Continuous Query Evaluation over Distributed Sensor Networks

Oana Jurca, Sebastian Michel, Alexandre Herrmann, Karl Aberer

Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

***Abstract*— In this paper[1] we address the problem of processing continuous multi-join queries, over distributed data streams. Our approach makes use of existing work in the field of publish/subscribe systems. We show how these principles can be ported to our envisioned architectural model by enriching the common query model with location dependent attributes. We allow users to subscribe to a set of sensor attributes, a service that requires processing multi-join correlation queries. The goal is to decrease the overall network traffic consumption by removing redundant subscriptions and eliminating unrequested events close to the publishing sensors. This is non-trivial, especially in the presence of multi-join queries without any central control mechanism. Our approach is based on the concept of filter-split-forward phases for efficient subscription filtering and placement inside the network. We report on a performance evaluation using a real-world dataset, showing the improvements over the state-of-the-art, as we reduce the overall data traffic by half.**

I. Introduction

The advances in sensing technologies, in particular wireless sensors, allow for continuous and detailed monitoring of the real world. Application scenarios are manifold, ranging from traffic or health monitoring, over monitoring of industrial sites and logistics, to environmental monitoring for scientific purposes. In environmental monitoring, scientists use a wide variety of observation networks, ranging from deployments with limited spatial extension, to interconnected sensor stations that often cover large regions, countries or continents, with a large selection of different sensors. Data is no more collected from a single dedicated observation network, but aggregated from heterogeneous measurements, typically performed by many different organizations. Similar trends can also be observed for the other application scenarios mentioned above.

We consider streams with high data rates and that data forwarding is expensive. Queries that target multiple streams will often focus on a particular region, but using sensor deployments operated by different organizations. Thus joins over data streams will frequently occur on geographically correlated data. For economical and organizational reasons, and security and privacy concerns, data providers will often be reluctant to share their data through central repositories. Our focus will be on solutions that support such settings where data stream processing occurs in a fully distributed environment and with only local knowledge available at the processing nodes.

[1]The main funding sources for this paper were 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 the Competence Center Environment and Sustainability of the ETH Domain (CCES).

As a concrete scenario, we will consider widespread sensor deployments in environmental science, as we encounter it in the *Swiss Experiment Project*, an interdisciplinary project involving more than 10 environmental research groups spread across Switzerland. Figure 1 gives an overview of the different field site locations. The different sensor deployments are run by a number of different organizations, that provide the often very expensive hardware devices and have the domain knowledge for their maintenance. The measured phenomena include meteorological data like temperature, snow height, air humidity, and solar radiation, but also more specific measurements for tasks like water shed monitoring, permafrost monitoring or land slide detection. The field locations vary from hydrological observations in the forealps to high alpine weather monitoring deployments such as those operated by the Snow and Avalanche Research Center (SLF) in Davos. Apart from scientific use we expect such data being made available for professional and personal use in large scale user communities. This is already true for meteorological data (e.g. storm, frost, floods warnings), but personalization capabilities are limited and integrated access to different operators is not supported.

Within the Swiss Experiment we use a sensor data stream processing middleware, GSN [2], as a common data gathering and processing platform. It opens the possibility to issue sophisticated continuous queries over the data streams, resulting in complex event descriptions that involve multiple sensors. Sensors can be addressed directly, but it is more likely that users are interested in one or more sensors within a particular spatial region. These interests are expressed as subscriptions. Such subscriptions are processed through continuous queries that report results back to the subscriber whenever a particular condition (event) is satisfied. The main difficulty is to effectively place subscriptions at carefully selected positions inside the network, such that sensor readings are filtered close to their sources, as the number of readings is typically several orders of magnitude larger than the number of subscriptions.

This becomes challenging in the case of complex subscriptions that involve different data sources, because sensor readings (events) have to be "joined" to match the subscription (event description). When there are large numbers of subscriptions in the system, they will naturally overlap, and so the same result sets will be repeatedly transmitted wasting network resources. This is in particular problematic in distributed, large-scale networks connected over the Internet, where network bandwidth is the limiting factor for achieving acceptable response times.

Fig. 1. Google maps view of current field sites within the Swiss Experiment project. Each point denotes one particular experimental site with in most cases several sensor stations installed, each having several single sensors attached.

The work in the fields of operator placement and publish/subscribe for large-scale streaming systems are the closest to our approach, as they consider joins over multiple streams, but they often require some form of global knowledge or even a centralized server for managing joins, and do not take complete advantage of the existing subscription overlaps.

We consider building a publish/subscribe system on top of the sensor data stream processing infrastructure without making any assumption on the network layout, the semantics of the sensor readings etc.. The proposed approach is based on the concept of filter-split-forward phases for efficient subscription filtering and placement inside the network, and an efficient, publish/subscribe forwarding of matching events. Note that we do not consider optimizing in-sensor-network data processing, but focus on the sensors or sensor stations connected through a Internet-based data stream processing middleware.

Our contributions can be summarized as follows:

- We show why state of the art approaches from the publish/subscribe domain are insufficient, and we tailor them for sensor networks, by extending the subscription language, and by adapting subscription filtering and event matching processes.
- We adapt state-of-the-art solutions for continuous query processing over multiple data sources to work in a distributed setting, requiring local interactions only, without relying on global knowledge or a centralized server.
- We propose an algorithm for distributing the user subscriptions in the system composed of filtering, splitting and forwarding phases: it injects fewer subscriptions, which in turn generate less result sets overlap. Coupled with our event propagation algorithm which avoids result redundancy, it drastically lowers network traffic.
- We report on a performance evaluation replaying a real-world dataset, showing the improvements of the three distributed approaches on network traffic.

The paper is organized as follows. Section II reviews related work and positions our paper. Section III details how the state of the art can be adapted to function based only on local interactions between nodes in the system. Section IV gives a detailed overview of the considered model. Section V presents our proposed algorithms. Section VI presents the experimental results. Section VII concludes the paper and gives an overview of ongoing work.

II. Related Work

Several architectures have been proposed for interconnected sensor networks. Sgroi et al. [18] suggest basic abstractions, a standard set of services, and an API to free application developers from the details of the underlying sensor networks, focusing on systematic definition and classification of abstractions and services. Both IrisNet [12] and Hourglass [19] propose an Internet-based infrastructure for connecting sensor networks to applications. In Hourglass, the focus is on maintaining quality of service of data streams in the presence of disconnections. In IrisNet, the authors propose a two-tier architecture consisting of sensing agents, which collect and pre-process sensor data and organizing agents, which store sensor data in a hierarchical, distributed XML database. [11] and [17] provide hierarchical data stream query processing to acquire, filter, and aggregate data from multiple devices in a static environment, with the focus on filtering the useful data as close as possible from the producing device. Besides these architectures, a large number of systems for query processing in sensor networks exist. Most notably, Aurora [9], GSN [2], STREAM [5], TelegraphCQ [8], and Cougar [25]. Our method offers efficient query processing in a distributed setting and is independent of the underlying platform; thus it can be implemented on top of any system that offers sensor network access and interconnection.

