Context-Aware Publish-Subscribe: Model, Implementation, and Evaluation

Gianpaolo Cugola, Alessandro Margara
Dip. di Elettronica e Informazione
Politecnico di Milano, Italy

Matteo Migliavacca
Department of Computing
Imperial College London, UK

Abstract

Complex communication patterns often need to take into account the situation in which the information to be communicated is produced or consumed. Publish-subscribe, and particularly its content-based incarnation, is often used to convey this information by encoding the "context" of the publisher into the published messages. In this paper we claim that this approach is limiting and inefficient and propose a context-aware publish-subscribe model of communication as a better alternative. We describe a protocol that implements such model in a distributed publish-subscribe middleware, and analyze how it performs w.r.t. traditional content-based routing.

1. Introduction

During the decade, the publish-subscribe paradigm of communication [1], and particularly its content-based incarnation [2], has shown its effectiveness in a large number of domains. The ability to address messages based on their content results in a strong decoupling among communicating parties, which provides a great flexibility in adapting the system architecture and the communication patterns to the various situations that applications may encounter.

In most domains in which content-based publish-subscribe finds its natural application, an effective communication paradigm requires to take into account the *situation* in which the information to be communicated is produced or consumed. As an example, the users of a wireless sensor network monitoring the temperature in a building could be interested in receiving periodic readings from those sensors whose battery is well-charged, while they could be satisfied of receiving only fire alarms from those sensors being low on batteries. Similarly, the announcement about a new product has to specify different prices for the different countries in which the interested receivers reside.

What these examples have in common is the fact that not only the informative content of messages is relevant to determine the information flow, but also the *context* in which this information has been produced and its relationship with the context of the consumer.

This need for *context-awareness* is so common in publish-subscribe applications that it is not unusual to see the context of the publisher encoded into the published messages, as a way to implement context-aware interactions by taking advantage of the expressiveness of content-based addressing. In this paper we argue that this approach leads to strong inefficiencies in routing messages from publishers to subscribers, particularly in those situation involving a large number of communicating parties and consequently a distributed dispatching system. Our claim is that it should be replaced by the adoption of a context-aware publish-subscribe model of communication.

In particular, in Section 2 we explain why content-based addressing alone cannot entirely answer the need for context-awareness typical of large-scale scenarios. In Section 3 we describe a new publish-subscribe model that is both content and context-based, together with a routing protocol to implement the new model in a distributed publish-subscribe system. In Section 4 we evaluate such protocol, as implemented in our publish-subscribe middleware REDS [3], measuring its effectiveness w.r.t. traditional content-based routing. Finally, in Section 5 we survey related work, providing some concluding remarks in Section 6.

2. Why a New Model

Consider a Fire Monitoring System (FMS) deployed in a large building, which consists of smoke detectors, light signals, and sprinklers. Whenever smoke is detected, an alert message is published, causing sprinklers at the same floor and within 30 meters from the smoke to automatically activate. Meanwhile, a planner computes an evacuation plan and publishes a message toward the signals at the east of the detected fire to let them display an eastbound arrow towards the eastern emergency exit. A similar message is published toward the signals located in the area at the west of the fire to direct people toward the western exit.

Despite its simplicity, this example highlights a characteristics that is common to several publish-subscribe applications: both the subscribing and publishing of messages are tied to the context in which producing and consuming entities are. Once we accept this, we might wonder whether a new communication paradigm that explicitly takes context into account (and the middleware system implementing it) is really required. Isn't content-based addressing enough?

978-1-4244-4671-1/09/$25.00 ©2009 IEEE

sprinklers	subscribe($msgType = Alert \wedge$ $myPos.x - 30 \leq x \leq myPos.x + 30 \wedge$ $myPos.y - 30 \leq y \leq myPos.y + 30$)
smoke detectors	publish($msgType = Alert$, $x = myPos.x$, $y = myPos.y$)
signals	subscribe($msgType = Evacuate* \wedge$ $x = myPos.x \wedge y = myPos.y$)
planner	subscribe($msgType = Alert$) publish($msgType = EvacuateL$, $x < fire.x$) publish($msgType = EvacuateR$, $x > fire.x$)

Table 1. The FMS example using traditional API

Smoke detectors might publish their location as part of the alarm message, while sprinklers might subscribe to messages generated in an area close enough to them. Similarly, light signals could include their location into the subscription used to receive activation messages, while the planner could publish activation messages that appropriately filter the interested signals based on their location.

