Efficient Probabilistic Subsumption Checking for Content-based Publish/Subscribe Systems

Aris M. Ouksel*[1], Oana Jurca[2], Ivana Podnar[2], and Karl Aberer**[2]

[1] The University of Illinois at Chicago
Depts. Of Information and Decision Sciences and Computer Science
aris@uic.edu
[2] School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
{oana.jurca, ivana.podnar, karl.aberer}@epfl.ch

Abstract. Efficient subsumption checking, deciding whether a subscription or publication is subsumed by a set of previously defined subscriptions, is of paramount importance for publish/subscribe systems. It provides the core system functionality—matching of publications to subscriber needs expressed as subscriptions—and additionally, reduces the overall system load and generated traffic since the covered subscriptions are not propagated in distributed environments. As the subsumption problem was shown previously to be co-NP complete and existing solutions typically apply pairwise comparisons to detect the subsumption relationship, we propose a 'Monte Carlo type' probabilistic algorithm for the general subsumption problem. It determines whether a publication/subscription is covered by a disjunction of subscriptions in $O(k\ m\ d)$, where k is the number of subscriptions, m is the number of distinct attributes in subscriptions, and d is the number of tests performed to answer a subsumption question. The probability of error is problem specific and typically very small, and determines an upper bound on d in polynomial time prior to the algorithm execution. Our experimental results show significant gains in term of subscription set reduction which has favorable impact on the overall system performance as it reduces the total computational costs and networking traffic. Furthermore, the expected theoretical bounds underestimate algorithm performance because it performs much better in practice due to introduced optimizations, and is adequate for fast forwarding of subscriptions, especially in resource scarce environments.

1 Introduction

A large number of applications require, for performance or semantic considerations, determining efficiently whether a logical expression subsumes another. These include, for

* Research supported in part by the National Science Foundation grant IIS-0326284.
** Research supported in part by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and carried out (partly) in the framework of the EPFL Center for Global Computing.

example, stock tickers, RSS news feeds, network monitoring, traffic monitoring, electronic auctions, recommendation systems and user modeling in electronic commerce. In turn, these applications fueled interest in the publish/subscribe communication paradigm [1] as a content-based selective information dissemination service that delivers information to subscribers based on whether the content of the information, or publications, subsumes their subscriptions.

Content-based publish/subscribe systems aim at providing expressive means for content filtering. Traditional systems usually employ high-performance servers to handle high rates of publications and serve millions of subscribers in static environments. They have been optimized for fast matching of publications to subscriptions [2–5] and typically maintain a special subscription index that does not frequently change as the rate of subscription changes is negligible compared to the publication rate. Distributed systems traditionally assume static environments, and use a network of brokers to divide the publication and subscription load. Brokers implement routing protocols to provide a consistent service with a goal of reducing the networking costs generated by publications and subscriptions [6, 7]: Subscriptions are typically routed through the network toward publishers to enable filtering of publications close to their sources. Subscription traffic, on the other hand, is reduced by not propagating covered subscriptions as they are redundant, or by merging subscriptions [8, 9].

Although the importance of subscription reduction has been stressed, e.g. in [9], existing deterministic algorithms [10, 11, 8] focus on efficient matching of publications to subscriptions and use pair-wise comparisons to detect the subsumption relationship between two subscriptions or to merge two similar subscriptions. Publications and subscriptions are typically modeled as logical expressions–conjunctions of predicates– where each predicate defines a linear constraint on an attribute. Geometrically, publications are points in a multi-dimensional space, while subscriptions can be viewed as convex polyhedra. We consider publications also as convex polyhedra, to support environments with imprecise data sources, as it is advocated in recent publish/subscribe models with approximate matching [12]. In this context, the general subsumption checks whether a disjunction of subscriptions covers a subscription/publication, and can geometrically be interpreted as checking whether a convex polyhedron is contained within a finite union of convex polyhedra. The problem was proven to be co-NP complete in [13].

The importance of subscription set reduction becomes more significant in environments with highly-changeable subscriptions. Examples are MANETs, and sensor networks where the assumption of both network [14] and subscription stability no longer holds. The rate of subscription changes may drastically increase as a consequence of both changing user interests and context changes; therefore, novel indexing techniques have been investigated that trade-off precision to performance [15], however it does not tackle the essential problem of subscription set reduction.

In this paper we propose a probabilistic ‘Monte Carlo type’ algorithm for the general subsumption problem. This is the first probabilistic approach to test the subscription coverage by a union of subscriptions. The algorithm solves the subsumption problem in $O(k \cdot m \cdot d)$, where k is the number of subscriptions, m is the number of distinct attributes in subscriptions and d is the number of tests performed to answer the sub-

scription coverage question. The value of parameter d is dependent on an acceptable predefined probability of error which is problem specific and can be computed in polynomial time prior to the execution of the algorithm. Next, we define a minimal cover set algorithm that reduces the problem complexity by identifying a minimal set of relevant subscriptions sufficient to determine the subsumption relationship in $O(m^2\ k^3)$. As a result, the process of subsumption checking is accelerated because a new subscription is compared against a reduced set of subscriptions which also influences d. Finally, we list several sufficient conditions to quickly produce a deterministic answer to the coverage question in specific subscription settings. Our experiments show that our algorithmic approach performs much better in practice than the theoretical $O(k \cdot m \cdot d)$, and can on average efficiently produce an answer to a given coverage question.

As publish/subscribe systems typically target usage scenarios where a subscription space is moderately populated, and subscriptions typically overlap due to similar but not equal interest, there is a higher probability of a subscription being covered by a set of subscriptions rather than a single one. Covered subscriptions are redundant and not propagated which reduces the total number of subscriptions in the system saving memory and reducing traffic, and in turn reduces computational costs for matching publications to subscriptions as the set of subscriptions is reduced. The advantages of the algorithm are in the following:

- It efficiently checks the coverage relationship between a subscription and a set of subscriptions, and therefore
- reduces the set of active subscriptions in the overall system because covered subscriptions are not further propagated, and finally
- provides gains in terms of publication matching because the tested subscription set is reduced.

The algorithm introduces an error because it can assume a covering relationship and prevent forwarding of a subscription which leads to loss of publications. This occurs when the algorithm fails to detect a '*point witness*', a point in a multidimensional space proving the coverage relationship. The probability of error is problem specific and becomes negligible for large d, as shown by our evaluations. Concerns about lost publications are legitimate in case of highly-reliable systems with delivery guarantees. Such guarantees require costly mechanisms, especially in distributed environments where publications may not reach a subscriber simply because its subscriptions have not propagated through the system. However, most applications can accept potential loss of publications to gain on performance. An example are sensor networks where the published content is often inaccurate or redundant.

To summarize, the algorithm has the potential to significantly decrease the costs in terms of computation, memory, and bandwidth consumption in content-based and distributed publish/subscribe systems. The main contributions are as follows:

1. We design a novel probabilistic algorithm for solving the general subsumption problem, and introduce additional optimizations for efficient algorithm performance.
2. We test the performance of the proposed algorithmic approach in a number of subscription generation scenarios where the pair-wise coverage cannot reduce the subscription set, and show the algorithm can effectively reduce the number of subscriptions with acceptable costs within required error bounds.

3. We compare the performance of the algorithms to the standard pair-wise coverage algorithm in a realistic setting to investigate potential gains in terms of subscription set reduction.

The remainder of the paper is structured in the following way. We review the basic principles of content-based publish/subscribe communication style in Section 2. To motivate the presentation, Section 3 sketches a usage scenario and formally defines the subsumption problem. Section 4 presents our novel probabilistic algorithm with specific optimizations, and we investigate it's properties in a distributed setting in Section 5. Section 6 presents an evaluation of the algorithm using extensive experimentation, and in Section 7 we compare it to the related work in the field. We complete the paper with our conclusions in Section 8.

2 Distributed Publish/Subscribe Communication

The publish/subscribe interaction model enables asynchronous communication between information *publishers* and *subscribers*. Subscribers express interest in receiving publications that comply to specific criteria by defining *subscriptions* which changes the set of active subscriptions maintained by the publish/subscribe system. When a publisher defines a new *publication*, it is compared against all active subscriptions, and the system notifies subscribers with a matching subscription about the published content. Thus, the publish/subscribe service performs content filtering and enables push-style group communication, where group members are determined dynamically per each publication.

The communication is *event-driven* with a characteristic *subscribe-publish-notify* pattern: *subscribe* occurs as a result of changed subscriber's information needs, while *publish* is caused by the availability of a new or modified information item, or by a publisher's state change. Subscriptions and publications are typically generated at random, while *notify* is a conditional event which always occurs as a consequence of a publication. The system performance is largely influenced by the rate of publications, and the rate of subscription changes.

The simplest approach to route notifications in distributed is notification flooding: each published notification is sent to all system brokers, and brokers perform the matching of notifications to subscriptions of their local subscribers. This approach is an obvious solution for scenarios with a densely covered subscription space where most brokers have interested subscribers for all published notifications, but it wastes a lot of bandwidth in cases with few or no subscribers interested in a large fraction of published notifications.

To minimize the notification traffic, the information about subscriptions is disseminated through the network. Each broker receiving a new subscription informs other brokers that are potential publishers of notifications matching the defined subscription about its new subscription. A commonly used technique for subscription dissemination is flooding: subscription s is flooded to all potential information publishers, i.e. neighboring brokers, that flood s further to their neighbors. Published notifications will follow the reverse direction of subscriptions. The technique is commonly known as *reverse path forwarding* [6, 7], and is used to create *delivery trees* connecting publishers to a set of potential subscribers. To further reduce the number of subscription/unsubscription

messages exchanged between the brokers, subscription covering and merging is applied [11, 8].