The work in the field of operator placement ([22], [3], [16]) is similar to ours, as it proposes algorithms and metrics to decide on the placement of join operators shared between overlapping subscriptions in order to reduce data processing. However, they might require some form of global knowledge and the query rewriting techniques do not achieve a high query load reduction, while the result sets still overlap; thus both queries and, more importantly, result tuples are redundantly pushed in the system. In [26], the authors improve general operator placement techniques to allow for dynamic load balancing while minimizing communication costs; identical operators or groups of operators are shared between queries, but no complex sharing can be achieved, thus still duplicating result sets. Xing et al. [24], [23] discuss stream processing on a cluster of computing devices, with the aim to achieve faster processing by parallelization, while ensuring load balancing, such that no device can become overloaded and degrade the overall latency. These approaches assume network traffic is not an issue and choose processing nodes independent of data stream localization. Thus, they are not suitable for query processing in distributed networks. REED [1] takes into account that centralized query processing creates a high overhead, when there are few satisfying tuples, and processes queries inside the sensor network; it filters the streams with query condition(s), but it does not support queries over multiple, independent streams. Our approach can join tuples from distributed streams, while taking into account stream localization to reduce network traffic.

Publish/subscribe has been recognized as an efficient communication paradigm for sensor networks ([20], [13], [10]). [20] 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 [13] the authors aim at reducing 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, [10], combines deterministic forwarding of publications (those that match subscriptions 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. More recently, in [7], the authors achieve less duplication, by approximating subscriptions over multiple attributes, called multi-joins, with binary joins and by using a publish/subscribe dissemination of results. However, they assume a centralized processing location, and the join approximation technique introduces false positives.

III. Tailoring existing approaches for "local knowledge"-only interaction

We have explained in the previous section that existing approaches cannot be applied in a truly distributed environment, as they require some form of global knowledge (e.g., for global query plans) or do not consider the cost of streaming data in the network (e.g., for centralized or cluster processing). They cannot support distributed, processing nodes, with knowledge emerging only from local interaction, between neighboring (linked) nodes. We have analyzed traditional operator placement techniques, and adapted them to be applicable without global query plans. As for the recently proposed multi-join technique [7], we have distributed the processing tasks and the forwarding of data to all nodes.

A. Operator Placement Based on Local Interaction

In general, operator placement techniques construct a global query execution plan that reuses identical filters between different queries, and places these reduced number of filters (operators) on nodes, with the goal to minimize processing tasks and, possibly, to reach a balanced processing load between nodes. They achieve filter reuse between different queries by joining the corresponding query plans for the processing of that filter, while load balancing is achieved by relocating filters depending on a node's processing load. An operator takes as input one or several data streams, and generates one result set stream, which can be duplicated if downstream operators are part of different query plans.

Placing Operators Based on Local Interaction: We have designed a distributed operator placement and improved the reuse of operators. By taking advantage of well established publish/subscribe techniques, that achieve pairwise subscription reduction, we reuse wider filters for the more restrictive ones, which they cover entirely. Thus, different query plans can share not only identical operators, but also covering operators, which results in smaller number of operators and larger portions of shared query plans. To place operators, we do not construct a global query plan, instead we rely on local knowledge and interaction. We distribute query plans following subscription paths from users to corresponding data sources, which ensures streams are processed only on nodes that would have to relay them anyway. User subscriptions are placed as operators at nodes on the reverse data sources advertisement path, and are split into simpler join operators, each time the corresponding advertisement paths diverge. Due to the pairwise coverage check, split operators are filtered out at nodes where a covering operator resides.

Constructing the Result Sets: Each operator generates its own result set, and forwards it on the reverse subscription path. We assume a publish/subscribe forwarding which recognizes the covered (and filtered out) operators and generates the missing result set at the node where covering was detected. In the setting of traditional operator placement techniques, this can be seen as placing the more restrictive operator downstream from the covering operator, to filter the larger result set to extract the smaller result set. Thus, covered operators generate traffic only from the node where coverage was detected, to the user's node. This mechanism reduces the result set redundancy typical to operator placement techniques in the case of joins over multiple attributes. On the other hand, redundancies still exist: an attribute's publications can appear in different results sets, if the corresponding operators have different, but overlapping attribute sets. This drawback also pertains to typical operator placement techniques, and it is not reduced by having global knowledge.

B. Distributed Multi-join Processing

In most application scenarios the amount of publications largely exceeds the amount of user queries or subscriptions. Thus, the above mentioned drawback prompted further research among which, recently, Chandramouli et al. propose a more efficient approach in [7]. It splits multi-join queries into binary joins over a main attribute that generates the result set, and a filtering attribute, which sanctions the main attribute's matching events. Result sets for such binary joins are easier to dispatch efficiently to the users (thus less redundancy between result sets), but on the other hand generate false positives, which are forwarded all the way to the user and create additional network traffic. Although the authors discuss the importance of a distributed network of processing nodes, it is a centralized approach, where a processing server has full knowledge of users subscriptions and published data.

Placing Binary Joins Based on Local Interaction: We have designed a distributed version of the algorithm in [7], as the application in a distributed environment is not detailed. Data streams in our considered setting do not include a payload of data, so we treat all streams equally in the filtering process and when choosing the pairs of binary joins into which a multi join is split. We forward subscriptions following the reverse advertisement path, and start the splitting process only at the first node where the path diverges. We preserve the natural splitting into simple operators, according to the network connections behind this node, to send the individual filters to the corresponding data source. Splitting into binary joins results in higher number of forwarded subscriptions, but only from the divergence node towards the sensors (it acts in a way as the centralized server).

Keeping in line with the original algorithm, which processes together binary joins with the same signature, we perform pairwise covering filtering, for both binary joins and multi-

joins: covered multi-joins would eventually split into covered binary joins.

Constructing the Result Sets: Similar to the operator placement techniques, each binary join generates its own result set, comprised of the main attribute's matching values for which there exist matching values of the filtering attribute. Since each result set stream has only one attribute, it is easy to check whether one data item has been already forwarded over a link, which avoids result set overlap. Events traverse the network following the matching subscriptions (and corresponding binary joins) reverse path. We employ a publish/subscribe forwarding, which recognizes covered binary joins and subscriptions and generates the missing result set at the node where covering was detected. Matching data streams against binary joins creates false positives, which traverse the network all the way to the user, affecting the overall traffic.

IV. Model

A. Publish-Subscribe Model

We model sensors as data sources, where each sensor produces data of a fixed type and has a known location. The set of data types produced by sensors is denoted by $\mathcal{A}$, and a given sensor d has an attribute type $a_d \in \mathcal{A}$. Each attribute a_d has values v from a corresponding value domain $\mathcal{D}_{a_d}$. The location $location(d)$ of a sensor d is given as a value $p_d \in \mathcal{L}$ from the location domain, e.g., 2D or 3D space. To make its presence known, a sensor d produces a *data source advertisement* or short advertisement $DSA_d = (a_d, p_d)$. Measurements of a sensor d result in *publications* of events $e_d = (a_d, p_d, v, t)$, where v is the measured value and t is the time of the measurement event.

Conditions on events published by sensors are expressed as filters. A *simple filter*, f_a, expresses a condition on attributes of a given type and is of the form $min \leq a \leq max$ (in ordered domains) or $a = v$. A *simple filter with identification*, f_d, specifies the attribute type and the location, and thus applies to a single sensor d. It is of the form $(min \leq a_d \leq max) \wedge (location(d) = p_d)$. A complex filter with identification is a set of simple filters with identification. For a set of sensors D we use the shorthand notation $F^D = \{f_d, d \in D\}$ to denote a complex filter with identification. An *abstract filter* allows users to constrain the type and value of data sources from a given region $L \subseteq \mathcal{L}$. For a given set of attributes $A \subseteq \mathcal{A}$, an abstract filter $F^{A,L}$ is of the form $F^{A,L} = \{f_a \wedge p_d \in L, a \in A\}$, where f_a is a simple filter over attributes of type a. The nature of region containment conditions checks depends on the location domain $\mathcal{L}$, e.g. L can be an area in 2D space, a volume in 3D space, or a sub-location in a hierarchically organized location domain.

