Publish/Subscribe in a Mobile Environment

Yongqiang Huang, Hector Garcia-Molina
Department of Computer Science
Stanford, CA 94305
{yhuang, hector}@cs.stanford.edu

ABSTRACT

A publish/subscribe system dynamically routes and delivers events from sources to interested users, and is an extremely useful communication service when it is not clear in advance who needs what information. In this paper we discuss how a publish/subscribe system can be extended to operate in a mobile environment, where events can be generated by moving sensors or users, and subscribers can request delivery at handheld and/or mobile devices. We describe how the publish/subscribe system itself can be distributed across multiple (possibly mobile) computers to distribute load, and how the system can be replicated to cope with failures, message loss, and disconnections.

1. INTRODUCTION

A *publish/subscribe* system connects together information providers and consumers by delivering *events* from sources to interested users. A user expresses his/her interest in receiving certain types of events by submitting a predicate defined on the event contents, called the user's *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 delivering it efficiently and reliably to all users whose subscriptions match the event.

The publish/subscribe communication paradigm differs from traditional point-to-point models in a number of ways. In publish/subscribe, communication is anonymous, inherently asynchronous and multicasting in nature. The system is able to quickly adapt in a dynamic environment. Anonymity means that the communication partners are not required to identify the party they want to talk to. For example, instead of naming a publisher to receive events from, the subscriber simply describes the characteristics of the events it wants to receive. 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 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. Finally, the system can cope with a dynamically changing operational environment where the publishers and subscribers frequently connect and disconnect.

This combination of unique characteristics makes the publish/subscribe model well suited to a variety of application areas, such as distributed information dissemination, financial analysis and factory automation. In the past decade many problems related to publish/subscribe have been tackled and solved, with some systems having reached commercial maturity ([19, 20]).

However, almost all of the research on publish/subscribe systems so far has concentrated on publish/subscribe systems in a fixed network. We argue that publish/subscribe systems are also advantageous in a mobile and/or wireless environment ([9]). The anonymity and dynamism of publish/subscribe allow the systems to adapt quickly to frequent connections and disconnections of mobile nodes, 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 helps a system scale to millions of units.

With increasing popularity of mobile handheld devices, there is a pressing need to extend publish/subscribe to a mobile environment. As a sample application, in a military battlefield, thousands of wireless and mobile sensors such as satellites and equipment sensors report all kinds of information ranging from the location of enemy troops to whether the engine of a tank has overheated. There are also many parties interested in receiving certain types of information. An individual soldier may need to know the location of the nearest enemy troops, or whenever a missile has been fired. The above scenario requires the deployment of a highly scalable and dynamic communication infrastructure, for which a mobile publish/subscribe system is an ideal candidate.

In this paper, we briefly review a few publish/subscribe models and discuss how they might be adapted to a mobile environment. After presenting our framework (Section 2), we start with the simplest model, namely, the centralized approach (Section 3). We then discuss distributed models that address the scalability problem (Section 4). Finally, we look at the use of replication and its impact (Section 5).

2. FRAMEWORK

The first generation of publish/subscribe systems use either group-based (also known as channel-based) or subject-based (also known as topic-based) addressing. In the former ([4, 10, 14, 18]), a set of "groups" (or "channels") are designated by the system. Each event is published to one of these groups by its publisher. A user subscribes to one or more groups, and will receive all events published to the subscribed groups. For example, in IP multicast ([10]), each group is identified by a class D IP address. Subject-based systems ([15, 13, 19, 20]) are slightly more flexible. Each event is tagged with a short "subject" (or "topic") that describes its content. The subject is either an arbitrary string or a string taken from an agreed-upon domain. The subscriber defines its subscription in terms of this subject line. In addition to an exact match, the subscriber can also ask for all subjects beginning with the word "jobs", for example.

In recent years a more flexible paradigm called content-based addressing has emerged. A content-based system gives more flexibility and control to the subscriber by allowing him/her to express his/her interest as an "arbitrary" query over the contents of the events. Therefore, instead of relying on the publisher to classify the events into groups or subjects, the subscriber is now able to define sophisticated subscriptions such as "give me all stock quotes for stock X issued between time A and time B if the price is larger than 35." A content-based publish/subscribe system has also been called condition monitoring systems or event notification/distribution/-delivery systems in various contexts.

