JION: A JavaSpaces Implementation for Opportunistic Networks

Abdulkader Benchi, Pascale Launay, Frédéric Guidec
IRISA, Université de Bretagne-Sud
Vannes, France
{abdulkader.benchi, pascale.launay, frederic.guidec}@univ-ubs.fr

***Abstract*—Disconnected mobile ad hoc networks (or D-MANETs) are partially or intermittently connected wireless networks, in which continuous end-to-end connectivity between mobile nodes is not guaranteed. The ability to self-form and self-manage brings great opportunities for D-MANETs, but developing distributed applications capable of running in such networks remains a major challenge. A middleware system is thus needed between network level and application level in order to ease application development, and help developers take advantage of D-MANETs' unique features. In this paper, we introduce JION (JavaSpaces Implementation for Opportunistic Networks), a coordination middleware specifically designed for D-MANETs, and with which pre-existing or new JavaSpaces-based applications can be easily deployed in such networks.**

***Index Terms*—peer-to-peer computing; opportunistic networking; D-MANETs; coordination middleware; JavaSpaces.**

I. Introduction

A mobile ad hoc network (or MANET) is a dynamic wireless network that requires no fixed infrastructure. It is generally formed on-the-fly by a collection of wireless nodes without the aid of any centralized administration. Each mobile host can communicate with its neighbors using direct pair-wise wireless links. Communications in MANETs have been enhanced over the years thanks to multi-hop forwarding protocols, such as OLSR, AODV, DYMO, DSR, etc. [1].

Yet most of these protocols rely on the assumption that the whole MANET remains continuously connected, i.e., between any pair of hosts in the MANET, there actually exists at least one temporaneous end-to-end path. Unfortunately, this assumption does not hold in real conditions; many real MANETs are, under the most favorable conditions, only partially or intermittently connected.

The sparsely or irregular distribution of a MANET's hosts can, for example, induce link disruptions in the whole MANET. These disruptions may in turn split the whole MANET into a collection of distinct, continuously changing, disconnected "islands" (connected components) as shown in Figure 1. This kind of MANET is called a Disconnected MANET (D-MANET). The "*store, carry and forward*" approach is the foundation of Delay/Disruption Tolerant Networking (DTN) [2]. In a D-MANET, it can help bridge the gap between non-connected parts of the network; the mobility of hosts makes it possible for messages to propagate network-wide by using mobile hosts as carriers (or *data mules*) that can move between network islands. As shown in Figure 1, connectivity disruptions between islands 1 and 2 can for example be tolerated thanks to users moving (deliberately or by chance) between these islands. The device of a user moving from island 1 to island 2 acts as a data mule for messages addressed to hosts located in island 2. Considering that many carriers may be involved successively for the transmission of a single message, this approach provides message delivery at the price of additional transmission delays. Figure 1 shows that the transmission of a message from island 1 to island 4 can for example involve two message carriers: first a carrier moving from island 1 to island 2, and then another carrier –or the same one– moving from island 2 to island 4.

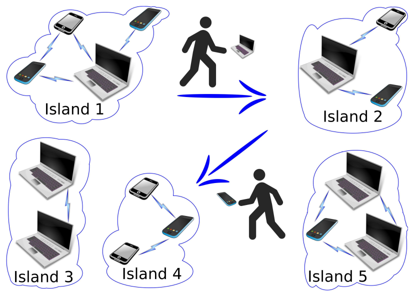

Figure 1. Example of a disconnected mobile ad hoc network

In the DTN community, some approaches make the assumption that communications between the hosts can be predicted accurately, and routing strategies can be thus devised based on contact predictions. But, in most real D-MANETs, communications are not planned in advance and can hardly be predicted, especially if the physical carriers are for example human beings carrying laptops or smartphones. The term opportunistic networking is often used to denote such disruption tolerant networks where the contacts must be exploited opportunistically [3]. In such wise, each contact represents an opportunity for two hosts to exchange messages. Consequently, communication protocols for D-MANETs usually provide no more than best-effort delivery. Consider again the example shown in Figure 1, and assume a message is addressed by a host in island 2 to a host in island 3. If no human carrier ever visits island 3, then there is no chance that the message ever gets delivered in this island.

The dynamic nature of D-MANETs creates many challenges for application developers. As a general rule, no host can be considered as stable and accessible enough to play the role of a server for all the other hosts. Consequently, applications developers should generally use a peer-to-peer model rather than a client-server one. Developers, while writing their applications, should additionally take into consideration occasional transmission failures and long transmission delays.

