

A Quantitative Comparison of Communication Paradigms for MANETs

Justin Collins and Rajive Bagrodia

University of California, Los Angeles
Los Angeles, CA
{collins,rajive}@cs.ucla.edu

Abstract. Mobile ad hoc networks (MANET) present a challenging area for application development. The combination of mobile nodes and wireless communication can create highly dynamic networks with frequent disconnections and unpredictable availability. Several language paradigms have been applied to MANETs, but there has been no quantitative comparison of alternative approaches. This paper presents the first quantitative evaluation of three common communication paradigms (publish/subscribe, RPC, and tuple spaces) compared within realistic MANET environments using real applications. We investigate the application-level performance of the paradigms and present a summary of their relative strengths and weaknesses. We also demonstrate the impact of wireless and mobility on application-level metrics, the most dramatic being delivery rates dropping to nearly 25% and round trip times increasing up to 2000% in a mobile scenario.

1 Introduction

In mobile ad hoc networks (MANET), high nodal mobility causes frequent topology and route changes, making it difficult to maintain connections between nodes. Many routes in the network span multiple wireless hops which may experience dramatic and unexpected fluctuations in quality. The combination of mobility and wireless communication creates highly dynamic network topologies in which frequent, possibly permanent disconnections are commonplace. The dynamics of the network and the wireless channel requires changes to the networking stack and alternative solutions at the application level.

Middleware, frameworks, libraries, and languages have been proposed for meeting the challenges of developing mobile and ubiquitous applications. Quantitatively comparing these projects is difficult, since they are implemented in different languages, with different feature sets, and with varying levels of completeness. In this paper, we examine the fundamental differences in communication paradigms commonly used in projects, rather than the projects themselves. We accomplish this by evaluating representative implementations of each paradigm.

We also use real applications utilizing each of the paradigms, allowing us to investigate the impact of the paradigms on application-level metrics. Of prime importance to MANET applications is the performance of the communication

model when the wireless channel and mobility are introduced. Therefore, we have evaluated the communication paradigms using a high-fidelity emulation of the network stack and detailed models of the wireless channel.

Since these communication models are common across multiple projects and are likely to be used in future projects, the results of this study have wide applicability. While previous work has qualitatively compared a subset of these paradigms or reported experimental results at the individual project level, we have investigated the performance characteristics of the underlying communication models themselves.

In this paper, we present the first quantitative comparison of three communication paradigms - publish/subscribe [1], remote procedure calls [2], and tuple spaces [3] - using canonical implementations within real applications. Our results show wide variation in paradigm performance within the same scenario. Publish/subscribe provides fast, cheap message delivery, with a message overhead of 357 bytes and median round trip times of $<400ms$ even with mobility. RPC supplies good delivery ratios when a reply is expected, achieving a 94% delivery ratio with mobility, while publish/subscribe and tuple spaces dropped to 75% and 72%. In the whiteboard application, however, tuple spaces were able to deliver all messages in 5 out of 6 scenarios, including a scenario in which RPC only delivered 25%.

2 Evaluation Architecture

The architecture used for this comparison has three layers: a network emulator, which provides a scalable and realistic MANET environment; the communication components, which implement the paradigms; and the applications which utilize the paradigms. The application is built on top of the communication component, which communicates over a regular wired network. The traffic from the network is routed through the network emulator, which provides a high fidelity simulation of the wireless network, intermediate nodes, and mobility. This allows applications to be written independently of whether the network is real or emulated.

Network Emulation Because application-level performance is affected by variations in the wireless channel, mobility, and disconnection patterns, it is essential to have an accurate representation of the network stack and the wireless channel [4, 5]. EXata provides a high fidelity emulation of the entire network stack, using real MAC and routing protocols, as well as detailed simulation of the effects of the wireless channel and mobility, such as fading, shadowing, and path loss [6]. The emulator allowed us to run actual applications, rather than models, in a realistic representation of the MANET environment while retaining precise control of the variables in each experiment. This ensures our results are a fair comparison and not influenced by transient environmental effects.