At first this approach, which is summarized in Table 1, seems reasonable, but a closer look shows its weaknesses, especially if we consider large enough scenarios to require a distributed dispatching system.

Matching Inversion. In conventional content-based publish-subscribe systems messages hold data (usually encoded as key-value pairs), while subscriptions hold constraints on these data. If we look at Table 1 we notice that our solution to the FMS example violates this assumption. It requires the subscribers (i.e., the signals) to specify their location (a datum) into subscriptions, while the publisher (i.e., the planner) adds a constraint into messages to reach some signals and not others (see the last two rows of Table 1). This means that both the data model and the matching semantics of conventional publish-subscribe systems are unsuited for the case under consideration. We need to invert the conventional matching process to consider constraints embedded into messages and data embedded into subscriptions[1].

Efficiency. The second reason to add context into publish-subscribe is the efficiency that this solution can bring when a distributed dispatching system is used. Indeed, managing context explicitly enables a dispatching strategy that limits the spreading of subscriptions only to those areas of the routing network where matching publishers exist (i.e., those whose context satisfies the context filter specified by the subscriber). This (i) reduces the overhead of the subscription and unsubscription processes (saving bandwidth), and (ii) reduces the time required to match messages, thanks to smaller routing tables.

Separation of concerns. As a final issue, we observe that usually the components in charge of publishing messages and subscribing to them differ from those in charge of detecting and communicating context changes. As an example, a GPS controller could be in charge of notifying the location of a mobile node (i.e., its context), while the components in charge of subscribing and publishing are those that implement the node's application logic. Tying the two concepts together might reduce the readability of code, forcing complex interaction among parts of the application that should be kept separate.

1. The few content-based publish-subscribe systems that do not suffer of this problem are those adopting a Turing-complete language to implement filters, which however are hard to optimize [4]

sprinklers	subscribe($msgType = Alert$; $myPos.x - 30 \leq x \leq myPos.x + 30 \wedge$ $myPos.y - 30 \leq y \leq myPos.y + 30$)
smoke detectors	setContext($\{x = myPos.x, y = myPos.y\}$), publish($\{msgType = Alert, x = myPos.x, y = myPos.y\}$; ALL)
signals	setContext($\{x = myPos.x, y = myPos.y\}$), subscribe($msgType = Evacuate*$; ALL)
planner	subscribe($msgType = Alert$; ALL) publish($\{msgType = EvacuateL\}$; $x < fire.x$) publish($\{msgType = EvacuateR\}$; $x > fire.x$)

Table 2. The FMS example using Context-Aware API

3. API and Routing

To overcome the limitations above, we propose to introduce context as a first class element into the publish-subscribe API. In particular, we allow each node n to set its current context by invoking the $setContext(c)$ operation. Additionally, n can subscribe to messages matching the *content filter* f_{msg} and coming from publishers whose context matches the *context filter* f_{ctx} through the $subscribe(f_{msg}; f_{ctx})$ operation, while the $unsubscribe(f_{msg}; f_{ctx})$ operation does the opposite. Finally, n can publish messages for subscribers whose context matches the context filter f_{ctx} by invoking the $publish(m; f_{ctx})$ operation. Table 2 shows how the Fire Monitoring System example can be easily implemented with these context-aware publish-subscribe primitives.

3.1. The SPCF Protocol

To support large scale scenarios that involve hundreds of nodes, we developed *SPCF*, a protocol defining how a set of brokers connected in an overlay network should cooperate in order to efficiently provide the context-aware publish-subscribe service above to their *clients* (see Figure 2).

As its name suggests, SPCF adopts a *Shortest Path Context Forwarding* approach: messages are forwarded along the shortest path tree rooted at the publisher, using information about the context and interests of downstream clients to decide the branches to follow and those to prune.

