JMS on Mobile Ad-hoc Networks

Einar Vollset[1], Dave Ingham[2] and Paul Ezhilchelvan[3]

[1,3]School of Computing Science, University of Newcastle, Newcastle upon Tyne, NE1 7RU, United Kingdom
{[1]einar.vollset, [3]paul.ezhilchelvan}@ncl.ac.uk
[2]Arjuna Technologies Ltd, Nanotechnology Centre, Herschel Building, Newcastle upon Tyne, NE1 7RU, United Kingdom
dave.ingham@arjuna.com

Abstract. The Java Message Service (JMS) provides a standard asynchronous messaging API, which simplifies the construction of loosely coupled, distributed applications. This paper describes the design and implementation of a pure Java JMS solution for mobile ad-hoc networks (MANETs). The resulting JMS solution simplifies not only the construction of MANET applications but also the re-deployment of any existing JMS applications into a MANET context. The central contribution of this paper lies in comprehensively identifying and addressing the design challenges encountered. For example, common JMS implementations rely, for reasons of efficiency, on a central, reliable server for maintaining dynamic state information, e.g. routing information and group configuration. Construction of a JMS solution for MANETs, on the other hand, cannot rely on such a centralized server. Our server-less implementation involves building a new MANET transport module that is incorporated into an existing JMS product. This transport module implements a MANET multicast routing protocol that provides publish/subscribe semantics by mapping JMS topics to multicast addresses. To our knowledge, our implementation is the first ever Java middleware platform built for MANETs.

Keywords: MANETs, JMS, Mobile Ad-hoc Networks, Message-oriented Middleware, Multicast.

---

[1] (Correspondence Author) Tel: +44 (0)191 2228546, Fax: +44 (0)191 2228232
This work has been supported in part by the EPSRC PACE project and Santosh Shrivastava at the Distributed Systems Research Group at the University of Newcastle upon Tyne, UK

1. Introduction

A mobile ad-hoc network (MANET) is a network made up of a collection of mobile, autonomous nodes that communicate using wireless multi-hop links without any fixed infrastructure such as base stations. Only pairs of nodes that lie within each other's transmission radius can communicate directly. However, each node in the network acts as a router and participates in forwarding packets for other nodes.

MANET technology is currently an active area of research. There is, for example, a large number of proposed routing protocols, such as Ad Hoc On-demand Distance Vector (AODV)[1] and Dynamic Source Routing (DSR)[2]. However, off-the-shelf wireless devices, such as PDAs with 802.11b networking, do not currently possess a standard built-in routing functionality necessary to form MANETs. This lack of standardized routing functionality can be argued to be holding back the development of applications for MANETs. In this paper we attempt to address the issue of simplifying the development of MANET applications. Our work is influenced by our experience in providing support for application development in the traditional wired networking world.

In the traditional wired networking world, distributed application development is greatly simplified through the use of middleware technologies, such as distributed objects systems, e.g., CORBA[3], and message-oriented middleware (MoM)[4]. Of these technologies, MoM appears to be the most appropriate model for the MANET environment, for two main reasons: a) The message exchange model of MoM is asynchronous (as opposed to RPC-style communication), which is the most appropriate model for MANETs. b) The most popular messaging model in MoM, namely publish/subscribe messaging, involves one-to-many communication which maps well to the broadcast nature of MANETs.

The goal of the research presented here was to develop MoM middleware for MANETs based on standard off-the-shelf hardware and operating systems. Specifically, this paper describes the design and implementation of a message-oriented middleware solution for MANETs that runs on top of a standard J2SE (Java 2 Standard Edition) virtual machine on a computer with 802.11b networking. The system is largely compliant with the Java Message Service (JMS) specification[5] which defines a standard asynchronous messaging API to message-oriented middleware. Designing such a system in the MANET context raises new challenges not present in the wired networking world. We formulate these challenges and describe how we addressed them in building a JMS system that supports publish/subscribe messaging. This system not only simplifies the development of new MANET applications, but also allows existing JMS applications to be deployed in a MANET environment. As a proof of concept we deployed an existing instant messaging application, originally developed for wired networks, on a real MANET.
The paper is structured as follows. To present the design challenges faced in designing a JMS system for MANETs into context, the next section provides a description of the JMS specification. This forms the background to section 3, which identifies the issues that need to be addressed when providing JMS on MANET infrastructure. In particular, we argue that the approaches adapted from wired networks are not appropriate. Sections 4 and 5 describe the design and

implementation of our system. Section 6 describes testing and the proof of concept deployment of an instant messaging application, while section 7 describes how the system could be extended to provide more powerful message delivery semantics. Finally, we present our concluding remarks.