Communication Components Each of the three communication paradigms are implemented in Java on top of Apache MINA¹, a high-performance networking library. A summary of the interaction between the application, the

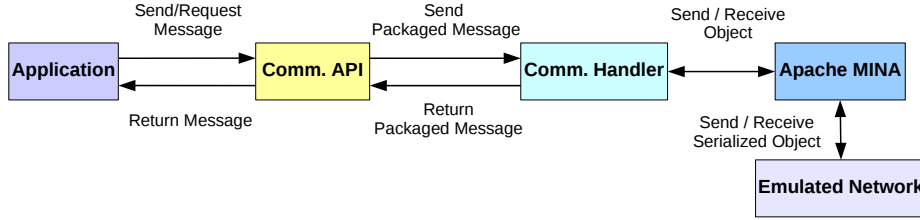


Fig. 1. Communication Components

communication components, and the networking library is illustrated in Figure 1. The libraries are intended to be functionally equivalent implementations of each paradigm to keep the comparison as fair as possible.

For publish/subscribe, we implemented a simple topic-based system. Publications to topics are broadcast to all subscribers available at time of publication. Our RPC implementation uses a reflection-based mechanism for invoking methods on remote objects, which are addressed by class. Parameters are copied to the remote machine and there is always a return value. The tuple space implementation we used is largely modeled on LIME [7] and uses the same local tuple space library. All three implementations support unicast, multicast, synchronous, and asynchronous communication.

Applications The first application used for these experiments is a simple client-server application which can send messages between hosts. This provides a baseline for the performance results and allows us to easily test performance with varying message sizes and frequency. The second application is a shared whiteboard. Collaborative applications are often cited as use cases for MANETs and the shared whiteboard is a common example [8–11]. This provides a non-trivial, realistic test case for each of the three communication paradigms.

3 Experimental Results

In the following sections, we present measurements of message delay and message delivery reliability for unicast and group communication, as well as for a non-trivial whiteboard application. We also examine the message overhead and the influence of routing algorithms. These experiments demonstrate the impact of the wireless network and mobility at the application level.

We compared application-level metrics using unicast and group communication in three network scenarios which are used throughout the experiments: a single hop, static network; a multi-hop, static network; and a fully mobile network. Each node in the emulated network is equipped with an 802.11b wireless interface. The two-ray model is used for path loss. Based on preliminary results, we used DSR [12] as the routing protocol for the static scenarios and AODV [13] for the mobile scenario.

¹ Multipurpose Infrastructure for Network Applications: <http://mina.apache.org/>

The mobile scenario uses random waypoint mobility with a pause time of 30s and maximum speed of 1 meter/second, representing pedestrians carrying handheld devices. The nodes move within a 1500m x 1500m indoor space where transmission range is limited to 50m. To avoid network segmentation, the scenario ensures there are always possible routes between any two nodes by having four fixed nodes. The mobility pattern in each experiment is identical.

3.1 Unicast Communication

Table 1. Message Sequence Overview

Paradigm	Sender	Receiver	Size (bytes)	Overhead (bytes)
Publish/Subscribe		Subscribe	175	
	Publish		1182	
		<i>Total</i>	1357	357
RPC	Search		146	
		Search Reply	187	
	Invoke		1238	
		Return Value	152	
		<i>Total</i>	1571	571
Tuple Space		Search	608	
		Search Reply	133	
		Tuple Request	588	
		Tuple Reply	1586	
		<i>Total</i>	2915	1915

Message Overhead

Application Overhead The first step of our experimental evaluation of these three paradigms is discovering the basic cost of communication. Table 1 provides an overview of the sequence of messages involved when using each of the communication paradigms in the simple case of a single sender and a single receiver sending a 1KB payload. The total size includes the 1KB payload. Publish/subscribe requires only two messages to be sent: one to subscribe to a topic and one to publish. Since publish/subscribe only needs to add a string indicating a topic, there is very little overhead added to the original message.

RPC first sends out a query to find the desired remote object. Once found, it sends a second message to invoke the method and transfer any arguments. The final message in the sequence is the return value from the method, which is dependent on the size of the return value.

Tuple spaces require the same number of messages as RPC, but the overhead is 2.3 times higher. Except for the search reply messages, all messages include a tuple object, making them larger than the simple messages exchanged in RPC.