However, the flexibility of content-based systems comes at the expense of added challenges in design and implementation ([5]). Intuitively, because the subscriptions can be complex, figuring out matches between events and subscriptions is a lot harder than in traditional group- or subject-based systems, where usually a simple table lookup is sufficient.

In this paper, we will assume a content-based system in our discussions, because it is the more general and powerful model. For instance, group- and subject- based systems can be regarded as special cases of content-based systems where the subscription syntax is restricted to simple tests on a specific field of the event.

2.1 Basic model

Figure 1 illustrates the schematic of a basic publish/subscribe system. It consists of one or more **Event Sources** (ES), an **Event Brokering System** (EBS), and one or more **Event Displayers** (ED). An Event Source generates *events* in response to changes to a real world variable that it monitors, such as the location of an enemy tank. We assume that each event is labeled with its source and a consecutive sequence number to facilitate our description. Other than that, we do not make any assumptions about an event's content. The events are *published* to the Event Brokering System, which matches them against a set of *subscriptions*, submitted by users in the system. For example, a soldier could subscribe to receive all events reporting the location of any tank within a certain range. Note that, as the core of the publish/subscribe system, the EBS could be implemented as either a single server (Event Broker) or multiple distributed ones working together. Additionally, an Event Broker can be replicated to increase availability. Sections 4 and 5 discuss distributed and replicated architectures and their mobile adaptations.

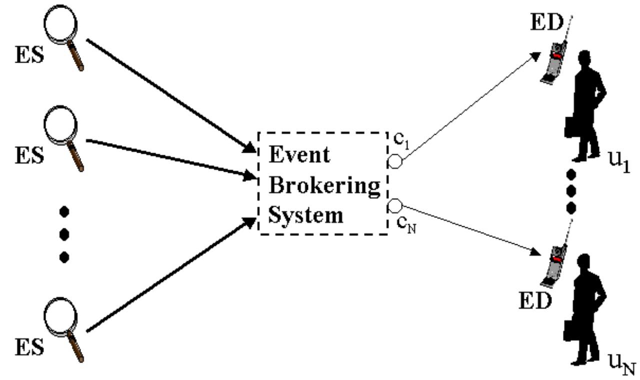

Figure 1: A content-based publish/subscribe system. The bubbles represent filtering of events, and are labeled with the respective filtering criteria.

In Figure 1, we use c_i to denote the subscription criterion of user i. In other words, user i wants all events and only those events that satisfy c_i. If a user's subscription matches, the event is forwarded to the Event Displayer associated with that user. The presence of a bubble labeled c_i in the link between the EBS and an ED implies that only events satisfying c_i passes through on this link. The Event Displayer is responsible for alerting the user. In our example, the soldier will be notified by a message on his/her mobile communication device.

Note that some of the event services surveyed in this paper provide additional functionality such as event stream manipulation. For example, some systems can trigger on events to generate new events. In this case, a subscription might look like: "generate a buy order when the price of stock X has climbed for more than 20 percent for three straight quotes." The ability to generate new events has been termed "content-based with patterns" ([6]), "event stream interpretation" ([3]) and "historical condition triggering" ([12]), among other things. In this paper, we do not take into account any of the specific system extensions such as this. Instead, we will focus on the most fundamental functionality, namely, to route events from their sources to their targets efficiently and reliably.

3. CENTRALIZED ARCHITECTURES

A centralized Event Brokering System consists of only one Event Broker (Figure 2). The central EB stores all currently active subscriptions in the system. Every new event is published to the EB, which is responsible for matching it against all the subscriptions. Afterwards the event is forwarded to all Event Displayers whose subscriptions match. Representative systems in this category include the SIFT Information Dissemination System ([21]) and active databases ([7]).