All these reasons result in an increasing need for a middleware system that, while coping with D-MANETs issues, provides the developers with a set of APIs that eases the development of distributed applications over D-MANETs. Moreover, any middleware system for D-MANETs must have an asynchronous nature in order to fit with the long transmission delays observed in such networks.

In the remainder of this paper, we present JION (JavaSpaces Implementation for Opportunistic Networks), a coordination middleware system we designed and implemented specifically for D-MANETs. JION is actually an implementation of the JavaSpaces specification [4], so any pre-existing distributed application basing on the JavaSpaces API can be executed in a D-MANET using JION, with no further development.

This paper is structured in the following way: the JavaSpaces specification is described briefly in Section II. Section III presents the JION's architecture along with details about its implementation. Evaluation results are shown in Section IV. Section V discusses related work. Section VI concludes this paper and describes our plans for future work.

II. JavaSpaces Background

The JavaSpaces technology, implemented by JION, is a Java specification of the concept of tuple space, which was originally introduced in the Linda programming language [5]. In this section, we provide a brief introduction to the tuple space as introduced in Linda. The JavaSpaces technology is presented as well.

A. Tuple space

The tuple space concept has its root in the Linda parallel programming language developed at Yale University [5]. A tuple space is a shared data space acting as an associative memory used by several processes for communication and/or coordination requirements. A Linda application is viewed as a collection of processes cooperating via the flow of data structures, called "*tuples*", into and out of a *tuple space*. Each tuple is a record of typed fields containing the information to be communicated. The coordination primitives provided by Linda allow processes to insert a tuple into the tuple space (*out*) or retrieve tuples from the tuple space, either removing those tuples (*in*) or preserving the tuples in the space (*read*). For retrieving operations, the tuples are selected using a simple pattern matching from a given set of parameters.

B. JavaSpaces

The JavaSpaces technology is a Java specification of the tuple space concept, implemented inside the JINI architecture. It defines a set of application programming interfaces (APIs) that extend the simple core of Linda primitives. The JavaSpaces version of Linda tuples, called "entries", are Java objects that contain public fields that act as Linda's typed fields. JavaSpaces provides *read*, *take* and *write* operations in order to implement Linda's *read*, *in*, and *out* operations respectively. Additionally, it provides a *notify* operation that allows processes to perform a lookup operation in an asynchronous manner. This operation notifies the processes by sending a special object called *event* containing information, to which the processes react. The matching in read, take and notify operations are performed using a special kind of entry, called a *template*, that characterizes the kind of entries the process wants to look for. The selected entries are those whose types and fields match the template. JavaSpaces also provides light versions of *read* and *take* operations, with which processes do not need to wait for the answer. These operations, called *readIfExist* and *takeIfExist*, can be useful when a process requires an answer without blocking. As JavaSpaces' entries are passive data, processes cannot perform operations on tuples directly. In order to modify an entry, a process must explicitly remove, update and reinsert it into the space. It is worth taking into consideration that till now there is no standard JINI security model to be used by JavaSpaces users.

III. JavaSpaces Implementation For Opportunistic Networks (JION)

The JavaSpaces technology was primarily designed to provide persistent object exchange areas (spaces), through which processes coordinate actions and exchange data. Most of the JavaSpaces implementations are server-based systems where centralized servers are used to manage such spaces. As explained in Section I, a server-based system is hardly compatible with the characteristics of D-MANETs, as no host in a D-MANET can act as a reliable server for all the other hosts. A server-less JavaSpaces implementation must then be developed in order to provide JavaSpaces services for D-MANETs.

JION, or JavaSpaces Implementation for Opportunistic Networks, is a JavaSpaces implementation that was designed along that line. Its architecture is composed of two basic modules: the communication middleware system, and the JavaSpaces services system.

A. Communication Middleware

Building any application for D-MANETs requires some communication middleware system, with which hosts can collaborate in a peer-to-peer manner to ensure message transportation and deal with high latency and high failure rate. JION relies on a communication middleware system called DoDWAN *(Document Dissemination in mobile Wireless Ad hoc Networks)* [6]. DoDWAN has been designed in our laboratory, and it is now distributed under the terms of the GNU General Public License[1].