We explain the approach using an example graph modeling nine brokers and the logical links between them in Figure 1. In this example network there are two publishers and two subscribers: S_1 is connected to B_1, S_2 is connected to B_6, P_1 is connected to B_9 and P_2 is connected to B_5. When a subscriber defines a new subscription, it is first submitted to the connecting broker and then further on flooded through the network of brokers, if not covered by another subscription. For example, when S_1 subscribes to s_1, this information is noted by B_1. B_1 sends a subscription request to its neighboring broker B_3, B_3 forwards it to its neighboring brokers B_2 and B_4, and so on, until the information about the subscription s_1 reaches all available brokers. When S_2 subscribes to s_2, $s_2 \sqsubseteq s_1$, this information also needs to flood the network checking the coverage relationship. Subscription s_2 reaches B_4 through B_6. B_4 will forward it to B_3, but not to B_5 nor B_7 because B_4 has previously subscribed to s_1 that covers s_2. Flooding the network with subscriptions enables the definition of delivery trees connecting a publisher with all interested subscribers. For example, the delivery tree of P_1 when publishing a notification n_1 that matches s_2 and, therefore, also s_1 connects brokers B_9, B_7, B_4, B_3, B_1, and B_6. The delivery tree for P_2 when publishing a notification n_2 that matches s_1 but not s_2 connects brokers B_5, B_4, B_3, and B_1. Note that delivery trees are computed using the local knowledge about subscriptions stored by each broker.

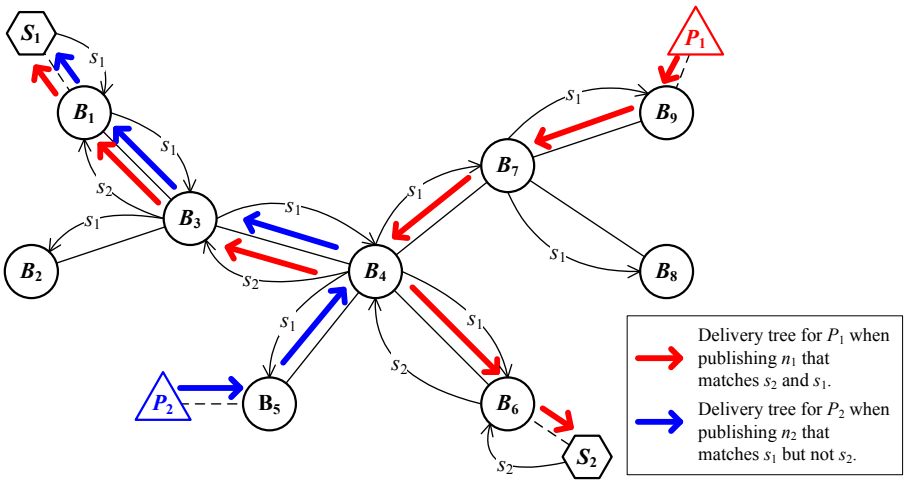

Fig. 1. Reverse path forwarding: Creating delivery trees for s_1 and s_2 ($s_2 \sqsubseteq s_1$)

From the given example it is obvious that the covering relationship can significantly reduce the subscription traffic, especially in usage scenarios with diverse subscriptions that partly cover the subscription space. Although the rate of subscription changes is typically lower than the publication rate, subscription dynamics can be significant if we

assume an extremely large number of subscribers. In addition, subscriber mobility introduces significant changes of subscriptions because they are triggered additionally by changes of location and network restructuring. Note that the publish/subscribe communication style is not suitable for the cases when the subscription space is either densely or sparsely covered by active subscriptions. In case of sparse subscriptions the probability of coverage relationship is low, and all subscriptions can be forwarded without checking the coverage relationship.

3 Problem Statement

Motivating scenarios. To illustrate the potential of publish/subscribe in resource constrained environments, we provide two motivating usage scenarios, a *sensor-enriched bicycle rental system* and *resource discovery in Grids*.

The envisioned bicycle rental system has an infrastructure consisting of several rental posts spread throughout a city offering various bicycles for rent. Each bicycle is equipped with a sensor (e.g. an RFID tag) storing bicycle-related data, and each registered user has a membership card storing his/her profile. Rental posts represent meeting places for bikes and people: They should enable the matching of available bikes to people using rental post in their vicinity. We assume rental posts have the means to detect bicycle sensors in their vicinity, they are equipped with card readers, and may even have a connection to the Internet.

Registered users may specify their long-term rental preferences, such as weekend bike rentals, or short-term needs when, for example, they decide to drop by a nearby supermarket during the lunch break. Long term rental preferences are part of user profiles and may even be stored on a membership card, while special applications customized for mobile phones may be used to define short-lived preferences. User profiles and preferences, together with contextual information are used to generate subscriptions.

Let us examine two simple example subscriptions:

s_1 Send an e-mail when a lady mountain bike size 19", preferable brand X, becomes available on Friday evenings within the area close to 'home'.

s_2 If today I have no appointments in my calendar between 12:00 and 2:00PM, send an SMS when a bike (sizes 17" and 19") is available in my current close vicinity (up to 500 m).

A publish/subscribe system will need to interpret such verbose user preferences extended by contextual information into machine understandable constraints over attributes, where attribute values are elements from (ordered) finite sets. For example, 'lady mountain bike' specifies bicycle type which may be interpreted as a range of unique bike identifiers (bID) that classify it in a particular category. Brand would be given as an element from a finite set, while a range of rental post identifiers ($rpID$) may encode the area in vicinity of home. A more formal representation of the two verbose subscriptions and two publications is presented in Table 1.

Even in this simple example subscriptions have 5 different attributes. The number of attributes used in real-world applications can be much larger. Contextual information related to e.g. user location, state, and available means of communication, largely

Table 1. Subscription and publication examples

	bID	$size$	$brand$	$rpID$	$date$
s_1	[1000, 1999]	19	X	[820, 840]	[2006-03-31T16:00:00, 2006-03-31T20:00:00]
s_2	[1, 1999]	[17, 19]	*	[10,12]	[2006-03-31T12:00:00, 2006-03-31T14:00:00]
p_1	1036	19	X	825	2006-03-31T18:23:05
p_2	1035	17	Y	11	2006-03-31T12:23:05

increase the number of constraints, and, at the same time, causes higher volatility of subscriptions. For example, s_1 should be activated only on Fridays to decrease the number of subscriptions in the system, and deactivated in the evening when a user browses through the list of received mails and chooses a bike for the weekend rental. On the other hand, s_2 is activated at noon when user's calendar is empty until 14:00, but will change with each significant change of user's geographical location as $rpID$ must encode the current user position. It can be deactivated as soon as the user rents a bicycle. Publications p_1 and p_2 are two example publications generated when a rental post detects an available bicycle. As p_1 matches s_1, and p_2 matches s_2, the publish/subscribe system should deliver them to the user using the preferred means of communication.

The example shows that a potentially large number of bike rental users may generate a huge number of constantly changing subscriptions, possibly in high-dimensional spaces. This can cause high update rate of subscription indexes maintained by the publish/subscribe service and consequently high subscription traffic. Therefore, we propose a method for reducing the total number of active subscriptions in the system by means of group coverage.

The second motivating scenario, *resource discovery in Grids* assigns computation requests (jobs) to available services. Current systems use server-based solutions and recently P2P-based solutions have been investigated [16] to deal with the scalability problem caused by a large number of jobs and services. Let us discuss the problem of resource discovery in terms of publish/subscribe. Services offering computational resources may announce their capabilities and availability through subscriptions to enable efficient matching and scheduling of jobs searching for available services. Jobs define their requirements from the services using publications. An example subscription with two publications are presented in Table 2.

Table 2. Subscription and publication examples

	$CPUcycles$	$disk$	$memory$	$service$	$time$
s_1	[3000, 3500]	[40, 50kB]	1GB	a.service.org	[2006-03-31T16:00:00, 2006-03-31T20:00:00]
p_1	3500	45kB	1GB	*.service.org	2006-03-31T16:00:00
p_2	1035	45kB	0.5GB	*.*.org	2006-03-31T12:23:05

The basic characteristic of this usage scenario are potentially large number of services and jobs that generate huge amounts of both subscriptions and publications. Dynamic changes of subscriptions are significant because as the context changes, i.e. services get allocated to new jobs, their subscriptions will consequently change. Therefore, this scenario exemplifies a setting where context changes induce higher subscription rate, as it can also be observed in mobile environments. Thus, a method for reducing the total number of active subscriptions in the system is highly needed and we can do it by taking advantage of group coverage. Due to large numbers and inherently distributed characteristics of Grid services, the publish/subscribe service for resource discovery would be distributed. As in this paper we are focusing on the subsumption process performed within a single node, we are not assuming neither an underlying network topology nor stability of the broker network. It can be applied with various routing protocols, and our goal is to point out potential impact of the proposed algorithm on the performance of a distributed system regardless of its topology and applied routing strategy.

Let us consider the following example of subscription coverage in a 2-dimensional subscription space. Table 3 defines two existing subscriptions, s_1 and s_2, and new subscription s. We want to determine whether s_1 and s_2 jointly cover s. As it is visible from the graphical representation of subscriptions in Figure 2, the subsumption relationship indeed exists. Even though neither s_1 nor s_2 cover s, their union entirely covers s. Note that constraints in this example define ranges to simplify the presentation, and can straightforwardly be extended to finite sets.