2. Java Message Service

The Java Message Service (JMS) is a specification developed by Sun Microsystems that provides a common way for Java applications to access each other via message-oriented middleware (MoM).
The JMS specification defines a set of interfaces and associated semantics that allow Java applications to create, send, receive, and read messages. The specification does not define how messages need to be transported with in a particular implementation, known as a *JMS Provider*. This clear separation of concerns was essential in order to allow vendors of existing messaging products, such as IBM MQSeries[6] and Tibco Rendezvous[7] to support the JMS specification. Furthermore, some JMS vendors provide multiple message-transportation implementations within the same product thereby providing the user with the ability to select the most appropriate transport technology for a particular deployment. An example is the Arjuna Message Service (Arjuna-MS)[8] (formerly the Hewlett Packard Message Service), which provides both server-based and server-less message transport technologies that can be mixed within the same application.

2.1 JMS Messaging Models

The JMS API includes support for the two most common enterprise-messaging models, namely, message queuing and publish/subscribe.

The message queuing model provides one-to-one delivery semantics. Clients send messages to, and receive messages from, *queues*. There may be multiple sender and receivers associated with a given queue, but each message sent by a sender is consumed by exactly one receiver. This means that if multiple receivers are associated with a queue, some sort of arbitration is required by the JMS Provider to decide which one will receive a given message.

The publish/subscribe (pub/sub) model complements the queuing model in that it provides one-to-many delivery semantics. Clients publish messages to and receive messages from *topics*. Each message may be consumed by zero, one, or more subscribers.

2.2 Reliability

JMS is able to support both best-effort and guaranteed delivery semantics called respectively *non-persistent* or *persistent* delivery modes. Non-persistent messages are delivered in a best-effort fashion by the JMS Provider. That is, they will generally be delivered but in the event of an abnormal condition, such as resource-exhaustion or a process failure, the message may be lost. Conversely, if a message producer specifies that persistent delivery mode should be used, then this is an indication to the JMS Provider that steps should be taken to ensure that the message is not lost in the presence of abnormal conditions.

Thus a full JMS implementation can provide 4 types of message delivery semantics: pub/sub non-persistent, pub/sub persistent, queuing non-persistent and queuing persistent.

3. Issues arising when providing JMS on MANETs

JMS products typically utilize a hub-and-spoke architecture in which clients connect to a central server (or cluster of servers) that manages the reliable message routing between them. However, a server-based JMS solution is not appropriate for mobile ad-hoc networks for a number of reasons. Mobile ad-hoc networks are aimed at environments where fixed infrastructure support is unavailable. The philosophy of MANET applications therefore is not to place reliance on centralized services, rather, such service functionality should be distributed across nodes in the MANET so as to minimize the dependency on any particular node and therefore improve overall system resilience. Furthermore, the nodes participating in MANETs are often resource-constrained devices such as PDAs or mobile phones. Because of this, it is advantageous for the processing and networking overheads to be *distributed* across the nodes rather than concentrated at a single server node.

Building a JMS solution for MANETs therefore requires a server-less architecture in which nodes collaborate in a peer-to-peer manner to provide the JMS functionality. We have identified the following issues (in increasing order of complexity), which need to be resolved in order to provide such a solution:

1. **Configuration**: Configuration is the means with which client discover the available queues and topics. A server-based JMS Provider can manage this configuration information on a central server and respond to client queries on configuration. In a server-less design, however, there is no such central server, and the information needs to be maintained in a distributed manner.

2. **Message transportation**: Server-based Providers typically use unicast communication for message transportation. Message producers send messages to the server, which stores and forwards them to the consumers, either a single receiver for the queuing model or several subscribers for the pub/sub model. This store-and-forward method is unsuitable for a MANET-based Provider as there can be no central message delivery mediator.

3. **Reliability**: Guaranteed message delivery in server-based JMS Provider is achieved by the server persisting the messages in a central database. Since the server manages the communication between all clients, it knows when a message has been consumed by all intended recipients and can be discarded from the database. Such a central database is not readily available in a MANET

environment, as the availability of any one node at all times cannot be relied upon. This greatly increases the complexity involved in achieving guaranteed message delivery.

