Adapting Asynchronous Messaging Middleware to Ad Hoc Networking

Mirco Musolesi
Dept. of Computer Science,
University College London
Gower Street, London
WC1E 6BT, United Kingdom
m.musolesi@cs.ucl.ac.uk

Cecilia Mascolo
Dept. of Computer Science,
University College London
Gower Street, London
WC1E 6BT, United Kingdom
c.mascolo@cs.ucl.ac.uk

Stephen Hailes
Dept. of Computer Science,
University College London
Gower Street, London
WC1E 6BT, United Kingdom
s.hailes@cs.ucl.ac.uk

ABSTRACT

The characteristics of mobile environments, with the possibility of frequent disconnections and fluctuating bandwidth, have forced a rethink of traditional middleware. In particular, the synchronous communication paradigms often employed in standard middleware do not appear to be particularly suited to ad hoc environments, in which not even the intermittent availability of a backbone network can be assumed. Instead, asynchronous communication seems to be a generally more suitable paradigm for such environments. Message oriented middleware for traditional systems has been developed and used to provide an asynchronous paradigm of communication for distributed systems, and, recently, also for some specific mobile computing systems.

In this paper, we present our experience in designing, implementing and evaluating EMMA (Epidemic Messaging Middleware for Ad hoc networks), an adaptation of Java Message Service (JMS) for mobile ad hoc environments. We discuss in detail the design challenges and some possible solutions, showing a concrete example of the feasibility and suitability of the application of the asynchronous paradigm in this setting and outlining a research roadmap for the coming years.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; C.2.1 [**Network Architecture and Design**]: Wireless Communication

General Terms

DESIGN, ALGORITHMS

Keywords

Message oriented middleware, middleware for mobile computing, epidemic protocol, mobile ad hoc networks, context awareness

2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing
Toronto, Canada

1. INTRODUCTION

With the increasing popularity of mobile devices and their widespread adoption, there is a clear need to allow the development of a broad spectrum of applications that operate effectively over such an environment. Unfortunately, this is far from simple: mobile devices are increasingly heterogeneous in terms of processing capabilities, memory size, battery capacity, and network interfaces. Each such configuration has substantially different characteristics that are both statically different – for example, there is a major difference in capability between a Berkeley mote and an 802.11g-equipped laptop – and that vary dynamically, as in situations of fluctuating bandwidth and intermittent connectivity. Mobile ad hoc environments have an additional element of complexity in that they are entirely decentralised.

In order to craft applications for such complex environments, an appropriate form of middleware is essential if cost effective development is to be achieved. In this paper, we examine one of the foundational aspects of middleware for mobile ad hoc environments: that of the communication primitives.

Traditionally, the most frequently used middleware primitives for communication assume the simultaneous presence of both end points on a network, since the stability and pervasiveness of the networking infrastructure is not an unreasonable assumption for most wired environments. In other words, most communication paradigms are synchronous: object oriented middleware such as CORBA and Java RMI are typical examples of middleware based on synchronous communication.

In recent years, there has been growing interest in platforms based on asynchronous communication paradigms, such as publish-subscribe systems [6]: these have been exploited very successfully where there is application level asynchronicity. From a Gartner Market Report [7]: "Given message-oriented-middleware's (MOM) popularity, scalability, flexibility, and affinity with mobile and wireless architectures, by 2004, MOM will emerge as the dominant form of communication middleware for linking mobile and enterprise applications (0.7 probability)...". Moreover, in mobile ad hoc systems, the likelihood of network fragmentation means that synchronous communication may in any case be impracticable, giving situations in which delay tolerant asynchronous traffic is the only form of traffic that could be supported.

Middleware for mobile ad hoc environments must therefore support semi-synchronous or completely asynchronous communication primitives if it is to avoid substantial limitations to its utility. Aside from the intellectual challenge in supporting this model, this work is also interesting because there are a number of practical application domains in allowing inter-community communication in undeveloped areas of the globe. Thus, for example, projects that have been carried out to help populations that live in remote places of the globe such as Lapland [3] or in poor areas that lack fixed connectivity infrastructure [9].

There have been attempts to provide mobile middleware with these properties, including STEAM, LIME, XMIDDLE, Bayou (see [11] for a more complete review of mobile middleware). These models differ quite considerably from the existing traditional middleware in terms of primitives provided. Furthermore, some of them fail in providing a solution for the true ad hoc scenarios.

