Noname manuscript No.
(will be inserted by the editor)

EMMA: Epidemic Messaging Middleware for Ad hoc networks

Mirco Musolesi, Cecilia Mascolo, Stephen Hailes

Department of Computer Science, University College London, Gower Street, London WC1E 6BT, United Kingdom
e-mail: {m.musolesi|c.mascolo|s.hailes}@cs.ucl.ac.uk

The date of receipt and acceptance will be inserted by the editor

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, discussing in details the design challenges and the solutions that have been adopted.

Key words Message oriented middleware – middleware for mobile computing – epidemic protocol – mobile ad hoc networks

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 Java RMI and CORBA are typical examples of middleware based on synchronous communication.

In recent years, there has been growing interest in platforms based on asynchronous communication, such as publish-subscribe systems [6]: these have been exploited very successfully where there is application level asynchronicity. This is an extract 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 prac-

tical 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 [20]. 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] and IBM WebSphere MQ [6], 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) [16].

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.

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).

In [10], for example, 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. In the following section, we will discuss our experience in designing and implementing JMS for mobile ad hoc networks.

3 Design of a Message Oriented Middleware for Mobile Ad hoc Networks

3.1 Adaptation of JMS for Mobile Ad Hoc Networks

We now describe EMMA (Epidemic Messaging Middleware for Ad hoc networks), our initial attempt to adapt the JMS specification to target the particular requirements related to ad hoc scenarios. As discussed in Section 2, 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 [17]). 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 3.2. 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 4). 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 3.4), 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 rather than the exception. In other words, we cannot consider disconnections as

[1] In theory, it is not possible to send a message to a peer that has never been reachable in the past, since there is no entry present in the registry. However, to overcome this limitation, we provide a primitive through which information can be added to the registry.

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.

3.2 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 [14].

The asynchronous message delivery described above is based on a typical pure epidemic-style routing protocol [18]. 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 [18]).

3.3 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 values 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.

3.4 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.

4 Implementation and Evaluation

We have implemented a prototype of our platform using the J2ME Personal Profile [15]. 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. The communication infrastructure is based on sockets. We have tested our prototype on HP IPaq PDAs running Linux and 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 [19] in order to have some simulation results considering scenario composed of a realistic number of hosts. This environment offers broad functionalities that facilitate the development and the optimisation of the simulation code.

4.1 Description of the simulation

We simulated the delivery of messages using the epidemic protocol in the case of one recipient (i.e., topic subscriptions and point to point message delivery) and in the case of multiple recipients (i.e, notifications to multiple subscribers). We assumed that no synchronous protocol is present in the underlying network layer. We used a group mobility model with movement patterns similar to those described in [13] We evaluated the performance of the system in terms of the delivery ratios and delays of persistent messages by sending 200 messages (50% persistent and 50% non persistent, with different priorities). Furthermore, we analysed the impact of the use of priorities in a different simulation scenario, sending 300 persistent messages in three priority classes (100 messages for each class). We performed this simulation in order to understand the influence of priorities; moreover, the case of persistent messages only in the system is an interesting limit case. In all the simulations, the priority and the type of persistence of each message are generated using uniform distributions.

The intervals between each message are modelled as a Poisson process. All the messages are sent in 20 seconds. The sender and receiver of each message are chosen randomly. The buffer for each node is set to 100 messages, unless otherwise specified. In the case of subscriber notifications, we set the number of recipients to 80% of the number of hosts; this scenario allows us to evaluate the performance of the delivery mechanisms based on the dissemination of the messages using the epidemic protocol. We consider three mobile scenarios composed of 16, 24 and 32 hosts in a 1 km^2 simulation area. We assume an omnidirectional antenna that transmits according to a free space model with a transmission range equal to 200 m. The maximum allowed delay time is set to four minutes.

