State-of-the-art in Publish/Subscribe Middleware for Supporting Mobility

Sumant Tambe
ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA
sutambe@dre.vanderbilt.edu

Abstract

Publish/subscribe is a widely used middleware architecture as it proliferates loose coupling between interacting components. It is particularly useful in mobile environment because communication in publish/subscribe is anonymous, asynchronous, and inherently multicasting in nature. In spite of these features, the requirements of mobile applications are hard to satisfy using publish/subscribe middleware for static systems. Frequent connections/disconnections, scarcity of bandwidth and power, and frequent changes in location are the major challenges in mobile environment. Several researchers have proposed solutions to address these challenges to build scalable publish/subscribe middleware in resource constrained environment. In this paper, we review the requirements of the mobile applications, survey the state-of-the-art of research that address some of the challenges in development and verification of publish/subscribe systems and discuss the open research issues that remain in the domain.

General Terms Mobility Support, Publish/Subscribe, Middleware, Location Awareness

Keywords Publish/Subscribe, Mobility, Middleware

1. Introduction

The traditional request/response paradigm, widely advocated as the distributed computing paradigm of choice, is not adequate for several different information dissemination applications such as stock price tracker, sports and news tracker, alert systems, pervasive computing, and autonomic systems. The request/response paradigm is not able satisfy the requirements of these applications because of loose coupling, high level of dynamism and flexibility required by these applications. Publish/subscribe paradigm was developed to address these challenges.

A *publish/subscribe* system connects together information providers and consumers by delivering events from sources to interested users. A publisher *advertises* the types of events it is going to publish. A user expresses his/her interest in receiving certain types of events by submitting a predicate defined on the event, called the users' *subscription*. When a new event is generated and published to the system, the publish/ subscribe infrastructure is responsible for checking the event against all current subscriptions and *dispatching* it to all users whose subscriptions match the event.

The publish/subscribe communication paradigm differs from traditional client-server model in a number of ways. In publish/subscribe, communication is *anonymous*, *asynchronous*, *implicit*, and *multicasting* in nature. Anonymity means that the communication partners are not required to identify the party they want to talk to. For example, instead of identifying a publisher to receive events from, the subscriber simply describes the characteristics of the events it wants to receive. The system is therefore able to quickly adapt in a dynamic environment as publishers come and go. Publish/subscribe is also inherently asynchronous because the sender (publisher) does not have to wait for an acknowledgment from the recipient (subscriber) before moving on. The publisher does not identify who the intended receivers are. The subscribers are implicitly selected based on their subscriptions. It is possible that sender publishes events when there are no subscribers. Such events are discarded by the infrastructure. The reliable transmission of events to the subscribers is taken care of by the infrastructure. Publish/subscribe resembles multicast because it allows a publisher to send the same event to many subscribers with only one publish operation.

These features of publish/subscribe architecture make them particularly useful in mobile environment. The anonymity and dynamism of publish/subscribe allow the systems to adapt quickly to frequent connections and disconnections of mobile nodes, a characteristic of a mobile network. Asynchrony is helpful because mobile devices are often turned off

or disconnected from the network for long periods of time. Wireless devices have limited capabilities and bandwidth. The multicasting nature of publish/subscribe can leverage the broadcast radio nature of many wireless technologies and helps a system scale to thousands of units.

In spite of the suitability of publish/subscribe paradigm to mobile applications, there are several challenges that emerge in mobile environment. Quite often mobile nodes leave a network at one place and reconnect again at another location. Mobility of clients must to be taken into account by the middleware. ToPSS [3] and JEDI [3] address different aspects of the mobility challenge including the problem of frequent connections and disconnections. In order to support existing applications on mobile platforms, *location transparency* must be provided by the underlying middleware. REBECA [6] platform provides location transparency in the context of physical mobility. In some extreme environments (*e.g.,* disaster recovery, war field) there is hardly any fixed network support available to host the infrastructure. The middleware suitable for applications running on ad-hoc networking technologies needs to be aware of such highly dynamic environments. Huang and Molina [8] describe how resource friendly dispatching trees can be constructed in highly dynamic environment of ad-hoc networks.

The rest of the paper is organized as follows. Section 2 gives some basic background and terminology of publish/subscribe middleware domain. In Section 3 we analyze the requirements of mobile computing as they pertain to the publish/subscribe paradigm. In Section 4 we discuss the solutions that are proposed in current literature. In Section 5, future directions for research are given and finally, Section 6 concludes the paper.

2. Publish/Subscribe Paradigm

In this section, comparison between publish/subscribe paradigm with other dominant communication paradigms is presented. It is also shown that publish/subscribe architecture is well suited for *identity*, *space* and *synchronization* decoupling of subscribers and publishers. We categorize existing publish/subscribe system design approaches along two important dimensions: expressiveness of subscription languages, and centralized/distributed infrastructure.