If the projected success of MOM becomes anything like a reality, there will be many programmers with experience of it. The ideal solution to the problem of middleware for ad hoc systems is, then, to allow programmers to utilise the same paradigms and models presented by common forms of MOM and to ensure that these paradigms are supportable within the mobile environment. This approach has clear advantages in allowing applications developed on standard middleware platforms to be easily deployed on mobile devices. Indeed, some research has already led to the adaptation of traditional middleware platforms to mobile settings, mainly to provide integration between mobile devices and existing fixed networks in a nomadic (i.e., mixed) environment [4]. With respect to message oriented middleware, the current implementations, however, either assume the existence of a backbone network to which the mobile hosts connect from time to time while roaming [10], or assume that nodes are always somehow reachable through a path [18]. No adaptation to heterogeneous or completely ad hoc scenarios, with frequent disconnection and periodically isolated clouds of hosts, has been attempted.

In the remainder of this paper we describe an initial attempt to adapt message oriented middleware to suit mobile and, more specifically, mobile ad hoc networks. In our case, we elected to examine JMS, as one of the most widely known MOM systems. In the latter part of this paper, we explore the limitations of our results and describe the plans we have to take the work further.

2. MESSAGE ORIENTED MIDDLEWARE AND JAVA MESSAGE SERVICE (JMS)

Message-oriented middleware systems support communication between distributed components via message-passing: the sender sends a message to identified *queues*, which usually reside on a server. A receiver retrieves the message from the queue at a different time and may acknowledge the reply using the same asynchronous mechanism. Message-oriented middleware thus supports asynchronous communication in a very natural way, achieving de-coupling of senders and receivers. A sender is able to continue processing as soon as the middleware has accepted the message; eventually, the receiver will send an acknowledgment message and the sender will be able to collect it at a convenient time. However, given the way they are implemented, these middleware systems usually require resource-rich devices, especially in terms of memory and disk space, where persistent queues of messages that have been received but not yet processed, are stored. Sun Java Message Service [5], IBM WebSphere MQ [6], Microsoft MSMQ [12] are examples of very successful message-oriented middleware for traditional distributed systems.

The Java Messaging Service (JMS) is a collection of interfaces for asynchronous communication between distributed components. It provides a common way for Java programs to create, send and receive messages. JMS users are usually referred to as *clients*. The JMS specification further defines *providers* as the components in charge of implementing the messaging system and providing the administrative and control functionality (i.e., persistence and reliability) required by the system. Clients can send and receive messages, asynchronously, through the JMS provider, which is in charge of the delivery and, possibly, of the persistence of the messages.

There are two types of communication supported: *point to point* and *publish-subscribe* models. In the point to point model, hosts send messages to *queues*. Receivers can be registered with some specific queues, and can asynchronously retrieve the messages and then acknowledge them. The publish-subscribe model is based on the use of *topics* that can be subscribed to by clients. Messages are sent to topics by other clients and are then received in an asynchronous mode by all the subscribed clients. Clients learn about the available topics and queues through Java Naming and Directory Interface (JNDI) [14]. Queues and topics are created by an administrator on the provider and are registered with the JNDI interface for look-up.

In the next section, we introduce the challenges of mobile networks, and show how JMS can be adapted to cope with these requirements.

3. JMS FOR MOBILE COMPUTING

Mobile networks vary very widely in their characteristics, from nomadic networks in which modes relocate whilst offline through to ad hoc networks in which modes move freely and in which there is no infrastructure. Mobile ad hoc networks are most generally applicable in situations where survivability and instant deployability are key: most notably in military applications and disaster relief. In between these two types of 'mobile' networks, there are, however, a number of possible heterogeneous combinations, where nomadic and ad hoc paradigms are used to interconnect totally unwired areas to more structured networks (such as a LAN or the Internet).

Whilst the JMS specification has been extensively implemented and used in traditional distributed systems, adaptations for mobile environments have been proposed only recently. The challenges of porting JMS to mobile settings are considerable; however, in view of its widespread acceptance and use, there are considerable advantages in allowing the adaptation of existing applications to mobile environments and in allowing the interoperation of applications in the wired and wireless regions of a network.

