# Programming in Mobile Ad Hoc Networks

Justin Collins
Mobile Systems Lab
Computer Science Department
University of California, Los Angeles
Los Angeles, California 90095
collins@cs.ucla.edu

Rajive Bagrodia
Mobile Systems Lab
Computer Science Department
University of California, Los Angeles
Los Angeles, California 90095
rajive@cs.ucla.edu

## ABSTRACT

The possibility for spontaneous ad hoc networks between mobile devices has been increasing as small devices become more capable of hosting useful networked applications. These applications face the challenges of frequent disconnections, highly dynamic network topologies, and varying communication patterns, a combination unique to mobile ad hoc networks. This is the first survey to examine current MANET programming approaches including tuple spaces, remote objects, publish/subscribe, and code migration through analysis and experimental results. We suggest that these approaches are essentially extensions to existing distributed and parallel computing concepts and new abstractions may be necessary to fully handle the programming issues presented by MANETs.

## Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.1.3 [**Concurrent Programming**]: Distributed Programming

## 1. INTRODUCTION

Mobile ad hoc networks (MANET) composed of small portable devices are becoming more and more common. Many PDAs, smartphones, portable game systems and most laptop computers now have the capability to form wireless ad hoc networks. Mobile applications are no longer limited to stand-alone or client-server programs, but can interact and form useful networks directly with each other. Such networks are ideal for situations in which there is no time to set up a fixed access point, or when there is no fixed infrastructure available.

The ad hoc nature of these networks, combined with the mobility of the nodes, introduces new challenges. Dynamically formed groups of mobile nodes must be able to coordinate between themselves to perform routing and resource discovery. Routes between nodes and available resources may change rapidly, requiring flexible protocols. Nodes cannot rely on knowing the addresses of remote resources ahead of time or from a central directory, but must discover them dynamically. Disconnections become commonplace rather than exceptional, due to mobility and wireless channel variations.

Many new applications, particularly in the consumer space, are being applied to MANETs. These include disaster recovery scenarios, collaborative software such as shared whiteboards, impromptu networks for communication and entertainment, and peer-to-peer applications for file sharing. Much work has been done on protocol development for MANETs, such as AODV [1] and OLSR [2], but systems supporting development of applications which can be used in a MANET are an active area of research.

This paper focuses on the approaches used in recent and ongoing projects which provide environments in which to develop generic applications for the general case of MANET. Section 2 provides an overview of the requirements and issues specific to developing applications to run on MANETs. Section 3 is an overview of several projects which have been proposed and developed to meet those requirements, while Section 4 discusses the various solutions used by the projects in more detail. In Section 5, we attempt to gain a better understanding of techniques required for MANET application development by writing two sample applications using three different projects. Experimental evaluation of those three projects are presented in Section 6. Section 7 lists some related projects and Section 8 presents our conclusions.

## 2. MANET REQUIREMENTS

Mobile applications face several challenges when compared with programs intended for standard desktops. Mobile devices are generally constrained in many ways: the screen size, processor power, memory, and battery power are often limited. Development platforms for mobile devices typically provide basic libraries for application support such as menus and access to data stored on the device. Networking, however, is generally limited to sockets, TCP/IP, and HTTP. In particular, applications are expected to either be standalone, like a calculator, or to only be using the network in a client-server manner, such as accessing websites or email servers.

However, the distinguishing characteristic of a MANET context is not the mobile device or platform itself, but the ad hoc communication between devices. The primary medium for communication is wireless, with no fixed infrastructure, and nodes which tend to be highly mobile. In such an


*WICON '08* November 17-19, 2008, Maui, Hawaii, USA


*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

environment, the following become important requirements when developing applications:

*Disconnection handling:* In a MANET, nodes are highly mobile and disconnections occur frequently, either due to channel condition variation or the mobility of destinations and intermediate nodes. Disconnections may be prolonged, brief, or intermittent and applications must handle all three. Traditional networking treats disconnections as failures, but a programming environment for MANETs needs to handle disconnections as a natural element of the environment.

*Addressing and discovery:* The lack of infrastructure in a MANET requires a decentralized method for finding and addressing resources. Traditional approaches such as DNS cannot be maintained in a MANET, so alternative means of discovering and addressing resources must be provided. The spontaneous nature of MANETs also dictates that discovery be dynamic, as the network topology cannot be known ahead of time and may change rapidly.

*Flexible communication:* Both unicast and multicast communications are common in MANET applications, which are often group-based or collaborative. Being able to provide flexible communication is crucial to developing applications for MANETs.

# 3. PROJECT OVERVIEWS

The current projects for developing software in MANETs fall into three broad categories: runtimes, languages, and middleware, which offer increasing levels of abstraction for the developer. They can also be combined: a middleware solution can be written in a language which uses one of the basic runtimes for mobile devices. In many cases projects also provide additional resources for software development such as debuggers and emulators for testing code. Table 1 summarizes the projects discussed in this paper with respect to the requirements in Section 2. A broader overview focusing only on middleware for MANETs can be found in [3]. The following projects were chosen because they are both recent and representative of varying approaches taken to address programming in a MANET.

## 3.1 Runtimes

Runtimes in this context are virtual machines for languages which are specifically intended for use on small, resource-constrained mobile devices. Runtimes are useful because they provide good portability for applications and thereby simplify some of the application development process.

