JOMS: a Java Message Service Provider for Disconnected MANETs

Abdulkader Benchi, Frédéric Guidec, Pascale Launay

JOMS: a Java Message Service Provider for Disconnected MANETs

Abdulkader Benchi, Frédéric Guidec, Pascale Launay
IRISA, Université de Bretagne-Sud
Vannes, France

***Abstract*—A disconnected mobile ad hoc network (or D-MANET) is a wireless network, which because of the sparse distribution of mobile hosts appears at best as a partially or intermittently connected network. Designing and implementing distributed applications capable of running in such a challenged environment is not a trivial task. Middleware systems such as Java Message Service (JMS) have made application development easy and cost-effective in traditional wired networks. It can be expected that middleware systems designed specifically for D-MANETs bring similar benefits. In this paper, we introduce JOMS (Java Opportunistic Message Service), a JMS provider for D-MANETs with which pre-existing and new JMS-based applications can be deployed simply in D-MANETs.**

***Index Terms*—Java Message Service, JMS provider, Message Oriented Middleware, disconnected MANET**

I. Introduction

In a mobile ad hoc network (or MANET), mobile devices can communicate with one another using direct wireless transmissions. Because the range of these transmissions is often quite short, a number of protocols (such as OLSR, AODV, DYMO, DSR...) have been developed during the last two decades in order to support multi-hop forwarding in MANETs. Yet most of these protocols rely on the assumption that the network remains connected, that is, a temporaneous end-to-end path exists at any time between any pair of hosts in the network. Unfortunately this assumption does not hold in many MANETs that are, at best, only partially or intermittently connected.

Figure 1 illustrates a typical disconnected MANET (or D-MANET). This network appears as a collection of distinct "*islands*" (or connected parts of the network) rather than as a single, connected network. Hosts can communicate within each island, but no temporaneous communication is possible between different islands. The concept of delay/disruption-tolerant networking makes it possible to bridge the gap between islands, though, using mobile hosts as message carriers as they move in the network [1]. A message can thus be *stored* temporarily on a host, and be *carried* for a while by this host before being *forwarded* to another host when circumstances permit. In Figure 1, a smartphone or laptop carried by a user moving –deliberately or by chance– from island 1 to island 2 can serve as a carrier for messages addressed to hosts located in island 2. Based on this "*store, carry and forward*" model, connectivity disruptions can be tolerated. Yet this approach yields long transmission delays because it depends on the –usually non-predicted– mobility of hosts.

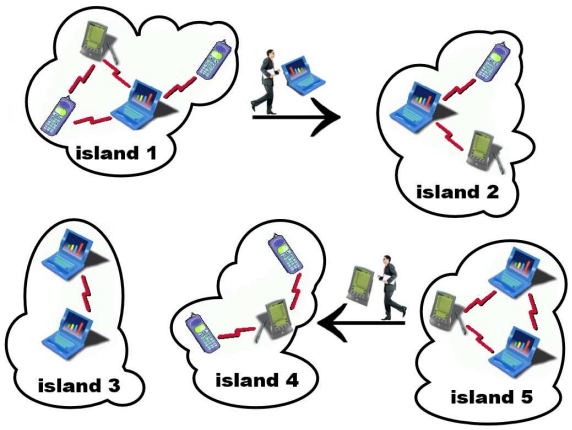

Figure 1. Example of a disconnected mobile ad hoc network

The term *opportunistic networking* is usually used in the literature to denote this kind of networking where non-predicted radio contacts between mobile hosts are used by these hosts to exchange messages, and thus contribute to the propagation of messages network-wide. A number of opportunistic networking protocols have been developed for D-MANETs during the last few years [2]. Yet developing distributed applications capable of running based on these protocols is not a trivial task: the dynamic nature of D-MANETs, long transmission delays, occasional transmission failures all constitute serious challenges developers must face. As a general rule, when designing an application for D-MANETs the peer-to-peer model should be preferred over the client-server one, because in most circumstances no host can be considered as being stable and accessible enough to play the role of a server for all other hosts.