A user subscription is a set of filters expressing ranges either over explicitly named sensors, as an *identified subscription* $S_{id} = (F^D, \delta_t)$, or over types of data streams, bounded to a specified region, as an *abstract subscription* $S_{ab} = (F^{A,L}, \delta_t, \delta_l)$. Typically, users are interested in correlated data from multiple data sources, but publication timestamps are seldom identical over different sensors. Therefore, a *temporal correlation distance* δ_t provides the maximum acceptable difference between publication timestamps from different sensors, such that they are considered as correlated. Similarly, when users are interested in data streams from a region, the *spatial correlation distance* δ_l gives the maximum acceptable difference between different sensor locations such that their publications are considered as being correlated. If event correlation is independent of spatial proximity we set $\delta_l = \infty$. Most applications set δ_t and δ_l as constants, because correlations have the same meaning through out the application.

We can now formally specify the matching of events to subscriptions in our model: a simple event $e_d = (a_d, p_d, v, t)$ generated by a sensor d *matches at time* t an identified subscription $s_{id} = (F^D, \delta_t)$, if $d \in D$ and $f_d(v)$ evaluates to $true$. It matches at time t an abstract subscription $s_{ab} = (F^{A,L}, \delta_t, \delta_l)$ if $a_d \in A$, $p_d \in L$ and $f_{a_d}(v)$ evaluates to $true$. Subscriptions can also match complex correlated events $E = \{e_1, e_2, \ldots, e_n\}$. A complex event E matches a subscription s (identified or abstract) at time t, if the following conditions hold:

1) Completeness: There is **one** simple event for each sensor (identified subscription) or each attribute type (abstract subscription)
2) Each simple event $e_i = (a_i, p_i, v_i, t_i)$ matches the subscription s
3) $t = max_{i=1\ldots n} t_i$
4) $|t - t_i| < \delta_t$ for all $i = 1 \ldots n$

Additionally, for abstract subscriptions, the condition $|max(p_i - p_j)| < \delta_l, i, j = 1 \ldots n$ needs to be satisfied.

For optimizing the processing of event and subscription propagation, we will rely on *subscription subsumption*, which is defined as follows. A subscription s is subsumed by subscriptions s_i, $i = 1 \ldots k$, if for all complex events E matching s, there exists a subscription s_i, such that E also matches s_i.

B. System Model

Our system model consists of processing nodes connected in an acyclic graph. Nodes having sensors attached are also called sensor nodes, and represent sources of data streams, while other nodes are relaying nodes, as they relay data received from other nodes. Data propagation in our network happens three-fold, for each group of data: advertisements, subscriptions, and events. Similar to the generic propagation methods introduced by directed diffusion [14], subscriptions in our system follow the reverse dissemination path of advertisements, and events following the reverse dissemination path of subscriptions.

Each node maintains three types of information, the set of received advertisements from each neighbor, the set of received subscriptions from neighbors and local users, and the set of received events. While advertisements and subscriptions are expected to be valid until explicitly removed, or have predefined life times, events are assumed to be valid for a short amount of time. Their validity should, nevertheless, be longer than δ_t to allow potentially correlating events to traverse the network. On the other hand, having a finite event validity reflects the expectation that, after a given time, no further time-correlations will appear (all time correlated events have already traversed the network). As a consequence, simple events can be dropped from the local storage after they are no longer valid, thus storage space requirements are reduced.

We assume sensor stations to be connected via a wide area network where network bandwidth consumption is of major concern. However, note that we do not consider any in-network data exchange issues like network traffic reduction or message lost, assuming this is handled by the employed sensing infrastructure.

V. Filter-Split-Forward Processing

Network traffic is generated by advertisement, subscription, and event propagation, with the latter having the greater impact, as events are several orders of magnitude more numerous than subscriptions, which themselves can be orders of magnitude more numerous than sensors. For an efficient data traffic, received data must be stored at each node, in order to reduce forwarding of data, similar to general publish/subscribe systems.

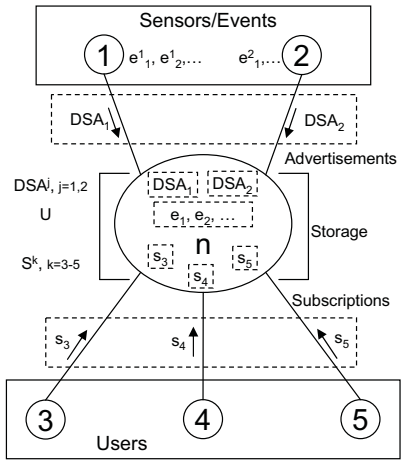

Fig. 2. Representation of data structures required at node n to process data sources advertisements, subscriptions and events from its neighbors.

We exemplify the data storage in Figure 2, for a node n with five neighbors, two of which are sensor nodes (nodes 1 and 2), while the others host user subscriptions. For simplicity, we do not show data relayed by n to its neighbors. First, n stores the advertisements received from each neighbor m or local sensors, in separate structures, identified with DSA^m, or DSA^{local}, respectively. Thus, the content of DSA^1 is DSA_1 and the content of DSA^2 is DSA_2, while DSA^{local} is empty (and not shown). Second, n must store the subscriptions received from its local users and its neighbors (more on when and how subscriptions are forwarded in Subsection V-B) in structures identified with S^{local} and S^m, respectively. In Figure 2, each of the neighbors 3, 4, 5 forwards one subscription, s_3, s_4, and s_5, respectively, stored at n in structures identified with S^m: $S^3 = \{s_3\}$, $S^4 = \{s_4\}$ and $S^5 = \{s_5\}$. Because there are no local users, S^{local} is empty and not shown. And third, all received events (forwarded by neighbors or published by local sensors) are stored together, in a structure identified with U.

In the following subsections, we will describe the propagation algorithms and the corresponding storage needs for each type of data in our systems. Subsection V-A describes advertisement processing. Subsection V-B presents our algorithms for subscription processing, discussing an advanced publish/subscribe technique for subscription filtering, **set filtering**, and why it cannot be directly applied to sensor networks. We detail how we develop this technique for the considered setting and exemplify with a simple subscription set over a small network. Event processing is presented in Subsection V-C, where we show how we match and detect complex events, while avoiding result set redundancies.

A. Advertisement Propagation

Advertisement propagation can be a straight forward flooding, as the number of data sources is expected to be small compared to subscriptions and events. Thus, a node propagates all received advertisements (from neighbors and those of local data sources) to all its neighbors, except the originating one. Also, all received advertisements are stored in the corresponding DSA data structure, per neighboring node, plus local data sources. This ensures that any incoming subscription can immediately follow the matching advertisements' reverse dissemination path.