2.1 Alternative Communication Paradigms

Eugster et al. [4] compare publish/subscribe paradigm with alternative communication paradigms: *Message Passing*, *Remote Invocation*, *Notification*, *Shared Spaces*, and finally, *Message Queuing*. Message passing is the most fundamental communication primitive based on which all the distributed systems are developed. Remote Invocation builds upon message passing and gives the impression of local function call where in fact the the execution happens at a remote location. Notifications are used when most of the communication is done using remote invocation style but when synchronization decoupling is desired, a synchronous remote invocation is split into a one way forward invocation coupled with a callback from the remote site later on. Shared spaces is a simple and powerful abstraction for accessing shared memory. A shared space is composed of a collection of ordered tuples and communication between two hosts takes place through insertion and removal of tuples to/from the shared space. Finally, message queuing is an abstraction that is closely related to publish/subscribe paradigm. In message queuing a central FIFO queue is maintained to synchronize the communication between producers and consumers. Table 1 highlights the main differences between various communication paradigms with examples.

2.2 Variations of Publish/Subscribe Paradigm

Eugster et al. [4, 5] and Carzaniga et al., [2] categorize publish/subscribe schemes depending upon the expressive power of the subscription languages. *Channel-based*, *subject-based* (topic-based), *type-based*, *content-based*, *content-based with patterns*, and *local context-based* publish/subscribe schemes, ordered in the increasing order of expressiveness, have been developed.

- In channel-based [2] model of event notification, notifications are fed into what amounts to a discrete communication pipe. The notion of filtering reduces to channel selection.
- In subject-based [4] (topic-based) notification scheme, recipients subscribe to one or more topics identified by *keywords*. Events belonging to a topic are dispatched to the interested recipients based on matching subscriptions.
- In type-based [4] notification scheme, events are filtered according to their programming language level types. This enables closer integration of the language with the middleware. Moreover, type safety can be achieved at compile time.
- Content-based [3] scheme is by far the most popular scheme of designing publish/subscribe infrastructure. Thanks to its flexibility and expressive power. A predicate on the contents of the events is given with the subscription and the infrastructure delivers events to consumers if the given predicate matches an event. Events are usually structured as unordered sets of typed attribute-value pairs.
- Content-based with patterns [2] scheme is a generalization of the content-based scheme wherein the subscriber provides a pattern of more than one notifications to match against. The notifications are delivered to the subscriber only if they satisfy the registered pattern.
- Context-based publish/subscribe [5] scheme is the most general form of subscription language. The infrastructure has to take into account the state of the subscriber before delivering an event. The subscriber context can be physi-

Communication Paradigm	Identity decoupling	Time decoupling	Synchronization decoupling	Examples
Message Passing	No	No	Producer-side	HTTP on TCP
Remote Invocation	No	No	Producer-side	CORBA,
Notifications	No	No	Yes	CORBA AMI
Shared Spaces	Yes	Yes	Producer-side	Linda, Java Spaces
Message Queuing	Yes	Yes	Producer-side	IBM MQ series
Publish/Subscribe	Yes	Yes	Yes	REBECA [6], JEDI [3]

Table 1: Comparison of Abilities of Communication Paradigms [4]

cal values such as location, speed, direction, temperature as well as resource availability such as battery power.

It is generally desirable to build sophisticated event filtering mechanisms in the middleware itself and not in the higher level application because, performing event filtering in the infrastructure saves bandwidth and avoids unnecessary processing complexity at the client side. It is especially important in mobile environment because, mobile clients are constrained on bandwidth as well as processor resources.

2.3 Centralized vs. Distributed Publish/Subscribe Architecture

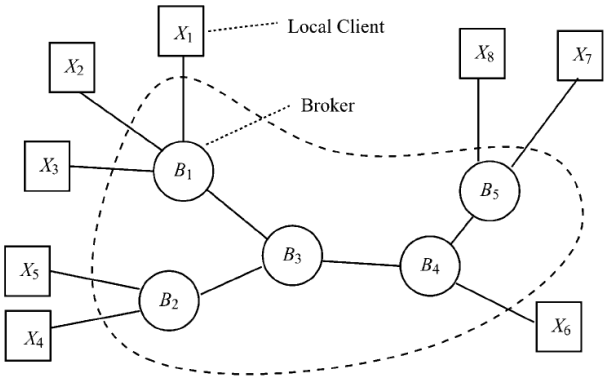

Figure 1: A Distributed Event Broker Network [6]

The publish/subscribe infrastructure components that are responsible for storing consumer subscriptions and matching events against them are known as Event Brokers[1] [8]. A centralized publish/subscribe system consists of only one central broker component running on a dedicate host. The central broker stores all currently active subscriptions in the system. Every new event is published to the broker, which is responsible for matching it against all the subscriptions. Afterwards the event is forwarded to all consumers whose subscriptions match. A distributed publish/subscribe infrastructure consists of more than one Event Brokers arranged in a hierarchical fashion or generic connected graph representation. Figure 1 shows an example of a distributed publish/subscribe infrastructure. Distributed event broker architectures can scale to large number of publishers, consumers and subscriptions. Matching and delivering workload at each event broker is reduced as there are fewer subscriptions to match at every event broker.