Two common runtimes for mobile devices are Java ME [4] and the .NET Compact Framework [5]. A third runtime, BREW [6], is a proprietary product from Qualcomm. These runtimes focus on using few resources and providing libraries for application development, especially user interfaces. They do not provide much networking support beyond basic sockets and HTTP support. While it is possible to use these runtimes as foundations for better abstractions, they provide little on their own to support MANET applications and will not be considered in the following comparisons.

## 3.2 Languages

A language in this paper refers to any language, language extension, or library which provides new language constructs for programming in a MANET. Languages often include their own runtime or are built on top of existing runtimes. Libraries and language extensions are likely to be easier for developers to use if they are already familiar with the base language.

M2MI [7], AmbientTalk [8], and SpatialViews [9] are three language-based projects intended for MANETs. Many-to-Many Invocation (M2MI) avoids costly ad hoc routing and discovery by broadcasting messages. Messages are addressed by object type, so if a device hosts an object of the addressed type, it will pass the message to that object.

The advantage of M2MI is simplicity. As messages are simply broadcast without expectation of reply, there is no need to worry about return values or blocking while waiting for confirmation. At the language level, there is no difference on the sender's side between a message which is actually received and one which is not received by anyone. Though this provides simplicity, it also means more work for the programmer. As there is no guarantee of message delivery, any functionality beyond simple unidirectional message passing must be implemented on top of M2MI.

AmbientTalk is a complete object oriented language inspired in part by M2MI's message passing. AmbientTalk implements a higher level abstraction of resource discovery and disconnection handling which is absent from M2MI, but retains the idea of object handles and remote method invocations. All remote events are handled asynchronously by AmbientTalk through the registration of callbacks. A block of code may be registered to be invoked when discovering a certain resource type. AmbientTalk also adds the ability to receive values from method invocations on remote objects through the use of futures. By default, messages sent to remote objects are buffered until they can be sent. The programmer can also choose to break the connection and recall buffered messages.

SpatialViews takes a completely different approach than M2MI and AmbientTalk. SpatialViews is a language extension to Java ME which allows programs to iterate over groups of devices. The code inside the loop is executed on the initial device and then migrates to the next, eventually making its way back to the initial node. This allows for complex operations to be written easily, as the language has built-in support for such things actions as reduction operations. The iteration itself is generally done according to some physical layout, although it is possible to iterate over all objects or to use logical locations instead.

## 3.3 Middleware

Middleware is software which manages interaction and communication between applications, as well as providing various services which may be used by applications. Middleware may also include supporting libraries which can be used by applications.

LIME (Linda in mobile environment) [10] is a well-established implementation of tuple spaces [11] for mobile environments. Each device or agent has its own tuple space, which can merge with remote tuple spaces when they come into range of each other. Tuples can be read and written from specific locations, but can also be read or written to the "federated" tuple space which includes the local tuple space and any tuple spaces which are currently merged with it. However, the tuple will reside in a particular tuple space, so when that device or agent moves away, the tuples in that tuple space will move with it and be out of reach. LIME does not currently have an implementation intended for mobile devices smaller than laptops, though there are variations of LIME

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

**Table 1: Projects Summary**

| Project | Category | Disconnection Handling | Addressing and Discovery | Communication |
|---|---|---|---|---|
| LIME | Middleware | Tuple removal | Merged tuple spaces | Tuple space |
| MESH*Mdl* | Middleware | Tuple removal | Tuple exchange | Tuple space |
| TOTA | Middleware | Connectionless | Tuple propagation | Tuple space |
| STEAM | Middleware | Connectionless | Event content | Publish/subscribe |
| SyD | Middleware | Object proxies | Object type | Message passing |
| M2MI | Language | Connectionless | Object type | Message passing |
| AmbientTalk | Language | Flexible references | Object type | Message passing |
| SpatialViews | Language | Connectionless | Object type | Code migration |
| .NET CF | Runtime | None | URL | Sockets |
| Java ME | Runtime | None | URL | Sockets |

intended for sensors.

MESH*Mdl* [12] is another tuple space implementation, but varies slightly from the LIME model. In MESH*Mdl*, there is a single tuple space shared between all applications on a device. All communication between applications is performed via this shared tuple space. Remote tuple spaces are not shared like in LIME, but are accessible for reads and writes only: it is not possible to remove tuples from a remote tuple space. MESH*Mdl* supports mobile agents and recommends using them if actions need to be performed on a remote tuple space. MESH*Mdl* also adds the idea of being able to automatically write, read, or block tuples from other tuple spaces.

Tuples on the Air (TOTA) [13] also implements a tuple space for MANETs, but differs from LIME and MESH*Mdl*. Rather than storing tuples on a particular device, tuples in TOTA are propagated through the network according to rules specified per tuple. As the tuples move through the network, they can acquire context information about the network, such as how many hops they have traveled from the source.

SyD (System on Mobile Devices) [14] is a complete middleware solution for MANETs. The middleware centers around the idea of object registries which allows service registration and look up. Methods can then be invoked on these remote objects. Disconnection is handled by allowing objects to also provide proxy objects. If an object is unavailable, the method invocation will be handled by the proxy object, which can then perform an action specific to that service. For example, the proxy may buffer the request and send it later, or send back a cached or default response.

STEAM (Scalable Timed Events and Mobility) [15] is an event-driven middleware which uses a publish/subscribe [16] mechanism for propagating events. STEAM uses the concept of proximity groups for communication, limiting events to the local geographic area. Events are propagated by subscribers only when the subject and proximity match. Events are further filtered on the subscriber side by content, which determines if an event is delivered to the local application.