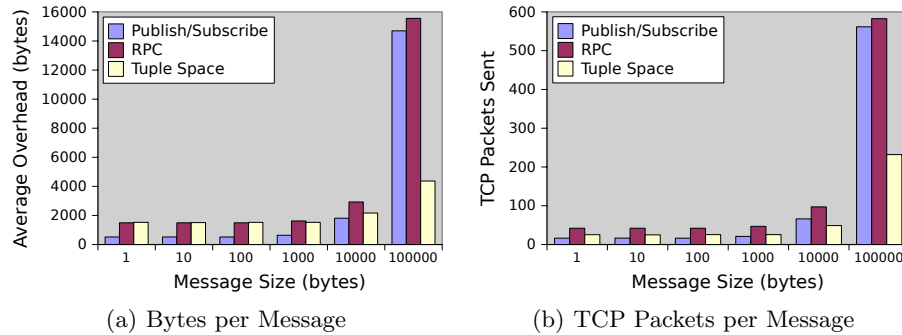


Fig. 2. Average Message Overhead

Network Overhead While Table 1 indicated the overhead added at the application layer, Figure 2(a) shows the average amount of TCP traffic which is sent over the network for a single message, calculated as *bytes sent - message size*. These results use the single hop static scenario and are averaged from 50 messages.

The results are fairly constant until the packet size is exceeded. There is some increase at 10KB, and a dramatic increase at 100KB. Figure 2(b) shows the same data in terms of TCP packets and indicates the cause of the sharp increase in traffic at 100KB is the result of packet fragmentation.

Despite having large message sizes, tuple spaces have much lower overhead in terms of TCP traffic. This difference arises from a side issue related to TCP send window sizes. For tuple spaces, where the receiver initiates the connection, the TCP send window size grows to accommodate larger packet sizes. With RPC and publish/subscribe, the send window size remains constant, causing the large messages to be split into many more packets.

Message Reliability How reliably a communication paradigm handles message delivery has a direct impact on the application layer. The more reliable the communication paradigm, the less responsible the application is for handling lost messages. We measured reliability in terms of message delivery. In the single hop scenario, all paradigms achieved 100% delivery and figures 3(a) and 3(b) indicate nearly perfect message delivery for all the paradigms in the unicast scenario. Publish/subscribe performed the worst and still only lost 4 messages.

Message Delay Message delay is another important application-level metric, as it determines how quickly information is transferred and the freshness of the application's information. Figures 4(a), 4(c), and 4(e) show delay in terms of round trip times for each paradigm in a single hop scenario. The majority of the messages in each paradigm are under the 200ms mark, with just a few wayward messages taking longer. Even for tuple spaces, 80% of the messages take less than 400ms to complete their round trip. However, some messages take much longer, up to 8s. For tuple spaces, this is partially due to the complexity and overhead of the messages required to perform the round trip message delivery.

However, the time delay for tuple spaces in the single hop scenario is also related to the pull (rather than push) nature of the paradigm. A tuple is timestamped when it is output, but the tuple is not actually sent to the receiver until the receiver requests it. The same situation happens on the return trip, when the tuple must be pulled back to the original sender. Any delays in this process cause the round trip time to increase.

On the other hand, publish/subscribe messages are sent out almost immediately after being timestamped. Nearly all the delay is caused by the network itself. RPC has more potential for delays since it must find the remote method before invoking it. However, the return message can reuse the existing TCP connection, which appears to provide an advantage over tuple spaces.

The mobile scenario introduces even greater delays. Routes are changing frequently and may be several hops long. While the publish/subscribe and RPC results are clustered around $100ms$ and remain under $500ms$, the tuple space values are considerably higher with a median at $256ms$ and a high of nearly $20s$. This is again due to the pull nature of tuple spaces and the overhead seen in Section 3.1.

3.2 Group Communication

Group or multicast communication is a useful but more complex part of MANETs, where information and resources are often disseminated in a peer-to-peer manner. Group communication differs significantly from unicast communication. Given the mobile characteristics and decentralized nature of MANETs, a group's membership may be in constant flux, so it is unlikely a sender has perfect knowledge of the members of the group. The time difference between replies from members of the group may vary greatly, and the initiating node cannot know how many replies to expect.

We have investigated how well each paradigm handled group communication by again evaluating message delay and message delivery reliability, but with multiple receivers.