Table 3. Subsumption example: $s \sqsubseteq (s_1 \vee s_2)$

$Subscription\ s$
$[x_1 \geq 830 \wedge x_1 \leq 870 \wedge$
$x_2 \geq 1003 \wedge x_2 \leq 1006]$
$Subscription\ s_1$
$[x_1 \geq 820 \wedge x_1 \leq 850 \wedge$
$x_2 \geq 1001 \wedge x_2 \leq 1007]$
$Subscription\ s_2$
$[x_1 \geq 840 \wedge x_1 \leq 880 \wedge$
$x_2 \geq 1002 \wedge x_2 \leq 1009]$

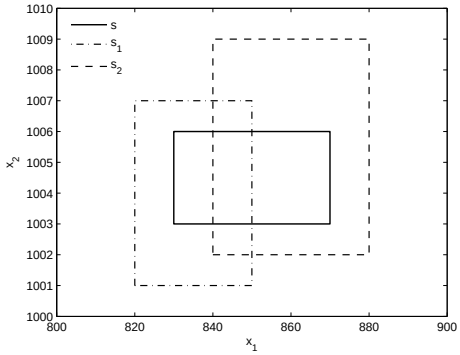

Fig. 2. Graphical representation of subscriptions in Table 3

Table 4 lists the notation used in the paper.

Definition 1. *Subscription* s_i is a conjunction of predicates $s_i = s_i^1 \wedge s_i^2 \wedge \ldots \wedge s_i^{r_i}$ where each s_i^j is a simple predicate, and $r_i \geq 1$, where r_i is the number of simple predicates forming subscription s_i. Let us define m, as the number of distinct attributes in the set of k subscriptions s_i, $1 \leq i \leq k$.

Table 4. Notations

Symbol	Meaning
s	New subscription
p	Publication
S	Disjunction of existing subscriptions s_i, $1 \leq i \leq k$
k	$\|S\|$
s_i	Existing subscription $s_i \in S$
s_i^j	j^{th} predicate in s_i
x_j	Attribute j
m	number of distinct attributes in S
T	Conflict table
T_i^j	Value in row i, column j of T
t_i	Number of defined elements in row i of T
f_{c_i}	Number of conflict-free elements in row i of T
δ	Error probability
ρ_w	Probability of guessing a point witness

Without restricting the applicability of the algorithm and to simplify the analysis, we consider that each simple predicate defines a constraint on an attribute x_j, $1 \leq i \leq m$, where each x_j has a lower ($x_j \geq low_i$) and upper limit ($x_j \leq high_i$). Each attribute is therefore defined as a range. Furthermore, we assume that all subscriptions define constraints for the same number of attributes $m_1 = m_2 = \ldots = m_k = m$, and since there is a lower and upper bound on each x_j, $r = 2 \cdot m$. Note that this in fact is not a restriction as the bounds $(-\infty, +\infty)$ mean the attributed is not significant for a particular subscription, and remains undefined.

The *general subsumption* problem tests whether a subscription s is covered by a disjunction of subscriptions, $s \sqsubseteq (s_1 \vee s_2 \vee \ldots \vee s_k)$, where k is the total number of existing subscriptions.

Definition 2. A *conflict table* T is a $k \times (2 \cdot m)$ table relating a subscription s to all simple predicates defined by $S = \{s_1 \vee s_2 \vee \ldots \vee s_k\}$. An element in table T, T_i^j is $\neg s_i^j$ if $s \wedge \neg s_i^j$ is satisfiable or is otherwise *undefined*.

A conflict table points out conflicting and not covered intervals between a tested subscription and a set of subscriptions. To construct the conflict table, we process each subscription $s_i \in S$ to verify the satisfiability of the negation of each simple predicate s_i^j against subscription s. If the condition is true, T_i^j is assigned the value $\neg s_i^j$, otherwise it is assigned the *undefined* value. Thus, the decision whether a specific T_i^j is defined is done in $O(1)$ and the construction of the table requires $O(m \cdot k)$.

For the example in Table 3, $s \wedge \neg s_1^1$ is *not satisfiable*, because the the intersection between s and $\neg s_1^1 = \{x_1 < 820\}$ is empty, while $s \wedge \neg s_1^2$ is *satisfiable* because the intersection between s and $\neg s_1^2 = \{x_1 > 850\}$ is non-empty. Both $s \wedge \neg s_1^3$ and $s \wedge \neg s_1^4$ are *not satisfiable* and thus the corresponding table cells are $undefined$. The same procedure is performed to compare s to s_2.

The conflict table relating subscription s from Table 3 to the set of subscriptions s_1 and s_2 is given in Table 5. The first presented row represents a template for the content

Table 5. Conflict table for the example in Figure 2

s_i	$x_1 < low_i^1$	$x_1 > high_i^1$	$x_2 < low_i^2$	$x_2 > high_i^2$
s_1	$undefined$	$x_1 > 850$	$undefined$	$undefined$
s_2	$x_1 < 840$	$undefined$	$undefined$	$undefined$

of the actual conflict table relating s to s_1 and s_2. The first line corresponding to s_1 has only one defined element, $\neg s_1^2 = \{x_1 > 850\}$ because, as it is visible in the graphical representation, s_1 does not cover s for $x_1 > 850$. Analogously, the only defined element in the second line corresponding to s_2 is $\neg s_2^1 = \{x_1 < 840\}$.

Definition 3. A *polyhedron witness* to non-cover is a set of elements from a conflict table T, $\left\{T_1^{j_1}, \ldots, T_k^{j_k}\right\}$, such that $s \wedge \neg s_1^{j_1} \wedge \ldots \wedge \neg s_k^{j_k}$ is satisfiable, defining a convex polyhedron. In other words, a polyhedron witness is a convex polyhedron contained in s, but not in S.

Let us consider the example graphically represented in Figure 3, defining two subscriptions s_1 and s_2 that do not cover subscription s. The *polyhedron witness* to non-cover is a rectangle in this case, and is defined by the intersection of s and the element $\neg s_2^2 = \{x_1 > 870\}$. This rectangle is contained in s, but not in s_1 nor s_2.

Table 6. Non cover example: subscriptions

$Subscription\ s$
$[x_1 \geq 830 \wedge x_1 \leq 890 \wedge$
$x_2 \geq 1003 \wedge x_2 \leq 1006]$
$Subscription\ s_1$
$[x_1 \geq 820 \wedge x_1 \leq 850 \wedge$
$x_2 \geq 1002 \wedge x_2 \leq 1009]$
$Subscription\ s_2$
$[x_1 \geq 840 \wedge x_1 \leq 870 \wedge$
$x_2 \geq 1001 \wedge x_2 \leq 1007]$

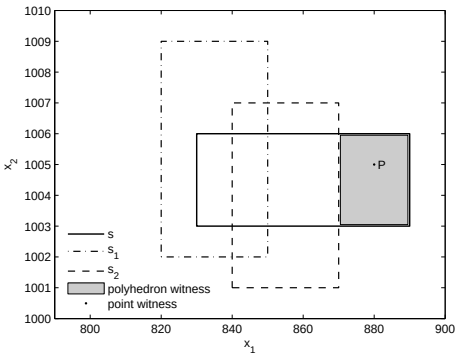

Fig. 3. Non-cover example: graphical presentation of a polyhedron witness and point witness

Definition 4. A *point witness* to non-cover is a point that satisfies s, but does not satisfy S. A point witness is inside a polyhedron witness, but not inside S.

In the previous example, any point inside the polyhedron witness rectangle defined by $s \wedge \neg s_2^2$ is a point witness. The following 2 corollaries are based on the properties of the conflict table, polyhedron witness and point witness.

Corollary 1. If all T_i^j for $1 \leq j \leq r$ are undefined, then s is covered by s_i.

Proof. If all $T_i^1, \ldots, T_i^r$ are undefined, then $(s \wedge \neg s_i^1, \ldots, s \wedge \neg s_i^r)$ are all not satisfiable, and thus $(s \sqsubseteq s_i^1) \wedge \ldots \wedge (s \sqsubseteq s_i^r)$, or alternatively, $s \sqsubseteq (s_i^1 \wedge \ldots \wedge s_i^r)$. In

effect s is covered by s_i. Thus, as a side-effect, the use of the conflict table provides a sufficient condition, tested in $O(m \cdot k)$, to check whether s is covered by any of the subscriptions individually.

Corollary 2. If all T_i^j for $1 \leq j \leq r$ are defined, then s covers s_i.

Proof sketch. If all $T_i^1, \ldots, T_i^r$ are defined, then $(s \wedge \neg s_i^1, \ldots, s \wedge \neg s_i^r)$ are all satisfiable, and thus s includes s_i on all attributes.

Corollary 3. Let $t_{i_1}, t_{i_2} \ldots t_{i_k}$ be the list resulting from sorting $t_1, t_2 \ldots t_k$ in ascending order, where t_i represents the number of defined entries in row i of the conflict table T. If $t_{i_j} \geq j$ for $1 \leq i_j \leq k$, then s is not covered by S.

Proof sketch. If $t_{i_j} \geq j$ for $1 \leq i_j \leq k$, then a polyhedron witness exists. It can be constructed in the following way: Choose any element $s_{i_1}^{j_{i_1}}$ to be part of a polyhedron witness, and then eliminate any conflicting entries from other rows. Since each row will have a maximum of one conflicting element with $s_{i_1}^{j_{i_1}}$, then at most one element in each row will be eliminated. If this step is repeated k times a polyhedron witness will be derived. Thus, s is not covered by S.