# 4. APPROACHES TO REQUIREMENTS

## 4.1 Disconnection Handling

The main challenge in MANETs is handling disconnections, which may be intermittent, prolonged, or permanent. For example, at a busy conference there may be many mobile devices in contact with each other, but distance and physical obstacles may cause intermittent disconnections. Routes may also break and re-form due to mobility or channel variations, possibly causing prolonged disconnections, but connections are eventually regained. When the attendees all leave, it becomes unlikely their devices will ever be in contact with each other again, making the disconnection permanent. A programming environment for MANETs must be able to handle all three kinds of disconnections.

One solution, used in LIME, MESH*Mdl*, and TOTA, is to use tuple spaces for communication. Tuple spaces exhibit both spatial and temporal decoupling, meaning that messages being sent do not need to be addressed to a particular recipient nor does the recipient need to be present when the message is sent. Tuple spaces generally operate by reading, writing, and taking tuples to and from a shared location. Rather than sending a message directly to a recipient, a tuple is written to the tuple space and can be read or taken from the tuple space by other clients. This allows the tuple space to withstand disconnections.

For example, a tuple may be written out to the tuple space and then retrieved by a different client an arbitrary amount of time later. The client which retrieves the tuple may not even be in existence when the tuple was written. However, there is still a problem if the writer of the tuple disconnects before the tuple is read by a receiver. For LIME and MESH*Mdl*, where the tuple space is associated with a particular devices, the tuple space is only available when the sender and the receiver are able to communicate directly with each other. In TOTA, tuples are disseminated throughout the network and can survive even if the original sender disconnects.

A different approach, used by M2MI and STEAM, is to forgo connections completely. Messages in M2MI are sent with no expectation of reply. In the general case, messages are sent to a particular object type, to be processed by any device hosting an object of that type. Messages are broadcast with no buffering whether or not there is a receiver available. This provides even more decoupling than tuple spaces, but tuple spaces have the advantage of having some feedback about a tuple's status. The sender can check if a tuple has been removed from the tuple space or not. If it has, the sender can have some assurance the tuple was received by someone, otherwise it will be available until removed by the original sender or another client.

STEAM avoids connections by filtering events on the subscriber's side. This eliminates the need for publishers to keep track of subscribers and completely decouples the two. However, publish/subscribe in a MANET environment does not provide any message reliability. Any message reliability

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

```
1. def print(doc) {
2.  when: Printer discovered: { |printer|
3.    when: (printer<-print(doc)) becomes: {|res|
4.      system.println("Status: " + res);
5.    };
6.  };
7. };
```

**Figure 1: Printer Discovery in AmbientTalk**

or disconnection feedback would need to be implemented on top of the publish/subscribe framework.

Code migration, the approach used by SpatialViews, does not maintain connections, but can be affected by disconnections if the device currently executing the mobile code fails or leaves the network before completion. Most of the devices in the network will not be involved in executing code at any particular moment, in which case their failure or disconnection from the network would not have an effect. When it does have an effect, however, it may cause the entire iteration to fail. This can be mitigated by using a form of parallel iteration over the devices. Since the iteration order in SpatialViews is nondeterministic, it does not provide message reliability.

A third approach, implemented in AmbientTalk, relies on event handling and futures. Event handlers can be registered for various events, such as discovery of a service or disconnection of a remote object. Figure 1 illustrates the use of two of these callbacks to discover a printer service. Once a printer is discovered, a document is sent to be printed and a status message is returned. If a remote object is discovered and later moves out of range, AmbientTalk can call the disconnection code. By default, messages sent to a disconnected remote object will be buffered until it is possible to send them. This provides a solution for intermittent and even prolonged disconnections. If a remote object is disconnected for too long, the programmer can recall all buffered messages and close the connection.

AmbientTalk also offers AmbientReferences [17], which are related to the M2MI model of object handles. AmbientReferences have a specified flexibility which determines how disconnections are handled. Sturdy is the default model of using buffered messages which will be delivered upon reconnection. Elastic references wait a specified amount of time before severing the connection and rebinding to another object of the same type. Fragile references will break immediately upon disconnection and rebind to another object.

The approach used by SyD is to offer the ability for the application designer to specify how to handle disconnections. In SyD, it is possible to provide proxy objects which will be called when the actual remote object is unavailable. These objects can then handle the invocation in an object-specific manner, such as buffering or returning a default value.

## 4.2 Addressing and Discovery

Unlike a wired network with a fixed infrastructure, MANETs cannot depend on centralized look up services like DNS to find peers. Since devices are constantly joining and leaving the network and it is not possible to maintain IP addresses or URLs to locate resources, applications must be able to locate them dynamically.

```
1. spatialview v = ChatService;
2.
3. visiteach c : v {
4.   c.receive(sender, message);
5. }
```

**Figure 2: Simple Messaging in SpatialViews**

The tuple space implementations of LIME, MESH*Mdl*, and TOTA automatically discover neighboring tuple spaces. LIME will merge tuple spaces with the same name, while MESH*Mdl* does not merge tuple spaces, but uses special tuples to provide a method of addressing a remote tuple space. Tuple spaces can be used for service discovery by writing out tuples which describe available services, or by writing out tuples intended for a specific service, which will read the tuples when it is available. Addressing is not necessary in general in tuple spaces, as it can be assumed a given service and a client will have pre-agreed upon tuple template to use for communication.