In [10], JMS was adapted to a nomadic mobile setting, where mobile hosts can be JMS clients and communicate through the JMS provider that, however, sits on a backbone network, providing reliability and persistence. The client prototype presented in [10] is very lightweight, due to the delegation of all the heavyweight functionality to the

provider on the wired network. However, this approach is somewhat limited in terms of widespread applicability and scalability as a consequence of the concentration of functionality in the wired portion of the network.

If JMS is to be adapted to completely ad hoc environments, where no fixed infrastructure is available, and where nodes change location and status very dynamically, more issues must be taken into consideration. Firstly, discovery needs to use a resilient but distributed model: in this extremely dynamic environment, static solutions are unacceptable. As discussed in Section 2, a JMS administrator defines queues and topics on the provider. Clients can then learn about them using the Java Naming and Directory Interface (JNDI). However, due to the way JNDI is designed, a JNDI node (or more than one) needs to be in reach in order to obtain a binding of a name to an address (i.e., knowing where a specific queue/topic is). In mobile ad hoc environments, the discovery process cannot assume the existence of a fixed set of discovery servers that are always reachable, as this would not match the dynamicity of ad hoc networks.

Secondly, a JMS Provider, as suggested by the JMS specification, also needs to be reachable by each node in the network, in order to communicate. This assumes a very centralised architecture, which again does not match the requirements of a mobile ad hoc setting, in which nodes may be moving and sparse: a more distributed and dynamic solution is needed. Persistence is, however, essential functionality in asynchronous communication environments as hosts are, by definition, connected at different times.

In the following section, we will discuss our experience in designing and implementing JMS for mobile ad hoc networks.

4. JMS FOR MOBILE AD HOC NETWORKS

4.1 Adaptation of JMS for Mobile Ad Hoc Networks

Developing applications for mobile networks is yet more challenging: in addition to the same considerations as for infrastructured wireless environments, such as the limited device capabilities and power constraints, there are issues of rate of change of network connectivity, and the lack of a static routing infrastructure. Consequently, we now describe an initial attempt to adapt the JMS specification to target the particular requirements related to ad hoc scenarios. As discussed in Section 3, a JMS application can use either the *point to point* and the *publish-subscribe* styles of messaging.

Point to Point Model The *point to point model* is based on the concept of queues, that are used to enable asynchronous communication between the producer of a message and possible different consumers. In our solution, the location of queues is determined by a negotiation process that is application dependent. For example, let us suppose that it is possible to know *a priori*, or it is possible to determine dynamically, that a certain host is the receiver of the most part of messages sent to a particular queue. In this case, the optimum location of the queue may well be on this particular host. In general, it is worth noting that, according to the JMS specification and suggested design patterns, it is common and preferable for a client to have all of its messages delivered to a single queue.

Queues are advertised periodically to the hosts that are within transmission range or that are reachable by means of the underlying synchronous communication protocol, if provided. It is important to note that, at the middleware level, it is logically irrelevant whether or not the network layer implements some form of ad hoc routing (though considerably more efficient if it does); the middleware only considers information about which nodes are actively reachable at any point in time. The hosts that receive advertisement messages add entries to their JNDI registry. Each entry is characterized by a lease (a mechanism similar to that present in Jini [15]). A lease represents the time of validity of a particular entry. If a lease is not refreshed (i.e, its life is not extended), it can expire and, consequently, the entry is deleted from the registry. In other words, the host assumes that the queue will be unreachable from that point in time. This may be caused, for example, if a host storing the queue becomes unreachable. A host that initiates a discovery process will find the topics and the queues present in its connected portion of the network in a straightforward manner.

In order to deliver a message to a host that is not currently in reach[1], we use an asynchronous *epidemic routing protocol* that will be discussed in detail in Section 4.2. If two hosts are in the same cloud (i.e., a connected path exists between them), but no synchronous protocol is available, the messages are sent using the epidemic protocol. In this case, the delivery latency will be low as a result of the rapidity of propagation of the *infection* in the connected cloud (see also the simulation results in Section 5). Given the existence of an epidemic protocol, the discovery mechanism consists of advertising the queues to the hosts that are currently unreachable using analogous mechanisms.