4. **Coordination**: As previously noted, if there are several receivers associated with a queue, then for each message, the provider has to select a single receiver to consume the message. With a server-based architecture it is straightforward for the server to make this decision. In a server-less architecture, distributed coordination mechanisms are required if fair, non-simplistic queue semantics are to be achieved (i.e. the messages are evenly distributed among the associated receivers). Such mechanisms require the use of an agreement (consensus) protocol. Note that the consensus problem cannot be deterministically solved in an asynchronous environment [23] and protocols are possible only when certain assumptions restricting asynchrony are possible to make.

In the following section, we describe how we addressed the first two of these issues, resulting in the construction of JMS system for MANETs with pub/sub non-persistent message delivery semantics. Section 6 addresses how the third issue can be solved and incorporated into the implemented system. We then indicate the ways of solving or circumventing the distributed coordination problem in MANETs. Our ongoing work involves the former.

4. Designing a JMS Provider for MANETs

In order to provide a complete JMS solution for MANETs, the design issues described in the previous section would all have to be dealt with.
The next two sections describe how we have designed a system based on an existing messaging product, which provides non-persistent JMS topic semantics by addressing the two primary issues (configuration and message transportation) required in order to develop a JMS solution for MANETs.

4.1 Configuration

The server-less nature of our JMS solution leads to the somewhat meta-physical question: when do queues and topics come into being, and how do the JMS clients find out about them? In traditional server-based JMS providers, an administrator creates a destination on the JMS server. It thus resides on the server as a tangible entity from creation until it is destroyed by an administrator. All clients then typically use JNDI to lookup by name, the queues and topics they require. With MANETs it is not so straightforward, as there is no central entity to lookup topics and queues from.

The approach taken in our system is to require all clients participating in a message exchange to have a local copy of an identical configuration file. This configuration file then contains all information the client needs to lookup and use the destination. This resolves both of the above issues, as a lookup of destinations only requires reading the configuration file, and a queue or a topic will have existed "forever" as far as the client is concerned. Although inflexible, it is worth considering that the creation of queues and topics on server-based JMS providers requires the intervention of an administrator on the JMS Server. Of course, there are other possible and more flexible solutions, including broadcasting destination information on a well-known multicast address, but these are outside the scope of this paper.

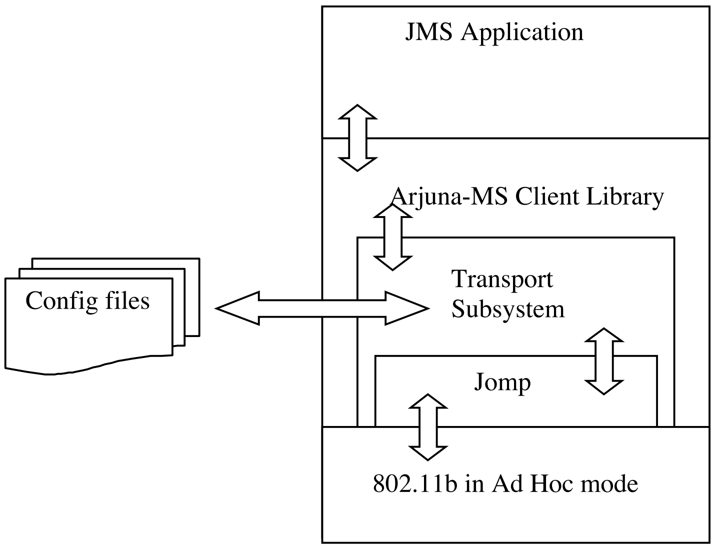
4.2 Message transportation

In order to provide the required message transportation, an application-level implementation of a multi-hop routing protocol had to be developed. The reason for this is the lack of readily available MANET routing protocols integrated into the network layer in most commercially available mobile devices. As the main goal of this research was to provide JMS *topic* semantics to MANETs, it was decided to implement a *multicast* multi-hop routing protocol. In multicast, the key concept is that of a multicast group, which any node can send and receive packets to and from. In order for a packet to be delivered to all the members of a multicast group, a node only needs to send a packet once to a given multicast address. Clearly this functionality closely mirrors that of JMS topics, where a message producer only needs to send a message once to a given topic in order for all subscribers on that topic to receive the message.
Several multicast protocols specifically designed for MANETs have been proposed. These include ODMRP[9], AMRIS[10], CAMP[11] and MAODV[12]. ODMRP, or the On Demand Multicast Routing Protocol, was chosen because it has performed well with regards to throughput and control packet overhead in several simulation studies[13, 14].