[1] In the surveyed literature, both the terms – "Event Broker" and "Dispatching Server" – represent the same thing conceptually.

3. Publish/Subscribe Systems and Mobility

In this section, we analyze the requirements of publish/subscribe middleware for large mobile systems and issues that any publish/subscribe middleware must address to overcome them. We also discuss an approach to verify applications built using publish/subscribe middleware.

3.1 Requirements of Publish/Subscribe Middleware for Mobility [3]

- Manage physical mobility of application components. Users can move while being online or offline, the middleware must take this into account.
- Manage changes in the underlying network topology that may occur in very dynamic settings like ad-hoc networking.
- Manage large number of information consumers and consequently large number of subscriptions. Similarly, manage a potentially very large number of information providers
- Propagate notifications for thousands of information consumers simultaneously, which generally implies use of some sort of multicasting or limited broadcasting like technique under the hood.
- Manage high volatility of users' interests (subscription update, insertion, deletion).
- Manage heterogeneity of content formats, ranging from topic tagged blobs and attribute-value pairs to HTML and XML marked-up data. Also support heterogeneous notification channels (email, Internet-protocols, WiFi, phone, WAP, imode, ICQ etc.)
- Support *approximate subscriptions* and *approximate events* to enhance system flexibility by increasing the

expressiveness of the filtering language and the publication language. This feature is particularly useful for *location-based services*.

- Support existing (legacy) applications.
- Support high availability despite node and/or link failures.

It clearly suggests that innovative and flexible middleware policies and mechanisms are required to support such a wide range of application requirements.

3.2 Mobility Issues in Publish/Subscribe Middleware

There are two main issues in middleware for supporting mobility: *physical mobility* [6] and *logical mobility* [6]

Physical Mobility. A good mobile publish/subscribe system has to deal gracefully with both publishers and consumers going offline. For example, when a consumer is out of reach, it is reasonable to expect that the event broker to log and queue the consumers' events so that they can be delivered later when the consumer comes back online. It is quite possible that when the consumer comes back online, it may not reconnect to the old event broker. It may connect to a different event broker on the same broker connectivity graph. In such situations, the main concern of physical mobility is *location transparency*. The middleware should completely shield away the change of location in a seamless manner. Applications that were not designed taking into account mobility should be able to interact with the infrastructure in a seamless manner. The middleware needs to satisfy at least following requirements to support true *location transparency*:

- **Interface.** The interface of publish/subscribe system must not change as legacy applications are not aware of mobility.
- **Completeness.** Despite intermittent disconnects, the middleware should deliver all notifications for a client eventually.
- **Ordering.** Producers' ordering of event should be maintained during connections and disconnections.
- **No duplicates.** In large distributed event broker networks, rapid location changes coupled with network latency due to link congestion may result into events begin delivered twice. Middleware should guard applications against such situations.
- **Responsiveness.** The quality of service (QoS) guarantees of the application should not have noticeable degradation. Often, Mobile IP [9] based solutions to mobility do not guarantee responsiveness because of extra indirection in the middle.

Providing above guarantees in very dynamic environments such as ad-hoc networking is a very challenging task, if not impossible. Often, persistence storage is used to provide *completeness* guarantee to the applications. Mobile nodes in ad-hoc network rarely can afford resources to support a persistent storage. Discarding out-of-order events and requesting retransmission of missed events is not an option for bandwidth constrained mobile nodes. In Section 4.1, we discuss how a dynamic publish/subscribe tree (PST) [8] can be created in ad-hoc wireless environment to reduce the overhead of event dispatching in a distributed event broker system. Similarly, to maintain responsiveness at different physical locations after repeated disconnections requires quick adaptation of the delivery path from producer to consumer in a distributed broker network. A handoff protocol for consumers subscriptions is often necessary in such situations. In Section 4.1, we discuss a handoff protocol implemented in REBECA [6] that allows seamless location transparency. The subscription handoff protocol needs to be designed carefully so that, as the new routing information (delivery path) percolates up the distributed event brokers, no event from any potential source is lost or delivered twice.

Logical Mobility. While physical mobility is a rather technical issue invisible to the application, logical mobility involves location awareness. An example for logical mobility is when clients move around a house or building that is served by only one event broker. In this case, the user might be interested to receive just those notifications that refer to the room he is currently located in. It is possible for a client to be both logically and physically mobile at the same time.