DoDWAN supports content-based information dissemination in D-MANETs. In content-based networking, information flows towards interested receivers rather than towards specifically set destinations. This approach notably fits the needs of

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

Board= “Sports”
Topic= “football”
Language= “English”

(a)

Board= “Sports”
Topic= “.*ball”
Language= “English | French | German”

(b)

Figure 2. A message descriptor and a message selector

applications and services dedicated to information sharing or event distribution. It can also be used for destination-driven message forwarding, though, considering that destination-driven forwarding is simply a particular case of content-driven forwarding where the only significant parameter for message processing is the identifier of the destination host (or user).

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 Figure 2a. These attributes can be defined freely by the developers of application services built on top of DoDWAN.

DoDWAN implements a selective version of the epidemic routing model proposed in [7]. It provides application services with a publish/subscribe API. When a message is published on a host, it is simply put in the local cache maintained on this host. Afterwards, each radio contact with another host is an opportunity for the DoDWAN system to transfer a copy of the message to that host whenever it is interested.

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, as shown in Figure 2b. 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. Details about this interaction scheme and about how it performs in real conditions can be found in [6].

As a general rule, a mobile host that defines a specific interest profile is expected to serve as a mobile carrier for all messages that match this profile. 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 setting DoDWAN’s configuration parameters accordingly.

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 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.

B. JION Implementation

According to the JavaSpaces specification, processes coordinate by exchanging entries through the space using a simple set of operations. Entries and operations represent the basic JavaSpaces’ elements. This section describes the JION’s architecture and its entry module, along with the supported operations.

1) JION’s architecture: As mentioned in Section III, JION is a server-less JavaSpaces implementation, as it is intended to be used in D-MANETs where server centralization is impractical. Each host maintains a local space, in which JION stores the entries produced locally (that is, entries produced by write operations, which have been invoked by local processes). If entries were propagated all over the D-MANET and managed in a collaborative manner, this could result in *orphan entries*; as stated in [8], it is impossible to obtain a consensus between hosts in a distributed disconnected environment. Consequently, D-MANET’s hosts could not agree to remove any entry from the space, for example when a process wants to take it. Imagine that an entry has been propagated over the hosts in the islands shown in Figure 1, and a process in island 1 takes this entry. If no user ever visits island 5 for example, the copies of this entry in this island will become orphan entries. In JION, the write operations are only processed locally, while matching and fetching operations (read, take and notify) are processed by querying hosts over the network for the entries they own.

2) Entries and Templates: According to the JavaSpaces specification, an *entry* is an object reference characterized by its “*fields*”. In the JavaSpaces terminology, entry fields refer only to the public fields of the entry objects. In fact, entry fields are meant to act as a set of attributes characterizing an entry, and are used to perform matching operations while retrieving entries from the space.

As mentioned before, a DoDWAN message has two parts: a descriptor and a payload. Since DoDWAN’s descriptor is also meant to characterize the content of the message, JION maps the entry’s fields to the DoDWAN’s descriptor, as their content is needed by JION to do match operations. The rest of the entry (that is, non-public fields) is simply carried in the message as its payload, and considered as a simple byte array.

Templates are special entry objects whose fields’ values are used in match operation. This notion of a template in JavaSpaces is mapped to that of DoDWAN’s selection pattern, which is used by JION’s pattern matching operation.

3) Operations: According to the JavaSpaces specification, access to entries must be done through a set of basic operations, which are: write, read, take and notify. Below is a description of the way JION supports these operations.

write: this operation stores a new entry into the host’s local space for a specific period of time, called a lease. A lease represents the lifetime of the associated entry. As mentioned above, each entry is only stored on the local host and is not replicated over the D-MANET. Therefore, it is up to each host to monitor its local space and manage its own entries, and especially ensure that out-of-date entries are not used anymore.

read: this operation requests the JION service to locate an entry that matches the template provided as a parameter. When a process on a host performs a read operation, the host’s local space is queried first in order to find a matching entry. If no matching entry is found, JION disseminates the specified template over the D-MANET. Each host, when it receives this template, queries its local space to find a matching entry and forward a copy of this entry back to the requesting process. It is then up to the requesting process to choose one entry as an answer to its read request. Operation *readIfExists* is also supported by JION. This version queries only the local space to find an entry that matches the specified template. The request is not disseminated over the D-MANET.

take: this operation basically performs the same function as *read*, except that it removes the matching entry from the space. JION first searches the local space. If no match is found, it queries all the hosts over the D-MANET in order to discover which hosts (if any) have a matching entry. Upon receiving the proposals, JION selects one host, from which the entry should be taken. JION then asks the chosen host to permanently remove the matching entry from its local space and hand it back to the requesting process. The entire operation may take more time than the read operation since it needs four messages while only two messages are required in the *read* operation. JION also supports a *takeIfExists* operation, which performs exactly like the corresponding *readIfExists*, except that a matching entry is only requested from the local space.

notify: this operation notifies a process when entries that match a given template are written into a space. When a notify operation is performed by a process, JION disseminates the given template all over the hosts in the D-MANET. The hosts register the notify request in the hope that a matching entry will be written before the request's lifetime expires. Consequently, when a matching entry is written in a host, the host forwards an event object containing information about this entry and its location to the requesting process.