4 Probabilistic Cover Algorithm

In this section we describe the probabilistic cover algorithm to solve the defined subsumption problem. This algorithm has direct implications on the effectiveness of routing both publications and subscriptions in a distributed environment, and the reliability of discovering the matching publications. The probabilistic core of the algorithm is the 'Monte Carlo type' Random-Simple-Predicates-Cover part. It runs in a fixed number of iterations, but may produce an incorrect result with a certain pre-determined probability of error. The probability of error is problem specific, and we show that an upper bound on this error is derived in polynomial time prior to the execution of the algorithm. Thus, the performance of the algorithm can be decided in advance based on the reliability desired. The Random-Simple-Predicates-Cover can be executed independently or in conjunction with the minimal cover set algorithm which reduces the original set of subscriptions S to a minimal set of subscriptions against which a new subscription s has to be checked. We also introduce a number of optimizations used for making fast decisions under specific conditions that can be detected from the conflict table.

4.1 Random Simple Predicates Cover

The Random Simple Predicates Cover (RSPC) algorithm exploits the property of *point witnesses*. If the algorithm guesses a point in s that is a *point witness* to non-cover for the set of subscriptions S, then the subsumption problem is solved with a definite NO, i.e. $s \not\sqsubseteq S$. On the other hand, in case a subsumption relationship exists, the algorithm would try in vain to find such a witness. To prevent this situation, we define a threshold d for the number of guesses, and the algorithm outputs a probabilistic YES, i.e. $s \sqsubseteq S$ with a predefined probability of error.

Algorithm 1 defines the RSPC algorithm which executes a number of iterations d to randomly generate a point satisfying subscription s and checks whether it is a *point witness*. To generate a point within s costs $O(m)$, and verifying whether it lies inside

Algorithm 1 Random-Simple-Predicates-Cover

```
1: /* Decide whether a subscription s is covered by the existing subscriptions set S */
2: for i = 1 to d do
3:    GUESS a point P inside s
4:    if P does not satisfy subscriptions set S then
5:       RETURN false
6:    end if
7: end for
8: RETURN true
```

any of $s_1, s_2, \ldots s_k$ can be done in $O(m \cdot k)$ steps. Overall, the algorithmic complexity of RSPC is $d(m + m \cdot k)$, or $O(d \cdot m \cdot k)$. However, our experiments in Section 6 show that this is a pessimistic upper bound since at any iteration, RSPC can output a definite NO if the guessed point is indeed a point witness. In addition, the complexity can greatly be reduced using the optimizations presented in Sections 4.2 and 4.3.

Proposition 1. RSPC returns NO when s is definitely not covered by S. It returns YES with a probability error δ upper bounded by

$$\delta = (1 - \rho_w)^d, \tag{1}$$

where ρ_w is the probability that a randomly generated point P inside s is a point witness.

Proof. If RSPC returns NO then a point witness was found, and thus s is definitely not subsumed by S. Therefore, the answer is correct. If s is not subsumed then RSPC returns YES only if none of the guessed points is a point witness. For each trial this happens with probability less than $1 - \rho_w$, therefore for d trials the probability RSPC returns YES is less than $(1 - \rho_w)^d$, since d trials are randomly generated and are thus assumed to be independent. □

In problems with specific probability of error δ, we can compute the necessary number of trials, d, to answer the subsumption question with the required δ, using Equation 1 beforehand in polynomial time. The number of trials increases with a decrease of the error probability.

The value of ρ_w depends on the number of existing point witnesses for the particular subscription s related to the set of subscriptions S, and the 'size' of subscription s. Let N_w be the number of existing point witnesses and $I(s)$ the size of s. Then, $\rho_w = N_w / I(s)$.

Proposition 2. If the subsumption relationship holds, $N_w = 0$. Otherwise, there exists at least one polyhedron witness s_w and $I(s_w) \leq N_w \leq I(s)$. This implies that $I(s_w)/I(s) \leq \rho_w \leq 1$.

Proof. If there is union cover, then clearly all the points inside s will also be inside S and hence $N_w = 0$. If there is no union cover, then there exists at least one polyhedron witness. If s_w is the witness with the smallest number of integer solutions then $N_w \geq I(s_w)$, and hence $\rho_w \geq I(s_w)/I(s)$.□

Since the probabilistic algorithm may produce a wrong answer only if s is not subsumed by S, the worst situation is to assume that s is indeed not subsumed by the set. To compute the upper bound on d, we need to determine the lower bound on ρ_w which is set by the lower bound on N_w for the smallest polyhedron witness.

Algorithm 2 Compute ρ_w

/* Compute the probability of guessing a point witness */
/* Construct and use the conflict table T */
$I(s_w) = 1$
$I(s) =$ number of points in s
$p_w = 0$
$min = 0$
for $i = 1$ to m **do**
  $aux = min = s^{(2*i-1)} - s^{(2*i)}$
  for $j = 1$ to k **do**
    if $T_j^{(2*i-1)}$ is defined **then**
      $aux = T_j^{(2*i-1)} - s^{(2*i-1)}$
    end if
    if $aux < min$ **then**
      $min = aux$
    end if
    if $T_j^{(2*i)}$ is defined **then**
      $aux = s^{(2*i)} - T_j^{(2*i)}$
    end if
    if $aux < min$ **then**
      $min = aux$
    end if
  end for
  $I(s_w) = I(s_w) * min$
end for
$\rho_w = I(s_w)/I(s)$
RETURN ρ_w

Algorithm sketch for computing d. First, the algorithm computes ρ_w, for which the lower bound depends on the size of the smallest existing polyhedron witness s_w, (its $I(s_w)$), and $I(s)$. $I(s_w)$ can be approximated by multiplying the minimum non-covered ranges on each attribute over all subscriptions in the set. Then, we can determine the upper bound on d by extracting it from the Eq. 1 using the computed value for ρ_w and δ.

4.2 Minimized cover set of subscriptions

To further reduce the number of subscriptions against which s needs to be checked, we introduce another algorithm, the minimized cover set algorithm (MCS). From the set of subscriptions S, MCS constructs a non-reducible set of subscriptions, by ignoring those that are redundant for the covering detection problem. MCS selects a set of subscriptions that could jointly cover s and filters out duplicate subscriptions, those covering the same parts of s, and subscriptions that do not intersect with s. The remaining subscriptions form the non-reducible set S' against which s is subsequently checked by RSPC.

Definition 5. Two defined entries in the table, $T_{i_1}^{j_{i_1}}$ and $T_{i_2}^{j_{i_2}}$ are said to be conflicting if $i_1 \neq i_2$, and $s \wedge T_{i_1}^{j_{i_1}} \wedge T_{i_2}^{j_{i_2}}$ is not satisfiable. A defined entry $T_i^{j_i'}$ is said to be *conflict-free* if given any polyhedron witness $W_1 = \{s_1^{j_1}, \ldots, s_i^{j_i}, \ldots, s_k^{j_k}\}$, then $W_2 = W_1 - T_i^{j_i} \bigcup T_i^{j_i'}$ is also a witness. Let f_{c_i} denote the number of conflict free entries in row i.

Lemma 1. Given the set $W = \{T_1^{j_1}, \ldots T_k^{j_k}\}$ of defined entries in the conflict table, then, $s \wedge T_1^{j_1} \wedge \ldots \wedge T_k^{j_k}$ is not satisfiable if and only if there exists at least one conflicting pair in W.

Proof. If W has a conflicting pair, say $T_{i_1}^{j_{i_1}}$, $T_{i_2}^{j_{i_2}}$ then $(s \wedge \neg s_{i_1}^{j_{i_1}} \wedge \ldots \wedge \neg s_{i_k}^{j_{i_k}})$ is not satisfiable. If W does not contain any conflicting pair, then all range constraints in $(s \wedge \neg s_{i_1}^{j_{i_1}} \wedge \ldots \wedge \neg s_{i_k}^{j_{i_k}})$ are pairwise intersecting. Thus $(s \wedge \neg s_{i_1}^{j_{i_1}} \wedge \ldots \wedge \neg s_{i_k}^{j_{i_k}})$ defines a nonempty hyper rectangle and is satisfiable. □

Proposition 3. A *conflict free* entry from a conflict table T is any defined element $T_{i_1}^{j_{i_1}}$ that does not conflict with any other defined element $T_{i_2}^{j_{i_2}}$, where $i_1 \neq i_2$.

Proof. Assume $W = \{T_1^{j_1}, \ldots, T_{i_1}^{j_{i_1}}, \ldots, T_k^{j_k}\}$ is a witness. Since W has no conflicting elements, $T_{i_2}^{j_{i_2}}$ does not conflict with any element in W, then $W_1 = W - \{T_{i_1}^{j_{i_1}}\} \bigcup \{T_{i_2}^{j_{i_2}}\}$ has no conflicting pairs. From Lemma 1, W_1 is a witness and $T_{i_2}^{j_{i_2}}$ is conflict free. □

Conflict free entries are determined by comparing entries from the conflict table for different subscriptions. If a constraint on an attribute conflicts with any other constraint defined by another subscription, the entry is conflicting. It is conflict free otherwise.

Figure 4 presents a set of 3 subscriptions, s_1, s_2 and s_3, as well as the a subscription s, and Table 8 the corresponding conflict table. We can observe that the defined entries for s_3 are conflict free: they are not conflicting with the entries from s_1 and s_2. On the other hand, s_1 and s_2 have conflicting entries because x_1 cannot simultaneously satisfy both conditions, $x_1 > 850$ and $x_1 < 840$.