Algorithm 1 Advertisement propagation

```
1: /* Run each time a new sensor d of type a appears at node n
   with location(n) = l_n */
2: if receive("sensor", d) at node n then
3:     append_to(DSA^local, (a, l_n))
4:     for all j ∈ neighbor(n) do
5:         send_advertisement(a, l_n)
6:     end for
7: end if
8: if on_receive("advertisement", a_d, l_d) at any node m from
   neighbor k then
9:     append_to(DSA^k, (a_d, l_d))
10:    for all j ∈ neighbor(m) \ {k} do
11:        send_advertisement(a_d, l_d)
12:    end for
13: end if
```

B. Subscription Propagation

A node can receive subscriptions from local users and from its neighboring nodes, and should forward them toward corresponding sensors, by following the reverse path of the respective sensor advertisements. A subscription cannot match events unless all of its attributes have corresponding sensors. Furthermore, to reduce the storage needs at each node and the subscription traffic in the system, nodes try to filter out redundant subscriptions, that is subscriptions subsumed by the set of already stored subscriptions. Any type of subscription filtering can be applied for this purpose, after which, only unfiltered subscriptions are forwarded to those neighboring nodes that have matching advertisements.

Set Filtering: To reduce subscription traffic, we have looked into the most efficient subscription filtering techniques, known in general as **set coverage** or **set subsumption**, proven to be co-NP complete [21]. In [15], we propose a probabilistic algorithm, which guarantees detection of set subsumption with a configurable probability of error. The probabilistic aspect can generate false positive decisions that a new subscription is covered by existing ones. The uncovered parts, or gaps, of such subscriptions generate false negative events, if events fall into these gaps. The algorithm takes as input as a parameter

the error probability as specified by users or applications. This parameter controls the tradeoff between processing cost and false negatives, i.e., smaller probabilities generate more processing load but less false negatives, and vice versa. We show in [15] that, in practice, the actual error is even smaller and decreases with larger subscription sets. We call this algorithm **set filtering**, and find it appropriate for distributed data streams, like sensor networks, where data sources could be unreliable, user subscriptions numerous and transmitted data can be lost due to in-network traffic congestion and link failure ([4], [13]).

However, **set filtering** as defined in the context of content-based publish/subscribe systems cannot be directly applied to distributed sensor networks, because it considers all subscriptions and events defined over the exact same set of attributes; in fact, all published data in such systems is already correlated. To apply **set filtering** in distributed data streams, we define subsumption within our model, considering also the location and timestamp meta-attributes, aside from the non-correlated data attributes. First of all, the timestamp attribute is orthogonal to the subsumption process, because it only affects the data content (the streams) and its correlation, and not user subscriptions (as long as δ_t is constant). On the other hand, the location meta-attribute has a direct impact, even more so as we support identified and abstract subscriptions.

For identified subscriptions, each sensor in the system acts as one attribute, while for abstract subscriptions, the data types act as data attributes. In fact, the location meta-attribute adds another dimension to the problem, and due to the location domain's containment property, it can be treated as just another data attribute, e.g., a location region can be contained in another region, or union of such regions. This translation of attributes and meta-attributes into data attributes partly reduces the difficulty of applying set subsumption in the context of distributed data streams, but we still face the problem of non-correlated data: subscriptions are defined over different attribute subsets. We cannot assume that a missing attribute in a subscription implies a request for the whole corresponding value space (instead, values for such an attribute are unrequested and should not be forwarded to the end user).

Illustrative Example: In Table I, three subscriptions query three previously advertised sensors, and are registered in the following order: s_1, s_2, s_3. Neither s_1 nor s_2 can be subsumed, because they request data from sensors not queried before. However, all of s_3's sensors have already been requested, and, furthermore, all data matching s_3 can be obtained by joining and filtering the result sets of s_1 and s_2. Thus, s_3 is subsumed by existing subscriptions, but this decision cannot be reached with **set filtering**.

TABLE I
SUBSCRIPTION SUBSUMPTION EXAMPLE

Subscriptions	*Sensor a*	*Sensor b*	*Sensor c*
s_1	$50 < a < 80$	$10 < b < 30$	
s_2		$20 < b < 40$	$2 < c < 20$
s_3	$55 < a < 75$	$15 < b < 35$	$5 < c < 15$

Propagation of subsumed subscriptions generates redundant processing and traffic in the network, which increase with the length of a subscription's path from the user to its matching sensors. We consider the previous subscription set appearing at the same node (n_6) in the small-scale setting of Figure 3, a 6-node network, with 3 sensor nodes (n_1, n_2, n_3). In this setting, even though s_3 is subsumed at n_6, it will traverse the whole network.

Divide and Conquer: We have designed a "divide-and-conquer" solution, which takes into account that subscriptions with more attributes are more restrictive than subscriptions with fewer attributes, but demand more processing for data correlation. For each new subscription, we first check subsumption against subscriptions with the same attribute set, and in case of a negative answer, we split it into simpler subscriptions, to compare with subscriptions over subsets of the initial attribute set. We use data source advertisements to drop subscriptions without sources and to guide answerable subscriptions towards the matching sensors. Keeping in mind that a sensor is affected only by one simple filter, part of the original subscription, we use the advertisement reverse path as a deterministic choice for subscription splitting. At the same time, we can postpone subsumption processing to the node where advertisement paths diverge. If all subscription subparts are subsumed along the paths, this process can actually detect subsumption against subscriptions over different attribute subsets.

Set Filtering Applied: We detail in Algorithm 2 how we can apply **set filtering** in our setting. We assume the same δ_t for all subscriptions in our system and in the case of abstract subscriptions, the same δ_l. We compare only subscriptions over the same attributes, i.e., both the set of subscriptions and the tested subscription have the same set of attributes, and only subscriptions of the same type (exceptions: simple operators and identified subscriptions where all attribute locations are logically the same). Under these conditions, we can apply **set filtering**, by either treating each different sensor type and, separately, the location information, as individual attributes (line 4), for abstract subscriptions, either treating each $(attribute, location)$ pair as an attribute (line 7), for identified subscriptions. We are not limited to filter out subscriptions to the arrival order; however, since traffic has already been generated by the existing subscriptions, we do not actively remove them.

Algorithm 2 filter($s, \mathcal{S}$): subscription filtering with **set filtering**

```
1: /* Check if a new subscription s is filtered by S */
2: if s = (F^{A,L}, δ_t, δ_l) then //s is abstract
3:     S = {s_i ∈ S, s_i = (F_i^{A_i,L_i}, δ_t, δ_l), A_i = A}
4:     return set_filtering(s, S) //Treat L as another attribute
5: else //s = (F^D, δ_t) is identified
6:     S = {s_i ∈ S, s_i = (F_i^{D_i}, δ_t), D_i = D}
7:     return set_filtering(s, S) //Each d (sensor) is one attribute
8: end if
```

Subscription Placement: A subscription might be forwarded as a whole, but could also be split into subsets of filters: one subset for each neighbor with matching advertisements, over the attributes common to those advertisements and the processed subscription. As a consequence of this split and forward process, nodes forward subscriptions either as the