The adaptation of some location dependent subscription should take place "instantaneously". Propagation delay of events and subscriptions between distributed nodes should not cause loss of location specific events and blackout periods. Such location-aware subscriptions are commonly known as *location dependent filters*. Middleware should provide support (using policies and mechanisms) for applications that use location dependent filters in such a way that discrete publication events should be delivered to mobile subscribers subject to their *continuous* motion. As shown in Figure 2, simple routing technique may result into loss of some location dependent events. Flooding can solve the problem but it is very expensive in large networks. We discuss a solution to the problem of loss of events using *restricted flooding* [6] technique in Section 4.2.

Effective implementation of location dependent filters require *approximate matching* [3, 10] technique. An example of an approximate subscription is shown in Figure 3.

```
S:        (close to downtown Toronto) AND
          (about 75 square meters in size) AND
          (no more than $1000) AND
          (close to major grocery shopping)
```

Figure 3: An Example Approximate Subscription [3]

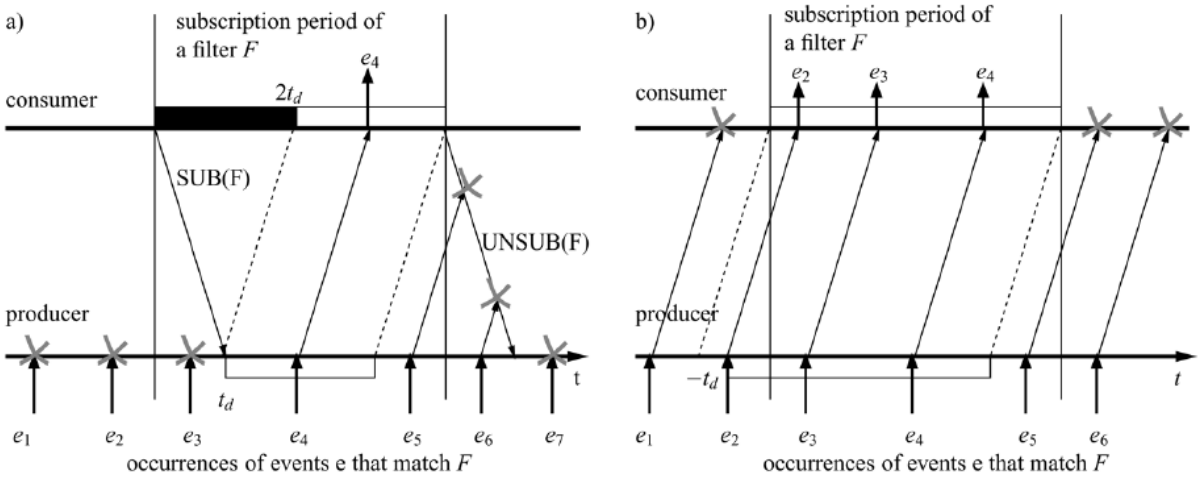

Figure 2: Blackout Period After Subscribing with (a) Simple Routing and (b) Flooding with Consumer-side Filtering. [6]

Unlike approximate matching, conventional publish/subscribe systems use strict boolean matching. They are either matched or not matched by a given event. This matching semantic is often too restrictive, because in many situations only imprecise knowledge about the exact value of a location variable is available, or because subscribers may often be satisfied with an event that matches a subscription partially or only to a certain degree. For example, it is often enough to indicate that a parking space is available within a few hundred feet from the current location of a vehicle. Probability theory, fuzzy set theory, or possibility theory based technique should be incorporated into middleware for supporting location-aware subscriptions.

Middleware for supporting location-aware applications also need to provide support for *location-dependent events* or in general, *approximate events* similar to approximate subscriptions. For example, an approximate location-dependent event may look like as shown in Figure 4.

```
E:        (location, close to downtown)
          (size, big)
          (price, expensive)
          (shopping, close-by)
```

Figure 4: An Example Approximate Event [3]

The notions of location dependent events and location dependent subscriptions are generalized to *publication spaces* and *subscription spaces* in LPS [5] as shown in Figure 5. Modeling of imprecise sensor readings, approximate location information, location information annotated with range, and stochastic environmental conditions are further examples that require an approximate matching-based approach for processing with a publish/subscribe system. We discuss the design of an approximate *matching engine kernel* [3, 10] in Section 4.2.

