A Message Service for Opportunistic Computing in Disconnected MANETs

Abdulkader Benchi, Frédéric Guidec, Pascale Launay

A Message Service for Opportunistic Computing in Disconnected MANETs

Abdulkader Benchi, Frédéric Guidec, and Pascale Launay

IRISA-UBS, Université de Bretagne-Sud, 56000 Vannes, France

Abstract. Disconnected mobile ad hoc networks (or D-MANETs) are partially or intermittently connected wireless networks in which instant end-to-end connectivity between any pair of mobile hosts is never guaranteed. Recent advances in delay/disruption-tolerant networking make it possible to support communication in such conditions, but designing and implementing distributed applications for D-MANETs is still a challenging task. Middleware systems such as the Java Message Service (JMS) have made application development easy and cost-effective in traditional wired networks. In this paper, we introduce JOMS (Java Opportunistic Message Service), a JMS provider specifically designed for D-MANETs, and with which pre-existing or new JMS-based applications can be easily deployed in such networks.

Keywords: Java Message Service, JMS provider, Message Oriented Middleware, disconnected MANET, JNDI

1 Introduction

Mobile ad hoc networks (MANETs) have justified a fair amount of research activity during the last two decades. A MANET is a wireless network that requires no fixed infrastructure. Each mobile host can communicate with its neighbors using direct pair-wise wireless links. Because the range of wireless transmissions is often quite short, many protocols (such as OLSR, AODV, DYMO, DSR...) have been designed over the years in order to support multi-hop forwarding in MANETs.

Yet most of these protocols rely on the assumption that the whole network remains continuously connected, so that whenever one mobile host must send a message to another host, there actually exists at least one end-to-end path between these hosts in the network. Unfortunately this assumption does not hold in many real MANETs that are, at best, only partially or intermittently connected.

A MANET can for example become disconnected when the mobile hosts that compose this network are sparsely or irregularly distributed, as shown in Fig. 1. The whole network then appears as a collection of distinct, continuously changing "islands" –or connected components– rather than as a single, connected network. Communication between hosts that belong to the same island is possible –using multi-hop forwarding if needed– but no instant communication is

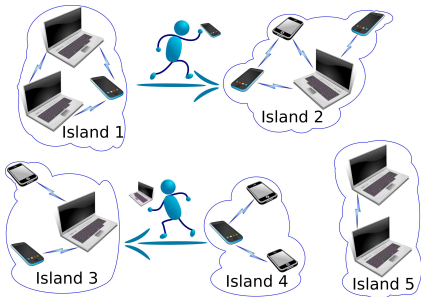


Fig. 1: Example of a D-MANET (disconnected mobile ad hoc network)

possible between hosts that reside on different islands. In such conditions, a method must be devised in order to bridge the gap between non-connected parts of the network. The mobility of hosts can here be considered as an advantage, as it makes it possible for messages to propagate network-wide, using mobile hosts as carriers –or *data mules*– that can move between non-connected parts of the network. A message can thus be stored temporarily on a host while this host is moving in the network, and be forwarded later to another host when circumstances permit. In Fig. 1, a smartphone or laptop carried by a user moving (deliberately or by chance) from island 1 to island 2 can serve as a data mule for messages addressed to hosts located in island 2. This *"store, carry, and forward"* approach fits in the general category of disruption-tolerant –or delay-tolerant– networking (DTN [1]): connectivity disruptions are indeed tolerated thanks to the mobility of hosts, but this approach generally yields long transmission delays because transporting a message from source to destination can take a long time when one or several data mules are involved during the transport. The term *opportunistic networking* is often used in the literature to denote delay/disruption-tolerant networking when radio contacts between mobile hosts are not planned in advance and must thus be exploited opportunistically. This is typically the case in many kinds of disconnected MANETs (or D-MANETs), where the mobility pattern of each host is actually that of its physical carrier. If the physical carriers in a D-MANET are vehicles or robots with pre-defined travel plans, then radio contacts between the hosts they carry can be predicted accurately, and routing strategies can be devised based on contact predictions. But if the physical carriers are for example human beings carrying laptops or smartphones, or animals carrying sensors, then radio contacts cannot always be predicted, although each contact is still an opportunity for two hosts to exchange messages.