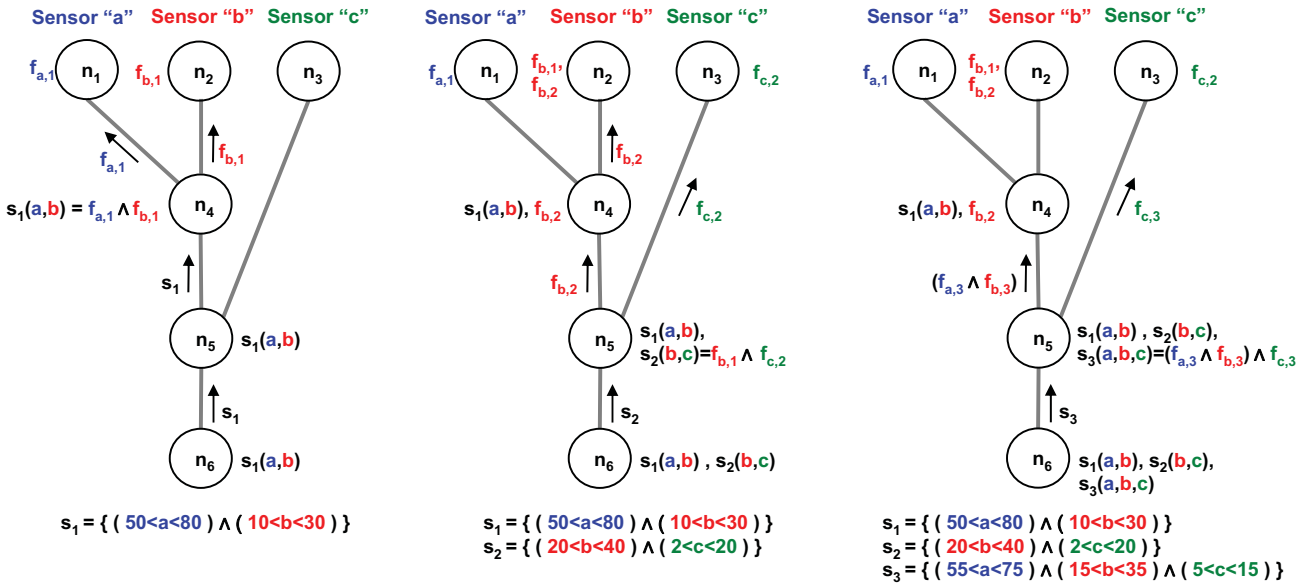

Fig. 3. Depiction of set filtering for the Table I subscription set in a small network of 6 nodes, when subscriptions appear sequentially at one node and sensors are places at the other side of the network. Node data structures reflect the forwarding, splitting and placement of operators after s_1 (left), after s_2 (center) and after s_3 (right).

complete set of filters given by a user, or as filters subsets. We refer to a (sub)set of filters as a correlation operator, because it requires data over different streams and must correlate them on time (and maybe also location). When such an operator is addressing a single attribute, we call it a simple operator, and it does not suffer further splitting on its way to the matching sensor(s), though it might be filtered out along the way.

Algorithm 3 split_and_forward(s, m): split and forward at node n, subscription s coming from node m

```
1:  /* If there are matching sources for s, split and forward towards
    sources' nodes*/
2:  if n == m then //s from local users
3:      if matching_sources(s,neighbor(n)) = false then //absent
        sources
4:          return
5:      end if
6:  end if
7:  for all j ∈ neighbor(n) \ {m} do
8:      operator = project(s, j)
9:      send_subscription(operator)
10: end for
```

Subscription forwarding makes sense only if the subscription can be answered by the existing data sources. In Algorithm 3, we perform this check only at the originating node, where new subscriptions are first checked against the stored data sources advertisements, and subsequently dropped if some source is missing (line 3). Only if all sources are present, or the subscription (or operator) came from a neighbor, which means its sources were already checked, must the subscription be forwarded to the node's neighbors, according to the matching source advertisements' reverse dissemination path. One can imagine this operation as a projection of the subscription on the neighbor's data space, as defined by its advertisements (line 8). Finally, for each neighbor with matching advertisements, the resulting projected operator is forwarded (line 9).

Algorithm 4 Subscription propagation

```
1:  /* Run each time a new subscription s appears at node n*/
2:  if receive("subscription", s) at node n from node m then
3:      if n == m then //s from local users
4:          S = S^local //all local subscriptions
5:      else
6:          S = S^m //all subscriptions from m
7:      end if
8:      if filter(s, S_uncovered) = false then
9:          append_to(S_uncovered, s)
10:         split_and_forward(s, m)
11:     else
12:         append_to(S_covered, s) //set of covered subscriptions
13:     end if
14: end if
```

Subscription Propagation Overview: Subscription propagation is described in Algorithm 4, where both the filtering and the split and forward procedures are depicted as black boxes. Incoming subscriptions are the filtering set, whether they are from local users (line 4) or a given neighbor (line 6). Furthermore, only the previously uncovered subscriptions (line 8) need to be considered, because the covered subscriptions are redundant to the filtering problem. Both covered and uncovered subscriptions must be stored: even though only uncovered subscriptions are candidates for forwarding to neighbors, all subscriptions define the correlations needs of the neighbors or local users. User subscriptions that were not filtered out by the **filter** procedure might require some other information than existing subscriptions and must be forwarded in the system towards the matching data sources. If these data sources have different paths, the subscription is correspondingly divided, and subsequent filtering and splitting will eventually determine whether it is subsumed.

Looking back at our subscription set example, we detail in Figure 3 the processing of each subscription, and the state of the system after each processing is completed. Near each node we put the operators it stores, and near each link, the forwarded operators, while the forwarding sense is depicted by an arrow. Figure 3 (left) illustrates the system's state after s_1 is processed, (center) after s_2 is processed, and (right) after s_3 is processed. The gain in memory space, traffic and processing can be immediately observed, as no further storage, processing and forwarding is done in relation to s_3 from n_2 towards the sensor nodes.

C. Event Propagation

A sensor node generates (simple) events each time there is a sensor reading. Since events can be very numerous, for efficiency reasons they should be forwarded in the system, only if there are interested users. This is reflected by the presence at a sensor node, of a subscription or correlation operator addressing the corresponding attribute. At each node, on the path from the publishing node to the matching users, an event is forwarded (or on the contrary, dropped) only if it is part of a complex event matching a correlation operator, or if it matches a simple operator. Furthermore, in accordance to the publish/subscribe paradigm, we ensure each simple event is forwarded only once over a network link.

Algorithm 5 Event-propagation

```
1:  /* Run each time a new event e = (a_d, p_d, v, t) appears at node
    n from m*/
2:  initialize(e.sendTo) //neighbors having received e
3:  t_insert(U, e) //timestamp ordered set of unexpired events
4:  T = t − δ_t
5:  X =t_correlated(U, T) //time correlated events, from T
6:  while not_empty(X) do
7:     for all j ∈ neighbor(n) \ {m} ∪ {n} do
8:        if new_events(X, j) then //X has simple events not seen
          by j
9:           S = S^j_uncovered //S = S^local for j == m
10:       end if
11:       for all s_k ∈ S do
12:          X_k = complex_match(X, s_k)
13:          if Y_k =new_events(X_k, j) then
14:             if (j == n) then
15:                send_complex_event(X_k,user(s_k))
16:             else
17:                send_events(Y_k, j) //also updates sendTo
18:             end if
19:          end if
20:       end for
21:    end for
22:    X =t_correlated(U,next(T)) //increase T to next timeunit
23: end while
```

Algorithm 5 presents the forwarding of events, how to detect and match a complex event. At each node, all received simple events are stored and indexed by their timestamps (line 3), to facilitate time correlation. Furthermore, each event has a corresponding array of flags (line 2: one flag per neighbor), tracking whether it was forwarded to neighbors, to ensure that no data unit is sent more than once to the same neighbor.

A potential complex event, containing the incoming simple event e, can only be present in a δ_t length sliding window over the received events set, with a start point set by the first simple event preceding e by δ_t (line 4). Each set of simple events, from an instance of the sliding window, must be checked against the set of uncovered subscriptions from each neighbor or against all local subscriptions (which are all whole), in order to construct the complex event users were looking for (line 9). For each subscription, the set of simple events is checked against the filters, and if at least one event per subscription attribute is still left (line 12), we know a matching complex event exists. Simple events from the matching complex events are sent to the neighbor having forwarded the subscription, if they have not been previously forwarded to that neighbor (line 13).