More specifically, each broker runs a link state protocol [5] to build its own view of the dispatching network[2] and calculates, using a local algorithm like Dijkstra or Floyd-Warshall, the *shortest path trees* (SPTs) rooted at each

2. Not considering the clients, which are not relevant in this phase.

Context table	
broker id	$\{c_1, ..., c_n\}$

Content table			
broker id	context	neighbor id	$\{(f_{msg_1}, c_1), ..., (f_{msg_n}, c_n)\}$

Figure 1. Tables kept by each broker.

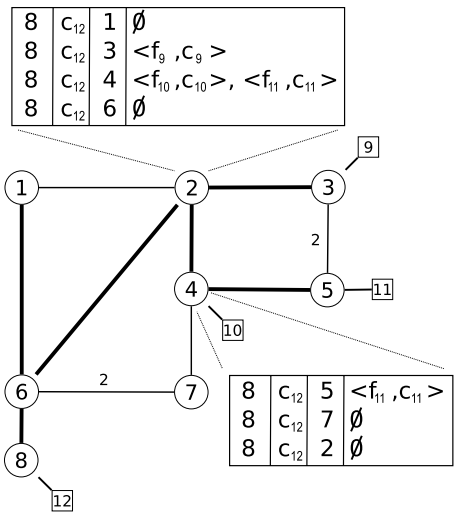


Figure 2. A dispatching network with eight brokers (the circles) and four clients (the squares).

broker in the network. Message forwarding uses these trees together with two tables, which are kept by each broker (see Figure 1): a *context table* and a *content table*. The former maps brokers (i.e., their identifiers) to the set of contexts of their clients. The latter stores, for each other broker B_p, each context c_p among those of the clients attached to B_p, and each neighbor N, the set of content filters and contexts coming from clients attached to brokers that are downstream along N in the SPT rooted at B_p.

3.1.1. Forwarding. To explain how SPTs plus context and content tables are used to forward messages, we use the example in Figure 2. It shows a publish-subscribe network with eight brokers and four clients, among which only clients 9, 10, and 11 subscribed, each issuing a single subscription. The cost of links connecting brokers is shown when greater than one, while the SPT rooted at broker 8 is shown using thick lines. Finally, the figure shows part of the content tables of brokers 2 and 4: the part concerning broker 8.

When client 12 with context c_{12} invokes the $publish(m; f_{ctx})$ operation, broker 8 matches the context filter f_{ctx} against the different contexts that are part of its context table and encodes the matching ones in a bloom filter b. Afterwards, broker 8 builds a packet $(8, c_{12}, m, f_{ctx}, b)$ and routes it along its own SPT. When the packet reaches broker 2, the first two fields are used to isolate the relevant part of the content table (the part shown in figure), then a "standard" content matching is performed to find the neighbors that have to receive the packet. At this step, the bloom filter b is used to reduce the number of content filters to consider, i.e., only those associated with contexts that are part of the bloom filter. As an example, if we suppose that m matches both f_9 and f_{10} while b includes c_{10} but not c_9 (i.e., at broker 8 f_{ctx} has matched c_{10} but not c_9) then the packet is forwarded only toward broker 4. There the message is first forwarded to the local client 10[3], then the same forwarding algorithm above is run. In particular, if we suppose that either f_{11} does not match m or c_{11} is not part of b the forwarding stops, otherwise the packet is sent to broker 5.

3.1.2. Routing. We describe here how the context and content tables are built and maintained. To do so we start describing how subscriptions are managed.

When a client s with context c_s invokes the $subscribe(f_{msg_s}; f_{ctx_s})$ operation, the broker B_s it is attached to operates as follow:

1) it records the subscription in a client table (omitted in the discussion so far since it is used only to deliver messages to clients at the last hop);
2) it determines the set D of brokers that must receive the subscription by matching f_{ctx_s} against the contexts in its context table;
3) for each neighbor N, it calculates the subset D_N of D that includes only those brokers whose SPT has N as the parent node of B_s;
4) if $D_N \neq \emptyset$ it forwards a packet $(B_s, c_s, f_{msg_s}, f_{ctx_s}, D_N)$ toward N.