Many papers published in the last few years address the problem of supporting communication in D-MANETs (good surveys can notably be found in [2] and [3]). In any case, it should be noticed that communication protocols for D-MANETs usually provide no more than best-effort delivery: they cannot guarantee that a message will be delivered. This is a consequence of the disconnected nature of the networks considered. Consider again the example shown in Fig. 1,

and assume a message is addressed by a host in island 1 to a host in island 5 (or to all hosts in the network, including those in island 5). If no human carrier ever visits island 5, then there is no chance that the message ever gets delivered in this island. A communication protocol running in a D-MANET can do no magic if mobile hosts do not move in such a way that messages can be transported between non-connected fragments of the network.

A D-MANET is therefore a network in which transmission delays can be very long, and transmissions can even fail altogether. Designing and implementing distributed applications capable of running in such conditions is quite a challenge. The peer-to-peer model should generally be preferred over the client-server model, because in many real D-MANETs no host can be considered as stable and accessible enough to play the role of a server for all other hosts. Additionally, any distributed application running in a D-MANET must obviously be able to tolerate long transmission delays, and occasional transmission failures as well.

The concept of middleware has long proved efficient in easing the development of distributed applications for "traditional" wired networks. It can be expected that carefully designed middleware systems might bring similar benefits for D-MANETs. According to Hurwitz there are four types of middleware, which are transactional, procedural, message-oriented and object-oriented middleware [4]. Message-oriented middleware appears to be the most appropriate model for D-MANETs because it generally relies on asynchronous message-passing, which fits perfectly with the long transmission delays that can be observed in such networks.

In the remainder of this paper we present JOMS *(Java Opportunistic Message Service)*, a message-oriented middleware system we designed and implemented specifically for D-MANETs. JOMS is actually a provider for the standard Java Message Service (JMS), so any pre-existing distributed application using the JMS API can be executed in a D-MANET with little or no further development. New applications can also be developed easily, taking benefit from the server-less directory service along with the administration tools provided with JOMS.

This paper is structured in the following way. An overview of the JMS specification is provided in Section 2. Section 3 presents the general architecture and principles of our system JOMS, whose implementation is then evaluated in Section 4. Related work is later discussed in Section 5. Section 6 concludes this paper, and describes our plans for future work.

2 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. As a MoM standard, it supports distributed communication in a loosely-coupled, reliable, and asynchronous manner [5]. A client sends (*produces*) a message to another client, which can then try to retrieve (*consume*) it *asynchronously*. The producer and the con-

sumer 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.

2.1 JMS API overview

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.

2.2 Java Naming and Directory Interface (JNDI)

According to the JMS specification, applications learn about the available topics and queues –or so-called *destination objects*– through the Java Naming and Directory Interface (JNDI). JNDI is a Java API that provides a common interface to access various naming and directory services [6]. It is independent from the underlying directory services, which could be implemented using a server, a plain file, or a database. According to the JNDI specification, any directory service must provide a hierarchical structure, referred to as a *namespace*. JNDI uses this namespace in order to map any name to the corresponding object. Applications can thus discover names and lookup any data or object via its name. In traditional server-based JNDI implementations, a client creates a destination

object via some administration tool and binds it to a central JNDI server. This object then resides there until it is unbound. From then on, applications typically use JNDI to lookup the destination objects they require.

3 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. Furthermore, a server is responsible to provide a directory service that allows JMS clients to discover queues and topics. As explained in Section 1 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 server-less 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 three basic modules: a communication middleware system, a directory service, and the JMS provider per se.

3.1 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 [7].

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. 2a. 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. 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

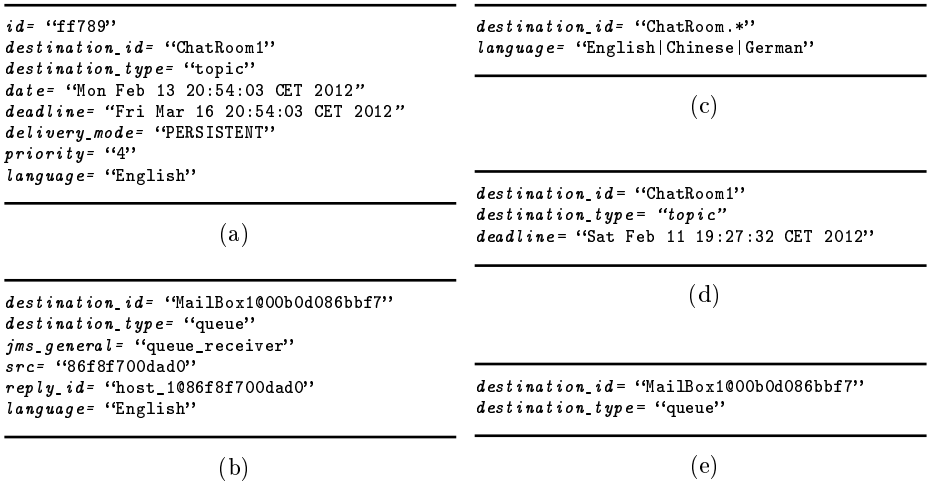


Fig. 2: Examples of message descriptors and message selectors

message descriptor, except that the *value* field of each attribute contains a regular expression. Fig. 2c shows a selection pattern, which would for example match the message descriptor shown in Fig. 2a.