An important problem that any centralized system would

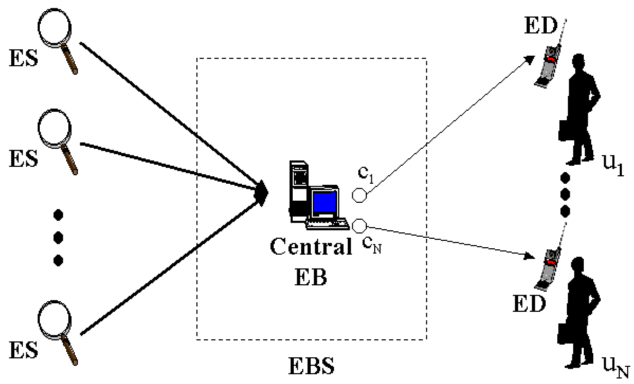

Figure 2: Centralized architecture: one server does all the matching and filtering.

need to address is how to efficiently match a new event against a large set of subscriptions to figure out which ones match. Although the matching problem is challenging and interesting to study, it is beyond the scope of this paper. Interested readers are referred to [21, 1, 7] for more detailed discussions.

Even though the central EB may be a performance bottleneck and a single point of failure, it is important to understand how it could operate in a mobile environment. In later sections, we will look at how distribution alleviates the scalability problem, and how replication helps with reliability.

3.1 Mobile adaptation

When adapting a centralized architecture to the mobile environment, we argue that, while the Event Sources and the Event Displayers can reside on a mobile device, the central Event Broker server should be placed on a separate computer in the fixed network if possible. Typically an Event Source resides near the real world variable it monitors, while an Event Displayer resides near the end user (e.g., on a PDA). Since in a mobile environment both the information providers and the consumers tend to be mobile, the ES and the ED are likely to be placed on a mobile device.

The central EB, however, should reside on a computer separate from the ESs or the EDs. There are three reasons why the EB should not in general be placed on the same device as an ES. Firstly, the EB will likely require a fair amount of computing resource for data logging and subscription matching, while an ES is usually a simple sensor device. Secondly, the Event Sources can be autonomous and do not allow the users to store their own subscriptions there. For example, the ES can be a stock trading center giving out stock quotes. Individual investors usually cannot ask these sources to monitor a stock condition for them. Finally, the EB may need to store a matched event and repeatedly attempt to resend it to its target as the target has gone offline. It is unreasonable to require a mobile Event Source to prolong its connection to the fixed network just because the event recipient is not connected for the moment.

Likewise, the EB should be hosted on a separate device from an ED as well. The PDA can be powered off or disconnected from the network most of the time to conserve battery, making it unsuitable to host the EB, which needs to listen constantly for new events. Furthermore, the computer hosting the EB should be placed in the fixed network if possible, because otherwise when the central EB is disconnected, the whole system is paralyzed.

Once we figure out where each part of the system resides, it would seem that we can simply rely on previous work on mobile networking ([16]) to provide us with connectivity between the components and to hide the idiosyncrasies of mobile communication. However, as we will illustrate next, there are issues unique to publish/subscribe in a mobile environment that we have to consider.

Mobile/wireless devices can be frequently disconnected from the fixed network because they are off (running out of battery or turned off to conserve battery), or they cannot be contacted (transient wireless communication problems or wandering into an area without radio reception etc.). A good mobile publish/subscribe system has to deal gracefully with both the ES's and the ED's going offline. For example, when a user is out of reach, it is reasonable to expect the EBS to log and queue the user's events so that they can be delivered later when the user comes back online.

More issues arise when an Event Source is disconnected. One option of course is to have the ES queue all the events that are generated when it is not connected. Such an option may not be feasible, however, as the ES is often a low capability device without too much storage. Consequently, the ES may have to discard older events once the buffer fills up.

Ad-hoc networks pose additional challenges. An ad-hoc network is formed by wireless devices wanting to talk to each other without the benefit of a fixed network infrastructure. Ad-hoc networks are extremely useful in scenarios where a natural disaster has wiped out the infrastructure, or where rapid deployment is required and an infrastructure is not possible, for example in the battlefield.