Table 7. Conflict-free example: subscriptions

$Subscription\ s$
$[x_1 \geq 830 \wedge x_1 \leq 870 \wedge$
$x_2 \geq 1003 \wedge x_2 \leq 1006]$
$Subscription\ s_1$
$[x_1 \geq 820 \wedge x_1 \leq 850 \wedge$
$x_2 \geq 1001 \wedge x_2 \leq 1007]$
$Subscription\ s_2$
$[x_1 \geq 840 \wedge x_1 \leq 880 \wedge$
$x_2 \geq 1002 \wedge x_2 \leq 1009]$
$Subscription\ s_3$
$[x_1 \geq 810 \wedge x_1 \leq 890 \wedge$
$x_2 \geq 100 \wedge x_2 \leq 10054]$

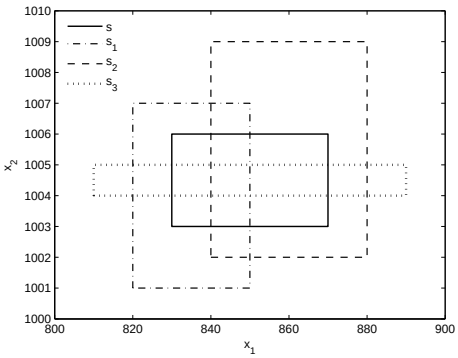

Fig. 4. An example with conflict free entries

Table 8. Conflict table for the example in Figure 4

s_i	$x_1 < low_i^1$	$x_1 > high_i^1$	$x_2 < low_i^2$	$x_2 > high_i^2$
s_1	$undefined$	$x_1 > 850$	$undefined$	$undefined$
s_2	$x_1 < 840$	$undefined$	$undefined$	$undefined$
s_3	$undefined$	$undefined$	$x_2 < 1004$	$x_2 > 1005$

Proposition 4. If $f_{c_i} \geq 1$, or $t_i \geq k$, then s_i is redundant.

Proof. If s is subsumed by $(s_1 \vee \ldots \vee s_{i-1} \vee s_{i+1} \vee \ldots \vee s_k)$ then it is trivially subsumed by $(s_1 \vee \ldots \vee s_{i-1} \vee s_i \vee s_{i+1} \vee \ldots \vee s_k)$. For the reverse to be true, some conditions must be satisfied. Let us assume that s is not subsumed by $(s_1 \vee \ldots \vee s_{i-1} \vee s_{i+1} \vee \ldots \vee s_k)$, then there exists a polyhedron witness $W = \{T_1^{j_1}, \ldots, T_{i-1}^{j_{i-1}}, T_{i+1}^{j_{i+1}}, \ldots, T_k^{j_k}\}$.

- $f_{c_i} \geq 1$. Let $T_i^{j_i}$ be one of the conflict free elements in s_i. From the definition of conflict free it follows that $W = \{T_1^{j_1}, \ldots, T_{i-1}^{j_{i-1}}, T_i^{j_i}, T_{i+1}^{j_{i+1}}, \ldots, T_k^{j_k}\}$ is also a witness, and hence s is not subsumed by $(s_1 \vee s_2 \vee \ldots \vee s_k)$.
- $m_i \geq k$. Since each element in W conflicts at most with one element in row i, there exists at least $m_i - (k-1)$ elements in row i which do not conflict with any of W. Given $m_i \geq k$, then $m_i - (k-1) \geq 1$, implying that there exists at least one element in row i not conflicting with any element in W. Let that element in row i be $T_i^{j_i}$. It follows that $\{T_1^{j_1}, \ldots, T_{i-1}^{j_{i-1}}, T_i^{j_i}, T_{i+1}^{j_{i+1}}, \ldots, T_k^{j_k}\}$ is also a witness, and hence s is not subsumed by $(s_1 \vee s_2 \vee \ldots \vee s_k)$. □

The MSC algorithm consists of two main steps, as defined in Algorithm 3. First, starting from the conflict table T, it counts the number of defined elements for all subscriptions s_i in the corresponding rows, t_i and computes the number of conflict free elements, f_{c_i}. Then, it removes from the set all subscriptions for which t_i is equal to or greater than the current number of subscriptions in the set. It also removes subscriptions that have at least one conflict free element in the corresponding row of the conflict

table. These two steps are repeated until there are no more subscriptions that fulfill any of the two conditions. The remaining subscriptions form the non-reducible cover set S' for answering the union covering problem.

Algorithm 3 Minimized Cover Set

```
1: /* Find the minimized set of subscriptions S' relevant for subsumption detection */
2: /* Construct and use the conflict table T */
3: repeat
4:    S' = S
5:    for every row i in T do
6:       compute f_ci /* number of conflict-free elements in row i in T */
7:       if f_ci ≥ 0 or t_i ≥ k then
8:          remove row i from T
9:          remove subscription s_i from S'
10:         k = k − 1
11:      end if
12:   end for
13: until no s_i can be removed
14: RETURN S'
```

Considering the conflict table from Table 8, in the first step, we can see that no subscription has more defined entries than the total number of subscriptions ($t_1 = t_2 = 1$ and $t_3 = 2$ which is smaller than 3), while only s_3 has conflict free entries. Based on the elimination conditions (in this case, $f_{c_3} = 2 > 0$), in the first iteration, MCS can remove subscription s_3. In the second iteration, still no subscription has more defined entries than the total number of subscriptions ($t_1 = t_2 = 1 < 2$) and there are no conflict free entries, so the algorithm stops. The minimized cover set is $S' = \{s_1, s_2\}$.

Determining if a table entry is conflict free is $O(m \cdot k)$. Therefore computing each f_{c_i} costs $O(m^2\ k)$, and in turn steps 1 and 2 in each iteration of the minimized set cover algorithm costs $O(m^2\ k^2)$. Steps 1 and 2 may be repeated k times since each time step 2 is performed at least one s_i is filtered out. As a result, the overall cost of the algorithm reduction is $O(m^2\ k^3)$ in the worst case.

4.3 Fast decisions based on sufficient conditions

To summarize, in order to answer the subsumption problem, the algorithm first constructs the conflict table, runs the MSC algorithm to reduce the subscription set, and then applies the probabilistic RSPC algorithm which produces a either definite NO or a probabilistic YES. Nevertheless, for some specific cases, the algorithms can efficiently give a deterministic answer. Here we briefly present three specific cases.

1. Pairwise subsumption: As stated in Corollary 1, it is possible to detect if a subscription s is entirely covered by another subscription and produce a definite YES by analyzing the conflict table. If the row in the conflict table corresponding to subscription s_i contains only *undefined* values, then s_i covers the new subscription.

2. The outcome of the MCS algorithm can be an empty set, which means that there are no candidate subscriptions that could jointly cover s, and the algorithm will produce a definite NO.
3. Polyhedron witness: Detecting the existence of a polyhedron witness suffices to detect a non-cover relationship and output a definite NO as stated in Corollary 2. Based on the definitions of the polyhedron witness and conflict free entries, we can detect the presence of such a witness, depending on the number of defined entries in the conflict table without using either RSPC or MSC. The rows of the conflict table are sorted in ascending order of the number of defined table entries per row. In the ordered conflict table, if for each row, the number of defined entries is greater than the row number, the new subscription is not covered.

Algorithm 4 Fast decisions based on sufficient conditions

```
/* Check conditions for fast deterministic answers to the covering problem */
/* Construct and use the conflict table T */
for i = 1 to k do
    for j = 1 to p do
        /* Set flag to true to determine pairwise coverage */
        flag = true
        if T_i^j ≠ undefined then
            flag = false
        end if
        if flag == true then
            RETURN true
        end if
    end for
end for
SORT T into ascending order of defined entries per row
/* Set flag to true to detect the presence of a polyhedron witness */
flag = true
for i = 1 to k do
    if t_i < i then
        flag = false
    end if
end for
if flag == true then
    RETURN false
end if
S' = MCS()
if S' = ∅ then
    RETURN false
end if
RUN RSPC
```

4.4 Matching publications to a set of active subscriptions

Definition 6. A *publication* p is a point in the attribute space. It has values for all defined attributes.

When a publication p arrives, it has to be matched against the subscriptions in the system. By checking only the uncovered subscriptions first, (the set S), we can avoid checking *all* existing subscriptions. If there is no matching to any of these subscriptions, it cannot match any subscription that is covered and therefore removed from the set S. If there is a match, the publication must also be checked against the set of covered subscriptions SS.

Algorithm 5 Matching Publications to Subscriptions

```
1: /* Match a publication p to the set of uncovered subscriptions S and covered subscriptions
   SS*/
2: /* Set flag to true to remember any match */
3: flag = false
4: for every subscription s_i in S do
5:    if p is_covered by s_i then
6:       flag = true
7:       send p to s_i
8:    end if
9: end for
10: if flag == true then
11:    for every subscription s_i in SS do
12:       if p is_covered by s_i then
13:          send p to s_i
14:       end if
15:    end for
16: end if
```

Algorithm sketch. Check the publication p against the uncovered subscriptions. Whenever a subscription s_i covers p, send a notification to the associated subscriber. If there was a match, all covered subscriptions must be checked; otherwise, the publication is ignored without checking the covered subscriptions.

The cost of checking a publication against a subscription is $O(m)$, thus, the cost for matching a publication to the set of subscriptions is $O(m \cdot k)$, where k is the number of uncovered subscriptions. If there was a match, the covered subscriptions must also be checked, so the cost is at most $O(m \cdot N)$, where N is the total number of subscriptions.

Optimization. The covered subscriptions set can be organized by remembering for each element, the subscription(s) that cover it. This will create a (possible) multi-level structure. Then, publications are checked against the next level of subscriptions only if there was a match at the higher level. On the other hand, whenever a subscription is matched, the other subscriptions coming from the same neighbor (broker) need not be checked, as the publication will be forwarded to that broker anyway.