In traditional networking environments the concept of middleware gained popularity as a solution to ease the development of distributed applications. It can be expected that middleware systems designed specifically for D-MANETs should bring developers similar benefits. According to Hurwitz [3] there are four main types of middleware: transactional, procedural, message-oriented and object-oriented middleware. The asynchronous message-passing feature of message-oriented middleware makes it an appropriate model for D-MANETs, for the long transmission delays imposed by the "store, carry and forward" scheme can be easily tolerated through asynchronous messaging.

In the remainder of this paper we present JOMS *(Java Opportunistic Message Service)*, a message-oriented middleware system we designed and deployed for D-MANETs. JOMS is actually a provider for the standard Java Message Service (JMS), so the developers of JMS applications can use JOMS just like they use any other JMS provider.

This paper is structured in the following way: an overview of the JMS specification is provided in Section II. Section III presents the general architecture and principles of our system JOMS, and details about the implementation of this system are provided in Section IV. Related work is discussed in Section V. Section VI concludes this paper, and describes our plans for future work.

II. Java Message Service (JMS)

The Java Message Service (JMS) is a Message-oriented Middleware (MoM) standard that allows application components based on the Java 2 Platform Enterprise Edition (J2EE) to create, send, receive, and read messages. Since it is a MoM standard, it supports distributed communication in a loosely-coupled, reliable, and asynchronous manner [4]. In general, MoM enables a *loosely-coupled* type of distributed communication. A client sends (*produce*s) a message to another client, which can then try to retrieve (*consume*) it *asynchronously*. The producer and the consumer do not have to be available at the same time in order to communicate. In fact, the producer does not need to know anything about the consumer, nor does the consumer need to know anything about the producer. This brings a major benefit to MoM, where the producer and the consumer need to know only what message format and what channel to use in order to communicate.

The JMS API defines a common set of interfaces including the associated semantics that allows programs written in Java to communicate. The JMS specification does not define how messages are transported within a particular implementation, known as a *JMS provider*. Because of the lack of unified implementation, each major vendor proposes its own JMS provider along with the associated management tools. Each JMS provider supplies the user with an appropriate transport technology for a particular deployment environment.

The JMS API defines two communication models: point-to-point and publish-subscribe. The point-to-point model is built around the concept of *queues*. A *queue sender* sends a message to a specific queue, from which a *queue receiver* can receive it asynchronously. This model provides a *one-to-one* communication model. In other words, a given queue may have multiple senders and multiple receivers, but each message sent by a sender to this queue is consumed by one receiver. This means that some mechanism is required to decide which receiver candidate will be the actual receiver of a given message.

The publish-subscribe model is based on the use of *topics* that can be subscribed to by *topic subscribers*. Messages are published to a topic by *topic publishers* and are then received in an asynchronous mode by all the corresponding topic subscribers. Each message may thus be consumed by multiple subscribers. This model complements the point-to-point model in that it provides a *one-to-many* communication model.

JMS supports two delivery semantics, through the so-called persistent and non-persistent delivery modes. A non-persistent message should be delivered in a best-effort mode. Conversely, a persistent message must be delivered in a guaranteed mode. The JMS specification does not define how messages are transported. It is up to the JMS provider to define how these two types of delivery semantics are implemented.

III. Java Opportunistic Message Service (JOMS)

JMS was primarily designed for systems where clients connect to central servers via traditional networks and this has remained its typical usage scenario. In most implementations of JMS, message producers send messages to a server that *stores* these messages and *forwards* (delivers) them later to the consumers. As explained in Section I a server-based model is hardly compatible with the characteristics of D-MANETs. No host can act as a reliable server for all other hosts. A serverless JMS implementation must thus be developed in order to provide JMS services in D-MANETs.

JOMS, or Java Opportunistic Message Service, is a JMS provider that was designed along that line. Its architecture is composed of two basic modules: a communication middleware system and the JMS provider per se.

A. Communication layer

Building any application for D-MANETs requires some communication middleware system with which mobile hosts can collaborate in a peer-to-peer manner to ensure message transportation. JOMS relies on a communication middleware system called DoDWAN *(Document Dissemination in mobile Wireless Ad hoc Networks).* This system was designed in our laboratory in order to support content-based information dissemination in D-MANETs [5].