The lack of a fixed network in ad-hoc networks implies that the central EB must also be mobile. Hence we can no longer assume that it is always connected. When an Event Source wants to publish, it must first search for an existing EB. If one cannot be found, a new EB might have to be elected. Likewise, an Event Displayer must periodically poll the EB and refresh its subscription information. Otherwise, the old EB could have gone out of reach and a new EB elected without knowing about this ED's subscription. Finally, when one EB becomes aware of the existence of another EB (for example when two previously partitioned wireless subnets come into physical proximity), a merging protocol might have to be invoked to combine them into one central EB. Alternatively, both could operate in a coordinated fashion as discussed in Section 4.

3.2 Centralized with quenching

Quenching has been proposed ([17]) as an enhancement to the straightforward centralized approach in fixed networks (Figure 3). An Event Source is given a "combined active subscription expression" (c_{all}), which represents the logical

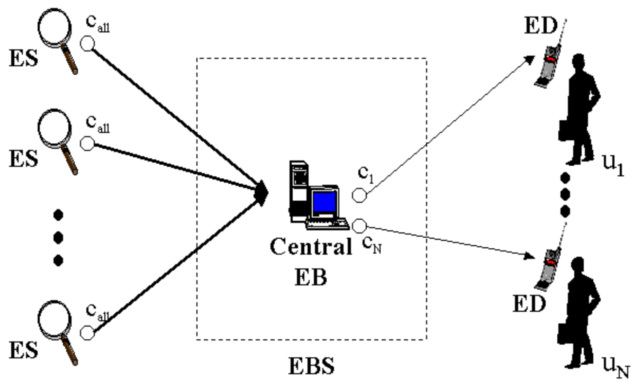

Figure 3: Centralized architecture with quenching.

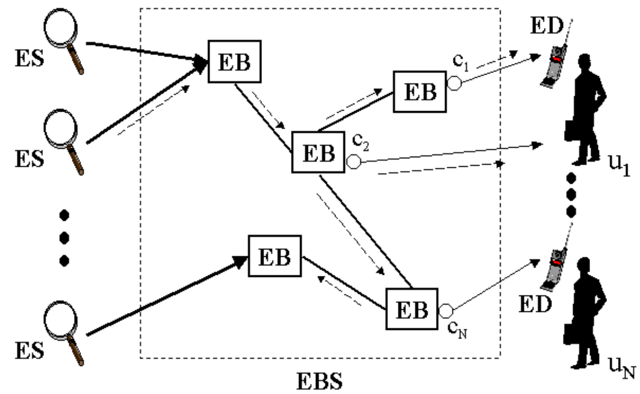

Figure 4: Distributed broadcast. The dotted lines are the path of an example event which satisfies c_1 and c_2.

OR of all the currently active subscriptions on the Event Broker. Essentially, we have $c_{all} = c_1 \vee c_2 \vee \ldots \vee c_N$. When a new event e is generated, the ES first checks it against c_{all}. If $c_{all}(e) = false$, that means no subscription will match e at the EB. Hence the event is discarded (quenched) at the source. If e matches c_{all}, then at least one subscription will match, and the event is forwarded to the EB as usual. This quenching behavior is represented by the bubbles labeled c_{all} in Figure 3.

Note that in order for quenching to make sense it must be much easier to figure out whether or not an event e matches the combined c_{all} than to figure out the exact subset of $\{c_1, c_2, \ldots, c_N\}$ that matches, so that the Event Source does not have to duplicate all the work that is being done at the EB[1]. Quenching has proved to be particularly effective in reducing network traffic and the load of the central EB if a significant portion of the events generated do not match any subscriptions.

However, the appropriateness of using quenching in a mobile environment needs to be further examined. We have said previously that an ES can be a wireless low capability sensor device. Thus it might not be feasible for the ES to evaluate a complicated condition for every event generated. Moreover, informing the sensor of newly added or deleted subscription could consume valuable wireless bandwidth. On the other hand, effective quenching can also significantly reduce the bandwidth needed to transmit events. Fundamentally, quenching represents a tradeoff between the bandwidth required to send all events and the computation power needed to match and filter events. Since a mobile device is usually limited in both resources, the answer is not apparent.