5 Subscription propagation in a distributed system

As in a distributed system subscription propagation affects the overall system performance, here we analyze the implications of incorrectly declaring a subscription as covered. Equation 1 gives the upper bound for the probability of error in incorrectly withholding the forwarding of a subscription, and therefore, it represents the likelihood of not finding a matching publication if it is available at the next broker. In a distributed publish/subscribe system, data is routed throughout the system, and we need to analyze the influence of our probabilistic algorithm on subscription propagation. We consider in Figure 5 a simple and illustrative case, where the new subscription s should be propagated along a chain of brokers $B_1, B_2, \ldots, B_n$.

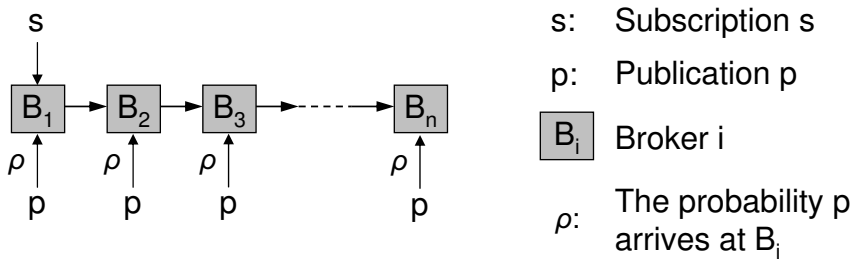

Fig. 5. New subscription propagation

We assume that the new subscription s is issued at broker B_1, while subscriptions $s_1, s_2, \ldots, s_k$ have already been propagated down the path to all brokers. Let ρ be the probability that a matching publication p (matches s but no s_i) is issued at any of the brokers B_i. The overall performance of the probabilistic algorithm is given by the probability of finding the matching publication, wherever it resides.

Proposition 5. The probability of finding the matching publication p under the condition that s is erroneously found to be covered by $s_1 \wedge s_2 \wedge \ldots \wedge s_k$, where $s_1, s_2, \ldots, s_k$ have been propagated to all brokers along the path, and all brokers have equal probability of ρ of receiving publication p is:

$$\sum_{i=1}^{n} \rho[(1-\rho)(1-(1-p_w)^d)]^{i-1}, \tag{2}$$

where ρ is determined by the network density and the communication distance of two neighboring brokers, and n is the total number of brokers in the path.

Equation 2 gives the lower bound for the overall algorithm performance. However, as we will show in the next chapter, the effective performance is much better in practice, even for loose error probabilities.probabilities. On the other hand, the longer the broker

path, the more important is the reduction in the global subscription traffic along the path, which reflects the local reduction at each broker, exponentially amplified in the network diameter.

Note that we do not present in this paper the mechanism for dealing with subscriptions cancelation. This issue can be tackled by explicit forwarding of unsubscriptions between brokers or by associating an expiration time with each new subscription. According to our approach, the canceled subscription can either be covered, and then cancelation has only the effect of removing it from the passive set, either be present in the (active) subscription set, and then its covered subscription must be promoted to this set, to replace it.

6 Experimental Evaluation

In this section, we evaluate the performance of the proposed probabilistic approach using simulations. We investigate algorithm performance in terms of *efficiency* and *effectiveness* for a number of subscription generation scenarios. Efficiency is analyzed in terms of the number of actual algorithm steps performed to answer the subsumption question, and effectiveness as the ratio of recognized redundant subscriptions to the total number of redundant subscriptions. Especially, we are interested in potential gains when using the MCS algorithm: We want to quantify both the costs and gains when using MSC in specific subscription generation scenarios. Next, we analyze the number of false decisions declaring a subsumption relationship when there was no subsumption. Finally, we compare our approach with the existing one for pair-wise coverage detection.

There are two specific categories of subscription settings, for which we want to investigate the performance:

(1) Covering: s is covered by the set of subscriptions (with some of $s_i \in S$ being redundant).
(2) Non-cover: s is not covered by the set S (as such, all subscriptions are redundant).

In particular, we have analyzed the algorithm performance using the following subscription generation scenarios:

(1.a) Pairwise covering scenario; s is entirely covered by at least one subscription from the set of existing subscriptions.
(1.b) Redundant covering scenario; s is not covered by any single subscription, but is covered by the set, with a lot of subscriptions being redundant.
(2.a) No intersection scenario; s does not intersect with any existing subscription.
(2.b) Non-cover scenario; s is not covered by the set S, but overlaps with existing subscriptions over many attributes.
(2.c) Extreme non-cover scenario; similar to (2.b), but s has only a very small non covered gap.
(1-2) Comparison scenario; generate incoming subscriptions randomly.

Scenario (1.a) is straightforward as the subsumption relationship is determined efficiently by applying Corollary 1 after the construction of the conflict table, therefore the cost of detecting pairwise coverage is $O(m \cdot k)$ (equals the cost of constructing the conflict table). Scenario (2.a) is also straightforward because the MCS algorithm determines non subsumption after the first iteration, which removes all subscriptions from the set S' because all $s_i \in S'$ have conflict-free elements in the conflict table. The remaining three scenarios are difficult settings for checking the subsumption relationship, as there are no pairwise subsumptions which could help to reduce the set S', and the classical approach cannot give an answer to the group covering problem.

We tested the scenarios (1.b), (2.b), and (2.c) using the following subscription generation principle: Existing subscriptions overlap with a new subscription and each other for many attributes, but there are no pairwise subsumptions. In the experiments, s, as well as $s_1, s_2, \ldots, s_k$ were generated such that all of them are satisfiable (not empty), each s_i intersects with s and finally all s_i's are pairwise intersecting for at least one of the attributes. The last scenario (1-2) simulates a realistic setting assuming that user interest are similar, and that the popularity of attributes appearing in subscriptions is Zipfian. Additional restrictions to the selection of s_is were applied depending on the simulation scenario.

In the redundant covering and non-cover scenarios, experiments were done with an increasing number of subscriptions, k, from 10 to 310 in steps of 30, for different number of attributes m: 10, 15, and 20. The probability of error δ was very low, 10^{-10}. For each measurement, 1000 algorithm runs were performed to compute the average performance. The overall cost of the algorithms was measured as the number of trials needed to answer the covering problem.

The extreme non-cover scenario aims at analyzing the importance of the probability of error and the non-cover gap size. The extremeness of the scenario implies that the new subscription is covered entirely, except for a narrow slice over one attribute, where we enforce a gap. For the measurements were conducted for a fixed number of subscriptions, $k = 50$, and fixed number of attributes per subscription, $m = 5$; the gap over the non-cover attribute was increased from 0.5% to 4.5% of the interval in steps of 0.5% and there were 3 probabilities of error, δ: 10^{-10}, 10^{-6}, and 10^{-3}. For each possible combination of tested parameters, 3000 algorithm runs were performed, to observe the number of false decisions and the average number of trials performed.

The comparison scenario is performed in a single run by generating a sequence of 5000 subscriptions and quantifying the sizes of the subscription sets for pair-wise and group coverage checking. Different number of attributes m: 10, 15, and 20 were analyzed and the probability of error δ was set to 10^{-6}.

6.1 Redundant covering scenario

This simulation scenario investigates the algorithm performance when the subscription set S subsumes s. A high rate of redundant subscriptions is introduced to test the influence of the MCS algorithm on the overall performance and its efficiency.

The experimental setup was constructed such that s was covered by the first 20% of the generated subscriptions forming S. The remaining 80% of subscriptions from

S were then generated to partly cover s, and are actually redundant since s is already covered by the first 20% of subscriptions.

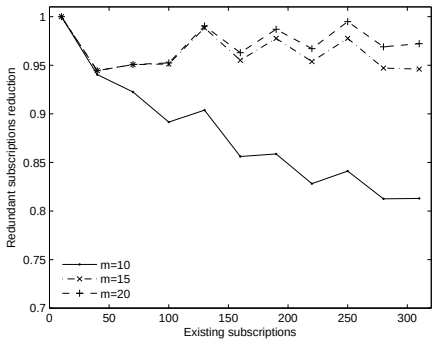

Fig. 6. Reduction for the redundant covering scenario

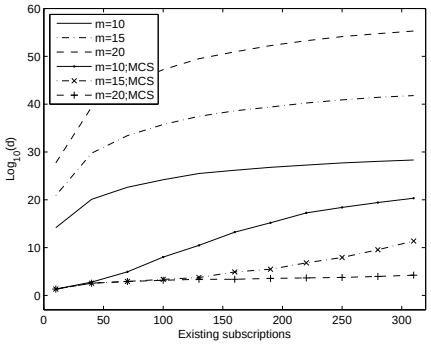

Fig. 7. Theoretical number of iterations for the redundant covering scenario

Figure 6 shows the efficiency of the MCS algorithm measured as the percentage of removed redundant subscriptions. The degree of reduction is extremely high: the algorithm successfully removed between 80% and 100% of redundant subscriptions. The performance decreases for small number of attributes (10) when increasing k, but increases for higher number of attributes. MSC removes almost all redundant subscriptions for small number of subscriptions, then the performance drops, as more computations are needed, but it improves with increasing number of subscriptions and attributes.

Figure 7 shows the theoretically predicted number of iterations d needed to answer the subsumption question. The $log(d)$ plot is shown as a function of k, and is calculated using Equation 1. The plot is given for the initial set of subscriptions S, and the reduced set of subscriptions S' after running the MCS. Due to the low error probability, d is extremely high if we use only the RSPC algorithm. However, MSC significantly reduces the number of needed iterations and becomes practically feasible: $d < 10^5$ for 100 subscriptions with 10 attributes, and decreases significantly for larger number of attributes.