Messages in DoDWAN are composed of two parts: a descriptor and a payload. The payload is simply perceived as a byte array. The descriptor is a collection of attributes expressed as *(name, value)* tuples, as illustrated in Fig. 2. These attributes can be defined freely by the developers of application services built on top on DoDWAN. The only exceptions to this rule are a message identifier and a deadline, that must systematically appear in any descriptor. The identifier must be unique, for it allows DoDWAN to differentiate messages while detecting duplicate copies of the same message. The deadline is meant to specify how long a message should be allowed to disseminate in the network, and therefore how long copies of this message should be stored by mobile hosts in their local cache.

DoDWAN provides application services with a publish/subscribe API. When a message is published by a local application service, it is simply put in the local cache maintained by DoDWAN. Afterwards each radio contact with another host will be an opportunity for DoDWAN to transfer a copy of the message to that host.

id= “ff789”
destination_id= “ChatRoom1”
destination_type= “topic”
date= “Mon Oct 17 20:54:03 CET 2011”
deadline= “Fri Nov 18 20:54:03 CET 2011”
delivery_mode= “PERSISTENT”
priority= “4”
language= “English”
locked= “true”

Figure 2. Example of a message descriptor

In order to receive messages an application service must subscribe with DoDWAN and provide a *selection pattern* that characterizes the kind of messages it would like to receive. A selection pattern is expressed just like a message descriptor, except that the *value* field of each attribute contains a regular expression. Fig. 3 shows a selection pattern, which would for example match the message descriptor shown in Fig. 2.

The selection patterns specified by all local application services running on the same host define this host's *interest profile*. DoDWAN uses this profile to determine which messages should be exchanged whenever a radio contact is established between two hosts. The interaction scheme implemented in DoDWAN takes inspiration from the Autonomous Gossiping (A/G) algorithm [6], which itself defines a selective version of the epidemic routing model proposed in [7]. Each host periodically broadcasts an announcement in order to inform its neighbors (if any) about its identity and interest profile. By sending such an announcement periodically, a node informs its neighbors about its presence and about the kinds of messages it is interested in. Conversely, by receiving similar announcements a host discovers its neighbors, and learns about their own interest profiles. By matching its neighbor's profiles against the descriptors of the messages it maintains in its cache, a host can select descriptors of messages that might be of interest to at least one of its current neighbors. It can thus build a catalog containing these descriptors, and incorporate this catalog in its next announcement. Upon receiving such a catalog, each host matches the descriptors it contains against its own interest profile in order to identify messages that match this profile and that are not already present in its local cache. If such messages are identified, then a request for these messages is sent to the announcer, which complies by sending the missing messages on the radio channel. Finally, when a host receives a message it has requested, this message is put in the local cache so it can later be proposed to other hosts met while moving in the network.

destination_id= “ChatRoom.*”
language= “English|Chinese|German”

Figure 3. Example of a selection pattern

As a general rule, a host that subscribes to receive a particular kind of message is expected to serve as a mobile carrier for this kind of message. Yet a host can also be configured so as to serve as an altruistic carrier for messages that present no interest to the application services it runs locally. This behavior is optional, though, and it must be enabled explicitly by an administrator of the DoDWAN platform.

Mobile hosts running DoDWAN only interact by exchanging control and data messages encapsulated in UDP datagrams, which can themselves be transported either in IPv4 or IPv6 packets. When a message is published on a host, its descriptor and its payload are both compressed in order to reduce the bandwidth required for their transmission. Large messages are additionally segmented so that each fragment can fit in a single UDP datagram. Fragments of a large message all contain a copy of the original message's compressed descriptor, so they can propagate independently in the network and be reassembled only on destination hosts, where the payload is eventually uncompressed.

Interactions between neighbor hosts rely on an opportunistic scheme rather than on a strict transactional scheme. No session –and especially no TCP session– is ever established between neighbor hosts because of the high level of connectivity disruptions expected between these hosts. Each host only maintains soft-state information about its neighbors. Thus, whenever a host broadcasts an announcement, for example, some of its neighbors may fail to receive this announcement, without ever compromising either the sender or any potential receiver. Likewise, whenever a host requests a message and fails to obtain this message, it simply waits until it can get another chance to grab this message (either from the same neighbor, or from a different one).