Object types for discovery and addressing is used by M2MI, AmbientTalk, and SpatialViews. This is based on the assumption that objects with the same name will implement the same services. M2MI and AmbientTalk use this with object handles which refer to a specific object type. When methods are invoked on a given handle, the remote object will correspond to the type of the object handle. M2MI, however, does not provide a method for discovery beyond manually sending out messages periodically and waiting for replies. AmbientTalk offers event handlers to be automatically called when objects of a specific type are discovered. These can be called exactly once or each time one is discovered.

SpatialViews uses object types along with spacial properties to define a "view" of the network. Once a view is created containing a given object type, SpatialViews provides a method of iterating over the available nodes within that view. The code within the iterator is executed locally on the remote devices. After the code is run, the device locates another nearby node hosting an object of the correct type and the code migrates there. Within the iteration, the code can synchronously invoke methods on the local service through an object handle. In Figure 2, a simple SpatialView is created to broadcast a message in a chat application. The code within the `visiteach` block (line 4) is executed locally. Therefore, unlike the AmbientTalk printer example, `c.receive(...)` is a local method call, not a remote call.

SyD also uses objects to invoke remote services, but it requires that these objects register themselves with neighboring devices, as well as locally.

The publish/subscribe model used by STEAM relies on subscribers knowing ahead of time what subscriptions are interesting to them. The publishers do not need to explicitly know who is subscribed, as messages are simply broadcast. However, it is possible to send out messages periodically or on demand which describe available subscriptions.

## 4.3 Flexible Communication

Basic communication between devices in a network is generally accomplished in a one-to-one unicast manner. However, in a MANET, group communication is also common, due to the broadcast nature of wireless networking and the

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

```
1. Container result = new Container();
2. spatialview v = Printer;
3. visiteach p : v {
4.    if(result.isEmpty()) {
5.       String result = p.print(document);
6.     if(result == "success")
7.        result.addElement(p.getName());
8.    }
9. }
```

**Figure 3: Printer Discovery in SpatialViews**

```
1. LimeTupleSpace lts =
2.   new LimeTupleSpace();
3. lts.setShared(true);
4. ITuple printjob =
5.   new Tuple().addFormal(PrintJob.class);
6. UbiquitousReaction ur =
7.   new UbiquitousReaction(printjob,
8.     this, Reaction.ONCEPERTUPLE);
9. lts.addWeakReaction(new Reaction[] {ur});
```

**Figure 4: LIME: Print Job Reaction**

limitations of bandwidth. Collaborative applications, networked games, and streaming media also benefit from group communication. Having both one-to-one and group communication available in the programming environment is necessary, though it may be possible to implement one with the other.

Tuple spaces lend themselves naturally to group communication. Tuples are written to shared storage space, which is globally accessible. Since tuples can be read without being removed from the tuple space, tuples are inherently one-to-many. One-to-one communication is not as directly supported by tuple spaces. However, tuples can be sent to a specific recipient by setting one of the fields in the tuple to an agreed-upon address. The specified recipient can look for tuples addressed to itself and take them from the tuple space.LIME, MESH*Mdl*, and TOTA support this type of communication.

M2MI and AmbientTalk support object references which can refer to all objects of a type, a selected subset of those objects, or a particular object. These handles directly correspond to broadcast, multicast, and unicast. Since AmbientTalk expects return values from messages, it is possible to receive multiple replies when sending a multicast or broadcast message, resulting in event handlers running multiple times or the return value being set more than once.

Communication in SpatialViews is done through code and variable migration. This makes it very simple to perform complex group operations such as reductions over several devices, but it makes one-to-one communication difficult. Figure 3 shows how it is necessary to set a variable to ensure a document is only printed by a single printer. There is also no method to provide reliable message delivery, other than iterating until a prearranged flag is set.

Similarly, publish/subscribe naturally supports group communication, but attempting to send a message to a particular recipient is not directly supported by the middleware. Publish/subscribe is intended to be used in situations with a single sender and multiple receivers and does not adapt well to sending a message to a single receiver. To do so would require the sender and receiver using a predefined addressing, similar to setting an agreed-upon tuple value in tuple spaces. Successful message delivery in the publish/subscribe is less likely than in a tuple space, since messages are not persistent in the way that tuples are.

# 5. APPLICATIONS

To better understand the effect of using different programming approaches when developing applications, two separate applications were written using AmbientTalk, LIME, and SpatialViews. These projects were selected because they represent very different approaches and had publicly available implementations. For each application, we discuss issues with disconnections, discovery, and communication.

## 5.1 Printer Discovery

The printer discovery application illustrates how the different projects can be used to approach the problem of resource discovery in a changing network. The client needs to locate a device offering a printer service. Next it sends the print job to a printer it has found, then waits for a reply. The printer processes the job and sends back a success or failure message.

Disconnection can occur at different points in this process. The printer may go out of range after the client has discovered it, but before the job is sent, or it may go out of range after the job is sent, but before the result is returned. In AmbientTalk, the default way of handling both cases is to wait until the printer can be contacted again and then resume the connection. The print job or result message will be buffered until the connection can be made again and then the message will be delivered. This works well in the case where there is only transient disconnection, but if a client has permanently left the area of the printer the application may wait forever unless the programmer explicitly uses a timeout.