Quenching can be a particularly attractive option when an ES is disconnected, since it allows the ES to discard certain events on the fly, thus reducing the potential size needed for the event queue. However, quenching is also problematic since the system cannot contact an ES about newly added subscriptions when the ES is disconnected. A reasonable strategy might be to save all events in the buffer at the beginning of a disconnection in case a new subscription not known to the ES matches them. When the buffer overflows, however, the ES can then start to discard older events according to the quenching criteria it has.

4. DISTRIBUTION

As explained in Section 3, centralized systems are limited by the capability of a single server, beyond which distribution has to be used. This section illustrates two typical ways that work is partitioned among multiple servers, and their extensions to the mobile world.

4.1 Distributed broadcast

In distributed broadcast (Figure 4), the EBS consists of M Event Brokers, each responsible for a portion of the N total subscriptions. The EBs are connected to each other by the network and form a general connected graph.[2] An Event Source publishes a new event to any one of the M EBs. (In reality, the ES probably connects to the nearest EB in the network.) That EB is then responsible for forwarding the event to all other EBs in the system (hence the name "distributed broadcast"). The forwarding "broadcast" can be implemented with network layer multicasting ([10]). Alternatively, the event could be sent along a "forwarding tree" rooted at the originating EB, using unicast at each leg of the trip.

When a new event arrives, each EB matches the new event against all subscriptions it is responsible for, and delivers the event as necessary. Note that the matching and delivering workload at each EB is reduced compared to a centralized approach because, although each EB still processes all the events generated in the system, it only has to match them against a fraction of the total subscriptions. The dotted lines in Figure 4 give an example of the path traversed by an event which matches c_1 and c_2. An example of a distributed publish/subscribe system using broadcast is the SIFT Grid ([21]).

[1] A trivial example where this is true is the following. Suppose $c_1 = (e.value > 10)$ and $c_2 = (e.value > 20)$. To figure out whether an event matches either c_1 or c_2, it is sufficient to only test whether its "value" is larger than 10.

[2] Actual systems may connect the EBs in other typologies such as a hierarchical tree instead of a peer-to-peer graph. To be most general, our paper assumes a graph structure, although our discussion is equally valid for other structures.

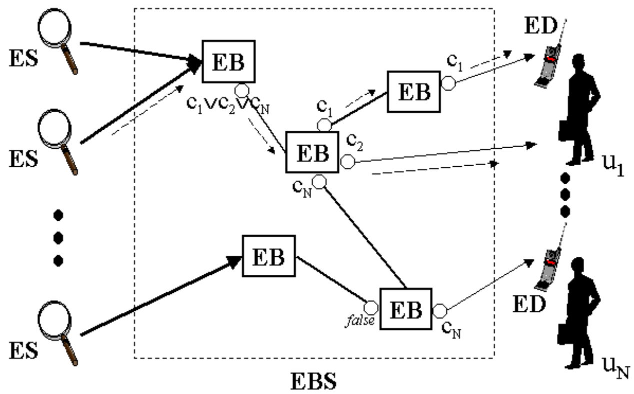

Figure 5: Distributed multicast.

4.2 Distributed multicast

Distributed broadcast can create a lot of network traffic because events are flooded to all the Event Brokers. An alternative approach, called distributed multicast (Figure 5), prunes the forwarding tree. Specifically, when an event arrives at each EB in the forwarding tree, it is forwarded onto one of the EB's outgoing branches only if the event might match a subscription at some EB leading from this branch. In other words, the EB selectively forwards an event based on the result of "partial matching." In effect, the event is matched against the logical OR of all the subscriptions stored at all the EBs downstream from a particular branch. If the result is false, that branch is "pruned" for this event. The behavior is very similar to what happens at the routers in IP multicast, hence the name "distributed multicast."

The need for partial matching implies that, unlike in distributed broadcast, it is no longer sufficient for each Event Broker to know about only its share of the active subscriptions. In the worst case, each Event Broker may need to store all the currently active subscriptions in the system. The Siena system ([6]) proposes a solution where multiple subscriptions can be collapsed into one condition, at the expense of restricted subscription syntax.