As an optimization, checking the set of subscriptions from a given neighbor can be stopped early, if all simple events in the current window instance have been forwarded to that neighbor. The overall procedure, launched for a new event, ends after having checked the last sliding window instance (line 22).

VI. Experiments

Our experimental study consists of the following algorithms:

- **Centralized approach**. Although we focus in this paper on efficient decentralized continuous query processing, we have implemented a fully centralized approach to compare against. Using the network topology, all subscribers forward their subscription queries on the shortest path to the central node (the node with the minimum pairwise distance to all other nodes). Sensors send their events in the same way to the central node which does the matching. Matching events will be sent on the shortest path from the central node to the owner of the matching subscription.
- **Naive approach**. This approach emphasizes the problem of high network load in multi-join query processing, as it forwards all received queries (no filtering) and constructs result sets per query (no optimization for result set overlap). It provides a baseline against which we check the proposed solutions.
- **Distributed operator placement**. This approach was described in Section III-A where we adapt operator placement techniques to a distributed setting without global knowledge. The traffic is reduced by sharing common operators (identical or covered) between queries and publish/subscribe forwarding of result sets.
- **Distributed multi-join processing**. This approach was described in Section III-B where we adapt the technique of splitting multi-joins into binary joins from [7] to a distributed setting: joins splitting and processing, as well as event processing are no longer performed on a centralized server, but distributed on the network nodes and driven by local interaction. The traffic is reduced by detecting coverage of multi-joins and binary-joins and publish/subscribe forwarding of events.
- **Filter-Split-Forward**. This is our novel approach and was described in Section V, where we detail how we process correlation operators, through efficient filtering, system state driven splitting and placement, all based strictly on local interaction. The traffic is reduced by detecting correlation operators subsumption and publish/subscribe forwarding of events.

TABLE II
IMPLEMENTED APPROACHES

Approach	*Subscription Filtering*	*Subscription Splitting*	*Event propagation*
Centralized	*None*	*None*	*Full result sets*
Naive	*None*	*Simple*	*Full result sets*
Operator placement	*Pair wise*	*Simple*	*Per subscription*
Multi joins	*Pair wise*	*Binary joins*	*Per neighbor*
Filter split forward	*Set filtering*	*Simple*	*Per neighbor*

Table II summarizes the differences between the five approaches, in terms of subscription filtering and splitting (and consequent placing of operators) and event processing, excluding the advertisement propagation, that does not happen for the centralized approach and is the same for all distributed approaches.

We have implemented all approaches in Java (JSE5.0) and have evaluated them in a distributed environment, on a cluster of quadcore 2.4GHz machines running Linux (Debian Etch). Each node in our system runs on a virtual machine, simulating a less powerful device, with 256MB RAM, and 1.5GB HDD. The virtual nodes were created with paravirtualization (Xen No HW emulation) at a ratio of 30 nodes per quadcore.

A. Network Settings and Data Set

In our experiments we replay a real data set over a network of nodes emulating a real setup, obtained through a sensor network deployment of EPFL's SensorScope team [6] which took place in September and October 2007 on the Grand St. Bernard pass. We have selected 5 measurement data types from this project, as attributes in our setting, and as data sources 10, respectively 20 base stations, depending on the number of sensors in each experiment. The selected attributes represent the following measurements: Ambient Temperature, Surface Temperature, Relative Humidity, Wind Speed, and Wind Direction. Subscriptions are generated by creating ranges over 5 attributes, centered around the median values in the corresponding stream, with an offset drawn from a Pareto distribution with a skew factor of 1. We target all locations with the same number of subscriptions, which we vary across each experiment.

We emulate the real deployment setup by grouping nodes with sensors from the same base station in a vicinity, such that they are neighbors. We vary the network size in our experiments from 60 to 200 nodes, out of which 50, or 100 have a sensor attached, evenly distributed over 5 attribute types. As such we replay 10, respectively 20 different base stations.

For each of the four experiment setups, we ensure that the four approaches are tested in the same network settings (localization of data sources, of subscriptions, network connection between nodes), that the subscription sets and subscription registration order ar the same, and, of course, we replay the same event sets. During an experiment, we only vary the number of user subscriptions, to check on the approaches' performance with increasing subscription (and user) load.

B. Metrics

For all experiments, we will focus on *subscription load* and *publication load*, which are the main contributors to the overall data traffic.

Subscription load reflects the entire load of subscriptions forwarded in the network, and increases every time an operator is forwarded to a neighboring node. It is influenced by the number of user subscriptions with matching data sources, by node connectivity and distribution, and also by the subscription propagation policy of each approach. Due to the different efficiency of the filtering algorithms, we expect that the naive approach, with no filtering, behaves the worst, followed by the operator placement and multi-join techniques, with filtering based on pair-wise coverage, while the filter-split-forward approach should incur the smallest subscription load, due to to set subsumption.

The key issue of complex events traffic is analyzed through the **publication load** metric, which quantifies the forwarding of result sets. This metric is influenced by the number of operators residing at each node, because each one generates a result set, but also by the efficiency of event forwarding techniques. Similarly to subscription load, the naive approach should behave the worst, and the operator placement technique should benefit from the smaller number of present subscriptions. The multi-join technique should perform better than operator placement due to the publish/subscribe mechanism of forwarding events, which compacts the overlapping result sets. However, since it inherently forwards false positives (events matching binary joins, but not the originating multi joins), it should perform worse than filter-split-forward.

Because we employ a probabilistic algorithm for set subsumption, we also quantify **event recall**. When subscription subsumptions are falsely detected, events matching such subscriptions will not arrive to the user, unless they match unsubsumed, overlapping subscriptions.

C. Small Scale Experiment

In the first experiment we model a small scale deployment of a network of 60 nodes, with 50 nodes replaying sensor data (10 groups comprised of 5 sensors, one for each data type). We vary the number of user subscriptions from 100 to 1000, and we measure the performance of each approach after every new batch of 100 subscriptions. Location-wise, subscriptions are evenly targeting the 10 groups, and their attribute set varies between 3 to 5 attributes. The attributes are chosen randomly, thus creating a mixture of attribute subsets, which should decrease subscription overlap.