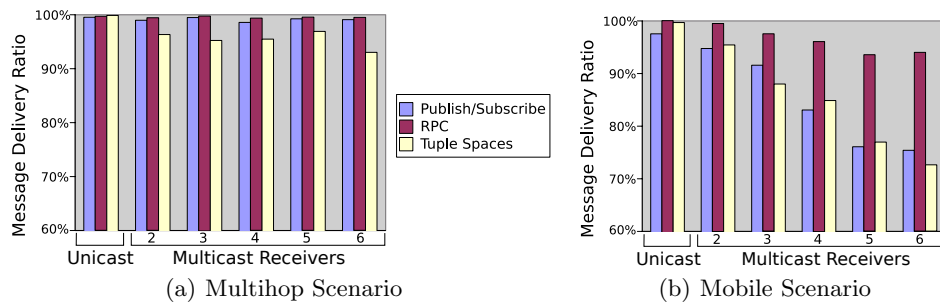


Fig. 3. Message Reliability

Message Reliability With this application, message reliability refers to messages which make the circuit from the sender to the receiver and back to the sender. This is useful, for example, in situations where a sink node aggregates information from other nodes.

Figures 3(a) and 3(b) show the percentage of messages successfully completing the round trip. The single hop scenario is not shown, as all paradigms achieved $> 99\%$ reliability in that scenario. In the multi-hop scenario, there are more losses even without mobility, but there is no significant trend as the number of receivers increases.

RPC has a slight advantage with this metric, as it will wait until at least one receiver is available. Publish/subscribe and tuple space will send out messages whether or not any receivers are available at the time. However, none of the communication paradigms will retry a message which is lost in transit. A message lost anywhere in the circuit causes the entire attempt to be reported as a failure.

This contributes to tuple spaces showing the lowest delivery ratio (93%) in the multi-hop scenario and a low delivery ratio (72.6%) in the mobile scenario. While tuple spaces can easily handle the delivery of the outgoing tuple, it is more difficult to guarantee the return of the reply tuple. If a node is not available to receive the request broadcast for a reply tuple, then the reply will never be sent even if the original outgoing tuple is received.

Message Delay We again consider round trip time for each of the paradigms, but this time with an increasing number of receivers. Figures 4(a) - 4(i) show the results for each paradigm and scenario.

For the single hop and multi-hop scenarios, where there is no mobility, the majority of the round trip times are fairly fast. The bottom 75% of the messages have very similar results, while the top 25% varies much more. This indicates that an application can expect most messages to be delivered quickly or not at all, but about a quarter of the messages may arrive up to minutes later.

The median delay does increase as receivers are added, especially in the mobile scenario. In the static scenario, the median delay publish/subscribe increased $121ms$ from two receivers to six receivers. RPC increased $147ms$, and tuple spaces increased $140ms$. For the mobile scenario, the median times for publish/subscribe increased $255ms$, RPC increased $237ms$, and tuple spaces increased by $2035ms$. The maximum delay values varied much less predictably. For tuple spaces, the static scenarios have unusually long delays with two and three receivers. In the static scenarios, the first three receivers are located in close proximity. One node would dominate the channel for several seconds before relinquishing it. Once again, this shows how influential the wireless channel is on the performance and behavior of applications in MANETs.

The median and maximum tuple space results are much longer than the other two paradigms. The median delay for tuple spaces ranges from twice as much as publish/subscribe in the single hop scenario up to 6 times as high in the mobile scenario. For publish/subscribe and RPC, the majority of delays can only be caused by the network, since they do not attempt to retransmit messages. Tuple spaces, on the other hand, can have very large delays due to the paradigm itself.