When a broker N receives such packet it updates its content table by adding the pair $\langle f_{msg_s}, c_s \rangle$ to all the rows tagged as (B_x, c_x, N_x) where $B_x \in D_N$, c_x is one of the contexts associated with B_x that also matches f_{ctx_s}, and N_x is the broker from which the packet arrived (i.e., B_s at the first hop). Then N forwards the packet by repeating the steps 3 and 4 above (this time using D_N as the initial set to partition).

Unsubscriptions are managed similarly, with the only difference that content filters are removed from instead of being added to the content tables.

The last point to describe is how context changes are managed. When a client n attached to B_n changes its context from c_n to c'_n by invoking the operation $setContext(c'_n)$, two things must happen: (i) the subscriptions of n stored in the various content tables must be changed since they recorded the "old" context; and (ii) the set of contexts associated with B_n in the context tables around the network must be changed, possibly attracting new subscriptions and removing existing ones.

3. To avoid the false positives potentially resulting from the use of bloom filters, at this step the context of clients potentially interested in the message is actually matched against the context filter f_{ctx} issued by the publisher.

The first step is easily managed by letting B_n unsubscribe from all the subscriptions previously issued by its client n and resubscribing with the new context. In practice, this can be done by repeating the protocol described above.

The second step is more complex and requires a new protocol. In particular, when n changes its context from c_n to c'_n three cases may happen: (i) the set of contexts associated with the broker B_n remains the same; (ii) c'_n must be added to the set of contexts associated with B_n and c_n be removed (because no other B_n client has c_n as its context); (iii) c'_n must be added and c_n must not be removed. This suggests to separately manage the actions of adding a new context and that of removing an existing one.

Removing a context c_n from the set of contexts associated with a broker B_n is simple: B_n builds a packet holding its identifier and the context c_n and routes it along its own SPT. Each broker processes this packet by updating the context table (removing c_n from the set of contexts associated with B_n) and removing the entries of the content table labelled with the pair (B_n, c_n). Indeed, these are subscriptions that reached B_n because of c_n and must be removed.

Adding a new context c'_n requires a similar processing, complicated by the fact that the new context must "attract" matching subscriptions that were not forwarded before. Each broker along the SPT rooted at B_n processes the packet to add the new context c'_n by updating the context table and reissuing those subscriptions previously sent by one of its clients, whose context filter matches c'_n.

As for the format of messages, contexts, and filters we notice that in principle the SPCF protocol is independent of them. On the other hand, to implement SPCF in a real middleware we had to choose one and in fact we adopted the most common in publish-subscribe middleware, which also allows for fast matching. It uses key-value pairs for messages and context descriptors and boolean predicates for filters.

4. Evaluation

To test the effectiveness of the SPCF protocol in real world scenarios we implemented it into the new version 2.0 of our middleware REDS [3] and used the Emulab [6] facility to test it. Since we were interested in comparing SPCF with traditional content-based routing protocols, we choose two of them and implemented both in REDS:

- ASF (Acyclic Subscription Forwarding) performs content-based routing on an acyclic topology by flooding subscriptions and routing messages only toward the interested clients. It is probably the most common approach adopted by distributed publish-subscribe middleware (e.g., see Siena [7]);
- GSF (Graph Subscription Forwarding) forwards messages along the SPT of the publisher, pruned by using subscriptions as done by SPCF. This approach is used by some advanced publish-subscribe systems like XNet [8] as it better exploits the network topology.

To encode our context-aware model in a purely content-based protocol, we used the approach suggested in Table 1. To solve the "matching inversion" problem we used the content-based routing protocol to transport messages ignoring the context filter specified by the publisher, matching it at the last broker only.

We consider a network composed of 20 brokers, each running on a different Emulab node. The overlay connects each broker with six others. 100 clients run on separate nodes and connect to brokers. Each client has 30 subscriptions each composed of several constraints on different attributes, for a total of about 10000 different constraints populating our tables (the number changes in the different scenarios we considered).