As the number of user subscriptions increases, so does the overall subscription overlap. Operator placement techniques and the multi-joins approach can reduce the subscription set up to a certain point (Figure 4), based on pair-wise coverage, but our approach performs better (on average 18%), due to group subsumption, associated with the splitting phase of the algorithm. Set filtering can only compare between subscriptions

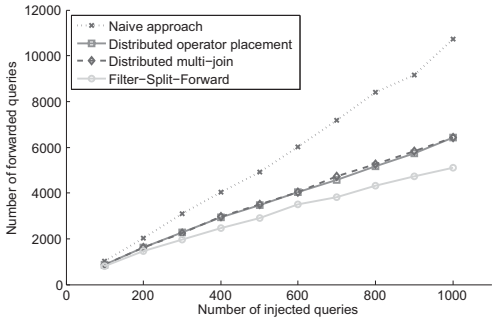

Fig. 4. Subscription load for the small scale experiment

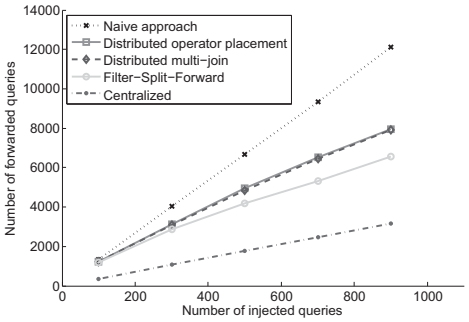

Fig. 6. Subscription load for the medium scale experiment

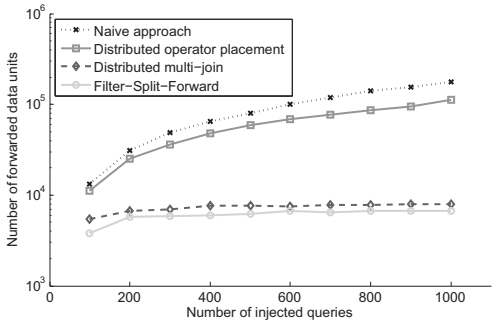

Fig. 5. Event load for the small scale experiment

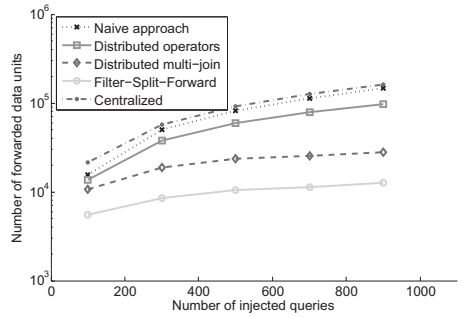

Fig. 7. Event load for the medium scale experiment

over the same subset of attributes and, since the subscriptions in this experiment have different attribute sets, there should be less subsumption than in cases where all subscriptions can be compared. The splitting phase actually allows the comparison between subscriptions over different (but overlapping) attribute sets, by comparing the simpler split operators and achieving reduction results similar to cases where all subscriptions are over the same attributes.

The *publication load* is depicted in Figure 5. The Filter-Split-Forward approach outperforms the operator placement method, due to a smaller number of forwarded operators and sharing of dissemination costs between overlapping operators. The performance is better (10% to 30%) than the multi-joins approach, due to the presence of fewer operators, and the fact that our approach does not introduce false positives. The reason why the improvement factor is not so big in this particular setting, is that event dissemination costs are almost similar for the two approaches, and for small number of attributes per subscription, binary-joins are good approximations of multi-joins, thus generating few false positives. In fact, binary joins are equivalent to multi-joins with two attributes, but become approximations for multi-joins over three attributes; the quality of the approximation degrades with increasing numbers of attributes.

D. Medium Scale Experiment

For the second experiment, we consider a network of 100 nodes, out of which 50 nodes are replaying sensor data (there are 10 groups comprised of 5 sensors, one for each data type). We vary the number of user subscriptions from 100 to 900 and we report the traffic load after each 100 subscriptions batch. Subscriptions are evenly targeting the 10 subnetworks, and have 5 attributes each, which is a realistic number of attributes, and is expected to further expose the weakness of the multi-join approach. We have chosen the medium size setting to compare against the centralized approach as it is a good representative and we have measured the same metrics. However, this does not reflect the extra strain the central node must support, as it is the only one processing the matching of events to subscriptions.

The subscription reduction in the second experiment (Figure 6) is similar for the distributed approaches to the first setting, establishing the same order, performance-wise, among these four approaches, with our approach outperforming state-of-the-art by 4.5% to 17.4%. For this setup we also look at the performance of the centralized approach which has by far the lowest subscription traffic. This is not a surprise as subscriptions are not forwarded (and split) all the way to the sensors, but only once to the central server, chosen to be exactly at the center of the network. However, it does not make up for the large drop in performance for the more important event traffic, as we will see in Figure 7. For the decentralized competitors, as before, operator placement and multi-join achieve more or less the same reduction in subscription traffic, because they use the same filtering technique (pair-wise covering detection). The slight differences come from the exact order in which subscriptions appear at nodes in the system: even though they are generated at the same intervals over all approaches, in each instance, nodes can be faster or slower in processing a new subscription or forwarding it over the next link, and we do not apply retroactive filtering (checking if a new subscription covers already forwarded subscriptions).

The publication load (Figure 7) is similar for the distributed

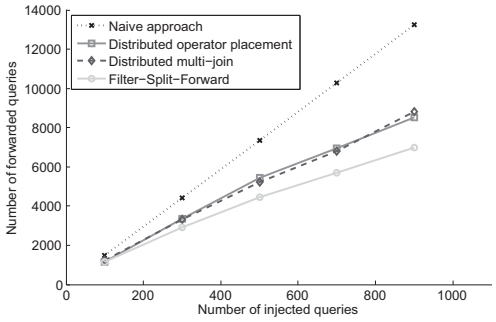

Fig. 8. Subscription load for the large (network) scale experiment

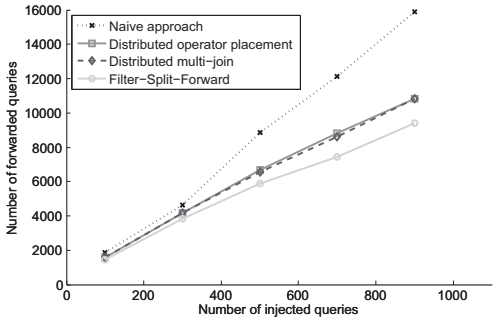

Fig. 10. Subscription load for the large (sources) scale experiment

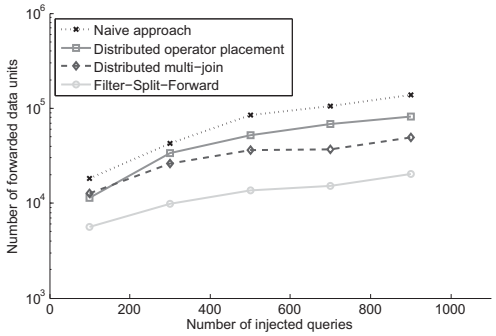

Fig. 9. Event load for the large (network) scale experiment

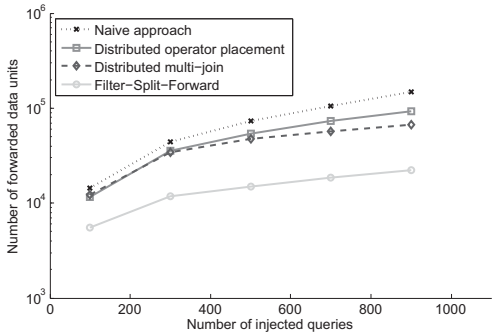

Fig. 11. Event load for the large (sources) scale experiment

approaches to the first experiment, however, our approach outperforms distributed multi-join by a wider margin (48% to 55.9%). This is due to binary joins approximations creating more false positives per multi-join than in the previous scenario, and also to forwarding them over more links, as the network is larger. Also, the event traffic for the centralized approach is the largest, and it largely outbalances the gains obtained by this approach for the less important subscription traffic. Its event traffic has a fixed component due to sending all events to the central node, and a variable component, for forwarding the results sets back to subscribers' original locations. The impact of the fixed component is more important the less events match subscriptions (highly selective subscriptions). In our scenario, we have chosen medium selective subscriptions, making sure each one has a minimum number of matching events, so we expect the centralized approach to deteriorate further compared to the distributed approaches for general sets of subscriptions.