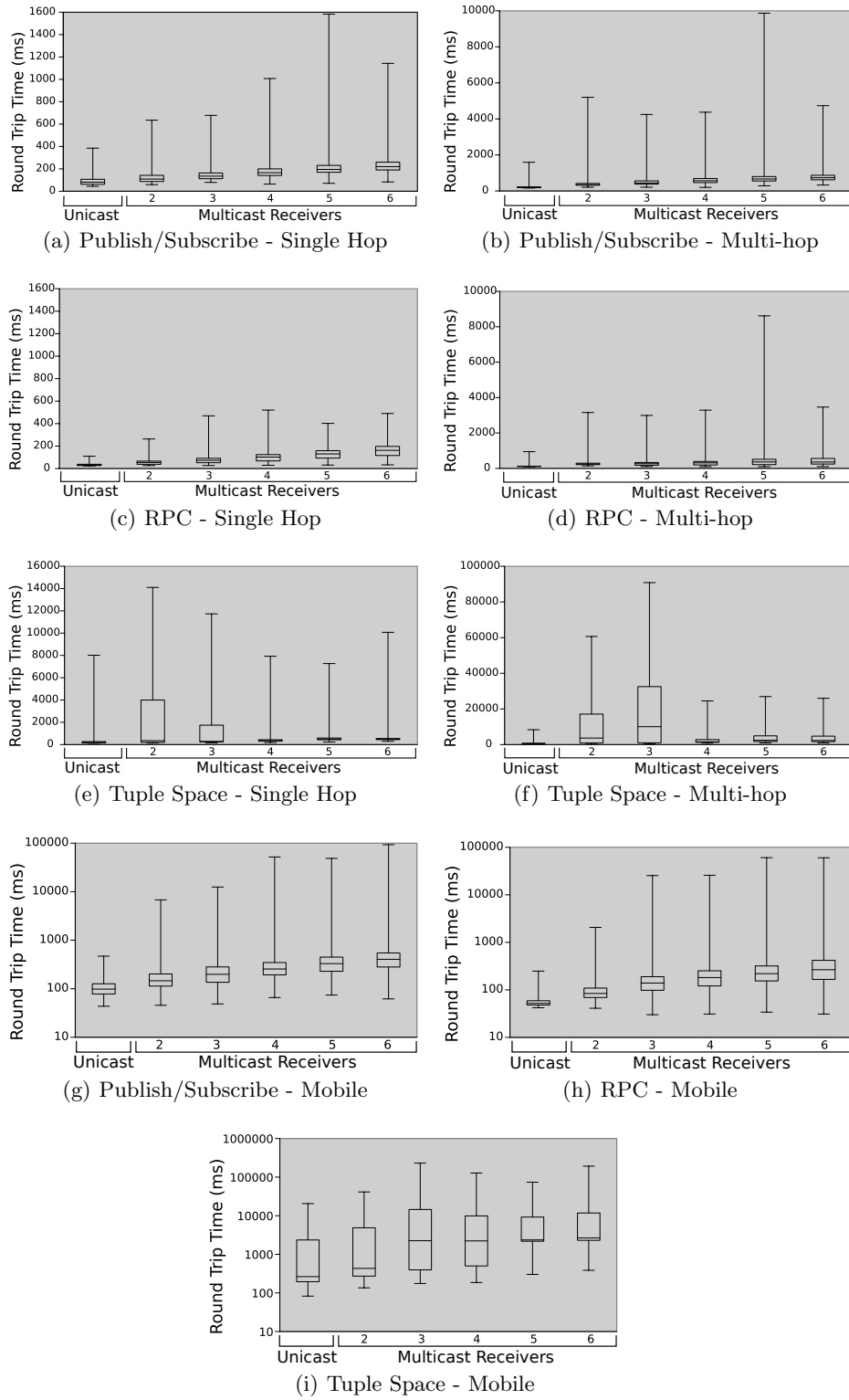


Fig. 4. Round Trip Times

If a receiver is “behind” it may spend time receiving older tuples before the newest tuple is requested. This causes the round trip times to increase while only improving one-way message delivery.

3.3 Shared Whiteboard Application

When testing the whiteboard application, we considered the metrics which a user might care about at the application level: how reliably and quickly users receive updates. In the results below, a single user is updating the whiteboard and the updates are propagated to 6 receivers. We used traffic traces from Coccinella² to ensure our implementation accurately represented a typical whiteboard application. For these experiments, 250 whiteboard update messages of varying sizes were sent out over a 10 minute period at varying intervals.

Furthermore, we tested the whiteboard application under the two different routing protocols we have been using, AODV and DSR. This is not meant to be an exhaustive comparison of the routing protocols themselves, but is intended to show how the choice in routing protocols might affect the performance of the communication models in a nontrivial application.

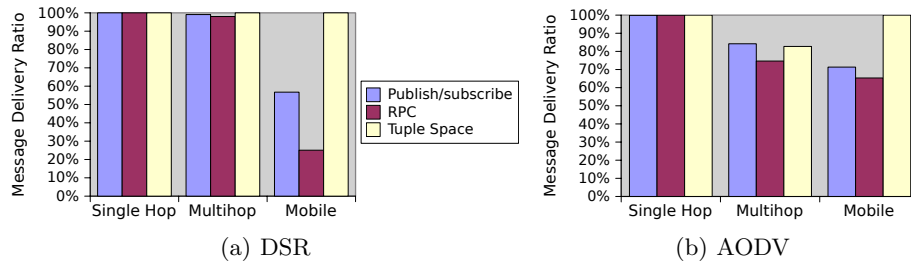


Fig. 5. Whiteboard Message Delivery

Message Reliability Unlike the previous results, these represent one-way communication from the whiteboard user to the receivers. Message reliability determines how accurately the receivers’ views reflect the state of the shared whiteboard.