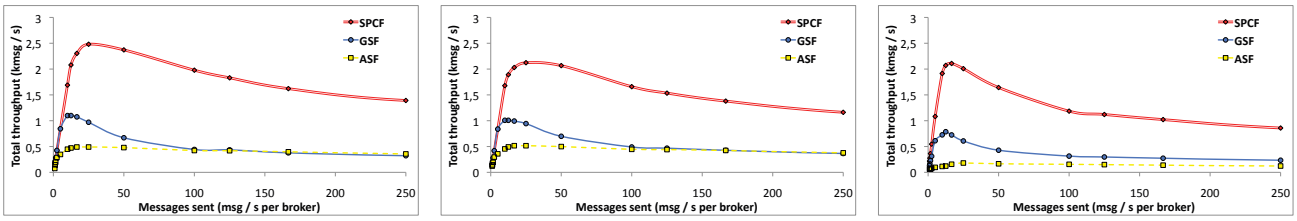
4.1. Forwarding

To study the impact of performing context-matching while forwarding messages we measured the throughput of the different systems, separately evaluating the impact of adding context filters in subscriptions from that of adding them in messages.

To investigate how SPCF behaves while changing the selectivity of filters (i.e. the percentage of clients a message has to be delivered to), we built up two different scenarios. The first maintains a fixed selectivity (of about 10%) for the overall system, decreasing the selectivity of context filters while correspondingly increasing that of content filters. The second keeps the selectivity of content filters fixed (at about 10%) while decreasing the selectivity of context filters. All experiments were performed using 100 clients (each with its own context) equally distributed between brokers.

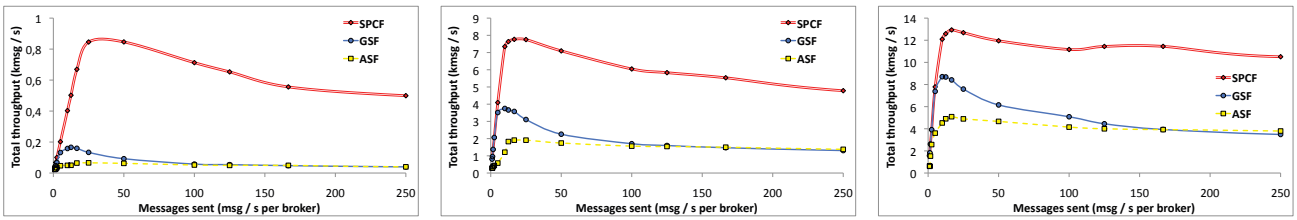
4.1.1. Context Filters in Subscriptions. As mentioned in Section 2, adding context filters to subscriptions does not add expressiveness to the model, but it allows for a more efficient forwarding since subscription tables are smaller and they include the content-filters only. This is confirmed by our tests, whose results are presented in Figure 3. SPCF has a much better throughput w.r.t. ASF and GSF in all scenarios. As expected, when the selectivity of context filters is low (i.e., each of them selects a large fraction of clients – around 80% in our tests) the differences between the three protocols decrease, with SPCF's subscription tables growing and approaching in size those of the other two protocols.

4.1.2. Context Filters in Publications. Adding context filters in publications allows publishers to filter out some subscribers and not others. Pure content-based protocols, like ASF and GSF, may perform such filtering at the last broker only, just before delivering messages to clients. This approach may result in misrouting messages toward areas



High context selectivity | Medium context selectivity | Low context selectivity

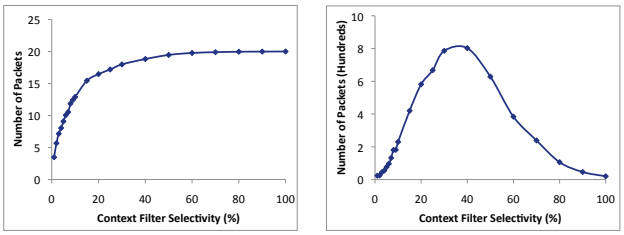
(a) Overall selectivity fixed



High context selectivity | Medium context selectivity | Low context selectivity

(b) Content selectivity fixed

Figure 3. Forwarding performance: Context filters in subscriptions



(a) *subscribe* (b) *setContext*

Figure 5. Cost of routing

of the network where the only subscribers are those whose context does not match the context filter specified by the publisher. SPCF does not suffer of this problem since it performs context filtering while routing. At the same time, this filtering has a cost that may reduce the advantage coming from not having misrouted messages. Figure 4 shows exactly this phenomenon, SPCF always performs better than traditional content-based routing but its advantage decreases as the selectivity of context filters becomes lower.