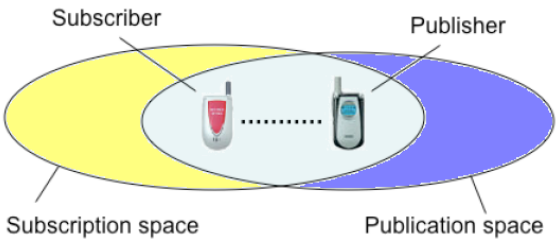

Figure 5: Location-based Event Matching [5]

3.3 Automatic Verification of Publish/Subscribe Architectures

It is clear from the previous sections that any publish/subscribe middleware that satisfies the wide range of requirements of the mobile applications has to be flexible and very dynamic in nature. Flexibility of the infrastructure hampers its validation. Therefore, verification of applications developed using this paradigm becomes a challenging task. We can easily reason on publishers and consumers in isolation, but the global picture is much more complex and dynamic. Components are often written independently of the way they are federated and their interactions can change dynamically. Moreover, the publish/subscribe paradigm can be instantiated in very different ways. The key features are preserved, but specific guarantees vary from infrastructure to infrastructure. For example, Table 2 summarizes the guarantees provided by publish/subscribe middleware. Available choices make verification process extremely difficult.

The problem becomes even harder if we want to model these features by means of existing model checkers. Detailed models cause the well-known *state space explosion* problem, which inherently leads to the inability to verify accurate models. The consequence is that simplified or partial models

Guarantee	Choices
Message Reliability	Absent, Present
Message Ordering	Random, Pair-wise FIFO, System-wide FIFO, Causal Order, Total Order
Filtering	Precise, Approximate
Real-time guarantees	None, Soft RT, Hard RT
Repliable Messages	Absent, Present

Table 2: Publish/Subscribe Guarantees [1]

are used, which limit the number of states generated during verification. In Section 4.3, we discuss the approach adopted by Baresi et al. [1] that allows creation of accurate finite state models of applications using Bogor [12] and detailed verification of them without incurring in the state space explosion problem.

4. Solutions

In this section we discuss existing solutions to address various issues in middleware for supporting mobility. The solutions can be categorized into three groups: (1) Algorithms for creating dynamic dispatching trees [3, 8] in ad-hoc networking environment and subscription handoff protocols [6] using dispatching trees, (2) Algorithms for creating location dependent filters [6] and approximate matching [3] techniques, and (3) techniques for automatic verification [1] of publish/subscribe systems. All the solutions are described in the context of distributed event broker network as shown in Figure 1.

4.1 Dynamic Dispatching Trees for Mobile Event Brokers

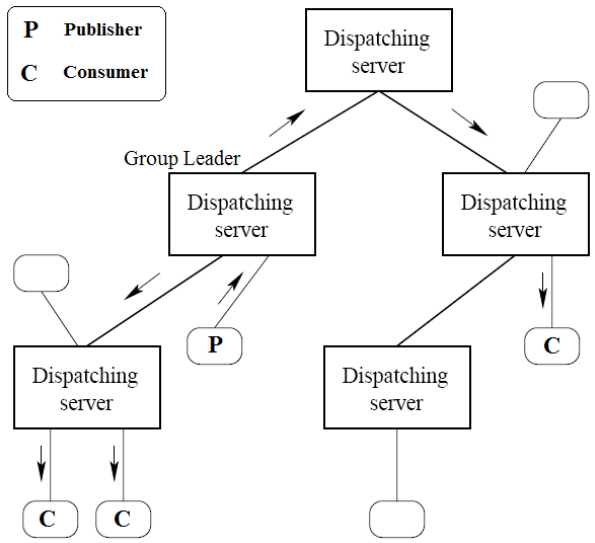

Figure 6: Dispatching Tree in JEDI [3]

Cugola and Jacobson describe an improvement over the existing static dispatching trees in JEDI [3] publish/subscribe system. Figure 6 shows an example of a dispatching tree in JEDI. In this approach, a dispatching server is selected to be a leader of a group of subscribers. It manages the access of other subscribers to the group and the distribution of group members in the network. Any new subscription is broadcasted to all other dispatching servers by the leader. Upon receipt of the subscription, non-leader servers point to the leader to forward the notifications they receive directly from publishers. Thus, any internal dispatching server knows the group leaders for all subscriptions. When an event notification is published, if the originator is not part of the corresponding group of subscribers, the notification is directly sent to the group leader. The group leader dispatches the event to all the interested consumers through the dispatching tree rooted at itself.

The algorithm in JEDI allows dynamic changes to the dispatching tree. A dispatching server can leave the dispatching tree for a particular subscription by sending control messages to its parent. A leaf dispatching server can easily leave the group by communicating its intent to its parent. An intermediate server has to wait till all its children (other dispatching servers and consumers) leave the group. A leader can also leave the group by broadcasting control messages to all other servers and follows the steps of either a leaf server or intermediate server depending upon its position.

The approach described above could be combined to dynamically adapt the dispatching network to changes in the workload that results from movement of consumers and to cope with mobile dispatching servers.