A Java application-level implementation of ODMRP has been developed. This implementation has been named the *Java on-demand multicast routing protocol (Jomp)*. Further details on Jomp can be found in the next section.

After developing Jomp, the required multicast functionality was now available for MANETs, and the Arjuna-MS client library has been extended to map between JMS publish-subscribe semantics and the multicast functionality provided by Jomp. Figure 1 shows a conceptual view of the Arjuna-MS client library and how it uses Jomp and its configuration file (as described above) to provide JMS messaging on MANETs.

Fig. 1. Conceptual view of the Arjuna-MS client library for MANETs.



5. Implementing a JMS Provider for MANETs

This section describes in more detail the implementation aspects of extending Arjuna-MS to MANETs. As mentioned in section 4.2, a Java application-level version of the ODMRP protocol (Jomp) was implemented in order to provide the required message transportation for the system. Section 5.1 describes the architecture and programming interface to Jomp, while section 5.2 describes our experiences implementing an application-level routing protocol in Java. A complete description of the system is available in [24].

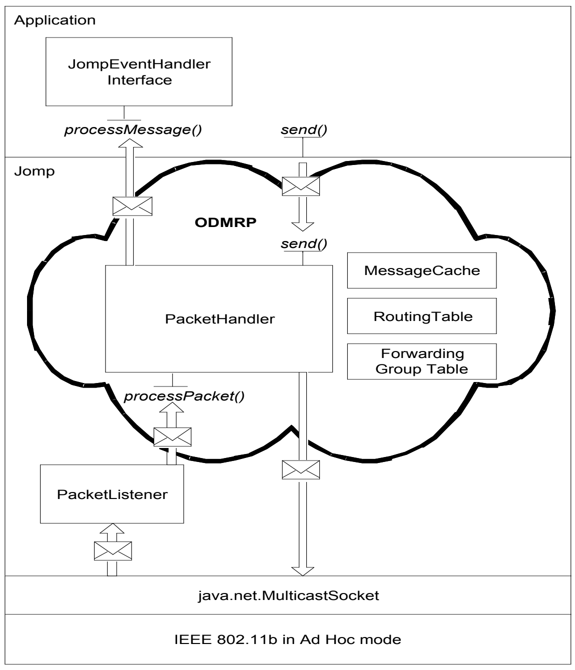
5.1 Jomp architecture and programming interface

Jomp was designed to provide a clean API for asynchronous message passing. This was achieved by designing a *Jomp object* for use when sending packets and a *JompEventHandler interface* which an application developer had to implement when using Jomp.

Jomp consists of 5 main components: the *PacketListener*, which is a thread that continuously listens on the network for any packets and passes any received packets up to the *PacketHandler*. The PacketHandler contains most of the ODMRP functionality and uses the *MessageCache, RoutingTable* and *ForwardingGroupTable* objects to make routing decisions (The last 3 are tables used by ODMRP, for further detail, see [9]). These routing decisions may result in the PacketHandler broadcasting a packet, passing the packet up to the application at the local host or discarding the packet.

Figure 2 shows a conceptual view of the architecture of Jomp, showing how these components interact and how Jomp interacts with the application using it and the network. The large "ODMRP cloud" indicates where the ODMRP protocol functionality is implemented.

Fig. 2. Jomp architecture



5.2 Experiences implementing a Java application-level routing protocol

Implementing anything in Java always raises the issues of performance, particularly when implementing something traditionally as low level as a routing protocol. However, based on the fact that Jomp is an application-level protocol, and on the observation that previous application-level implementations of routing protocols in Java, such as LRMP[15], has adequate performance, we have come to the conclusion that performance in most cases are not adversely affected by the choice of an interpreted programming language.

A more pertinent issue when using Java is the lack of direct access to devices such as the 802.11b network card. Most simulations and implementations of MANET routing protocols imply that the network card has to be set to promiscuous mode, where it will receive packets destined to all addresses. This is not possible in Java, and could have been problematic, as without a work-around, only nodes interested in a given multicast address would be aiding in forwarding packets on that multicast group.