4) Transactions: According to the JavaSpaces specification, it is possible to group multiple operations (participants) into a bundle that acts as a single atomic operation. This is done using the optional concept of transaction. Either all operations within the transaction will be performed or none will. In fully-connected stable networks, a transaction is controlled by a specific manager (server), which should always be reachable by all the participants. If a participant is momentarily disconnected, the whole transaction is aborted. Considering that hosts in D-MANETs can neither rely on a reliable server nor reach a consensus, it is not possible to ensure transactions as defined by JavaSpaces. For this reason, JION does not support the concept of transaction and each operation is considered as a singleton operation.

IV. Evaluation

A D-MANET is a wireless network whose topology is continuously changing, and where radio contacts between mobile hosts do not necessarily follow any predictable pattern. Therefore, protocols and systems designed for D-MANETs are usually evaluated using network simulators. The originality of our work lies in the fact that JION has been fully implemented in Java and is now distributed under the terms of the GNU General Public License[2].

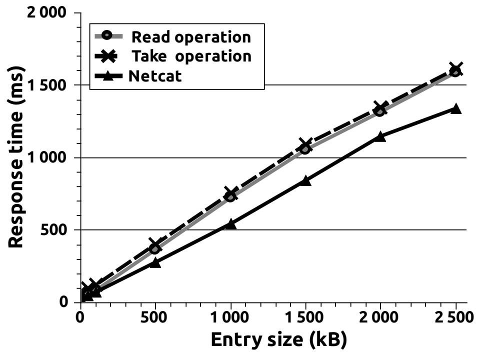

Figure 3. Response time observed between two netbooks

JION has seen extensive testing to examine how well it performs in D-MANETs. While conducting these tests we strived to evaluate how easy it is for an application developer to implement a distributed application using JION. Furthermore, we have evaluated the efficiency of JION in a real D-MANET.

A. Developing distributed applications with JION

JION implements Sun Microsystems' JavaSpaces Technology specification, provided as a part of the Java Jini Technology [4]. Since JION implements a well-known middleware specification, developers do not need to learn a new programming language, or get familiar with an exotic programming model or API. A developer can simply focus on writing a standard JavaSpaces application, and JION will take care of its execution in a D-MANET. Indeed, any pre-existing JavaSpaces application can be deployed using JION, without any change in its source code.

Developers should however be aware of the specific constraints posed by D-MANETs, where message delivery, message ordering, and transmission delays are usually not guaranteed. Such constraints are not due to limitations in JION; they are due to the very nature of D-MANETs. As explained in Section I, opportunistic protocols and middleware systems designed for D-MANETs can do no magic; they can support network-wide communication in a D-MANET, using mobile hosts as carriers that help to bridge the gap between non-connected parts of the network. Yet, unless otherwise specified they do not control how mobile hosts move in the network, so they cannot guarantee that a message will ever reach (or reach in time) any particular host in the network. A developer working on an application for D-MANETs should therefore assume that delivery failures and late deliveries may be more common than in-time deliveries, and design the distributed application or organize its deployment accordingly.

For testing and evaluation purposes, we have developed a distributed bulletin board system (D-BBS) inspired from the classical bulletin board system (BBS). A BBS typically consists of a number of bulletin boards, which serve as discussion areas relating to general themes. Each bulletin board is generally labeled by an expressive name describing