Unlike distributed broadcast, distributed multicast can no longer take advantage of network layer multicast directly to forward events to needed EBs, because potentially complex partial matching needs to be performed at each step. An efficient implementation of the forwarding tree without using IP multicast is given in [2].

4.3 Mobile adaptation

In addition to the challenges facing a mobile centralized system, there are more issues associated with adapting a distributed publish/subscribe architecture to a mobile environment. Because EDs often move around, an ED may disconnect and connect to a different EB quite often. When the ED reconnects to a different EB, two things need to happen. First, the new EB needs to be informed of the ED's subscription so that the routing tree can be adjusted to direct relevant events this way. Second, the new EB needs to obtain all the events queued on behalf of the ED while the ED was disconnected and deliver them to the ED. For both tasks, the new EB may contact the EB previously in charge of the user's subscription to obtain the information as part of a "handoff" protocol ([8]).

Alternatively, however, an ED can carry its own subscription information, and upload it onto the new EB when the ED reconnects. The advantage of this approach is that the ED can still receive new events even if the old EB is temporarily down or partitioned from the new EB. (Of course the new EB still needs to attempt contact with the old EB periodically to cancel the old subscriptions.)

The potential downside is that the ED may end up with more than one EBs monitoring the same subscription for it. Reference [11] proposes several schemes for mobile handheld devices which ensure that the ED receives the same message exactly once. For example, one variation requires the ED to keep a log of its past connections, which includes a timestamp and the id of the EB for each connection. Whenever the ED makes a new connection, this information is uploaded to the new EB, which uses it to check for any potential danger of duplicate delivery. For instance, events generated after the ED's last previous connection can safely be delivered. Moreover, if another EB cannot be contacted at the moment, but the log shows that the last connection to that EB happened "long enough" ago in the past, then queued events may still be delivered without worrying about duplication.

The subscription handoff protocol needs to be designed carefully so that, as the new routing information slowly percolates up the forwarding tree, no event from any potential source is lost. Ideally the same event should not be delivered both to the old and to the new EBs (unless the alternative approach above is taken). If that is impossible to guarantee, however, mechanisms to eliminate duplicates will be needed again.

Because a wireless device can be turned off or disconnected for long periods of time, a lot of missed events can accrue in the meantime. Even if storage at the EB is not a concern (which can be in an ad-hoc environment, for example), the sheer amount of time and precious wireless bandwidth required to transmit all of the queued events to the ED when it reconnects might be unreasonable. Again, knowledge about the semantics of a subscription often helps. For example, an EB can purge old events from the queue if it knows that the subscription is time-sensitive. Or it may keep only the more "important" events (e.g. the current high water mark if the client is interested in only maximums).

In a wireless system, it is sometimes possible to further optimize the connection behavior by using an "integrated" approach. Base stations are used as access points of wireless devices into the fixed network. A wireless device is controlled by one and only one base station at any time it is connected. When it moves out of the range of an old base station and into the range of a new one, a wireless handoff protocol is invoked. Naturally, the base stations are ideal candidates as Event Brokers in a distributed publish/subscribe system. In this case, subscription handoff can be handled as merely an additional step in wireless connectivity handoff, thus saving valuable time and resources.

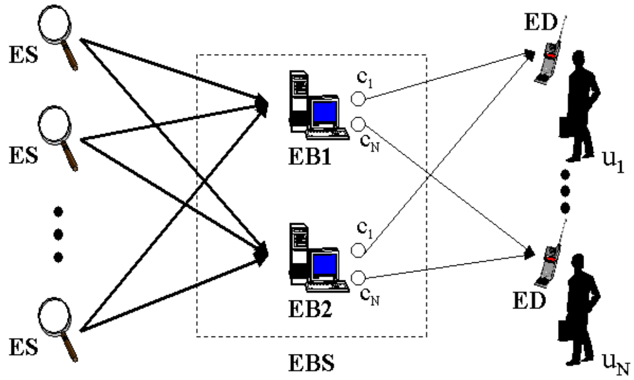

Figure 6: Replicated publish/subscribe.

The discussion thus far in this section has assumed that the EBs are placed in the fixed network for efficiency and robustness. In an ad-hoc network where the EBs have to be mobile, additional problems arise similar to those discussed in Section 3.1. We do not discuss these issues due to space limitations.