B. JMS provider

A JMS provider supports the *publish-subscribe* and the *point-to-point* styles of messaging. This Section describes the message model of JOMS and the way it supports the two models of communication.

1) Message model: according to the JMS specification [4], a JMS message has three parts: a header, properties, and a body. The JMS message header contains fields used by both clients and providers to control messages. The properties are extra header fields that act as a set of rules describing the message content. They are used by clients to filter messages via message selectors. It is worth noting that selection criteria cannot reference the message body, that carries the message content.

A DoDWAN message has two parts: a descriptor and a payload. Since JOMS is based on DoDWAN, it adopts this message model by mapping the JMS message's fields to the DoDWAN message. An example of a JOMS message is shown in Fig. 2. The JMS message's body is carried in a DoDWAN message as its *payload*, and considered as a simple byte array. The message descriptor is used by DoDWAN to manage the message dissemination and delivery, by selecting carriers or recipients whose interest profile match the descriptor. The JMS message's header and properties are mapped to the DoDWAN message's descriptor, as their content is needed by JOMS to

process the messages delivery. *MessageID* and *Destination* are standard JMS header fields used to identify and route the message. The message *identifier* is needed by DoDWAN to differentiate messages and avoid the dissemination of duplicate copies of the same message. The *destination name* is used by JOMS to route the message to its recipients having this criterion in their interest profile. We will explain later how these profiles are defined in JOMS to allow publish-subscribe and point-to-point communications. As those communication styles are quite different and implemented in JOMS using distinct models, an extra property, the *destination type*, is added to the JMS initial message. The JMS *Expiration* field is mapped to the DoDWAN message and called *deadline*. This field, optional according to the JMS specification, is mandatory while using DoDWAN, as it is used to avoid the overloading of radio channels and hosts' caches with out-of-date messages. Its value is set to a default value by JOMS if the JMS expiration field's value is zero. According to the JMS specification, the *DeliveryMode* and *Priority* properties express the expected degree of reliability and priority for transmitting messages. Given the disconnected nature of the environments targeted by JOMS, it is not possible to ensure reliability as defined by JMS. JOMS uses these properties to increase the delivery probability for the most important messages by modifying the DoDWAN's cache management policy in order to give them more chances to be opportunistically disseminated. Messages with a persistent delivery mode are favoured over non-persistent ones, and then the priority property is taken into account. All extra fields composing the JMS message's properties part are carried by the DoDWAN message's descriptor as they can act as a message selection criterion for DoDWAN while implementing the JMS selector mechanism.

2) Publish-subscribe model: this model is very close to the publish-subscribe API provided by DoDWAN. Usually, JMS providers implement this communication pattern using a server-based model: the publications and subscriptions to a given topic are managed by a central entity. However, the implementation of publish-subscribe communications using a server-less model is quite obvious and well suited: messages published to a given topic are disseminated over the network; thus, any application service interested in this topic is given the opportunity to receive its messages. DoDWAN supports content-based dissemination, rather than destination-based routing of messages. Therefore, JOMS tags a message published to a given topic with the topic name, and then publishes it using DoDWAN; DoDWAN manages the message dissemination and the message delivery to all interested hosts. JOMS expresses applications' interest in receiving messages published to a given topic (*topic subscribers*) by adding the topic's name in their interest profile. Moreover, JMS selectors, allowing topic subscribers to filter the messages they receive, are added to the applications' interest profile; thus, the message filtering is processed at the communication middleware level.

The JOMS message shown in Fig. 2, for example, is published to the topic "ChatRoom1". This read-only text message, labeled "ff789", has the priority 4 and is to be delivered in persistent mode. It has been published at "Mon Oct 17 20:54:03 CET 2011" and will die at "Fri Nov 18 20:54:03 CET 2011". The message selector "language= English" is a set of keywords characterizing this message.

destination_id= "MailBox1@00b0d086bbf7"
destination_type= "queue"

Figure 4. Example of a queue manager's interest profile

In D-MANETs, disconnections are the norm rather than the exception. As a result, the implementation of the JMS *non-durable* subscriptions concept, where messages are delivered only to active subscribers, is unsuitable and has no meaning for this environment. We deal with this problem by introducing a way to configure JOMS behaviour for non-durable subscriptions. By setting or unsetting some property, JOMS considers all non-durable subscriptions as durable ones, or refuses non-durable subscriptions and reports attempts to use them by throwing an exception.