The implication of this with respect to using Jomp as the transport for Arjuna-MS is that a node will only aid in forwarding messages destined for topics that the node is subscribed to. Clearly this is not beneficial for the MANET as a whole. In fact the problem with "selfish" nodes is something that has been highlighted as a problem for a number of routing algorithms and applications in MANETs[16].

However, we worked around this limitation by sending all packets on a single, configurable multicast address. The packets that were sent on this address were all tagged with the relevant multicast addresses mapping to the topic to which they were destined. This setup means that all nodes running Jomp will participate in the routing and forwarding of all packets, not just packets destined for the multicast addresses they were interested in. It is thought that there is no performance hit due to this, as the sending and receiving of multicast packets using an 802.11b network card is the same no matter what multicast address the packets are sent to.

6. Simulation and real world deployment

Testing real world implementations of any type of application built for MANETs presents a number of difficulties due to the dynamic nature of the MANET environment. One of the main problems is the lack of control the tester has over the accuracy of test parameters, with issues such as interference from other radio sources and the difficulties in managing a (potentially high) number of mobile units making the testers task very difficult indeed. Because of this, a network simulator, the Java Network Simulator (JNS)[18], has been modified to enable it to use real world implementations as nodes in a simulation. This allows tests of real world MANET applications to be carried out in a controlled environment. The next section describes how we used JNS to verify the correct operation of Jomp and to run a simple JMS application.

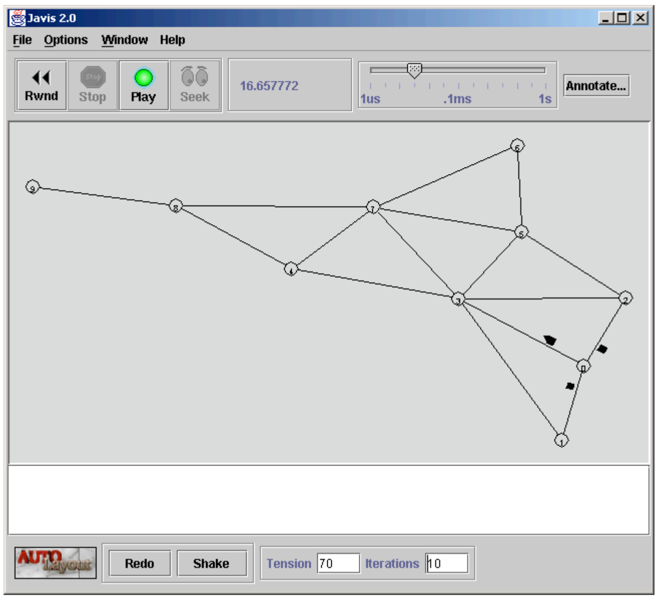
In addition to this verification through simulation, we realize that any MANET middleware needs to be tested in the real world. As an indication of the potential of Arjuna-MS for MANETs we have deployed Arjuna Chat Demonstrator, the instant messaging demonstration application that ships with the standard, wired version of Arjuna-MS, in a real world scenario. Details of this deployment are described in section 6.2.

6.1 Testing

As ODMRP has already been shown in literature [13,14] to perform well compared to other MANET multicast protocols, the emphasis of the testing was to satisfy ourselves that Jomp adhered to the ODMRP specification.

In order to verify the correct operation of both Jomp itself and Arjuna-MS using Jomp, we setup a simulation environment with 10 nodes with partial connectivity, as could be found in a MANET at any instant in time. In this scenario, we made one of the nodes publish messages to 9 subscribers. The trace files describing the message flow between the nodes verify that Jomp act according to the ODMRP specification. Additionally, all the JMS subscribers received the published message, which shows that Arjuna-MS using Jomp also works. Figure 3 shows the initial broadcast from the source node. Space limitations prevent us from presenting the complete sequence of message exchanges, but a video clip showing the interaction and several screenshots is available at [19].

Fig. 3. A simulation run of Arjuna-MS running on JNS



6.2 Proof of concept

In addition to the test runs on JNS, we have also deployed quite a large JMS application, the Arjuna Chat Demonstrator instant messaging demo application, in a real world scenario. This involved almost no change in the code base of the application, which shows the advantage of providing standard middleware for MANETs. The application, although not extensively tested, was successfully used in an office environment with several instances of the application running. Some anecdotal evidence of this successful deployment is available at [19].

7. Extensions and future work

7.1 Reliability

Currently, our JMS solution does not provide persistent messaging. This is because Jomp is an implementation of the basic operation of the ODMRP protocol, which only provides best effort message delivery guarantees.