Publish-Subscribe Model In the *publish-subscribe model*, some of the hosts are similarly designated to hold topics and store subscriptions, as before. Topics are advertised through the registry in the same way as are queues, and a client wishing to subscribe to a topic must register with the client holding the topic. When a client wishes to send a message to the topic list, it sends it to the topic holder (in the same way as it would send a message to a queue). The topic holder then forwards the message to all subscribers, using the synchronous protocol if possible, the epidemic protocol otherwise. It is worth noting that we use a *single* message with *multiple* recipients, instead of multiple messages with multiple recipients. When a message is delivered to one of the subscribers, this recipient is deleted from the list. In order to delete the other possible replicas, we employ acknowledgment messages (discussed in Section 4.4), returned in the same way as a normal message.

We have also adapted the concepts of *durable* and *non durable* subscriptions for ad hoc settings. In fixed platforms, durable subscriptions are maintained during the disconnections of the clients, whether these are intentional or are the result of failures. In traditional systems, while a durable subscriber is disconnected from the server, it is responsible for storing messages. When the durable subscriber reconnects, the server sends it all unexpired messages. The problem is that, in our scenario, disconnections are the norm

[1]In theory, it is not possible to send a message to a peer that has never been reachable in the past, since there can be no entry present in the registry. However, to overcome this possible limitation, we provide a primitive through which information can be added to the registry without using the normal channels.

rather than the exception. In other words, we cannot consider disconnections as failures. For these reasons, we adopt a slightly different semantics. With respect to durable subscriptions, if a subscriber becomes disconnected, notifications are not stored but are sent using the epidemic protocol rather than the synchronous protocol. In other words, durable notifications remain valid during the possible disconnections of the subscriber.

On the other hand, if a non-durable subscriber becomes disconnected, its subscription is deleted; in other words, during disconnections, notifications are not sent using the epidemic protocol but exploit only the synchronous protocol. If the topic becomes accessible to this host again, it must make another subscription in order to receive the notifications.

Unsubscription messages are delivered in the same way as are subscription messages. It is important to note that durable subscribers have explicitly to unsubscribe from a topic in order to stop the notification process; however, all durable subscriptions have a predefined expiration time in order to cope with the cases of subscribers that do not meet again because of their movements or failures. This feature is clearly provided to limit the number of the unnecessary messages sent around the network.

4.2 Message Delivery using Epidemic Routing

In this section, we examine one possible mechanism that will allow the delivery of messages in a partially connected network. The mechanism we discuss is intended for the purposes of demonstrating feasibility; more efficient communication mechanisms for this environment are themselves complex, and are the subject of another paper [13].

The asynchronous message delivery described above is based on a typical pure epidemic-style routing protocol [16]. A message that needs to be sent is replicated on each host in reach. In this way, copies of the messages are quickly spread through connected networks, like an infection. If a host becomes connected to another cloud of mobile nodes, during its movement, the message spreads through this collection of hosts. Epidemic-style replication of data and messages has been exploited in the past in many fields starting with the distributed database systems area [2].

Within epidemic routing, each host maintains a buffer containing the messages that it has created and the replicas of the messages generated by the other hosts. To improve the performance, a hash-table indexes the content of the buffer. When two hosts connect, the host with the smaller identifier initiates a so-called *anti-entropy session*, sending a list containing the unique identifiers of the messages that it currently stores. The other host evaluates this list and sends back a list containing the identifiers it is storing that are not present in the other host, together with the messages that the other does not have. The host that has started the session receives the list and, in the same way, sends the messages that are not present in the other host. Should buffer overflow occur, messages are dropped.

The reliability offered by this protocol is typically *best effort*, since there is no guarantee that a message will eventually be delivered to its recipient. Clearly, the delivery ratio of the protocol increases proportionally to the maximum allowed delay time and the buffer size in each host (interesting simulation results may be found in [16]).

4.3 Adaptation of the JMS Message Model

In this section, we will analyse the aspects of our adaptation of the specification related to the so-called *JMS Message Model* [5]. According to this, JMS messages are characterised by some properties defined using the header field, which contains values that are used by both clients and providers for their delivery. The aspects discussed in the remainder of this section are valid for both models (point to point and publish-subscribe).

A JMS message can be *persistent* or *non-persistent*. According to the JMS specification, persistent messages must be delivered with a higher degree of reliability than the non-persistent ones. However, it is worth noting that it is not possible to ensure *once-and-only-once* reliability for persistent messages as defined in the specification, since, as we discussed in the previous subsection, the underlying epidemic protocol can guarantee only best-effort delivery. However, clients maintain a list of the identifiers of the recently received messages to avoid the delivery of message duplicates. In other words, we provide the applications with *at-most-once* reliability for both types of messages.