In SpatialViews, the only way to communicate between nodes is to visit them in the course of an iteration over all nodes which offer a given service. With the SpatialViews implementation of printer discovery, disconnection after discovery and before sending back the success message are essentially the same. The iteration will never complete and the originating node will eventually timeout.

One of the strengths of tuple spaces is temporal decoupling. The sender and receiver do not both need to be present at the same time for a message to be sent. The LIME version of printer discovery does not face the disconnection issues above, partially because a print job remains in the tuple space until a result tuple is received. A printer which reads the print job and then goes out of range does not affect the operation. If the client is not in range when the result tuple is sent, but reconnects later, the result tuple will still be available for it to read. Even if the client permanently leaves an area, a different printer can pick up the print job instead. The downside of this approach is that multiple printers may process the same job, wasting resources.

In this example, addressing and discovery are needed to find a printer service and also to send the result message. AmbientTalk and SpatialViews handle addressing with interface types. The printer offers a service with a typed in-

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

terface and the client is able to find nearby services of a given type. Discovery is also built into both AmbientTalk and SpatialViews. In AmbientTalk, a callback function is set up to be called when the printer service is discovered, as shown previously in Figure 1. SpatialViews uses the idea of an iterator which loops over objects of a certain type nearby. In this respect, AmbientTalk is more reactive, while SpatialViews is proactive. The disadvantage to the SpatialViews approach is the service must be available at the time of the iteration, otherwise it will complete without a result. It will then be up to the programmer to retry the iteration until it is successful.

Discovery in LIME merely requires the registration of reactions to tuple templates. Addressing the printer is not necessary, but the result message contains a print job identifier so the client knows which job it represents.

Printer discovery does not really involve group communication, but there is one-to-one communication in the sending of the print job and the result message. In AmbientTalk, an object handle to the remote printer service is created when the the printer is found. The print job is then sent by invoking a method on the handle and waiting for a return value. On the printer side, it only needs to return a value, it does not require any knowledge of the client.

For SpatialViews, care must be taken to make sure the print job only goes to a single printer. Since the only method of communication is to visit every printer available, it is necessary to set a flag in a shared variable indicating the print job was already successfully printed and subsequent printers do not need to address it, as seen in Figure 3. Again, the printer does not need to know anything about the client.

In LIME, all communication is also inherently group communication, so the result tuple needs to be explicitly addressed to the client using a some sort of identification. The client will be waiting for a tuple with that specific ID.

## 5.2 Chat

The second example is a chat application. Clients can send out public or private messages. Public messages are delivered to all other chat clients nearby, while private messages are directed to a specific recipient. As in most chat applications, there is no history and clients do not expect to receive messages sent earlier or when disconnected. Disconnection can occur at any time while clients are exchanging messages.

Disconnection has less effect in this application than with printer discovery, as the clients do not depend on the delivery of messages to continue operating, although the AmbientTalk implementation does buffer both public and private messages. LIME also handles disconnection well in this case, since there is no need to guarantee message delivery.

SpatialViews suffers from the same issue in the printer discovery example: if the code migrates to a section of the network which then becomes disconnected from the rest of the network, or the current node goes down, the iteration just stops. Since each message is a separate iteration, it will not affect the overall operation of the application.

Addressing is handled similarly to the printer discovery example, except the user needs to know the names of other users when sending private messages. The AmbientTalk version notifies the user when other clients come into range and adds their name and object handle to a list. Figure 5 shows the implementation of the methods for sending messages.

```
 1. def all := ambient: Chatter
 2.     withCardinality: omni
 3.       withElasticity: fragile;
 4. def sendAll(message) {
 5.   all<-send(message, name);
 6. };
 8. def send(buddy, message) {
 9.   def b := buddy_list.get(buddy);
10.   b<-send_private(message, name);
11. };
```

**Figure 5: Chat in AmbientTalk**

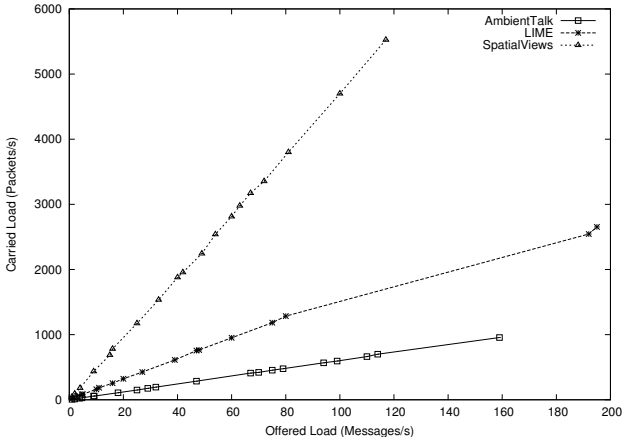


**Figure 6: Communication Overhead with Wired Links**

Public messages are sent out to an AmbientReference, defined in line 1, which only needs to know the interface name and implicitly tracks individual clients.

SpatialViews, like in the printer discovery application, iterates over all nodes with the chat interface, delivering the message to each as it visits. This is shown in Figure 2. For private messages, it still must iterate in the same manner, but the recipient is encoded in the message. Unfortunately, this means private messages still require visiting every node, possibly without even reaching the recipient in the case of disconnection. Likewise, LIME must rely on encoding the recipient in the message tuple and assuming no one but the intended client will read the message.