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 defines a selective version of the epidemic routing model proposed in [8]. Details about this interaction schema and about how it performs in real conditions can be found in [7].

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. Large messages are additionally segmented so that each fragment can fit in a single UDP datagram. Fragments of a large message can propagate independently in the network and be reassembled only on destination hosts.

3.2 Directory service (JNDI)

JOMS supports a distributed directory service, which implements a subset of the API of the standard JNDI, while preserving the semantics [6].

In JOMS, each host maintains a local directory that acts as a local namespace from which application services can lookup/retrieve destination objects. When a destination object is created, an entry for this destination object is added to the local namespace. Each entry in this namespace is characterized by a name, a type and a deadline.

Ensuring name unicity in a server-based configuration over traditionnal (connected) networking environments does not raise any major issue. In D-MANETs, no consensus between hosts can be reached [9], and ensuring name unicity in such conditions is not a trivial task. In JOMS, when topics with the same name are added to different local namespaces, they are considered as referring to the same topic. It is up to the applications to manage topic creations while avoiding name ambiguities. For queues, it is important to make each name unique. A queue's name in JOMS therefore has the form *queue_name@host_name*. Thus, since DoDWAN ensures that each host is assigned a unique name, then using this form helps to differentiate queues that have been created on different hosts.

The destination object's type can be *topic* or *queue*. This property has been added since the two modes of communication are managed by JOMS quite differently. Finally, the deadline represents the expiration date of the entry. An example of a destination object's entry is shown in Fig. 2d. This destination object describes a topic with name "ChatRoom1" and deadline "Sat Feb 11 19:27:32 CET 2012".

Once an object has been added in a host's local namespace, JNDI advertises this object using the underlying communication middleware. The hosts that receive an advertisement message that pertain to new destination objects (that is, destination objects that are not already available in their own namespaces) add these objects to their local namespace.

It is worth mentioning that in D-MANETs a strong consistency between several host's namespaces cannot be guaranteed since there is no central server. In practice, there is no way to unbind a destination object network-wide. For this reason, JOMS uses the deadline property introduced in each JNDI entry in order to delete obsolete destination objects. In addition, JOMS implements a refreshing mechanism: whenever a message is sent/received to/from a specified destination object, the lifetime of this destination object is automatically extended in the local namespace. Thus, if an entry is not refreshed for a long time, this entry is considered as obsolete and is automatically deleted from the local namespace. This mechanism prevents a host from advertising an out-of-date destination object and ensures that each host holds an up-to-date namespace.

3.3 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.

Message model according to the JMS specification [5], 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. 2a. 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. 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; they are used by JOMS to route the message to its recipients having these criteria 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. 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.

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. 2a, 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 Feb 13 20:54:03 CET 2012" and will die at "Fri Mar 16 20:54:03 CET 2012". The message selector "language= English" is a set of keywords characterizing this message.

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.

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 [9]. 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. 2e. This profile matches all messages sent to the queue "MailBox1@00b0d086bbf7", whose descriptors contain this property in the same way as the message shown in Fig. 2a. Now, when an application wants to be a receiver for this queue, JOMS sends a request to the QM as shown in Fig. 2b. 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 message 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.

4 Evaluation

Evaluating the performance of a middleware system capable of running in a D-MANET is obviously a tricky task. In the literature protocols and systems designed for D-MANETs are usually evaluated using network simulators, and little effort is devoted to investigate how they can perform when deployed in a real setting.

In contrast a major outcome of our work is the fact that JOMS[1] and DoDWAN[2] have both been fully implemented in Java, and are now distributed under the terms of the GNU General Public License. They can thus be tested in real conditions. Indeed DoDWAN has already been deployed in different kinds of environments, including military tactical networks, and proved robust and reliable in such harsh conditions [10].

As explained in Section 1 a middleware system designed for D-MANETs can do no magic: unless otherwise specified its does not control how mobile hosts move in the network, so it cannot guarantee that a message will ever reach (or reach in time) any particular destination. The behavior JOMS can show in a D-MANET is therefore highly dependent on how this network evolves over time. Based on this observation we conducted two series of experiments, first within a single connected island, and then in a real disconnected network involving several user-carried mobile hosts.