In order to implement different levels of reliability, EMMA treats persistent and non-persistent messages differently, during the execution of the anti-entropy epidemic protocol. Since the message buffer space is limited, persistent messages are preferentially replicated using the available free space. If this is insufficient and non-persistent messages are present in the buffer, these are replaced. Only the successful deliveries of the persistent messages are notified to the senders.

According to the JMS specification, it is possible to assign a priority to each message. The messages with higher priorities are delivered in a preferential way. As discussed above, persistent messages are prioritised above the non-persistent ones. Further selection is based on their priorities. Messages with higher priorities are treated in a preferential way. In fact, if there is not enough space to replicate all the persistent messages, a mechanism based on priorities is used to delete and replicate non-persistent messages (and, if necessary, persistent messages).

Messages are deleted from the buffers using the expiration time value that can be set by senders. This is a way to free space in the buffers (one preferentially deletes older messages in cases of conflict); to eliminate stale replicas in the system; and to limit the time for which destinations must hold message identifiers to dispose of duplicates.

4.4 Reliability and Acknowledgment Mechanisms

As already discussed, *at-most-once* message delivery is the best that can be achieved in terms of delivery semantics in partially connected ad hoc settings. However, it is possible to improve the reliability of the system with efficient acknowledgment mechanisms. EMMA provides a mechanism for failure notification to applications if the acknowledgment is not received within a given timeout (that can be configured by application developers). This mechanism is the one that distinguishes the delivery of *persistent* and *non-persistent* messages in our JMS implementation: the deliveries of the former are notified to the senders, whereas the latter are not.

We use acknowledgment messages not only to inform senders about the successful delivery of messages but also to delete the replicas of the delivered messages that are still present in the network. Each host maintains a list of the messages

successfully delivered that is updated as part of the normal process of information exchange between the hosts. The lists are exchanged during the first steps of the anti-entropic epidemic protocol with a certain predefined frequency. In the case of messages with multiple recipients, a list of the actual recipients is also stored. When a host receives the list, it checks its message buffer and updates it according to the following rules: (1) if a message has a single recipient and it has been delivered, it is deleted from the buffer; (2) if a message has multiple recipients, the identifiers of the delivered hosts are deleted from the associated list of recipients. If the resulting length of the list of recipients is zero, the message is deleted from the buffer.

These lists have, clearly, finite dimensions and are implemented as circular queues. This simple mechanism, together with the use of expiration timestamps, guarantees that the old acknowledgment notifications are deleted from the system after a limited period of time.

In order to improve the reliability of EMMA, a design mechanism for intelligent replication of queues and topics based on the context information could be developed. However this is not yet part of the current architecture of EMMA.

5. IMPLEMENTATION AND PRELIMINARY EVALUATION

We implemented a prototype of our platform using the J2ME Personal Profile. The size of the executable is about 250KB including the JMS 1.1 jar file; this is a perfectly acceptable figure given the available memory of the current mobile devices on the market. We tested our prototype on HP IPaq PDAs running Linux, interconnected with WaveLan, and on a number of laptops with the same network interface.

We also evaluated the middleware platform using the OMNET++ discrete event simulator [17] in order to explore a range of mobile scenarios that incorporated a more realistic number of hosts than was achievable experimentally. More specifically, we assessed the performance of the system in terms of delivery ratio and average delay, varying the density of population and the buffer size, and using persistent and non-persistent messages with different priorities.

The simulation results show that the EMMA's performance, in terms of delivery ratio and delay of persistent messages with higher priorities, is good. In general, it is evident that the delivery ratio is strongly related to the correct dimensioning of the buffers to the maximum acceptable delay. Moreover, the epidemic algorithms are able to guarantee a high delivery ratio if one evaluates performance over a time interval sufficient for the dissemination of the replicas of messages (i.e., the *infection spreading*) in a large portion of the ad hoc network.