4.2 Analysis of results

In this subsection we analyse the results of our simulations, presenting the performance of our platform and discussing the variation of some performance indicators as functions of the density of hosts (i.e., the number

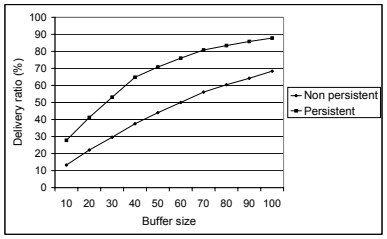

Fig. 1 Point to point model (scenario with 32 hosts): delivery ratio of persistent and non persistent messages vs buffer size.

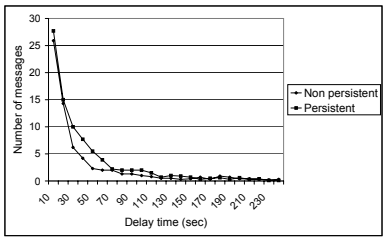

Fig. 2 Point to point model (32 hosts scenario): delay time distribution of persistent and non persistent messages.

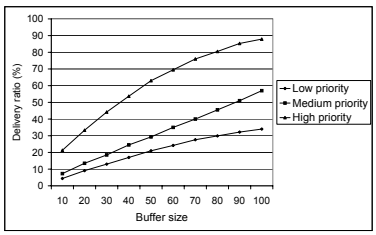

Fig. 3 Point to point model (32 hosts scenario): delivery ratio of persistent messages with different priorities vs buffer size.

of the hosts in the simulation area) and the size of the buffers used to store messages.

Point to point model Figure 1 shows the dependency of the delivery ratio of persistent and non persistent messages on the buffer size, in the case of a scenario with 32 hosts. As expected, the buffer size has a strong impact on the performance of the platform. Therefore, the choice of the correct dimension of the buffer is a key aspect of the deployment of the platform. However, in general, the maximum size of buffers is also constrained by the limited amount of available memory of mobile devices. Figure 2 shows the distribution of the average delay for the point to point delivery (32 hosts scenario); a proportion of the messages are delivered more or less immediately, since the recipients are in the same cloud

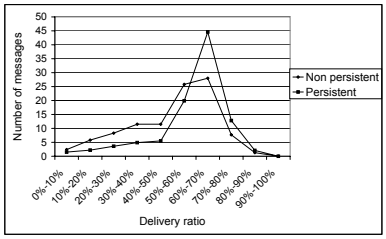

Fig. 4 Publish-Subscribe model (32 hosts scenario): delivery ratio distribution of persistent and non persistent messages.

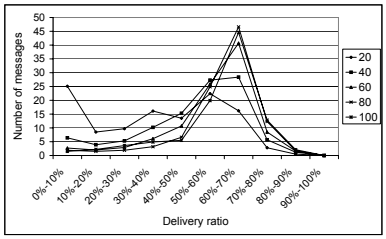

Fig. 5 Publish-subscribe model (32 hosts scenario): delivery ratio distribution of persistent messages vs buffer size.

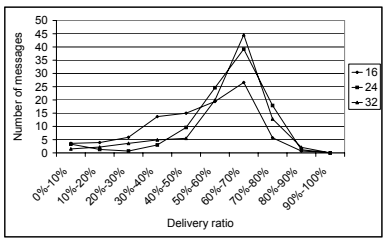

Fig. 6 Publish-subscribe model (32 hosts scenario): delivery ratio distribution of persistent messages vs population density.

as the sender. Figure 3 shows the delivery ratio of persistent messages with different priorities (300 persistent messages with three uniformly distributed levels of priorities as described above, with a buffer size equal to 100).

Publish-Subscribe model Figure 4 shows the distribution of the delivery ratio of persistent and non persistent messages in a 32 hosts scenario. In the case of the publish-subscribe model, the term delivery ratio indicates the average percentage of the potential recipients that actually received the message. Figures 5 shows the distributions of the delivery ratio of persistent and non persistent messages with multiple recipients with respect to buffer size, respectively (in the case of the scenario with 32 hosts). In Figure 6, a graphical representation of the variation of the delivery ratio with respect to

the population density (considering scenarios with 16, 24 and 32 hosts, with a buffer size equal to 100) is presented. As expected, the delivery ratio increases as the population density increases.