Typical server-based JMS providers usually implement persistent messaging by putting the messages on stable storage on a central server. However, there are weaknesses even with this approach, as the assumption made when making a message persistent on the server is that the disk will not fail. This assumption is not necessarily 100% accurate in all situations, as disks might indeed fail. However, the underlying assumption is that the level of fault tolerance is dependant on the environment in which the JMS provider is operating. This implies that in an inherently unreliable environment such as MANETs, a JMS provider does not need to provide the same level of fault tolerance as it would do in for example a wired environment where the JMS provider could make use of for example RAID disks or off-site replication.

Based on this observation, if it is possible to increase the packet delivery ratio of Jomp to a high enough level, it could be argued that both of JMS' message delivery guarantees could be provided by Arjuna-MS over Jomp.

One method of extending Jomp to reach the required level of reliability is by adding *Anonymous Gossip (AG)* [20] to Jomp. In general, gossip is the name given to the technique where nodes share the data they possess with other nearby nodes, in order for all nodes to share a consistent view of what has been communicated.

Simulation has shown[20] that the addition of Anonymous Gossip to an unreliable multicast protocol will substantially increase the reliability of the protocol. Based on this observation, it is our opinion that adding Anonymous Gossip to Jomp would sufficiently increase the reliability of Jomp in order for us to be able to claim that Arjuna-MS on MANETs support persistent messaging.

7.2 Coordination

Extending Arjuna-MS on mobile ad-hoc networks to support queues is a different matter altogether, and the main problem is achieving the required semantics of a JMS queue without incurring too much of an overhead.

Broadly speaking there are two main approaches to achieving JMS queue semantics. The first is to use a multicast protocol implementation, such as Jomp, to disseminate a message sent to a queue to all nodes containing receivers on that queue. The nodes would then need to agree amongst themselves which one receiver is to be handed the message. This would be achieved through some sort of consensus algorithm, for example through the use of a distributed leader election algorithm, where the elected leader would pass the message up to its queue receiver. This approach has the benefit that we can use Jomp without modification and that fair, non-simplistic queue semantics is achieved. However, it has the drawback that comes with any currently available consensus algorithms for MANETs, in that it can be very costly both in terms processing power and network bandwidth.

The other approach is to weaken the fairness of JMS queue semantics and thus circumventing the distributed coordination problem. One way to do this is to handle all synchronization on the sender side. For example, JMS queue semantics could be achieved by having the node containing the queue sender know the identity of all the nodes with queue receivers. The node containing the queue sender would then simply pick one queue receiver and unicast the packet to that receiver. This approach has the benefit that no processing is required on the nodes with the queue receivers, and the required bandwidth is substantially less compared to the first approach. However, this approach does require the implementation of a unicast protocol, as well as requiring the nodes with queue senders to maintain some sort of membership information of the queue receivers. This in itself adds an overhead both in processing and network bandwidth. Another way to circumvent the coordination problem is through the use of an *anycast protocol*. Anycasting protocols are a fairly recent development in the mobile networking community, and essentially provide one-to-any delivery semantics. That is, a packet that is being anycasted will be delivered to only one of a group of nodes, typically the closest to the sender. An example of an anycast protocol is the extensions made to the Temporally Ordered Routing Algorithm (TORA)[21] when it was extended to support geocasting (GeoTORA)[22].

Clearly the two approaches circumventing the distributed coordination problem are fairly similar, diluting the message queuing semantics and fairness in order to reduce overhead, and choosing between these two and the use of a consensus protocol will have to be based on the desired properties of the system.

Our current research involves developing consensus algorithms specifically for MANETs, which aim to reduce the required network and processing bandwidth, thus making it more suitable for the MANET environment, and perhaps making the above choice a little easier.

8. Conclusion

Mobile ad-hoc networking is likely to become an important future communication technology; many interesting application areas have been suggested, including applications in vehicular and sensor networks. However, the development of such applications is currently held up by the unavailability of MANET-capable mobile devices. Current wireless devices, such as PDAs with 802.11b networking, do not support the necessary multi-hop routing protocols as standard.

Despite such disadvantages and due to potential application benefits, this paper has met the objective of designing and implementing a Java Message Service (JMS) solution for MANETs using an application-level multicast routing protocol. Since JMS is known to simplify the construction of loosely coupled distributed applications using familiar APIs and messaging models, the provision of a JMS solution for MANETs greatly eases the effort required to build MANET applications.