One consequence of the dimensioning problem is that scalability may be seriously impacted in peer-to-peer middleware for mobile computing due to the resource poverty of the devices (limited memory to store temporarily messages) and the number of possible interconnections in ad hoc settings. What is worse is that common forms of commercial and social organisation (six degrees of separation) mean that even modest TTL values on messages will lead to widespread flooding of epidemic messages. This problem arises because of the lack of intelligence in the epidemic protocol, and can be addressed by selecting carrier nodes for messages with greater care. The details of this process are, however, outside the scope of this paper (but may be found in [13]) and do not affect the foundation on which the EMMA middleware is based: the ability to deliver messages asynchronously.

6. CRITICAL VIEW OF THE STATE OF THE ART

The design of middleware platforms for mobile computing requires researchers to answer new and fundamentally different questions; simply assuming the presence of wired portions of the network on which centralised functionality can reside is not generalisable. Thus, it is necessary to investigate novel design principles and to devise architectural patterns that differ from those traditionally exploited in the design of middleware for fixed systems.

As an example, consider the recent cross-layering trend in ad hoc networking [1]. This is a way of re-thinking software systems design, explicitly abandoning the classical forms of layering, since, although this separation of concerns afford portability, it does so at the expense of potential efficiency gains. We believe that it is possible to view our approach as an instance of cross-layering. In fact, we have added the epidemic network protocol at middleware level and, at the same time, we have used the existing synchronous network protocol if present both in delivering messages (traditional layering) and in informing the middleware about when messages may be delivered by revealing details of the forwarding tables (layer violation). For this reason, we prefer to consider them jointly as the *communication layer* of our platform together providing more efficient message delivery.

Another interesting aspect is the exploitation of context and system information to improve the performance of mobile middleware platforms. Again, as a result of adopting a cross-layering methodology, we are able to build systems that gather information from the underlying operating system and communication components in order to allow for adaptation of behaviour. We can summarise this conceptual design approach by saying that middleware platforms must be not only *context-aware* (i.e., they should be able to extract and analyse information from the surrounding context) but also *system-aware* (i.e., they should be able to gather information from the software and hardware components of the mobile system).

A number of middleware systems have been developed to support ad hoc networking with the use of asynchronous communication (such as LIME, XMIDDLE, STEAM [11]). In particular, the STEAM platform is an interesting example of event-based middleware for ad hoc networks, providing location-aware message delivery and an effective solution for event filtering.

A discussion of JMS, and its mobile realisation, has already been conducted in Sections 4 and 2. The Swiss company Softwired has developed the first JMS middleware for mobile computing, called iBus Mobile [10]. The main components of this typically infrastructure-based architecture are the JMS provider, the so-called mobile JMS gateway, which is deployed on a fixed host and a lightweight JMS client library. The gateway is used for the communication between the application server and mobile hosts. The gateway is seen by the JMS provider as a normal JMS client. The JMS provider can be any JMS-enabled application server, such as BEA Weblogic. Pronto [19] is an example of mid-

dleware system based on messaging that is specifically designed for mobile environments. The platform is composed of three classes of components: mobile clients implementing the JMS specification, gateways that control traffic, guaranteeing efficiency and possible user customizations using different plug-ins and JMS servers. Different configurations of these components are possible; with respect to mobile ad hoc networks applications, the most interesting is *Serverless JMS*. The aim of this configuration is to adapt JMS to a decentralized model. The publish-subscribe model exploits the efficiency and the scalability of the underlying IP multicast protocol. Unreliable and reliable message delivery services are provided: reliability is provided through a negative acknowledgment-based protocol. Pronto represents a good solution for infrastructure-based mobile networks but it does not adequately target ad hoc settings, since mobile nodes rely on fixed servers for the exchange of messages.

Other MOM implemented for mobile environments exist; however, they are usually straightforward extensions of existing middleware [8]. The only implementation of MOM specifically designed for mobile ad hoc networks was developed at the University of Newcastle [18]. This work is again a JMS adaptation; the focus of that implementation is on group communication and the use of application level routing algorithms for topic delivery of messages. However, there are a number of differences in the focus of our work. The importance that we attribute to disconnections makes persistence a vital requirement for any middleware that needs to be used in mobile ad hoc networks. The authors of [18] signal persistence as possible future work, not considering the fact that routing a message to a non-connected host will result in delivery failure. This is a remarkable limitation in mobile settings where unpredictable disconnections are the norm rather than the exception.