Huang and Molina [7, 8] describe an algorithm called SHOPPARENT to create a publish/subscribe tree (PST) to reduce the total amount of work performed by the dispatching servers. *Overhead* of a PST is defined to be a heuristic function of actions each dispatching server performs: checking for matches, buffering the events, and re-broadcasting the event for children in the PST. Key feature of SHOPPARENT algorithm is that it takes into account subscriptions and locality of subscribers while choosing the right PST to dispatch the events. It is a fully distributed, greedy tree construction algorithm in which no node needs to have global knowledge about the system.

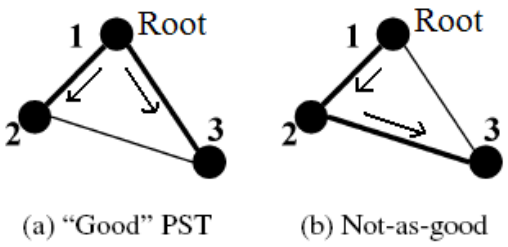

Figure 7: Comparison of Two PSTs [8]

Each dispatching server periodically broadcasts a PARENTPROBE to all dispatching servers in its immediate radio neighborhood, giving its desired subscription. A recipient of the probe replies with a PARENT-ADVERTISE message if it is capable of parenting the requester. The reply contains, in addition to other necessary information, an estimate of how much its overhead will increase if the requesting dispatching server connects to it. The original dispatching server selects among all the replies the one with the smallest expected overhead. Because the probe is periodic, a dispatching server is constantly searching for a better parent, and the tree can reconfigure itself as a result of changes in the system such as node movements and failures. Figure 7 shows an example of a better PST (a) than another PST (b).

In [7], performance of SHOPPARENT algorithms using different heuristic functions (SP-NHOP, SP-OVHD, SP-COMBO) is measured. It is shown that SP-COMBO algorithm performs within 15% of the optimal under normal configurations.

On one hand, dynamic tree construction/adaptation algorithms are required to address the challenge of moving infrastructure components in ad-hoc networking environments, whereas on the other hand subscription handoff protocols are required to efficiently migrate consumer subscriptions from one dispatching server to the another when the consumer physically moves in the vicinity of the later dispatching server. Fiege et al. [6] describe a subscription handoff protocol implemented in REBECA publish/subscribe middleware.

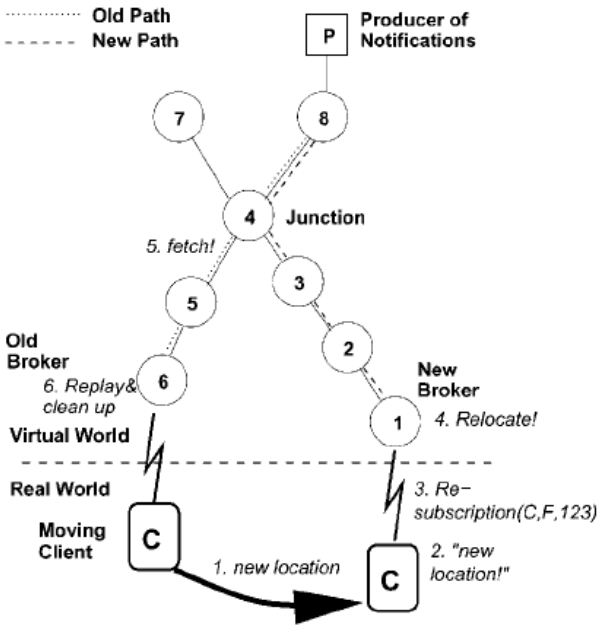

Figure 8: Subscription Hand-off from Broker 6 to Broker 1 [6]

Figure 8 shows an example of a client (C) that moves from dispatching server 6 to 1. The client is subscribed to events published by publisher attached to the dispatching server 8. The subscription handoff protocol is initiated by the dispatching server 1, when it realizes that client C moved from a different location. The subscription of the client is reissued at the new dispatcher (1). The dispatcher PROPAGATEs the subscription backwards to its parents till it finds the junction dispatcher server (4). The junction could be source dispatcher server as well. The junction dispatcher server sends FETCH request to the old dispatching server (6). Upon receiving FETCH request, dispatcher server 6 REPLAYs queued messages back to server 1 and un-subscribes client C from the set of subscribers at 6. Upon receiving replayed messages at the new server, it sends the received notifications to the roaming client in source FIFO order. Roaming client C receives the notifications in order as expected. This handoff protocol provides the *completeness*, *ordering*, *no duplicates* guarantees expected from the publish/subscribe middleware. The algorithm works correctly with the boundaries of finite storage space in the servers, speed of the roaming client, and the rate of event publication.

4.2 Location Dependent Filters and Approximate Matching

Fiege et al. [6] describe another algorithmic solution to a scenario where clients are only logically mobile, i.e., they remain attached to a single dispatching server. The idea is based on *covering* and *merging* of location dependent subscriptions. A filter F1 is said to *cover* another filter F2, if filter F1 accepts a superset of notifications of the second one. If no cover of a given filter can be exists, then two filters are *merged* to create a new filter that covers both the constituent filters. Merging technique simply uses disjunction of two filters.