3) Point-to-point model: this model is built around the concept of queue which has a central role in transmitting the messages from a queue sender to one and only one queue receiver. In fixed platforms, queues are maintained on a server which plays this central role in selecting the receiver of a message if there are multiple recipients associated with it. The main problem now is how to achieve this semantic of JMS queues in a D-MANET environment, where a server-based implementation is inappropriate, and where the consensus problem has not been however solved [8]. The approach to solve this problem is the so-called *quasi-central queue* approach: when an application creates a queue, its host will act as a queue manager (*QM*) for this queue. Thus, JOMS forwards to this QM all applications' requests to be receivers for this queue and all the messages sent to this queue. Then, it is up to the QM to decide to which receiver is to be handed the message, and to forward it using DoDWAN. Even if this QM is turned off or becomes unreachable, DoDWAN gives it a chance to receive later all the missing requests and messages by caching them on many other hosts. Thus, this queue acts as a *central decision-making* but not as a *central store*. This approach has the benefit that no consensus algorithm is required, thus making it more suitable for D-MANETs.

For the sake of illustration, a QM's profile is shown in Fig. 4. This profile matches all messages sent to the queue "MailBox1@00b0d086bbf7", which descriptors contain this property in the same way as the message shown in Fig. 2. Now, when an application wants to be a receiver for this queue, JOMS sends a request to the QM as shown in Fig. 5. The QM will use the *reply_id* property in order to address messages to that application, that has this property in its interest profile.

It is worth noting that each queue manager applies a selection policy in order to choose one receiver for each

destination_id= “MailBox1@00b0d086bbf7”
destination_type= “queue”
jms_general= “queue_receiver”
src= “86f8f700dad0”
reply_id= “host_1@86f8f700dad0”
language= “English”

Figure 5. Example of a request to be a queue receiver

Host 1

// Publish a message named “firstMSG” with payload “Hello”
// and selector “language=English” to “ChatRoom1”
% pub -t ChatRoom1 -id firstMSG -p Hello -ssl language=English
The Message has been published

Host 2

// Subscribe to the topic “ChatRoom1” to receive messages
// with properties “language=English or French”
% sub -t ChatRoom1 -ssl language=English|French
Waiting ....
You have received a new message with the content: Hello

Figure 6. Simple scenario

message. For each message, the queue manager chooses a receiver that matches the message properties in a fair way. The JOMS’s administrator can override this policy in order to have a more appropriate one regarding his requirements.

IV. Implementation details

The originality of our work lies in the fact that JOMS[1] and DoDWAN[2] have been fully implemented in Java and are now distributed under the terms of the GNU General Public License.

To date DoDWAN has been deployed and tested extensively using dozens of hand-held devices (such as laptops and smartphones) featuring Wi-Fi interfaces. It has also been used in a military tactical network involving VHF battlefield radios with built-in modems, and proved robust and reliable in such harsh conditions [9]. The scalability and stability of the algorithms DoDWAN relies on have been verified in simulations, using scenarios involving hundreds of mobile hosts [5].

JOMS implements Sun Microsystems’ Java Message Service API 1.1 specification on top of DoDWAN. An interactive command-line (console) is distributed with JOMS. The console is a tool that makes it easy and straightforward to test the performance of JOMS in real conditions. It provides the user with the most common JMS commands, i.e., destination-object management, publish/subscribe and send/receive commands. Using the console, JOMS has been tested using dozens of hand-held devices. For the sake of illustration, the following scenario demonstrates how to use the console. Assume that two hosts use JOMS. Fig. 6 shows commands executed on these two hosts along with the returned results.

JOMS is distributed with a number of example programs that demonstrate how to write simple applications. It is worth taking into consideration that nearly all JMS applications can be deployed perfectly over JOMS, even without changing their source code. This is an expected result of implementing a standard specification such as JMS. Application developers should take into consideration the characteristics of the networks supported by JOMS, where reliability, messages ordering and transmission delays cannot be guaranteed.