5. REPLICATION

Replication can be used in a publish/subscribe system to increase its availability and reliability when faced with server failures or network partitions. In a replicated publish/-subscribe system (Figure 6), a user's subscription is monitored by multiple Event Brokers independently. In particular, in Figure 6 we assume that two EBs, EB1 and EB2, simultaneously monitor the subscriptions for each user. On the other hand, we assume that there is still only one Event Displayer associated with each user, because, as we have discussed before, the ED is usually a program running on the PDA which the user carries with him/her. Hence, the two streams of events generated by EB1 and EB2 will merge at the ED. Note that for simplicity we use a centralized architecture as the basis for replication. Although we do not discuss it here, replication can also be introduced in a distributed system like the ones in Figures 4 and 5.

With replication, a user is less likely to miss events. For instance, suppose that EB1 misses some events from a particular mobile source which can only communicate with EB2 due to temporary network problems[3]. Then the events can still be matched by EB2 and delivered to the appropriate EDs. However, without any safeguards, replication can create "consistency" problems in a publish/subscribe system. Specifically, the user may receive a sequence of events that are confusing or even contradictory. As a simple example, without a mechanism to eliminate duplicates, the same event may get delivered to the user twice, once from each EB. The user will get confused if he/she relies on the events to keep track of an important count, such as the exact number of missiles that have been fired.

As another example, although it is not difficult to make a single EB always deliver events from the same source in order, replication can often result in an unordered event sequence when events from the two EBs are interleaved at the ED. For instance, suppose event number 3 is missed by EB1 but received by EB2. It is therefore entirely possible for EB1 to deliver the next event, say number 4, to the user before EB2 could have a chance to deliver event 3. Out-of-order event streams can be a problem if the order of events is significant, for example to establish a trend in the movements of a stock's price.

We can define three desirable properties for a replicated publish/subscribe system: **Orderedness**, **Consistency** and **Completeness**. The goal in general is to rule out deliveries of events to a user that could *not* have occurred with a non-replicated system. Intuitively, orderedness indicates that events from the same ES are delivered to the user in the order they are generated at the ES. Since a non-replicated system delivers events in this order, a replicated system that is ordered behaves similarly in this respect.

For a replicated system to be consistent, the set of events it displays to an end user over time must be a set that can possibly be generated by a non-replicated system (although perhaps in a different order). In other words, a user should not be able to tell, from observing the events that are displayed to him/her, that replication is being used (except for possibly increased reliability and responsiveness). For example, a replicated system that delivers duplicates to the end user is trivially not consistent.

Lastly, completeness requires a replicated system R to display all events that would be displayed by an equivalent non-replicated system N had the single EB in N received all events that were received by EB1 or EB2 in R. For example, if an event matching the user's subscription arrives at EB1 but is missed by EB2 due to network packet loss, then a complete replicated system will need to ensure that the event is not discarded (see next on Event Displayer filtering) and is ultimately delivered to the user. Completeness is a measurement of how effective a replicated system is at guarding against loss of events in the network. Our three notions of correctness are defined more formally in [12].

Obviously, if the Event Displayer simply passes along any event it receives to the user, the resulting replicated system will be neither consistent (due to duplicates) nor ordered. Table 1 summarizes properties satisfied by a replicated system under various configurations, with the first row being when no special processing is done by the ED. However, as we will see next, some system properties can be enhanced or enforced if the ED performs an extra step to filter out some events (e.g., duplicates) before passing them on to the user.

In the simplest example, the ED can implement a straightforward "exact duplicate elimination" algorithm, in which an event is discarded by the ED if an "identical" one has already been displayed previously. The exact definition of "identical" is given in [12]. The modified system properties under this ED filtering algorithm are listed in the second row of Table 1. As shown in the table, the system has gained consistency as a result.

[3]We assume that events can be lost when they are sent from their source to an EB. However, since we assume that the EB buffers and retransmits events as necessary, the link between the EB and the ED is assumed to be lossless.