The algorithm uses *restricted broadcasting* like approach to give a client fine-grained control over notification delivery in the form of location-dependent filters. It builds a series of location dependent filters with monotonically increasing coverage. Filters with larger coverage move toward the event publisher and filters with smaller coverage are stored in dispatching servers nearer to the consumer along the delivery path. Filters having larger and larger coverage are built using a *movement graph*. The movement graph is used to constrain the movement of a consumer as well as to determine the number of steps necessary to move from one location to another. These filters use a local context variable called *location variable*.

Such location dependent filters are often used in *Location dependent services* that rely on approximate matches of location variable. Generally, it is very difficult to cast vagueness inherent in real world applications into a crisp model of mathematical boolean operators that establishes exact limits. Such systems use approximate subscriptions and approximate events, as shown in Figure 3 and Figure 4.

ToPSS [3] publish/subscribe system has a *matching engine kernel* that implements an approximate matching algorithm for their subscription language and the publication model. The ToPSS kernel implements a minimal predicate language [10] on top of which various higher level languages are modeled. The minimal predicate language defines *crisp operators* (e.g., <, >, =, <=, >= and !=), which define crisp boolean predicates; *approximate operators*, which define fuzzy predicates; and *probability operators*, which define probabilistic predicates. An event matches a subscription to a certain degree, if the evaluation of its predicates' and the subscriptions' truth value result in a value less than one (i.e., in a logic where one would represent an exact match.) The interface exposed to the system developers is at a higher level of abstraction than the minimal predicate language. ToPSS provides a mapping of high-level subscription matching language to the low-level predicate language.

ToPSS publish/subscribe system has other features that help in solving other mobility related issues such as bandwidth conservation using a technique called *scheduled notifications*. Heterogeneous notification channels are also supported using the *pervasive notification engine*, which is a stand-alone component that supports a number of standard transport protocols for subscriber notification.

From the above survey of the literature it is clear that researchers have developed solutions to many critical problems in the domain of middleware for mobile systems, but they are scattered across multiple research prototype implementations and are not unified in single middleware platform. Often, combining solutions together is not feasible because of different trade-offs involved.

4.3 Automatic Verification of publish/subscribe architectures

In this section, we discuss how detailed verification of applications built using complex middleware platforms can be done efficiently. Baresi et al. [1] present a novel approach based on Bogor [12] for the accurate verification of applications based on publish/subscribe infrastructure. Instead of building on top of existing model checkers, asynchronous communication mechanisms of publish/subscribe infrastructures are embedded within Bogor. This way, publish/subscribe primitives become part of the specification language as additional, domain-specific constructs. It enables domain-experts to exploit their specific knowledge to better control the state space explosion. The domain-specific constructs hide state information, which does not affect the publish/subscribe semantics. Based on these semantics, the states that a standard model checker would consider as different are considered identical and thereby reducing the state space dramatically. Table 2 summarizes different guarantees provided by publish/subscribe middleware. Extended Bogor specification languages is useful to verify such a wide range of guarantees using higher level, domain-specific mechanisms.

Extended Bogor specification language shows how domain specific optimizations can be embedded in a model checker by raising the level of abstraction of specification language with domain specific constructs. It allows designers to parameterize the new publish/subscribe primitives with one or more of the guarantees shown in Table 2. For example, a publish/subscribe connection can be parameterized with *causal order* delivery option. Messages received out of order will not be delivered to the application using such a connection. This implies that receipt of an out of order message can't change the state of the application and therefore, no new state needs to be generated in this case. Another optimization that helps reduce number of states is when a point-to-multipoint communication mechanism comes into play. Message duplication and multicasting often happens transparently to the application. Therefore, it is possible to avoid to deal with mechanisms and data structures needed to duplicate and distribute messages from the computation of the system state.

This approach allows system designers to explore the trade-offs between the assumptions made on the underlying middleware system and the mechanisms explicitly implemented at application level. For each required guarantee, the developer can either pose new requirements on the publish/subscribe infrastructure by means of parameterization, or maintain the same assumptions on the publish/subscribe infrastructure and implement required mechanisms at application level. Besides flexibility, there is a gain in terms of computational time and memory use in analyzing the system. This means that models that would be too heavy for "conventional" model checkers become analyzable.

5. Discussion

Mobile computing applications raise a number of challenges for the middleware designer. Several solutions have been proposed in literature. We focused on solutions primarily related to (a) publish/subscribe dispatching tree construction and handoff protocols, (b) location dependent services and required mechanisms to support them, and (c) verification of publish/subscribe systems. However, there are several limitations of the approaches that we discussed.