Our implementation was integrated into the Arjuna Message Service product as a pluggable transport module. Currently, our JMS solution for MANETs supports non-persistent publish/subscribe messaging. Several existing JMS applications have been successfully deployed on top of it without the necessity for any code changes. Further verification of the correct operation of the underlying multicast routing protocol was obtained by extending an existing network simulator (JNS) to trace the behaviour of the executing JMS applications. Ideas for future work include support for the queuing messaging model through the development of distributed consensus algorithms for MANETs and reliable delivery through the use of message replication using the anonymous gossip protocol.

References

1. Charles E. Perkins, Elizabeth M. Royer, and S. Das,*Ad Hoc On Demand Distance Vector (AODV) Routing*. draft-ietf-manet-aodv-10.txt, 2002
2. Johnson, D.B. and D.A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, in *Mobile Computing*, H.F. Korth, Editor. 1996, KluwerAcademic Publishers Group. p. 153-179.
3. Richard E. Schantz and D.C. Schmidt, *Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications*, in *Encyclopedia of Software Engineering*. 2001, Wiley & Sons.
4. Banavar, G., et al., *A Case for Message Oriented Middleware*, in *Distributed Computing*, P. Jayanti, Editor. 1999, Springer Verlag Kg. p. 1-18.
5. Mark Hapner, et al., *Java Message Service*. Version 1.0.2b, http://java.sun.com/products/jms/docs.html, 2001
6. *IBM website*. http://www.ibm.com,
7. *Tibco website*. http://www.tibco.com,
8. *Arjuna website*. http://www.arjuna.com
9. Lee, S.J., M. Gerla, and C.C. Chiang. *On-Demand Multicast Routing Protocol*. in *Wireless communications and networking conference*. 1999. New Orleans; LA: IEEE Operations Center.
10. Wu, C. and Y. Tay, *AMRIS: A Multicast Protocol for Ad Hoc Wireless Networks*. Milcom, 1999. **1**: p. 25-29.
11. Garcia-Luna-Aceves, J.J. and E.L. Madruga, *The Core-Assisted Mesh Protocol*. Ieee Journal on Selected Areas in Communications Sac, 1999. **17**(8): p. 1380-1394.
12. Elizabeth M. Royer and C.E. Perkins, *Multicast Ad hoc On- Demand Distance Vector (MAODV) Routing*. Work in progress, 2000
13. Kunz, T. and E. Cheng. *On-Demand Multicasting in Ad-Hoc Networks: Comparing AODV and ODMRP*. in *Distributed computing systems*. 2002. Vienna: IEEE Computer Society.
14. Lee, S.J., et al. *A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols*. in *Computer communications; IEEE INFOCOM 2000*. 2000. Tel Aviv, Israel: Ieee.
15. Liao, T., *WebCanal: a multicast Web application*. Computer Networks and Isdn Systems, 1997. **29**(8/13): p. 1091-1102.
16. Sonja Buchegger and J.-Y.L. Boudec. *Performance Analysis of the CONFIDANT Protocol*. in *MobiHoc*. 2002. Lausanne, Switzerland.
17. S-J Lee, William Su, and M. Gerla, *On Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks*. http://www.ietf.org/proceedings/00jul/ID/ manet-odmrp-02.txt, Work in progress, 2000

18. *The Java Network Simulator (JNS)*. http://jns.sourceforge.net,
19. Arjuna-MS for MANETs webpage http://www.cs.ncl.ac.uk/people/einar.vollset/home.formal/arjunamanet.html, 2002
20. Chandra, R., V. Ramasubramanian, and K. Birman, *Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks*. International Conference on Distributed Computing Systems, 2001. **21**: p. 275-283.
21. Park, V.D. and M.S. Corson, *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*. Ieee Infocom, 1997. **3**: p. 1405-1413.
22. Ko, Y.B. and N. Vaidya. *GeoTORA: A Protocol for Geocasting in Mobile Ad Hoc Networks*. in *Network protocols*. 2000. Osaka, Japan: IEEE Computer Society.
23. M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with one faulty Process," Journal of the ACM, Vol. 32, No. 2, pp. 374-382, April 1985
24. Einar Vollset, "Extending an enterprise messaging system to support mobile devices", MSc thesis, University of Newcastle upon Tyne, September 2002.