how to incorporate publish/subscribe capabilities in an RDF-based P2P network, specified a formal framework for this integration, including an appropriate subscription language, 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 paper with more expressive relational algebra and IR operators, while still maintaining efficient subscription/notification processing.

7 Acknowledgements

The work of Stratos Idreos and Manolis Koubarakis is supported by project Evergrow funded by the European Commission under the 6th Framework Programme (IST/FET, Contract No 001935).

References

1. A. Campailla and S. Chaki and E. Clarke and S. Jha and H. Veith. Efficent filtering in publish-subscribe systems using binary decision diagrams. In *Proc. of 23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, 2001.
2. 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.
3. 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.
4. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22th International Conference on Distributed Computing Systems*, 2002.
5. M.J. Franklin and S.B. Zdonik. "Data In Your Face": Push Technology in Perspective. In *Proceedings ACM SIGMOD International Conference on Management of Data*, 1998.
6. B. Gedik and L. Liu. PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In *Proceedings of the 23rd IEEE International Conference on Distributed Computer Systems*, May 2003.
7. M. Hauswirth, A. Datta, and K. Aberer. Handling identity in peer-to-peer systems. Technical report, LSIR-EPFL.
8. M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou. Information alert in distributed digital libraries: Models, languages and architecture of dias. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, 2002.
9. 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), September 2003.
10. 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.
11. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjrn Naeve, Mikael Nilsson, Matthias Palmer, and Tore Risch. Edutella: A p2p networking infrastructure based on rdf. In *Proceedings of the 11th International World Wide Web Conference*, 2002.
12. 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.
13. 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.
14. 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.
15. Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang 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.
16. Bernd Simon, Zoltn Mikls, Wolfgang Nejdl, Michael Sintek, and Joaquin 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.
17. 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.
18. T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.
19. 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.