6.2 Non-cover scenario

For the-non cover scenario, the experiment is constructed by forcing the non-covering of s by leaving a small range over x_1 uncovered. The values over the other attributes are generated randomly. The whole set of subscriptions S is actually redundant, as s is not covered and there is no cover set for s. In this scenario, the algorithm has always detected the non-coverage relationship due to optimizations and a low probability of error.

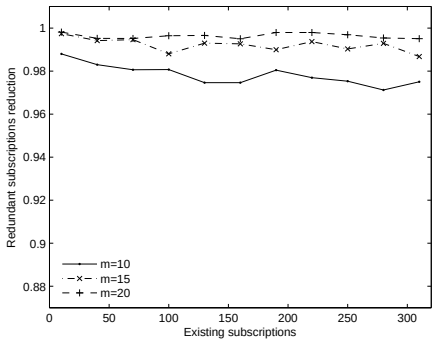

Fig. 8. Reduction for the non cover scenario

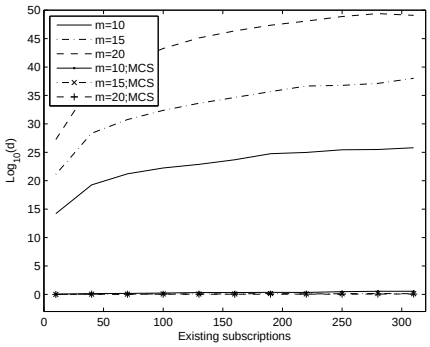

Fig. 9. Theoretical number of iterations for the non cover scenario

Figure 8 shows the efficiency of the MCS algorithm which performs even better than for the redundant covering scenario because most of the subscriptions are removed quickly due to the non covering relationship.

Figure 9 illustrates the theoretical number of iterations d which decreases tremendously after applying the MCS algorithm, proving that the algorithm needs few iterations to discover the non covering relationship.

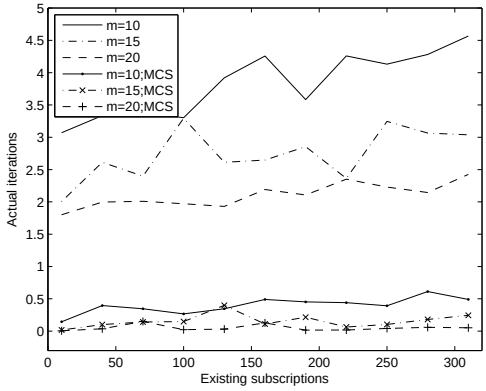

Fig. 10. Actual iterations for non cover

Since non cover can be detected prior to performing all d theoretical iterations, Figure 10 shows the actual number of iterations performed to discover a witness point. As it is visible, the average number of performed iterations is extremely low (< 0.5). This is due to the fact that in most of the cases, after running MCS there is no need to employ the probabilistic part of the algorithm to determine the non cover relationship

($d = 0$ because the reduced set is empty). There are some evident fluctuations in the number of d due to the probabilistic nature of the algorithm.

6.3 Extreme non-cover scenario

In the extreme non cover scenario, we generated the subscription set S such that it does not cover s over one attribute, for which we varied the size of a non covered range, while covering s entirely on all other attributes. The subscriptions in S are all intersecting with s; they are also pair-wise intersecting each-other, except at the bounds of the non covered range.

As performance metrics we illustrate the average number of guesses over 3000 runs needed to answer the covering problem and the total number of false decisions (that result in non forwarding of a non covered subscription) in 3000 runs.

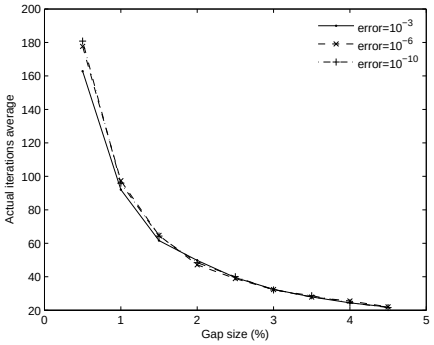

Fig. 11. Actual iterations performed

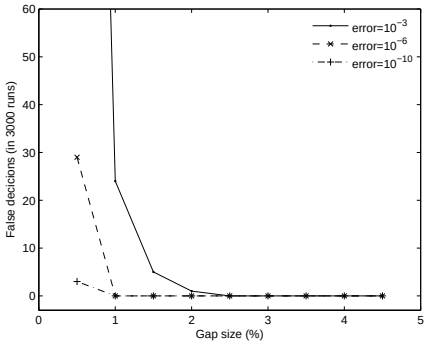

Fig. 12. Number of false negatives

Figure 11 shows that the average number of performed guesses is similar for all probabilities of error, even though the theoretical number of guesses increases for smaller error probabilities. This behavior is expected, as the chances of guessing a point witness depend on the ratio between the gap range size and the total range size of the non covered attribute, but it does not depend on the error probability, which we can choose arbitrarily low. Nevertheless, as the error probability is the same for covering and non cover cases, it should offer a compromise for the performance in both situations.

In Figure 12 we can see the total number of cases when the algorithm falsely decided, answering with a probabilistic YES, event though it analyzed a non cover situation. The number of false negatives increases with the error probability and decreases with larger gap sizes. In fact, for probabilities of error lower than 10^{-6} and gap range sizes of more than 1%, the algorithm always takes the right decision. Even for a higher probability of error (10^{-3}), the number of false negatives remains quite low, if the gap is at least 2%. The number of false negatives decreases fast with increasing number of attributes and subscriptions. The small values of k and m for this scenario are among

the largest that yield false negatives; for higher values, the algorithm is always right, because the number of allowed guesses (the computed threshold d) is also higher.

6.4 Comparison

Due to the lack of real-world subscription set, we have simulated a setting using power law distributions that are considered as good approximations of popularity both for the selection of attributes and attribute ranges. From the set of m attributes popular ones were chosen using a Zipf distribution (skew = 2.0). Attributes are generated in the following way: The center of a range is generated with a Pareto distribution (skew = 1.0) to simulate similar interests, while range sizes are generated with a normal distribution. The experiment compares the growth of subscription set sizes in case of the pair-wise and group subsumption reductions.

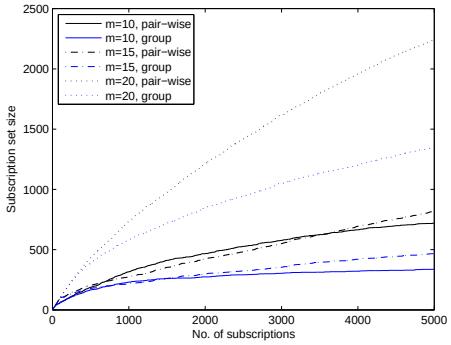

Fig. 13. Actual iterations performed

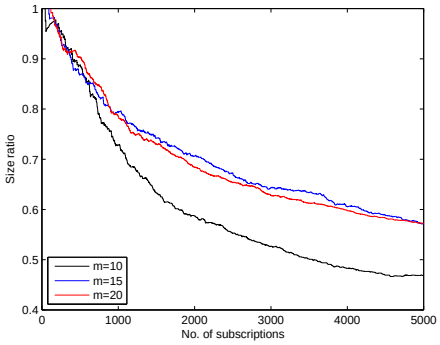

Fig. 14. Number of false negatives

Figure 13 shows the growth of the total number of active subscriptions when increasing the number of incoming subscriptions. It is interesting to observe the power of subscription set reduction using subscription coverage both for pair-wise and group coverage in case of partly covered subscription space. The group coverage shows greater reduction compared to the pair-wise algorithm for all values of m. For $m = 10$ and $m = 15$ group coverage has reduced the original set of 5000 subscriptions to less than 10%, and pair-wise coverage to approx. 15% of the entire set, while for $m = 20$ the reduction is still significant (around 33% for group and less than 50% for pair-wise coverage). The set reduction is very important for subscriptions with a large number of attributes which increases complexity because of the absolute subscription set size, e.g. some brokers have limited resources and may not handle more than 1000 active subscriptions. When increasing m, the actual number of active subscriptions is also larger, and this is due to the fact that the probability of subsumption generally decreases in the applied subscription generation scenario when increasing subscription space dimensionality.

Figure 14 quantifies the actual gain of group coverage compared to the pair-wise coverage by showing the ratio between the respective set sizes. The obtained results show the extreme reduction potential when increasing the number of incoming subscriptions. In case of 1000 received subscriptions, the ratio is between 70 and 80%, and keeps decreasing with new incoming subscriptions showing a stabilization tendency after 5000 subscriptions. The ratio is larger for large m, but still significant, and is almost similar for 15 and 20 attributes because the actual number of defined attributes does not significantly differ. Of course, the obtained results are highly dependent on subscription generation, but since our distributions follow a realistic popularity-based setting, it can be concluded that group coverage can greatly reduce the subscription set compared to the pair-wise approach.

6.5 Discussion

RSPC without MCS. The difference between the performances of RSPC in both settings, cover and non-cover, is statistically insignificant without applying MCS. The fact that the performance is similar in both cases indicates that the behavior of SPC is independent of the covering relationship. Our experimental results refer to three values of m, namely $m = 10$, 15 and 20. The results, however, can be readily extrapolated to all the intermediate values of m.

The following conclusions can be drawn:

1. The performance of the probabilistic algorithm deteriorates in both cases as m increases.
2. For both scenarios, the theoretical threshold d stabilizes when $k > r$. A possible explanation is that for a fixed value of m, as the value of k increases, the probability that the additional new predicates provide a smaller polyhedron witness is quite small.