In JEDI [3], every subscription in known to every dispatching server increasing processing time at every hop in the network of dispatching servers. Although, *covering* and *merging* based strategies can be used, it results into a kind of flooding in the overlay network of matching producers and consumers of similar interests. Effects of covering-based technique are discussed in REBECA [13]. Both covering and merging promise to increase routing efficiency but aggravate relocation management.

One of the drawbacks of the SHOPPARENT algorithm for dynamic dispatching tree construction in ad-hoc networking environment is that it only supports occasional reconfigurations of the tree followed by periods of stabil-

ity. This assumption may not hold in highly dynamic environments where frequent reconfigurations are necessary. Mesh-based instead of tree based approaches should be investigated. An undesirable consequence of using more (re)configuration resilient technique is that, reconfiguration latency increases resulting into need of larger persistent queues in the mobile hosts and reduces the upper bound on the maximum rate of change of location (speed).

Requirement of large persistent storage on mobile devices can be alleviated using semantic filtering [8]. Another technique that is based on semantics of events is to use *offline filters* or *durable subscriptions* [11]. Unlike durable subscriptions, offline filters are active only when the mobile subscriber is disconnected from its dispatching server. The offline filters are designed in such a way that only the most relevant events are preserved. For example, in a stock tracker application, only the latest quote is stored, or in a temperature sensor application, only the highest temperature is stored. Responsive handoff protocols can further reduce the size of persistent storage. Responsiveness of handoff protocols can be improved using network level quality of service (QoS) for control messages on fixed networks. Research along these directions seems to be missing.

In some worst case scenarios, the location based routing algorithm described in REBECA may lead to undesirable behavior like missing notifications or even starvation of a client because of the latency of the event middleware. Although their approach is based on "restricted flooding", flooding can be nearly eliminated if location change is predicated proactively and if the infrastructure reorganizes itself around it. Research seems to be missing along this dimension as well.

Achieving of high availability in spite of node failures in distributed publish/subscribe systems is also of concern, especially in mobile environments. Huang and Molina [8] describe how basic guarantees of publish/subscribe middleware are affected because of replication – a common technique to improve availability. Security is also an open issue for publish/subscribe middleware. In particular, it is critical to support confidentiality of events given that the content of events has to be available in some form to the event dispatcher to route them. It is clear that much work remains to be done in the broad space of publish/subscribe systems for supporing mobility.

6. Concluding Remarks

The proliferation of pervasive computing devices, the integration of network access technologies (mobile, wireless, and Internet) and the large amount of information providers offering content are driving the need to an information dissemination model that offers its users highly pertinent information in a demand-driven manner. This can be supported extremely well thorough the publish/subscribe paradigm, where content providers constitute the publishers of information, while content seekers constitute subscribers. Mobility support in publish/subscribe system raises new challenges because of highly dynamic nature of the underlying network and resource scarcity. We discussed requirements of the applications on mobile platform and issues that arise in middleware that promises to satisfy those requirements. We focused on various solutions pertaining to (a) publish/subscribe dispatching tree construction [3, 8] and handoff protocols [6] (b) location dependent filters [6] and approximate matching techniques [10] and (c) verification of publish/subscribe systems [1]. We also visited open challenges in the space of middleware support for mobility and sketched an outline of possible solutions to some of the problems.

References

[1] Luciano Baresi, Carlo Ghezzi, and Luca Mottola. On accurate automatic verification of publish-subscribe architectures. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 199–208, Washington, DC, USA, 2007. IEEE Computer Society.

[2] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 219–227, New York, NY, USA, 2000. ACM.

[3] G. Cugola and H.A. Jacobsen. Using publish/subscribe middleware for mobile systems. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):25–33, 2002.

[4] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[5] Patrick Th. Eugster, Benoît Garbinato, and Adrian Holzer. Location-based publish/subscribe. In *NCA '05: Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 279–282, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in content-based publish/subscribe middleware, 2003.

[7] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 122–140, London, UK, 2003. Springer-Verlag.

[8] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe in a mobile environment. *Wirel. Netw.*, 10(6):643–652, 2004.

[9] D. Johnson. Scalable support for transparent mobile host internetworking, 1995.

[10] Haifeng Liu and H.-Arno Jacobsen. A-TOPSS — a publish/subscribe system supporting approximate matching. In *Proceedings of the 28th Intl. Conference on Very Large*

Data Bases (VLDB), pages 1107–1110, 2002.

[11] Gero Muhl, Andreas Ulbrich, Klaus Herrmann, and Torben Weis. Disseminating information to mobile clients using publish-subscribe. *IEEE Internet Computing*, 8(3):46–53, 2004.

[12] M. Robby and J. Dwyer. Bogor: an extensible and highly-modular software model checking framework, 2003.

[13] A. Zeidler and L. Fiege. Mobility support with rebeca, 2003.