[2]http://www-irisa.univ-ubs.fr/CASA/JION

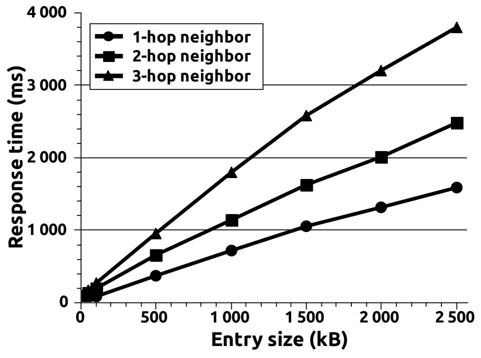

Figure 4. Response time between multi-hop neighbors

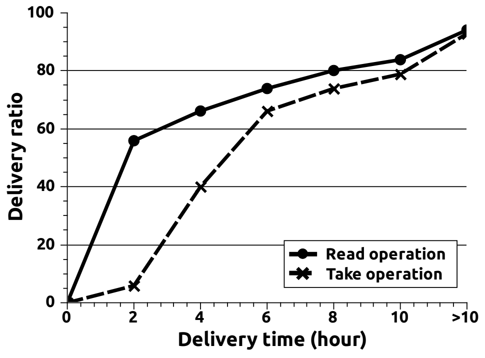

Figure 5. Entry delivery in a real D-MANET

its contents (e.g., Sports). On each board, users can post, read and delete messages under different topics. Since BBS typical implementations are usually server-based, they are not well adapted to D-MANETs, so we developed this application using the JION middleware.

In D-BBS, posting a message on a board under a topic is implemented by creating a tuple having the board's and the topic's names as fields, and writing it into the JION's space. Similarly, reading/deleting a message from a specific board under a specific topic is performed by creating a template having the board's and topic's name as fields. This template can then be passed as a parameter to JION's *read/take* function. Furthermore, the developer can get benefit from JION's *notify* function in order to keep users up to date with changes on a specific board under a specific topic using an appropriate template.

Developing and deploying this application for a D-MANET was a straightforward task. Using JION, the programmer focuses on the application features without paying attention to the specific issues of this kind of very challenging environment.

B. *JION's efficiency over D-MANETs*

Before trying to observe how entries can propagate between several islands in a disconnected network, one can first try to measure how fast they can propagate within a single island. Since JION is implemented on top of the DoDWAN communication system, which itself relies on UDP transmissions, our first objective was to evaluate how our multi-layer middleware architecture performs over the underlying wireless transmission medium.

We first used two netbooks A and B, running JION 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 actually focused on the response time, which is here defined as the time interval between the time netbook A writes entries of different sizes and the time when netbook B receives them using the read/take operations. 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. 200 series of tests were conducted in this scenario. The results of these tests are presented in Figure 3. Read and take operations show similar performance. However, JION shows about 20% overhead over Netcat. Considering that JION's communication middleware (DoDWAN) implements a sophisticated opportunistic protocol in order to orchestrate communications between neighbor hosts, we consider that these results are quite reasonable.

To increase realism, another real world test was conducted to investigate the behavior of JION when messages can propagate over multiple hops. The tests was carried out with four netbooks (A, B, C and D) distributed in our laboratory. Because of the effect of concrete walls on signal attenuation, the connectivity between these netbooks was such that netbook B could only communicate with A and C, while netbook D could only communicate with C. This test relied on write/read operations: a total amount of 225 entries were written on B, C, D, and host A was configured to read these entries. We measured the average time required for these entries to reach host A. The results of this test are shown in Figure 4. It can be observed that the delay before host A gets an entry changes with the size of the entry and the number of transmission hops. 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. It must also be considered that DoDWAN strives to ensure a high delivery ratio, so entries are retransmitted again and again if they are not received in the first place. Indeed, during the test all entries got received by host A.

After measuring how JION can perform in a single, connected island we used our D-BBS application to observe how it can perform in a real D-MANET. A dozen of volunteers in our laboratory were equipped with netbooks running D-BBS. Several one-day tests were conducted by asking the volunteers to carry their netbook whenever possible –and use D-BBS services of course– during a few days while roaming the laboratory building or its surroundings. To analyze the results special attention was paid to the cumulative delivery rates of *read/take* operations, as shown in Figure 5. The cumulative delivery rates were measured in terms of time slices where a measure of 2 hours means that the average delivery time observed was between 0 and 2 hours; a measure of 4 hours means it was between 0 and 4 hours, etc. Using the *read* operation, it can be noticed that nearly 59% of the entries were delivered in less than 2 hours. However, the majority of hosts had to wait between 4 and 6 hours in order to get the required entries using the *take* operation. This difference was expected since operation *take* requires more rounds than operation *read*. In general, the results show that most of the entries got delivered to their destination(s) in less than 10 hours. Yet, about 6% of the entries could not be delivered. This is the consequence of the unpredictable behavior of the users, which sometimes moved away from the laboratory or switched their netbook off unexpectedly.

V. Related Work

In the last few years, several projects revisited Linda [5], especially in the context of mobile ad hoc environments.