RSPC with MCS. The reduction algorithm performs in general very well, and significantly reduces the size of the subscription set with which a new subscriptions has to be compared. Note that even after the reduction the subsumption cannot be tested efficiently by a deterministic algorithm as the problem is still co-NP complete. The main conclusion is that neither the reduction algorithm, nor the probabilistic algorithm alone can be an efficient solution for the large class of problems we are considering. However, the combination of the two provides an efficient solution to both covering and non covering cases. The algorithm performs extremely well in cases with a large number of subscriptions, especially for $k >> m$, when a lot of subscriptions are redundant. Even if the theoretical d is high in some cases, it is usually overestimated. As Figures 10 and 11 show, the actual number of iterations for the non cover case is much lower than the expected d value and this scenario may frequently happen in real-world applications. Finally, the comparison shows the supremacy of the group coverage algorithm over the classical pair-wise approach that will in general largely decrease the number of subscriptions in different distributed publish/ subscribe systems.

7 Related Work

Most of the research efforts in publish/subscribe systems have so far focused on the problem of efficient matching and forwarding of publications [17, 10, 8]. Pairwise covering and merging of subscriptions are typically used to reduce the set of active subscriptions, and all algorithms rely on some version of the *counting algorithm*, originally defined in [18]. In [10], the authors use an index called *selectivity table* relating attribute $name$ to lists of constraints, which are related to the set of subscriptions. This approach supports pairwise subsumption, while the list of constrains resembles our conflict table. Modified binary decision diagrams are employed in [8], to achieve pairwise covering and merging of subscriptions. The merging operation is a precursor of group covering, merging two subscriptions having at most one mismatch in their predicates. None of these techniques supports group subsumption, and therefore can filter out far fewer subscriptions than our probabilistic algorithm. The RAPIDMatch algorithm defined in [17] is designed for a simplified data model where subscriptions and publications signal the presence or absence of an attribute. It discriminates subscriptions against a notification, similar to our comparison of a subscription against a set of subscriptions.

The importance of reducing the number of subscriptions in a distributed environment is stressed in [9]. The authors are dealing with a complementary problem - merging a set of subscriptions to reduce their number. As in [8], the trade-off is that the new subscription might contain parts of the subscription space not covered originally by the set, which leads to false positives (unrequested publications). A recently proposed solution relies on clustering of subscriptions based on a proximity metric in subscription space [19], and would greatly benefit from global subscription set reduction for both the total number of subscriptions and the generated traffic.

In [20], the publish/subscribe paradigm is enhanced by a DHT (Distributed Hash Table), ensuring fast matching. Subscriptions and publications are dispatched (through a hashing method) in a small number of steps to the corresponding responsible nodes, where the actual matching takes place. Other publish/subscribe approaches have concentrated on efficient routing and flexible expressiveness. The authors designed in [3, 21] an architecture which improves the subscription expressiveness and data semantic by exploring XML-based query filtering and transformation. Approximate information is introduced into publish/subscribe systems in [12], allowing users to formulate data in less precise terms, thus integrating uncertainties. Relevance based matching, characteristic of information retrieval systems is devised in [22], pairing publications to subscriptions based on some similarity metrics. The same trade-off apply for all these methods, as they forward publications in the system even in the absence of a covering subscription.

Sensor networks deal with resource scarce devices, in terms of memory, transmission and computational power, and transmitted data can be lost due to traffic congestion and link failure ([23, 24]). Publish/subscribe has been recognized as an efficient communication paradigm in such a setting ([25, 24, 26]). The solution presented in [25] employs topic-based publish/subscribe, on the account that sensors have well defined attributes, modeled as topics. It does not support more expressive content-based communication, and data is aggregated only over one topic. In [24] the authors aim at re-

ducing the communication traffic through the use of content-based publish/subscribe over a reduced number of paths inside the system, identified by an augmented distance vector protocol. A semi-probabilistic approach, defined in [26], combines deterministic forwarding of publications (matching subscription which are propagated only in a small vicinity) with probabilistic forwarding to a random number of neighbors (when there is no match). This approach cannot guarantee 100% delivery; to achieve better reliability, it must increase publication traffic (and, consequently, the number of false positives). With our solution we have good reliability, while totally minimizing the publication traffic (only required publications are forwarded).

8 Conclusion

The paper presents a novel probabilistic algorithm for determining whether a subscription is covered by a set of subscriptions. Theoretically it solves the problem in $O(k \cdot m \cdot d)$. The probability of error is problem specific and very small, and an upper bound on the threshold d is determined in polynomial time prior to the execution of the algorithm. Our experiments have shown that the algorithm performs even better in practice with the introduced optimizations. When combing the probabilistic algorithm with the reduction algorithm that removes redundant subscriptions against which a new subscription needs to be checked, the number of needed iterations for the probabilistic algorithm becomes reasonable. Even more, in case of the non covering relationship, it is possible to give a deterministic answer without applying the probabilistic tests. Therefore, we can conclude that the proposed algorithms can efficiently solve the subsumption problem which is important for fast subscription forwarding and network congestion control in distributed publish/subscribe systems.

References

1. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys **35** (2003) 114–131
2. Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., Chandra, T.D.: Matching events in a content-based subscription system. In: Proceedings of the 18th ACM Symposium on Principles of Distributed Computing, ACM Press (1999) 53–61
3. Altinel, M., Franklin, M.J.: Efficient filtering of XML documents for selective dissemination of information. In: The VLDB Journal. (2000) 53–64
4. Campailla, A., Chaki, S., Clarke, E., Jha, S., Veith, H.: Efficient filtering in publish-subscribe systems using binary decision diagrams. In: Proceedings of the 23rd International Conference on Software Engineering. (2001) 443–52
5. Fabret, F., Jacobsen, A., Llirbat, F., Pereira, J., Ross, K., Shasha, D.: Filtering algorithms and implementation for very fast publish/subscribe. In Sellis, T., Mehrotra, S., eds.: Proceedings of the 20th Intl. Conference on Management of Data (SIGMOD 2001), Santa Barbara, CA, USA (2001) 115–126
6. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems **19** (2001) 332–383

7. Mühl, G., L.Fiege, Gärtner, F.C., Buchmann, A.: Evaluating advanced routing algorithms for content-based publish/subscribe systems. In: Proceedings of the 10th IEEE International Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02), IEEE Computer Society (2002) 167–176
8. Li, G., Huo, S., Jacobsen, H.: A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In: 25th International Conference on Distributed Computing Systems (ICDCS). (2005)
9. Crespo, A., Buyukkokten, O., Garcia-Molina, H.: Query merging: Improving query subscription processing in a multicast environment. IEEE Transactions on Knowledge and Data Engineering **15** (2003)
10. Carzaniga, A., Wolf, A.L.: Forwarding in a content-based network. In: Proceedings of ACM SIGCOMM 2003, Karlsruhe, Germany (2003) 163–174
11. Mühl, G.: Large-Scale Content-Based Publish/Subscribe Systems. PhD thesis, Darmstadt University of Technology (2002)
12. Liu, H., Jacobsen, H.A.: Modeling uncertainties in publish/subscribe systems. In: ICDE '04: Proceedings of the 20th International Conference on Data Engineering, Washington, DC, USA, IEEE Computer Society (2004) 510
13. Srivastava, D.: Subsumption and indexing in constraint query languages with linear arithmetic constraints. Annals of Mathematics and Artificial Intelligence **8** (1992) 315–343
14. Chen, Y., Schwan, K.: Opportunistic overlays: Efficient content delivery in mobile ad hoc networks. In: Middleware 2005, ACM/IFIP/USENIX, 6th International Middleware Conference. (2005) 354–374
15. Dittrich, J.P., Fischer, P.M., Kossmann, D.: Agile: Adaptive indexing for context-aware information filters. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005. (2005) 215–226
16. Hauswirth, M., Schmidt, R.: An overlay network for resource discovery in Grids. In: 2nd International Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE'05). (2005)
17. Kale, S., Hazan, E., Cao, F., Singh, J.P.: Analysis and algorithms for content-based event matching. In: 4th International Workshop on Distributed Event-Based Systems (DEBS'05). (2005)
18. Yan, T.W., García-Molina, H.: Index structures for selective dissemination of information under the Boolean model. ACM Transactions on Database Systems **19** (1994) 332–334
19. Voulgaris, S., Rivire, E., Kermarrec, A.M., van Steen, M.: Sub-2-sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In: 5th Int'l Workshop on Peer-to-Peer Systems (IPTPS). (2005)
20. Aekaterinidis, I., Triantafillou, P.: Internet scale string attribute publish/subscribe data networks. In: Proceedongs of the ACM Fourteenth Conference on Information and Knowledge Management (CIKM05). (2005)
21. Diao, Y., Rizvi, S., Franklin, M.: Towards an internet-scale xml dissemination service (2004)
22. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: Publish/subscribe functionality in ir environments using structured overlay networks. In: SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval. (2005) 322–329
23. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine **40** (2002) 102–114
24. Hall, C., Carzaniga, A., Rose, J., Wolf, A.: A content-based networking protocol for sensor networks. Technical Report CU-CS-979-04, Department of Computer Science, University of Colorado (2004)

25. Souto, E., Guimares, G., Vasconcelos, G., Vieira, M., Rosa, N., Ferraz, C., Kelner, J.: Mires: a publish/subscribe middleware for sensor networks. Personal Ubiquitous Comput. **10** (2005) 37–44
26. Costa, P., Picco, G.P., Rossetto, S.: Publish-subscribe on sensor networks: A semi-probabilistic approach. In: Proceedongs of the 2nd IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS'05). (2005)