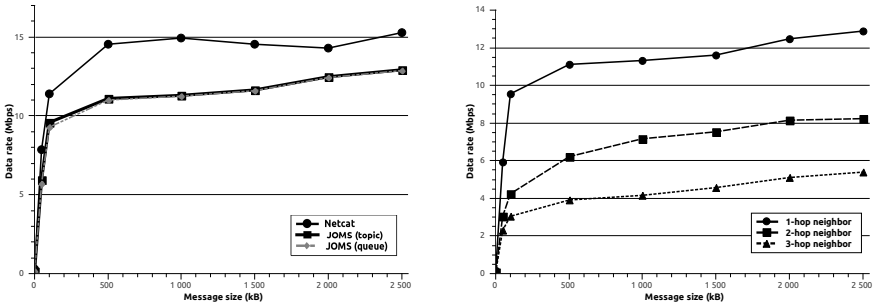
Efficiency of JOMS's messaging in a single connected island. We first used two netbooks A and B, running JOMS over a Linux operating system. These netbooks were installed next to each other in the same room, and their built-in Wi-Fi 802.11bg chipsets were configured to operate in ad hoc mode. We measured the time required to transmit messages of different sizes from one netbook to the other. These messages were produced by host A and received by host B, using either a topic-based model or a queue-based model. In the first case the messages were published to a topic by host A, and B was a subscriber to this topic. In the second case A hosted a local message queue, and B was a subscriber for messages deposited in this queue.

In order to get reference values regarding the capacity of the wireless link at application-level, we used the basic Netcat (nc) networking utility, that can read and write chunks of data across network connections.

Averaged over 150 rounds, the results of these tests are shown in Fig. 3-a, in terms of application-level data rates. Topic-based and queue-based messaging with JOMS obviously show similar performances. For large messages JOMS shows between 15% and 20% overhead over Netcat. Moreover the latency we measured when producing a message was around 10 ms with Netcat, and 25 ms with JOMS.

To complement these results we also investigated the behavior of JOMS when messages can propagate over multiple hops within a single connected island. The

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



(a) (b)

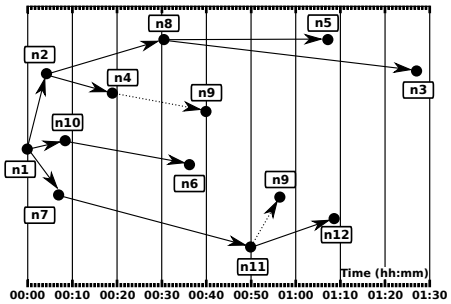
Fig. 3: Data throughputs observed within a single island

tests were conducted using four netbooks deployed so as to constitute a simple line A-B-C-D.Our test procedure relied on the queue-based model: message queues were created on B, C, D, and host A was configured as a subscriber to all these queues. Messages were deposited successively in each queue (with local deposits), and we measured the time required for these messages to reach subscriber A. Fig. 3-b shows the transmission data rates we calculated based on these transmission times, depending on the size of messages and on the number of hops required for each message to reach host A. It can be observed that the data rate gets lower as the number of hops increases. This is because when host B serves as a relay between its neighbors A and C, the radio channel around B is twice as busy as when B interacts only with host A. The same observation applies for host C when it must serve as a relay between hosts B and D.

Considering that DoDWAN implements a sophisticated opportunistic protocol in order to orchestrate communications between neighbor hosts, we consider that the above-mentioned results are quite reasonable. Yet JOMS's most salient feature is of course that it can ensure the delivery of messages in a disconnected network, where traditional JMS providers (designed for connected environments) are totally useless.

Efficiency of JOMS's messaging in a real D-MANET. In order to observe how JOMS can perform in such conditions we implemented a simple chat application based on JOMS. This application offers two services: public chat and private chat. The public chat service relies on the concept of topic: each message published on a topic can be received by all subscribers. With the private chat service, each message is addressed to a single message queue, which is hosted on the destination netbook.

A dozen of volunteers in our laboratory were equipped with netbooks running this application, and they were asked to carry their netbook whenever possible –and use both chat services of course– during a few days while roaming the laboratory building or its surroundings.