E. Large Scale Experiments

In the third experiment, we want to analyze the influence of the network size, hence construct a network of 200 nodes, out of which 50 nodes are replaying sensor data. We vary the number of user subscriptions from 100 to 900. Subscriptions are evenly targeting the 10 subnetworks, and have 5 attributes. The same conclusions can be drawn as in the previous experiment, for the subscription load reduction (Figure 8), as the approaches' relative improvement factors are the same; only the total number of subscriptions increases, which is normal as the average path from a user to the matching data sources increases with the network size. The reduction in publication load (Figure 9) shows even wider margins (56% to 62%), thus proving that the network size has a bigger influence for the larger traffic of events.

In the fourth experiment, we analyze the influence of the number of distinct data sources. We construct a network of 200 nodes, out of which 100 nodes are replaying sensor data. We vary the number of user subscriptions from 100 to 900. Subscriptions have 5 attributes, and are evenly targeting the 20 subnetworks, which should again decrease the set reduction opportunities, because candidate subscription set sizes are smaller than in the previous experiment. This is confirmed in Figure 10. Nevertheless, the publication load (Figure 11) achieved by Filter-Split-Forward decreases even more than in the previous experiment when compared with the multi-join approach (54% to 68%), which confirms the importance of the filter-split-forward phases, independent of the network size and candidate subscription set size.

F. Event Recall

For each of the four experiment settings, we have also measured the user perceived *event recall* of our approach, to compare with the other approaches which are deterministic, and have by design perfect accuracy (i.e., 100% event recall). As can be seen in Figure 12, the Filter-Split-Forward approach reaches an excellent accuracy, given the major gain in performance, illustrated by the subscription and even load metrics. The measured accuracy is 100% in some cases, and generally around 98%. However, for the small scale experiment and the large scale experiment with small number of subscriptions, the recall is around 93%, which we think is also acceptable in most application scenarios.

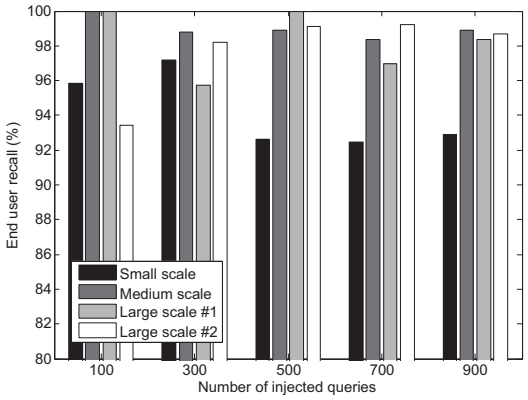

Fig. 12. End user event recall for the Filter-Split-Forward approach for all network settings

To achieve a higher recall, one can think of adapting the probabilistic filtering to be more conservative. In addition, in scenarios that require not to lose any event, the subscriptions can be made coarser, i.e., the specified filter ranges could be enlarged, requiring a final local filtering step. However, this would increase the number of forwarded events, and the overall traffic. Reducing either the traffic, either the number of missed events creates a tradeoff, upon which the user has to decide.

VII. Conclusion and Outlook

We have presented a general framework for processing publish/subscribe queries over distributed sensor networks. The requirements imposed on the underlying sensing infrastructure are almost negligible, hence, the integration of a large number of heterogeneous sensor networks under one common light-weight event processing model becomes possible. In our approach, multi-join queries are passed along communication links, from one node to other nodes, towards the data sources. We employ so called filter-split-forward phases for efficient subscription filtering and placement inside the network. In addition, we have tailored existing works in the area of operator placement and distributed multi-join processing to a distributed setting. We have conducted a performance evaluation by replaying a real data set to check the performance in traffic reduction achieved by the proposed and tailored distributed solutions for continuous query processing, compared to a baseline distributed approach and a centralized scheme. For low data rates, inexpensive data forwarding and providers sharing data, a centralized solution could work, too. Our approach is useful in such a setting as well: fewer subscriptions to check for event matching and efficient event delivery, if final users are reached by a distributed network. Also, if global knowledge is available, our approach can gain by either rewiring the network to connect sensors that frequently co-occur in user queries, if rewiring is possible, or placing the complex operators closer to the sensors with higher data rates. As future work, we will have a look at ranking batches of events, for more efficient event propagation, focusing only on the top-ranked items. This is in particular interesting for subscription queries posed by users with large numbers of matching events.

References

[1] D. J. Abadi, S. Madden, and W. Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. *VLDB*, 2005.

[2] K. Aberer, M. Hauswirth, and A. Salehi. A middleware for fast and flexible sensor network deployment. *VLDB*, 2006. Demonstration paper.

[3] Y. Ahmad and U. Çetintemel. Network-aware query processing for stream-based applications. *VLDB*, 2004.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(4), 2002.

[5] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. *Data-Stream Management: Processing High-Speed Data Streams*, chapter STREAM: The Stanford Data Stream Management System. Springer, 2006.

[6] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. Sensorscope: Out-of-the-box environmental monitoring. *IPSN*, 2008.

[7] B. Chandramouli and J. Yang. End-to-end support for joins in large-scale publish/subscribe systems. *VLDB 2008*, 2008.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. *CIDR*, 2003.

[9] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable Distributed Stream Processing. *CIDR*, 2003.

[10] P. Costa, G. P. Picco, and S. Rossetto. Publish-subscribe on sensor networks: A semi-probabilistic approach. *MASS*, 2005.

[11] M. Franklin, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design Considerations for High Fan-in Systems: The HiFi Approach. *CIDR*, 2005.

[12] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), 2003.

[13] C. Hall, A. Carzaniga, J. Rose, and A. Wolf. A content-based networking protocol for sensor networks. Technical Report CU-CS-979-04, Department of CS, University of Colorado, Aug. 2004.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. *MOBICOM*, 2000.

[15] A. M. Ouksel, O. Jurca, I. Podnar, and K. Aberer. Efficient Probabilistic Subsumption Checking for Content-Based Publish/Subscribe Systems. *Middleware*, 2006.

[16] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. *ICDE*, 2006.

[17] S. Rooney, D. Bauer, and P. Scotton. Techniques for Integrating Sensors into the Enterprise Network. *IEEE eTransactions on Network and Service Management*, 2(1), 2006.

[18] M. Sgroi, A. Wolisz, A. Sangiovanni-Vincentelli, and J. M. Rabaey. A service-based universal application interface for ad hoc wireless sensor and actuator networks. *Ambient Intelligence*. Springer Verlag, 2005.

[19] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Technical Report TR-21-04, Harvard University, EECS, 2004.

[20] E. Souto, G. Guimares, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal Ubiquitous Comput.*, 10(1):, 2005.

[21] D. Srivastava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence*, 8(3–4), 1992.

[22] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. *PODS*, 2005.

[23] Y. Xing, J.-H. Hwang, U. Çetintemel, and S. B. Zdonik. Providing resiliency to load variations in distributed stream processing. *VLDB*, 2006.

[24] Y. Xing, S. B. Zdonik, and J.-H. Hwang. Dynamic load distribution in the borealis stream processor. *ICDE*, 2005.

[25] Y. Yao and J. Gehrke. Query Processing in Sensor Networks. *CIDR*, 2003.

[26] Y. Zhou, K. L. Tan, and F. Yu. Leveraging distributed publish/subscribe systems for scalable stream query processing. *VLDB 2006 Workshop on Business Intelligence for the Real-Time Enterprise*, 2006.