4.2. Routing

To analyze the cost of routing for SPCF we took our test network and measured the traffic generated by a single call to the *subscribe* and *setContext* operations[4] for different selectivity of the context filters, measured as the (average) number of contexts matched by each context filter.

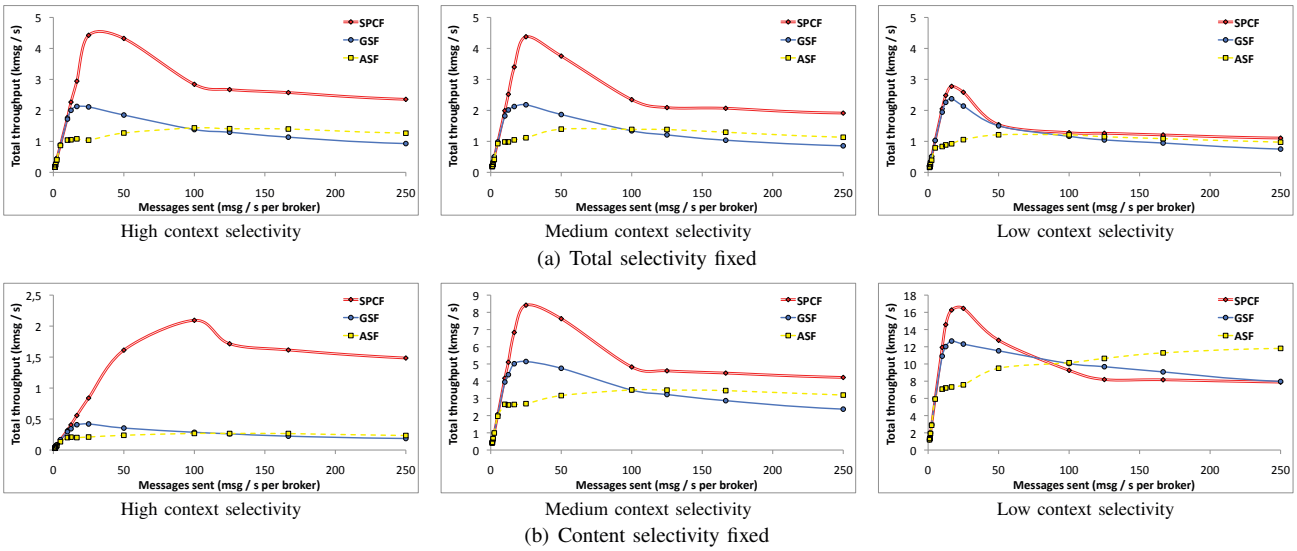
Figure 5(a) shows the cost for propagating a single subscription, which is lower when the context filters are very selective and consequently SPCF may propagate subscriptions to a smaller part of the dispatching network. When selectivity decreases (i.e. the percentage of selected contexts increases) SPCF approaches the behavior of ASF and GSF, which require 20 packets to flood the entire network.

Figure 5(b) analyzes what happens in our test network, with around 3000 subscriptions already deployed (30 for each of the 100 clients running), when a new client invokes the *setContext* operation. As we explained in Section 3, the new context has to attract, toward the invoking client, all the matching subscriptions that had been filtered out before. The resulting traffic is low when the selectivity of context filters is high (a few subscriptions match the new context), then increases up to a certain point, to decrease again when context filters become less selective. In the latter case, indeed, even if a lot of subscriptions match the new context there are good chances that they also matched an already existing context, having already travelled up to the broker close to the new client. When the selectivity is null (context filters match 100% of the contexts) SPCF performs as ASF and GSF apart from the cost of routing context information (20 packets).