(a)

Delivery time (hour)	Delivery ratio (Topic)	Delivery ratio (Queue)
$t \leq 2$	56%	6%
$t \leq 4$	66%	40%
$t \leq 6$	74%	66%
$t \leq 8$	80%	74%
$t \leq 10$	84%	79%
$t > 10$	94%	93%

(b)

Fig. 4: Message delivery in a real D-MANET

Several one-day trials were conducted during this test, and an average of 250 chat messages were produced by volunteers during each trial. Fig. 4a illustrates how one particular message disseminated during one of the trials. This message was first published in the public topic by netbook *n1*. After only a few minutes *n1* established radio contact with *n2*, which thus got a copy of the message and became a new carrier for this message. *n1* later managed to forward the message to *n7* and *n10* successively, while *n2* forwarded it to *n4*, and later to *n8*. The message thus kept disseminating, until it reached the last subscriber *n3*, about 90 minutes after it was initially published. This example shows how a message can disseminate in a D-MANET, using unpredicted radio contacts as opportunities to reach new mobile hosts in the network.

In Fig. 4b we present the cumulative delivery rates of topic-based and queue-based messages, as observed during these trials. It can be noticed that nearly 60% of topic-based messages got delivered in less than 2 hours, whereas most queue-based messages took between 4 and 6 hours to be delivered. This difference is due to the fact that reaching the holder of a given message-queue is more difficult than reaching any subscriber to a given topic. In any case the table shows that during the experiment most of the messages got delivered to their destination(s) in less than 10 hours. Yet, about 6% of the messages could not be delivered. This is the consequence of the unpredictable –yet perfectly legitimate– behavior of the users, which sometimes moved away from the laboratory or switched their netbook off unexpectedly. By doing so they prevented any further radio contact between their netbook and those of other users, and this of course lead to message loss.

5 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* [11]) 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 [12]. 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 E-JMS requires that all JMS clients have a local copy of an identical configuration file, which contains information about queues and topics. No directory service (such as JNDI) is provided, so queues and topics cannot be created or destroyed dynamically during the execution of a distributed application. This is clearly a serious constraint, which limits the usability of E-JMS for developers.

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 utilization, 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 [7]. Yet JOMS could theoretically be implemented above any other communication system. Some protocols have been actually implemented in middleware systems and are openly distributed : DTN2, a reference implementation of protocols designed by the Delay-Tolerant Networking Research Group (DTNRG) [13], and Haggle, a content-centric architecture for opportunistic communication among mobile users (or devices) [14].

6 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 distributed applications using message queues and topics –as defined in the JMS specification– can be deployed and executed in partially or intermittently connected ad hoc networks. An opportunistic, content-driven communication model is used to enable message forwarding in such networks, using mobile hosts as carriers that allow messages to propagate network-wide.

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 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. Fall, K.: A Delay-Tolerant Network Architecture for Challenged Internets, New York, USA, ACM (2003) 27–34
2. Pelusi, L., Passarella, A., Conti, M.: Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks. IEEE Communications Magazine (November 2006)
3. Zhang, Z.: Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks: Overview and Challenges. IEEE Communications Surveys and Tutorials **8**(1) (January 2006) 24–37
4. Hurwitz, J.: Sorting Out Middleware. DBMS **11**(1) (January 1998) 10–12
5. Hapner, M., Burridge, R., Sharma, R.: Java Message Service, Version 1.1 (2002)
6. Lee, R., Seligman, S.: JNDI API Tutorial and Reference: Building Directory-Enabled Java Applications. Addison-Wesley, Reading, USA (2000)
7. Haillot, J., Guidec, F.: A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks. Journal of Mobile Information Systems **6**(2) (2010) 123–154
8. Vahdat, A., Becker, D.: Epidemic Routing for Partially Connected Ad Hoc Networks. Technical report, Duke University (April 2000)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM **32**(2) (April 1985) 374–382
10. Haillot, J., Guidec, F., Corlay, S., Turbert, J.: 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 (October 2009)
11. Musolesi, M., Mascolo, C., Hailes, S.: EMMA: Epidemic Messaging Middleware for Ad Hoc Networks. Personal and Ubiquitous Computing **10**(1) (2005) 28–36
12. Vollset, E., Ingham, D., Ezhilchelvan, P.: JMS on Mobile Ad Hoc Networks. In Personal Wireless Communications (PWC) (2003) 40–52
13. Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., Weiss, H.: Delay-Tolerant Networking Architecture. IETF RFC 4838 (April 2007)
14. Nordström, E., Gunningberg, P., Rohner, C.: A search-based network architecture for mobile devices. Technical Report 2009-003, Department of Information Technology, Uppsala University (January 2009)