Group communication is natural for public messages and one-to-one communication for private messages. All three projects handle group communication well. AmbientTalk has omni-handles which refer to all interfaces of a given type and will broadcast the message to all nearby clients. The only communication in SpatialViews and LIME are essentially group communication, so for one-to-one communication, SpatialViews and LIME require the programmer to implement an addressing scheme on top of the group communication. The client side of the application needs to pick out private messages intended for it and ignore the rest.

# 6. EXPERIMENTAL RESULTS

This section compares the performance of AmbientTalk, LIME, and SpatialViews in a regular wired LAN and in a

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

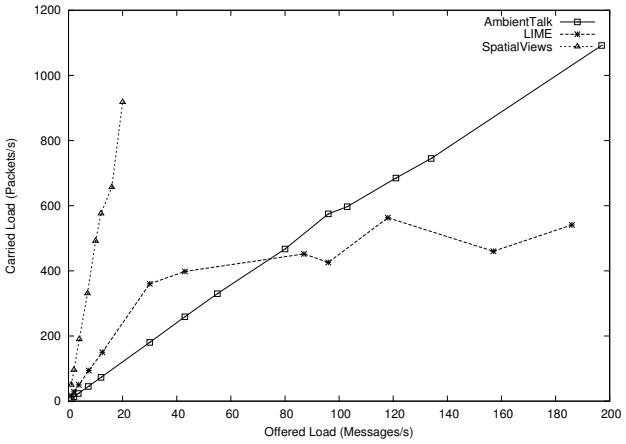


**Figure 7: Communication Overhead with Simulated Wireless Links**

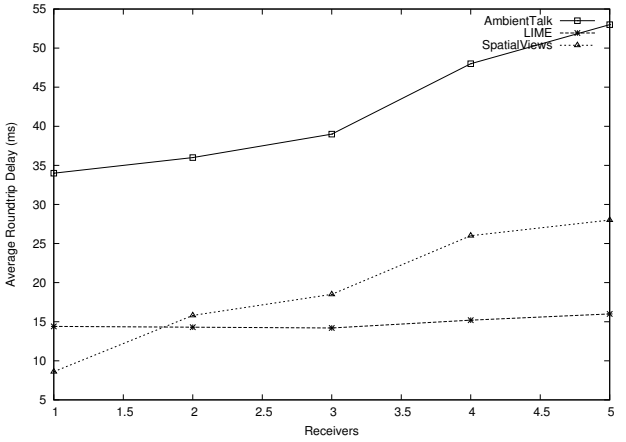


**Figure 8: Group Communication with Wired Links**

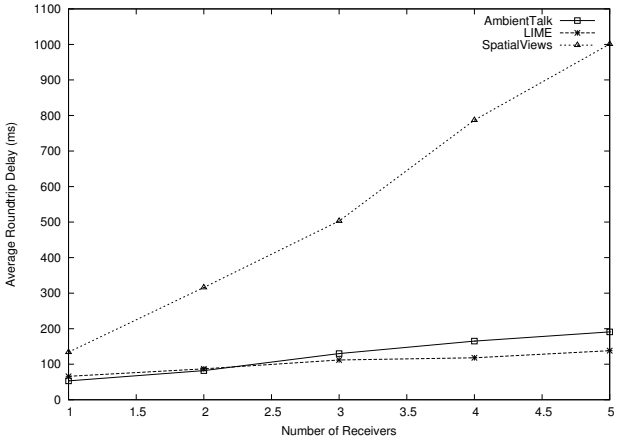


**Figure 9: Group Communication with Simulated Wireless Links**

MANET context using the QualNet and the IP Network Emulation library [18], which allowed us to evaluate real applications on actual nodes with simulated mobility and wireless network.

The wired LAN environment provides a nearly ideal network in which the cost of communication is very low, there is little contention for the communication channel, all nodes are connected directly to each other, and collisions are minimal. By minimizing these factors, it is possible to focus the experiment results on the overhead of the programming environments. In these experiments, all nodes were directed connected to a 100Mb/s switch, providing essentially an independent, one hop channel for each pair of nodes

However, it is also desirable to perform tests which reflect the requirements from Section 1. QualNet provides an emulated wireless protocol stack and simulation of the wireless channel and mobility. The results from the experiments using QualNet more accurately reflect the MANET environment. For these experiments, all communication was performed through QualNet, which emulated an 802.11 ad hoc wireless network with an available bandwidth of 11Mb/s.

## 6.1 Communication Overhead

AmbientTalk, LIME, and SpatialViews use very different messaging systems. This experiment demonstrates the overhead for each using a client-server setup as the simplest base case. Messages are sent out from the sever to the client at an increasing rate. The number of IP packets generated by doing so include control and discovery packets. Each node is within wireless range of the others so all communication is performed over single hop routes. Figure 7 and Figure 6 show the results from wired LAN and QualNet, respectively.

AmbientTalk has the lowest overhead, as it is simply performing a method call on a remote object and there is no return value. LIME requires some communication to alert merged tuple spaces of the messages' presence and then more communication to actually transfer the tuple. SpatialViews shows the highest amount of overhead, which is expected since it is migrating code and data to communicate a simple message.

Although the results were similar in the wired LAN and QualNet, the performance of SpatialViews was considerably slower, peaking at 20 msgs/s, while in the wired LAN it was possible to reach 117 msgs/s. This is due to contention for the wireless channel. It is also worth considering that LIME and AmbientTalk use asynchronous messages while SpatialViews uses a blocking synchronous message send. This allows LIME and AmbientTalk to take advantage of system level buffers, while SpatialViews cannot.