4.3. Summing Up

Overall, all the tests we run (including those we could not report here due to the limited space available) confirm that the context forwarding approach taken by SPCF provides the best results when: (i) the selectivity of context filters is high; and (ii) the information about the contexts of clients remains stable w.r.t. the changes that occur to their interests (i.e., the number of *setContext* is lower than the number of *subscribe* and *unsubscribe*).

4. The *unsubscribe* performs similarly to the *subscribe*.



High context selectivity | Medium context selectivity | Low context selectivity

(a) Total selectivity fixed

High context selectivity | Medium context selectivity | Low context selectivity

(b) Content selectivity fixed

Figure 4. Forwarding performance: Context filters in publications

As a final remark, we also observe that these results where obtained in the worst possible situation for SPCF, in which context filters select clients uniformly w.r.t. the network topology (i.e., there is no "locality" in the system). In real scenarios we may expect that clients with a similar context tend to be co-located and thus attach to the same broker or to close ones. In this case, the possibility of leveraging context information to reduce the areas of the network reached by subscriptions has an impact even greater than the (already very good one) we were able to measure.

5. Related work

The last ten years have seen the development of a large number of content-based publish-subscribe systems [1], [2], [9], [10] first exploiting a centralized dispatcher, then moving to distributed solutions for improved scalability.

Besides their differences, most of the systems available today share the same model of communication. In particular, all of them put the filtering ability in the hands of subscribers, while publishers do not have any possibility of adding "filters" into their messages. This results in the inversion of matching problem we identified in Section 2, which limits the expressiveness of the system, making it impossible to implement the model of communication we have in mind on top of them. The only systems that do not have this limitation are those (e.g., see [3] and [11]), which allow content filters to be expressed as executable code. On the other hand, executable code is not widely used in practice because the resulting filters are hard to optimize.

A first approach that goes in the direction of our model is the intentional naming system proposed in [12]. In this model a server announces its services with an "intentional" name, which encodes its properties, while a client addresses a message to a server with a query that species the desired properties of the services he is interested in. This approach can be seen as the reverse of publish-subscribe, in that it is the subscriber (i.e., the server) who specifies some piece of data, which is then matched by the filter provided by the publisher (i.e., the client). This results in a kind of inversion of matching problem that is specular to that found in publish-subscribe systems, if we were trying to use an intentional naming system to implement our model.

A nice step in the direction of our context-aware model is represented by the "symmetric" publish-subscribe model [13]. This work breaks the asymmetry of both traditional content-based publish-subscribe and intentional naming systems by merging them together. Indeed, both messages and subscriptions in symmetric publish-subscribe are specified through constraints, a matching being an "intersection" between the two. While elegant, this approach mix context and content parts, which makes it impossible to exploit the kind of optimizations put forth in this paper.

While symmetric publish-subscribe is more general, *scoping* [14] can instead be seen as a special case of context-aware publish-subscribe; however the authors concentrate on the model, without tackling efficiency issues.

A different path toward context-aware publish-subscribe system is represented by those systems that implement some form of location-aware publish-subscribe, e.g. see [15] which also contains a survey of previous work, the coeval [16] and the more recent [17]. These proposals, allowing subscriptions to filter messages based on the location of the publisher can also be seen as a special case of our model. Accordingly, not only their model focuses on location aspects but also their implementation mechanisms do, thus

being inapplicable to our general case.

Interestingly [17] present instead a generic context-aware publish-subscribe model, which is a superset of the one hereby presented, featuring, in addition to filters on publisher's and subscriber's context, (called publication and subscription domains), a "context of relevance" for publications and a "context of interest" for subscriptions which match positively when they overlap (analogously to symmetric publish subscribe). However the actual protocol implementation of the model is grounded in a location interpretation of the context, focusing on computing efficiently, by geocasting, the relevance-interest intersection in a mobile MANET setting, which is the target of that work.