[1]http://www-valoria.univ-ubs.fr/CASA/JOMS
[2]http://www-valoria.univ-ubs.fr/CASA/DoDWAN

V. Related work

A number of JMS providers have been developed in the last few years in order to support JMS in MANETs.

EMMA (*Epidemic Messaging Middleware for Ad hoc networks* [10]) is an adaptation of JMS that targets MANETs presenting connectivity disruptions. EMMA assumes the availability of a so-called *synchronous protocol*, which can be used to reach mobile hosts that belong to the same *cloud* –or island– as the sender. An asynchronous epidemic routing protocol is used to disseminate messages towards remote clouds. EMMA manages queues in a manner that is quite similar to that of JOMS: each queue is maintained by a single holder, which advertises this object periodically with a set lifetime, and which can accept subscriptions from other hosts. EMMA and JOMS however differ in the way they deal with topics. In EMMA topics are managed just like queues, with a single holder per topic. In JOMS topic subscriptions can be set locally on any host. Messages published in a topic propagate in the network by being stored, carried and forwarded by all hosts that have subscribed to this topic. Other hosts can additionally contribute to the dissemination of such messages, provided they have been configured so as to behave as altruistic carriers. Another difference between EMMA and JOMS is that in EMMA the gossiping mechanism between neighbor hosts is done in such a way that all messages are considered, so very large lists of message identifiers can be exchanged between neighbor hosts. In JOMS this gossiping is content-based – and thus more frugal– since neighbor hosts only exchange messages based on their respective interest profiles.

Extended JMS –or E-JMS– is another JMS provider, that uses an application-level multicast routing protocol that provides publish/subscribe semantics by mapping JMS topics to multicast addresses [11]. Since this protocol cannot disseminate messages beyond a single connected fragment of the network, E-JMS is hardly usable in D-MANETs. It could probably be adapted, though, using a disruption-tolerant version of the multicast routing protocol. Another problem is that the authors signal persistence as possible future work for developing consensus algorithms. We argue that such assumptions restrict MANET asynchronicity and limit the usability of E-JMS in D-MANETs.

EMMA and E-JMS both define their own communication protocols. In contrast JOMS presents a two-layer architecture: the upper layer is concerned with queue and topic management and utilisation, while the lower layer supports opportunistic communication. For the lower layer JOMS currently relies on DoDWAN, a middleware system we designed to support content-based information dissemination in D-MANETs [5]. Yet JOMS could theoretically be implemented above any

other communication system, provided this system can operate satisfactorily in D-MANETs.

Although many communication protocols for D-MANETs have been proposed during the last decade, most of these protocols have only been described in papers as abstract algorithms, and tested using pseudo-code in simulators. Only a handful of these protocols have been actually implemented in middleware systems (and can thus be used in real conditions), and only a couple of these systems are openly distributed and are thus accessible to developers.

DTN2 is the name for a reference implementation of protocols designed by the Delay-Tolerant Networking Research Group (DTNRG), a research group chartered as part of the Internet Research Task Force (IRTF). The DTN architecture operates as an overlay network, forwarding contiguous data blocks named *bundles* in a store and forward manner towards nodes identified by an EID (Endpoint Identifiers) [12]. Several convergence layers are defined, so bundles can for example be transported between two nodes using TCP sessions, UDP datagrams, plain files, or any other convenient transport protocol. The DTN2 reference implementation is available and can easily be deployed on standard workstations or laptops. However, this system has not primarily been designed to target highly dynamic D-MANETs presenting short, unpredictable radio contacts between mobile hosts, so it is still unclear to us if it could run satisfactorily in such conditions.