## 6.2 Group Communication

In this experiment we consider the common situation where one node needs to request information from the rest of the network and then collect the results, with increasing numbers of receivers. The application sends out a message then measures the time elapsed for responses. For SpatialViews, this involves visiting each node and that node then visiting the sending node. The AmbientTalk version uses an omnihandle, as in the chat application, to broadcast the handle of the sender and then the receivers use the handle to send a return message. For LIME, each message is sent as a tuple, to which the receivers send a response tuple. Again, the network is set up so that no node is farther than one hop

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

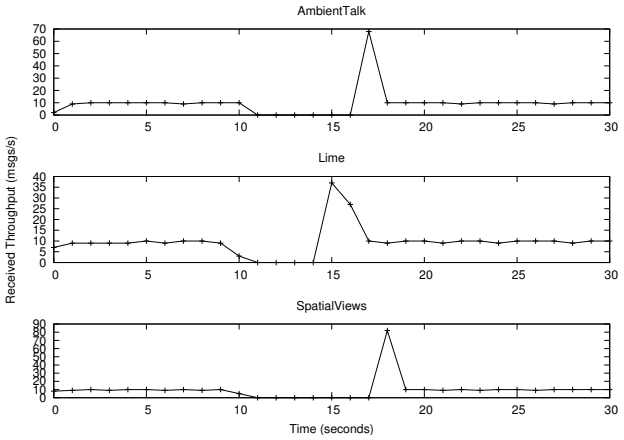


**Figure 10: Disconnection Recovery**

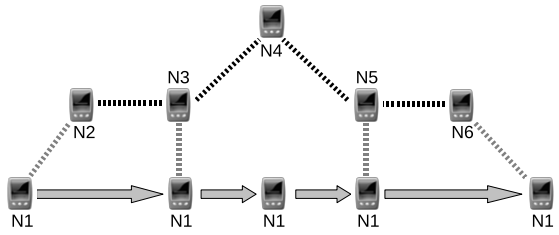


**Figure 11: Simulated Mobility Scenario**

from any other node. Figures 9 and 8 show the results.

In the wired LAN, LIME shows the least variation as the number of receivers increases. This is because the sender writes out a single tuple and each receiver can respond independently and in parallel. SpatialViews slows down considerably as the number of receivers increases, since SpatialViews visits each receiver in turn and waits on a response before continuing. The delay for AmbientTalk is the highest but does not increase quite as quickly as SpatialViews. Although AmbientTalk uses a single send at the application level, messages to individual receivers are sent serially, causing the delay for the last receiver to be higher than the first.

When run using QualNet, the effect of using the wireless channel is seen again. The delay with AmbientTalk and LIME increases, but not as dramatically as SpatialViews, which reaches a delay of about 1 second with 5 receivers, while a single receiver averages 134 ms. As in the previous experiment, the traffic generated by SpatialViews quickly creates conflicts in the wireless channel, causing retransmission and delay at the MAC layer.

## 6.3 Mobility and Disconnection

In order to isolate and examine disconnection recovery, a simpler experiment in a wired LAN was performed, still using the same client-server application. In this case, a 5 second disconnection was caused by turning the network interface off and then turning it back on. Each project reacted similarly, as shown in Figure 10 though LIME showed the fastest recovery time. Interestingly, SpatialViews exhibited

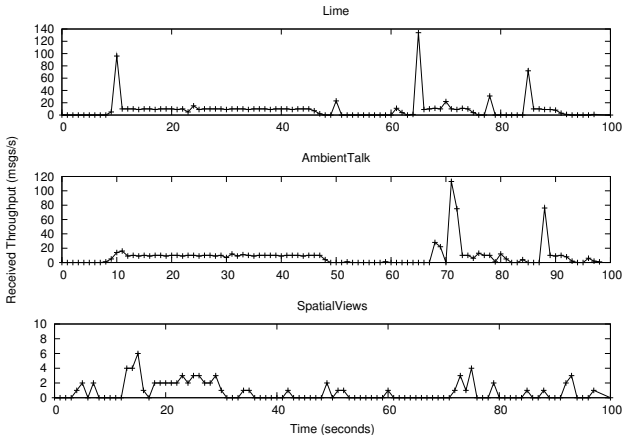


**Figure 12: Client-Server Throughput with Mobility**

delivery of buffered messages. As SpatialViews does not buffer messages itself, this buffering was the result of the operating system attempting to locate the remote node.

Using QualNet, it was possible to evaluate the projects in a mobile environment which provided disconnections, routing changes, and multi-hop communication. For this experiment, the network layout and mobility pattern shown in Figure 11 was used. Node 1 user the same client-server application as in the first experiment and is attempting to send messages to Node 4. The distance between the two nodes forces a multi-hop route through the intermediate nodes. Node 1 moves from left to right at a constant rate during a time period of 100 seconds. This experiment again measures message delivery rate. Results are shown in Figure 12.

The large spikes for the AmbientTalk and LIME results indicate the delivery of buffered messages. For AmbientTalk, the sender did not begin until the receiver was discovered, while LIME began sending messages immediately. The flat part of the graphs indicates when Node 1 was in between Node 3 and Node 5 and was outside the range of both. SpatialViews did not perform well in this experiment because it lacks the sophisticated disconnection handling and message buffering of the other two experiments. Also, the code migration was difficult, more time consuming, and more susceptible to disconnections. As in the previous experiments, this demonstrates the difference between experiments using a wired LAN compared to a simulated wireless network.