6. Conclusion

In most scenarios in which publish-subscribe is used, the context, being that of the publisher or that of the subscriber, would be a useful information, if available, to limit the scope of communication. Content-based publish-subscribe is a very expressive model of communication but it cannot entirely capture truly context-aware communications patterns. To overcome this limitation, we proposed a context-aware extension to the publish-subscribe model and a protocol to efficiently implement it in a distributed publish-subscribe system. Our tests with a publish-subscribe middleware that implements such protocols shows that it outperforms traditional content-based routing approaches.

Acknowledgments

This work was partially supported by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom; and by the Italian Government under the projects FIRB INSYEME and PRIN D-ASAP.

References

[1] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed Event-Based Systems*. Springer, 2006.

[2] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surveys*, vol. 2, no. 35, June 2003.

[3] G. Cugola and G. Picco, "REDS: A Reconfigurable Dispatching System," in *Proc. of the 6th Int. Workshop on Softw. Eng. and Middleware. (SEM06)*. Portland: ACM Press, nov 2006, pp. 9—16, available at www.elet.polimi.it/upload/cugola.

[4] G. Mühl and L. Fiege, "Supporting covering and merging in content-based publish/subscribe systems: Beyond name/value pairs," *IEEE Distributed Syst. Online (DSOnline)*, vol. 2, no. 7, 2001.

[5] J. McQuillan, I. Richer, and E. Rosen, "The new routing algorithm for the arpanet," *IEEE Trans. on Comm.*, vol. 28, no. 5, pp. 711–719, 1980.

[6] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*. Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[7] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Trans. on Comp. Syst.*, vol. 19, no. 3, pp. 332–383, 2001. [Online]. Available: citeseer.nj.nec.com/482106.html

[8] R. Chand and P. Felber, "A scalable protocol for content-based routing in overlay networks," in *Proc. of the 2nd IEEE Int. Symp. on Netw. Comput. and Appl.* Cambridge, MA: IEEE Computer Society, April 2003, pp. 123–130.

[9] R. Baldoni and A. Virgillito, "Distributed event routing in publish/subscribe communication systems: a survey," DIS, Università di Roma "La Sapienza", Tech. Rep., 2005.

[10] G. Mühl, L. Fiege, F. Gartner, and A. Buchmann, "Evaluating advanced routing algorithms for content-based publish/subscribe systems," in *Proc. of the* 10^{th} *IEEE Int. Symp. on Modeling, Analysis, and Simulation of Comput. and Telecommunications Syst. (MASCOTS02)*, 2002.

[11] P.Eugster and R. Guerraoui, "Content-based publish/subscribe with structural reflection," in *Proc. of the 6th Usenix Conf. on Object-Oriented Technol. and Syst. (COOTS01)*. San Antonio, Texas: USENIX Association, 2001.

[12] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proc. of the 17th ACM Symp. on Operating Syst. Principles (SOSP99)*. ACM Press, Dec 1999, pp. 186–201.

[13] W. Rjaibi, K. Dittrich, and D. Jaepel, "Event matching in symmetric subscription systems," in *Proc. of the 2002 Conf. of the Centre for Adv. Studies on Collaborative Research*. Toronto, Ontario, Canada: IBM Press, 2002.

[14] L. Fiege, M. Mezini, G. Mühl, and A. Buchmann, "Engineering Event-based Systems with Scopes," in *Proc. of the 16th Europ. Conf. on Object-Orient. Prog. (ECOOP02)*, ser. LNCS, vol. 2374. Springer, June 2002, pp. 309–333.

[15] G. Cugola and J. M. de Cote, "On introducing location awareness in publish-subscribe middleware," in *Proc. of the 4th Int. Workshop on Distributed Event-Based Systems*, June, Ed., Columbus, Ohio, USA 2005.

[16] T. Sivaharan, G. Blair, and G. Coulson, "Green: A configurable and re-configurable publish-subscribe middleware for pervasive computing," in *OTM 2005: CoopIS, DOA, and ODBASE*, ser. LNCS 3760. Springer, 2005, pp. 732–749.

[17] D. Frey and G.-C. Roman, "Context-aware publish subscribe in mobile ad hoc networks," in *Proc. of the 9th Int. Conf. on Coord. Models and Lang.*, ser. LNCS, vol. 4467. Springer, 2007, pp. 37–55.