Haggle is a content-centric architecture for opportunistic communication among mobile users (or devices) [13]. In fact Haggle and DoDWAN obviously share many common points. In both systems the information dissemination scheme is content-driven rather than destination-driven, and each mobile host can be characterised by an interest profile that determines the kinds of messages it is primarily interested in. A mobile host can additionally behave as an benevolent carrier for messages it is not interested in, although interesting messages are always favoured over less-interesting ones. DoDWAN and Haggle however differ significantly in the way interest profiles are dealt with. In Haggle the interest profile is disseminated network-wide, which may yield significant overheads in a large network involving hundreds or thousands of mobile devices. When the profile hence disseminated matches some data on another user's device, that device tries to *push* the matching data to the owner of the profile through a selected subset of its neighbors. In contrast DoDWAN does not attempt to disseminate profiles network-wide, but only up to a given horizon –defined as a maximum number of hops– around each host. Besides messages are not *pushed* by DoDWAN towards each potentially interested host, but a catalog of available messages (built according to their interest profiles) is proposed to all neighbors, and each neighbor can then request –or *pull*– the messages it is really interested in.

Another major difference between both systems is that Haggle was written in C and C++ (with language wrappers available for C# and Java) while DoDWAN was written directly in Java. DoDWAN is therefore highly portable, and readily interoperable with Java-based services such as JOMS.

VI. Conclusion

In this paper we have presented JOMS *(Java Opportunistic Message Service)*, a JMS provider we designed and implemented specifically for disconnected mobile ad hoc networks (D-MANETs). With JOMS pre-existing and new JMS applications can be deployed easily over D-MANETs, so the developers can simply focus on writing standard JMS applications which will be simply deployed over D-MANETs using JOMS.

JOMS is distributed under the terms of the GNU General Public License. It is currently compliant with version 1.1 of the JMS specification, which dates back to 2002. The next version of the JMS specification, namely JMS 2.0, should be issued at the end of 2012. JOMS shall be modified or extended so as to comply with this new specification. In future work we plan to add a directory service to JOMS so clients will be able to automatically discover queues and topics. We also plan to leverage on JOMS in order to implement other distributed programming abstractions for D-MANETs, such as tuple spaces and future objects.

References

[1] K. Fall, "A delay-tolerant network architecture for challenged internets." New York, USA: ACM, 2003, pp. 27–34.

[2] H. A. Nguyen and S. Giordano, "Routing in Opportunistic Networks," *International Journal of Ambient Computing and Intelligence (IJACI)*, vol. 1, 2009.

[3] J. Hurwitz, "Sorting out middleware," *DBMS*, vol. 11, no. 1, pp. 10–12, Jan. 1998. [Online]. Available: http://portal.acm.org/citation.cfm?id=284196.284204

[4] M. Hapner, R. Burridge, and R. Sharma, "Java message service, version 1.1," Apr. 2002.

[5] J. Haillot and F. Guidec, "A protocol for content-based communication in disconnected mobile ad hoc networks," *Journal of Mobile Information Systems*, vol. 6, no. 2, pp. 123–154, 2010.

[6] A. Datta, S. Quarteroni, and K. Aberer, "Autonomous gossiping: a self-organizing epidemic algorithm for selective information dissemination in mobile ad hoc networks," in *International Conference on Semantics of a Networked World*, ser. LNCS, no. 3226, Paris, France, Jun. 2004, pp. 126–143.

[7] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Duke University, Tech. Rep., Apr. 2000.

[8] 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, Apr. 1985. [Online]. Available: http://dl.acm.org/citation.cfm?id=214121

[9] J. Haillot, F. Guidec, S. Corlay, and J. Turbert, "Disruption-tolerant content-driven information dissemination in partially connected military tactical radio networks," in *28th IEEE Military Communication Conference (MILCOM'2009)*. Boston, USA: IEEE CS, Oct. 2009.

[10] M. Musolesi, C. Mascolo, and S. Hailes, "EMMA: Epidemic messaging middleware for ad hoc networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 28–36, Aug. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1103894

[11] E. Vollset, D. Ingham, and P. Ezhilchelvan, "Jms on mobile ad hoc networks," *In Personal Wireless Communications (PWC)*, pp. 40–52, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.63.4140

[12] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," IETF RFC 4838, Apr. 2007.

[13] E. Nordström, P. Gunningberg, and C. Rohner, "A search-based network architecture for mobile devices," Department of Information Technology, Uppsala University, Tech. Rep. 2009-003, Jan. 2009.