# 7. RELATED PROJECTS

The following projects have been included for completeness and because they exhibit unique features, but were excluded from the main discussion.

JANE [19] works by providing an event-based model with guaranteed network feedback. JANE also provides mobility and network emulation software which allows programs to be tested on simulated devices as well as real devices.

Pervaho [20] is another publish/subscribe middleware for MANETs, but includes the concept of persistent publications. This provides the temporal decoupling missing from STEAM.

TypeCast [21] is a recent project which uses type hierarchies to provide efficient routing for messages in a MANET.

*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*

The type of an object provides addressing and routing information.

EgoSpaces [22] is another tuple space middleware which focuses on context awareness.

# 8. CONCLUSIONS

This paper has discussed challenges specific to mobile ad hoc applications. We presented and evaluated existing projects based on tuple spaces, remote object handles, event handling, publish/subscribe, and code migration. Two applications were developed using three selected projects in order to evaluate their advantages and disadvantages with actual applications. Finally, several experiments were run on both a wired and simulated wireless network to quantify differences between AmbientTalk, LIME, and SpatialViews. The difference in resulting performance showed the necessity of testing these programming environments in a MANET context and the impact of expensive communication schemes. Future work would include more complex mobility and routing scenarios.

The approaches presented in this paper are not unique to MANETs, but have been adapted from familiar parallel and distributed computing concepts. Although the simple applications in Section 5 used languages specifically made for MANETs, it was still necessary to work around limitations of the approaches taken. Programming abstractions designed specifically for MANETs are necessary to fully handle the frequent disconnections, dynamic topologies, and varying communication patterns inherent to MANETs. Applications for MANETs are a currently expanding field and useful abstractions along with tools for testing can increase their rate of development and deployment.

# 9. REFERENCES


[1] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, 2003.

[2] T. Clausen, P. Jacquet, and P. Jacquet. Optimized link state routing protocol (olsr), 2003.

[3] S. Hadim, J. Al-Jaroodi, and N. Mohamed. Middleware issues and approaches for mobile ad hoc networks. *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, 1:431–436, 8-10 Jan. 2006.

[4] Sun Microsystems. http://java.sun.com/javame/.

[5] Microsoft. http://msdn.microsoft.com/en-us/netframework/.

[6] Qualcomm. http://brew.qualcomm.com/brew/.

[7] Alan Kaminsky and Hans-Peter Bischof. Many-to-many invocation: a new object oriented paradigm for ad hoc collaborative systems. In *OOPSLA '02: Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 72–73, New York, NY, USA, 2002. ACM.

[8] Tom Van Cutsem, Stijn Mostinckx, Elisa Gonzalez Boix, Jessie Dedecker, and Wolfgang De Meuter. Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks. In *SCCC '07: Proceedings of the XXVI Intern. Conf. of the Chilean Society of Comp. Sci.*, pages 3–12, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Yang Ni, Ulrich Kremer, Adrian Stere, and Liviu Iftode. Programming ad-hoc networks of mobile and resource-constrained devices. *SIGPLAN Not.*, 40(6):249–260, 2005.

[10] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.

[11] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *Computer*, 19(8):26–34, Aug. 1986.

[12] Klaus Herrmann, Klaus Herrmann, Gero Mühl, Gero Mühl, Michael A. Jaeger, and Michael A. Jaeger. Meshmdl event spaces - a coordination middleware for self-organizing applications in ad hoc networks. *Pervasive Mob. Comput.*, 3(4):467–487, 2007.

[13] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications with the tota middleware. *Perv. Comp. and Comm., 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 263–273, 14-17 March 2004.

[14] Sushil K. Prasad, Vijay Madisetti, et al. Syd: a middleware testbed for collaborative applications over small heterogeneous devices and data stores. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 352–371, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[15] René Meier and Vinny Cahill. Steam: Event-based middleware for wireless ad hoc network. In *ICDCSW '02: Proc. of the 22nd Intern. Conf. on Distributed Computing Systems*, pages 639–644, Washington, DC, USA, 2002. IEEE Computer Society.

[16] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[17] Tom Van Cutsem, Jessie Dedecker, et al. Ambient references: addressing objects in mobile networks. In *OOPSLA '06: ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 986–997, New York, NY, USA, 2006. ACM.

[18] Qualnet. http://www.scalable-networks.com.

[19] Daniel Gorgen, Hannes Frey, and Christian Hiedels. Jane-the java ad hoc network development environment. In *ANSS '07: Proceedings of the 40th Annual Simulation Symposium*, pages 163–176, Washington, DC, USA, 2007. IEEE Computer Society.

[20] Patrick Eugster, Benoit Garbinato, and Adrian Holzer. Pervaho: A development and test platform for mobile ad hoc applications. *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual Intern. Conf. on*, pages 1–5, 17-21 July 2006.

[21] Jinsong Lin, Thomas Phan, and Rajive Bagrodia. Typecast: Type-based routing in wireless ad-hoc networks. *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual Intern. Conf. on*, pages 1–10, July 2006.

[22] C. Julien and G.-C. Roman. Egospaces: Facilitating rapid development of context-aware mobile applications. *IEEE Transactions on Software Engineering*, 32(5):281–298, May 2006.



*Digital Object Identifier: 10.4108/ICST.WICON2008.4932*
*http://dx.doi.org/10.4108/ICST.WICON2008.4932*