The simulation results show that the performance provided by the platform in terms of delivery ratio and delay of persistent messages and messages with higher priorities is good. This is a direct consequence of the exploitation of epidemic techniques [18]. However, it is worth noting that, in general, it is quite difficult to offer high degree of scalability in peer-to-peer middleware for mobile computing due to the characteristics of the devices (limited memory to store temporarily messages) and the number of possible interconnections in ad hoc settings. Nevertheless, the number of nodes of many potential application scenarios that could be envisaged, is quite limited due to the intrinsic communication patterns and organisational boundaries. Moreover, it is worth noting that the dimension of the buffer may be chosen both in accordance with the application requirements and considering the resources of the devices.

For larger application scenarios, where the number of hosts is considerably higher or where the messages exchanged are in high number, we are studying a variation of the delivery mechanism presented that uses probabilistic and statistical techniques to reduce the number of message replicas present at the same time in the system [14]. The description of the protocol is however outside the scope of this paper.

5 Discussion and Related Work

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 [11] (such as LIME, XMIDDLE, STEAM). In particular, the STEAM [12] platform is an example of event-based middleware for ad hoc networks, providing location-aware message delivery and an effective solution for event filtering. In STEAM the communication is limited to the hosts that are in the same radio range. STEAM offers an interesting content-based model, but its possible applications are limited to specific scenarios, where the interaction among hosts belonging to different clouds is not necessary. EMMA, instead, supports communication also among hosts that are intermittently disconnected.

A discussion of JMS, and its mobile realisation, has already been conducted in Section 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.

Pronto [21] is an example of middleware 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. Moreover, different configurations of these components are possible. 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 such as [8]. The only implementation of MOM specifically designed for mobile ad hoc networks was developed at the University of Newcastle [20]. 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 [20] 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.

6 Conclusions

Asynchronous communication is a useful 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). We have described EMMA that represents 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. EMMA provides very good performance in terms of delivery ratio and latency. However, tradeoffs between application-level routing and resource usage should also be investigated, as mobile devices are commonly power/resource scarce. In fact, a key limitation of this work is the poorly performing epidemic algorithm in terms of the number of replicas that are spread across the network. 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 [14].

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 *Proceedings of MOBICOM'99)*, 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. R. Meier and V. Cahill. STEAM: Event-Based Middleware for Wireless Ad Hoc Networks. In *22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 639–644, June 2002.
13. M. Musolesi, S. Hailes, and C. Mascolo. An ad hoc mobility model founded on social network theory. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 20–24, Venice, Italy, October 2004. ACM Press.
14. M. Musolesi, S. Hailes, and C. Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. Technical report, UCL-CS Research Note, November 2004. Submitted for Publication.
15. Sun Microsystems. J2ME Personal Profile Documentation. http://java.sun.com/products/personalprofile/.
16. Sun Microsystems. Java Naming and Directory Interface (JNDI) Documentation Version 1.2. 2003. http://java.sun.com/products/jndi/.
17. Sun Microsystems. *Jini Specification Version 2.0*, 2003. http://java.sun.com/products/jini/.
18. A. Vahdat and D. Becker. Epidemic routing for Partially Connected Ad Hoc Networks. Technical Report CS-2000-06, Dept. of Computer Science, Duke University, 2000.
19. A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, June 2001.
20. E. Vollset, D. Ingham, and P. Ezhilchelvan. JMS on Mobile Ad-Hoc Networks. In *Personal Wireless Communications (PWC)*, pages 40–52, Venice, September 2003.
21. E. Yoneki. Pronto: MobileGateway with Publish-Subscribe Paradigm over Wireless Networks. In *Middleware'03 - Work in Progress Session*, number 4(5), IEEE Distributed Systems Online 2003.