7. ROADMAP AND CONCLUSIONS

Asynchronous communication is a useful communication paradigm for mobile ad hoc networks, as hosts are allowed to come, go and pick up messages when convenient, also taking account of their resource availability (e.g., power, connectivity levels). In this paper we have described the state of the art in terms of MOM for mobile systems. We have also shown a proof of concept adaptation of JMS to the extreme scenario of partially connected mobile ad hoc networks.

We have described and discussed the characteristics and differences of our solution with respect to traditional JMS implementations and the existing adaptations for mobile settings. However, trade-offs between application-level routing and resource usage should also be investigated, as mobile devices are commonly power/resource scarce. A key limitation of this work is the poorly performing epidemic algorithm and an important advance in the practicability of this work requires an algorithm that better balances the needs of efficiency and message delivery probability. We are currently working on algorithms and protocols that, exploiting probabilistic and statistical techniques on the basis of small amounts of exchanged information, are able to improve considerably the efficiency in terms of resources (memory, bandwidth, etc) and the reliability of our middleware platform [13].

One futuristic research development, which may take these ideas of adaptation of messaging middleware for mobile environments further is the introduction of more mobility oriented communication extensions, for instance the support of geocast (i.e., the ability to send messages to specific geographical areas).

8. REFERENCES

[1] M. Conti, G. Maselli, G. Turi, and S. Giordano. Cross-layering in Mobile ad Hoc Network Design. *IEEE Computer*, 37(2):48–51, February 2004.

[2] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Sixth Symposium on Principles of Distributed Computing*, pages 1–12, August 1987.

[3] A. Doria, M. Uden, and D. P. Pandey. Providing connectivity to the Saami nomadic community. In *Proceedings of the Second International Conference on Open Collaborative Design for Sustainable Innovation*, December 2002.

[4] M. Haahr, R. Cunningham, and V. Cahill. Supporting CORBA applications in a Mobile Environment. In *5th International Conference on Mobile Computing and Networking (MOBICOM99)*, pages 36–47. ACM, August 1999.

[5] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. *Java Message Service Specification Version 1.1*. Sun Microsystems, Inc., April 2002. http://java.sun.com/products/jms/.

[6] J. Hart. WebSphere MQ: Connecting your applications without complex programming. *IBM WebSphere Software White Papers*, 2003.

[7] S. Hayward and M. Pezzini. Marrying Middleware and Mobile Computing. *Gartner Group Research Report*, September 2001.

[8] IBM. *WebSphere MQ EveryPlace Version 2.0*, November 2002. http://www-3.ibm.com/software/integration/wmqe/.

[9] ITU. Connecting remote communities. *Documents of the World Summit on Information Society*, 2003. http://www.itu.int/osg/spu/wsis-themes.

[10] S. Maffeis. *Introducing Wireless JMS*. Softwired AG, www.sofwired-inc.com, 2002.

[11] C. Mascolo, L. Capra, and W. Emmerich. Middleware for Mobile Computing. In E. Gregori, G. Anastasi, and S. Basagni, editors, *Advanced Lectures on Networking*, volume 2497 of *Lecture Notes in Computer Science*, pages 20–58. Springer Verlag, 2002.

[12] Microsoft. *Microsoft Message Queuing (MSMQ) Version 2.0 Documentation*.

[13] M. Musolesi, S. Hailes, and C. Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. Technical report, UCL-CS Research Note, July 2004. Submitted for Publication.

[14] Sun Microsystems. Java Naming and Directory Interface (JNDI) Documentation Version 1.2. 2003. http://java.sun.com/products/jndi/.

[15] Sun Microsystems. *Jini Specification Version 2.0*, 2003. http://java.sun.com/products/jini/.

[16] A. Vahdat and D. Becker. Epidemic routing for Partially Connected Ad Hoc Networks. Technical Report CS-2000-06, Department of Computer Science, Duke University, 2000.

[17] A. Vargas. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, June 2001.

[18] E. Vollset, D. Ingham, and P. Ezhilchelvan. JMS on Mobile Ad-Hoc Networks. In *Personal Wireless Communications (PWC)*, pages 40–52, Venice, September 2003.

[19] E. Yoneki and J. Bacon. Pronto: Mobilegateway with publish-subscribe paradigm over wireless network. Technical Report 559, University of Cambridge, Computer Laboratory, February 2003.