Figures 5(a) and 5(b) show the percentage of whiteboard messages delivered for DSR and AODV, respectively. As before, the results are nearly 100% for all paradigms and both protocols in the single hop network. AODV performs poorly on the multi-hop scenario, while DSR achieves nearly 100% delivery for all paradigms. On the other hand, DSR performs much worse in the mobile scenario, with the delivery ratio for RPC only reaching 25%.

² <http://thecoccinella.org/>

The reliability of tuple spaces is considerably better in these experiments than in the round trip scenario, with 100% delivery in all but the AODV multi-hop scenario. The difference between these results and Section 3.2 is the lack of a return message. Each receiver is responsible for requesting the whiteboard updates, so the blocking request will be retried until the tuples are received. The only exception is the multi-hop scenario with AODV, in which all three paradigms perform much worse. Since all three paradigms are affected equally, these results must be directly due to the behavior of AODV in this scenario. Investigation of this phenomenon is outside the scope of this paper.

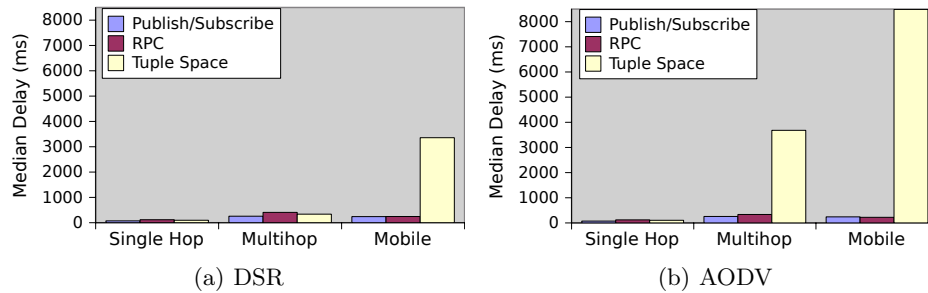


Fig. 6. Whiteboard Message Delay

Message Delay Message delay is measured as the time from when a whiteboard update is sent by the application until it is delivered to the receiver’s whiteboard. Update delays are very noticeable in a shared whiteboard application, so the delay time should be minimized.

Figure 6(a) shows the results when using DSR and Figure 6(b) shows the AODV results. Not unexpectedly, tuple spaces have the highest median latencies of $8,486ms$ with AODV and $3,357ms$ with DSR. For publish/subscribe and RPC, the median delay remained under $400ms$.

With DSR, tuple spaces report a nearly 100% delivery ratio in every scenario, yet the delay times are $<400ms$ in the static scenarios. In contrast, AODV causes long delays for tuple spaces in both the multi-hop and mobile scenarios. Since tuple spaces will repeatedly attempt to deliver messages, retries are expected to contribute to the majority of the delays. This is supported by the long delay times experienced by tuple spaces with AODV in the multi-hop scenario. However, in the mobile scenario tuple spaces achieve 100% delivery with AODV and DSR, but the median delay with DSR is less than half as with AODV.

From the mobile reliability results, we can infer that DSR does not maintain viable routes, because the results of publish/subscribe and RPC are poor. However, the delay results suggest DSR is faster than AODV at finding new routes when they become available.

4 Related Work

There are many projects using publish/subscribe, RPC, and tuple spaces specifically in MANETs. M2MI (Many-to-Many Invocation) [14] adapts RPC to a MANET context. LIME (Linda in a Mobile Environment) [7] is a tuple space implementation intended for mobile devices. STEAM (Scalable Timed Events and Mobility) [15] is an example of an event-based middleware which uses publish/subscribe for communication.

Middleware for MANETs is surveyed in [16] and [17], but no quantitative results are presented. Projects using different communication paradigms were compared in [18]. [19] implements tuple spaces in terms of a modified publish/subscribe, but does not provide quantitative results.