Both Ara [9] and LIME [10] are coordination middleware systems implementing tuple spaces stored on hosts acting as servers. These middleware systems target mobile ad hoc networks. They provide JavaSpaces services to mobile hosts that are in the servers' communication range. Since they rely on a server-based model, they are hardly usable in real D-MANETs. Limone [11] is a lightweight alternative to LIME requiring far less overhead. Limone is based on the premise that a single round-trip message exchange is always possible, making it impractical over D-MANETs, for in D-MANEs unpredictable disruptions are the norm rather than the exception. In contrast, CAST [12] is a server-less coordination middleware for MANETs. Since it does not rely on any centralized service, this middleware suits well the dynamics of wireless open networks. CAST makes it possible to process operations even when no end-to-end route exists between the involved hosts, by implementing a source routing algorithm. This routing strategy relies on the assumption that each host's motion profile is known. This is clearly a serious constraint, which limits the usability of CAST over the kind of D-MANETs JION targets, where hosts' motions are neither planned nor predictable. Tuple board (TB) [13] is another server-less coordination middleware for developing collaborative applications running in ad hoc networks of mobile computing devices. Like JION, this middleware has been fully implemented and distributed. It can thus be used and tested in real conditions. However, the proposal lacks flexibility, in that it is limited to a group of nearby connected devices: when a device leaves the network or turns off, all the tuples posted from this device are withdrawn. The importance that we attribute to disconnections make the disconnection tolerance a vital requirement for any middleware that is meant to support D-MANETs.

Furthermore, all the middleware systems mentioned above define their own communication protocol for route discovery and maintenance. Our work is different as JION presents a two-layer architecture: the upper layer is concerned with tuple space services, while the lower layer supports opportunistic communication. As mentioned above, DoDWAN has been chosen among a few opportunistic communication protocols that are openly distributed to support communications on D-MANETs. Yet, JION could theoretically be implemented above any other communication system, such as DTN2 (a reference implementation of protocols designed by the Delay-Tolerant Networking Research Group (DTNRG) [14]) or Haggle (a content-centric architecture for opportunistic communication among mobile devices [15]).

VI. Conclusion and Future Work

In this paper, we have introduced JION (JavaSpaces Implementation for Opportunistic Networks), a JavaSpaces implementation we designed and implemented specifically for disconnected mobile ad hoc networks (D-MANETs). Using JION, distributed applications based on the concept of tuple spaces (as defined in the JavaSpaces specification) can be deployed and executed in D-MANETs. JION provides an effective base, which eases the development of D-MANETs' applications. It has been tested in real conditions and is now distributed under the terms of the GNU General Public License.

Further tests are still under way in order to verify how stable JION is in different kinds of challenged environments. Future work should include the assessment of JION's portability (most likely by implementing it over the DTN2 reference implementation), and an investigation of security issues pertaining to the execution of JION-based distributed applications in D-MANETs.

References

[1] C. Liu and J. Kaiser, "A survey of mobile ad hoc network routing protocols," University of Magdeburg, Tech. Rep., 2005.

[2] K. Fall, "A delay-tolerant network architecture for challenged internets," in *ACM Annual Conference of the Special Interest Group on Data Communication*, Aug. 2003.

[3] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks," *IEEE Communications Magazine*, Nov. 2006.

[4] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces(TM) Principles, Patterns, and Practice*. Prentice Hall, Jun. 1999.

[5] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, no. 4, pp. 444–458, Apr. 1989.

[6] 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.

[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.

[9] H. Peine and T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents," in *Proceedings of the First International Workshop on Mobile Agents*. London, UK: Springer-Verlag, 1997, p. 50–61.

[10] A. L. Murphy, G. P. Picco, and G.-C. Roman, "Lime: A coordination model and middleware supporting mobility of hosts and agents," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, pp. 279–328, July 2006.

[11] C.-L. Fok, G.-C. Roman, and G. Hackmann, "A lightweight coordination middleware for mobile computing," *Coordination Models and Languages*, pp. 135–151, 2004.

[12] G.-C. Roman, R. Handorean, and R. Sen, "Tuple space coordination across space and time," in *COORDINATION*, 2006, pp. 266–280.

[13] A. Kaminsky and C. Bondada, "Tuple board: A new distributed computing paradigm for mobile ad hoc networks," *First Annual Conference on Computing and Information Sciences*, pp. 5–7, 2005.

[14] 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.

[15] 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.