ED filtering	Ord.	Cons.	Comp.
No filtering	X	X	√
Duplicate removal	X	√	√
Out-of-order and duplicate removal	√	√	X

Table 1: Properties satisfied by a replicated publish/subscribe system under various ED filtering algorithms.

For situations where an ordered event stream is imperative, an ED filtering algorithm has been proposed in [12] to enforce orderedness of a replicated system. Essentially, the ED records the last seen sequence number from each Event Source and discards any new event that arrives out of order. The disadvantage of this algorithm, however, is that the system is no longer complete, since some events may be "unnecessarily" filtered out based on their arrival order rather than their content. The tradeoff of completeness versus orderedness should be decided by the individual applications. The last row in Table 1 gives the system properties under a combined filtering algorithm that guarantees both orderedness and consistency.

Reference [12] offers an in-depth study of replication in publish/subscribe systems. For instance, it discusses systems with the ability to generate new events based on patterns in a stream of events. It is shown that such systems are usually inconsistent, because event loss can often lead to divergent perceptions between the two EBs about what constitutes a triggering pattern. Consequently, more sophisticated ED filtering algorithms are developed to guarantee consistency in such scenarios. Additionally, subscriptions defined on event "joins" from different streams are also studied. The paper also investigates multiple subscriptions submitted by the same user that are interrelated and need to be monitored in a coherent fashion.

6. CONCLUSION

In this paper we discussed how to adapt a publish/subscribe system to a mobile operating environment. We described several architectures of a publish/subscribe system, starting from the simple centralized approach, to distributed ones with improved scalability, and finally to replication that increases reliability but may cause consistency problems. We discussed issues and possible solutions specific to adapting the various architectures to a mobile and/or wireless environment. We also sketched solutions to the more challenging problems posed by ad-hoc networks. In presenting our work, we also surveyed some of the important work on content-based publish/subscribe systems in fixed networks.

7. REFERENCES

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, pages 53–61, 1999.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, pages 262–272, 1999.

[3] G. Banavar, M. Kaplan, K. Shaw, R. E. Strom, D. C. Sturman, and W. Tao. Information flow based event distribution middleware. In *Proceedings of the 1999 ICDCS Workshop on Electronic Commerce and Web-Based Applications*, 1999.

[4] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36.12:36–53, 1993.

[5] A. Carzaniga, E. Nitto, D. Rosenblum, and A. Wolf. Issues in supporting event-based architectural styles. In *3rd International Software Architecture Workshop*, 1998.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an Internet-scale event notification service. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, 2000.

[7] S. Ceri and J. Widow. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[8] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, to appear.

[9] G. Cugola, E. D. Nitto, and G. P. Picco. Content-based dispatching in a mobile environment. In *Workshop su Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi*, 2000.

[10] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[11] Y. Huang and H. Garcia-Molina. Exactly-once semantics in a replicated messaging system. In *Proceedings of the 17th International Conference on Data Engineering*, 2001.

[12] Y. Huang and H. Garcia-Molina. Replicated condition monitoring. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing*, 2001. To appear.

[13] B. Kantor and P. Lapsley. Network News Transfer Protocol: A proposed standard for the stream-based transmission of news. Request for Comments: 977, 1986.

[14] Object Management Group. CORBAservices - event service specification. Technical report, Object Management Group, 1997. `ftp://ftp.omg.org/pub/docs/formal/97-12-11.pdf`.

[15] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus - an architecture for extensible distributed systems. *Operating Systems Review*, 27.5:58–68, 1993.

[16] C. Perkins. IP mobility support. Request for Comments: 2002, 1996.

[17] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the 1997 Australian UNIX Users Group Technical Conference*, pages 243–255, 1997.

[18] Sun Microsystems, Inc. Jini(TM) technology core platform spec - distributed events. Technical report, Sun Microsystems, Inc., 2000. `http://www.sun.com/jini/specs/jini1.1html/event-spec.html`.

[19] TIBCO Inc. TIB/Rendezvous. `http://www.tibco.com/products/rv/index.html`.

[20] Vitria BusinessWare. `http://www.vitria.com/products/businessware.html`.

[21] T. W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24.4:529–565, 1999.