5 Conclusions

Publish/subscribe, remote procedure calls, and tuple spaces are three communication paradigms which have been applied to MANETs. They have been used as the basis for many projects and applications intended to operate in MANETs. This paper presented a quantitative comparison of these three paradigms in three different network scenarios, with a focus on application level metrics. The results show the relative strengths and weaknesses in each of the three paradigms, as well as how they varied within the same scenario.

Publish/subscribe and RPC provide fast delivery of messages (best times were $<100ms$), but provide little message reliability (as low as 25% delivery ratio for RPC). Tuple spaces, on the other hand, pay a speed penalty (median round-trip times are 2-6 times slower than publish/subscribe), but provide better reliability, since messages will persist until explicitly removed from the tuple space. When used to implement a whiteboard application, tuple spaces achieved a 100% delivery ratio in all but one scenario.

The wireless channel itself can cause unexpected delays in message delivery. In the single hop scenario, publish/subscribe had a maximum delay of 1.6s and tuple spaces had one message require 14s to deliver. Introducing multi-hop routes without mobility caused median delay times to double for publish/subscribe and increase by a factor of 10 for tuple spaces. Mobility and multi-hop wireless routes, both defining characteristics of MANETs, strongly influenced the application-level performance and reliability of these paradigms.

Our results provide essential quantitative data for deciding which communication model should be used for new projects. While the paradigms presented here are essentially interchangeable in terms of functionality, their performance varies widely according to traffic and wireless conditions. Since we have tested canonical implementations of each paradigm, these results are applicable to basic versions of the paradigms in general and can be used to inform future work in application development for MANETs.

² <http://www.isi.edu/nsnam/ns/>

References

1. Patrick Th. Eugster et al. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.
2. Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1):39–59, 1984.
3. David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.
4. M. Takai, J. Martin, and R. Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *MobiHoc '01: Proc. of the 2nd ACM Intl. Symp. on Mobile Ad Hoc Networking & Computing*, 2001.
5. M. Varshney and R. Bagrodia. Detailed models for sensor network simulations and their impact on network performance. In *MSWiM '04: Proc. of 7th ACM Intl. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2004.
6. Scalable Networks. Exata: An exact digital network replica for testing, training and operations of network-centric systems. Technical brief, 2008.
7. Amy L. Murphy et al. Lime: A coordination middleware supporting mobility of hosts and agents. *ACM Trans. on Software Engin. and Methodology*, July 2006.
8. Yao-Nan Lien et al. A manet based emergency communication and information system for catastrophic natural disasters. In *ICDCSW '09: Proc. of the 29th IEEE Intl. Conf. on Distributed Computing Systems Workshops*, pages 412–417, 2009.
9. Nadjib Badache. A distributed mutual exclusion algorithm over multi-routing protocol for mobile ad hoc networks. *IJPEDS*, 23(3):197–218, 2008.
10. Simone Leggio et al. Session initiation protocol deployment in ad-hoc networks: a decentralized approach. In *2nd Intl. Workshop on Wireless Ad-hoc Networks (IWWAN)*, 2005.
11. Mee Young Sung and Jong Hyuk Lee. Desirable mobile networking method for formulating an efficient mobile conferencing application. In *Embedded and Ubiquitous Computing*, 2004.
12. David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Springer US, 1996.
13. Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *WMCSA '99: Proc. of the 2nd IEEE Workshop on Mobile Computer Systems and Applications*, page 90, 1999.
14. Alan Kaminsky and Hans-Peter Bischof. Many-to-many invocation: a new object oriented paradigm for ad hoc collaborative systems. In *OOPSLA '02: 17th Conf. on Object-Oriented Programming, Systems, Langs, and Apps.*, 2002.
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, 2002.
16. S Hadim et al. Trends in middleware for mobile ad hoc networks. *Journal of Communication*, 1(4), 11-21 July 2006.
17. Guilhem Paroux et al. A survey of middleware for mobile ad hoc networks. Technical report, Ecole Nationale Supérieure des Télécommunications, January 2007.
18. Justin Collins and Rajive Bagrodia. Programming in mobile ad hoc networks. In *WICON '08: Proc. of the 4th Annual Intl. Conf. on Wireless Internet*, 2008.
19. Matteo Ceriotti et al. Data sharing vs. message passing: synergy or incompatibility?: an implementation-driven case study. In *SAC '08: Proc. of the ACM Symp. on Applied Computing*, pages 100–107, 2008.