

# New Architectures, Protocols, and Middleware for Ad Hoc Collaborative Computing

Alan Kaminsky  
*Department of Computer Science*  
*Rochester Institute of Technology*  
*Rochester, NY, USA*  
ark@cs.rit.edu

Hans-Peter Bischof  
*Department of Computer Science*  
*Rochester Institute of Technology*  
*Rochester, NY, USA*  
hpb@cs.rit.edu

## Abstract

The existing Internet architecture and protocols are not suitable for the newly emerging ad hoc collaborative computing systems. New architectures, protocols, and middleware specifically designed for the new ad hoc collaborative computing systems are needed. While work is currently being done to adapt the Internet architecture and protocols so the new systems can be developed and deployed, the fundamental characteristics that make the Internet unsuitable for the new systems — address-based messaging, one-to-one communication, and central servers — still remain. We envision alternative architectures, protocols, and middleware with the opposite characteristics — no central servers, broadcast communication, and content-based messaging. Many-to-Many Invocation (M2MI) and the Many-to-Many Protocol (M2MP) are an initial realization of this vision.

*“No one pours new wine into old wineskins. If he does, the new wine will burst the skins, the wine will run out and the wineskins will be ruined. No, new wine must be poured into new wineskins. And no one after drinking old wine wants the new, for he says, ‘The old is better.’ ”*

— Jesus of Nazareth (Luke 5:37–39)

## 1 Introduction

The network architecture and protocols that have evolved into today’s Internet have enjoyed fantastic success for over three decades. The Internet architecture and protocols are admirably suited for developing network applications where individual computers at known addresses communicate with other individual computers at known addresses. However, the Internet’s very success tends to press all network application development into the same mold, whether or not the application fits that mold. The newly emerging area of ad hoc collaborative computing is especially ill-suited to the Internet mold.

Jesus’ words about the folly of pouring new wine into old wineskins might well be addressed to today’s network researchers and developers. It makes no sense to pour new wine — the new ad hoc collaborative computing systems — into old wineskins — the existing Internet architecture and protocols. New wine needs new wineskins — new architectures, protocols, and middleware specifically designed for the new ad hoc collaborative systems.

We do not advocate abandoning the Internet. In the

applications for which they are well-suited, the Internet architecture and protocols can and should continue to be used. But in new kinds of computing systems for which the Internet architecture and protocols are not well-suited, it is better to direct our energies toward developing new architectures, protocols, and middleware than toward patching up the Internet.

This paper is organized as follows. Section 2 describes the ad hoc collaborative computing systems, the “new wine,” that we envision. Section 3 describes work currently being done to adapt, or in our view contort, the Internet architecture and protocols so the new systems can be developed and deployed, and discusses the negative consequences of this line of work. Section 4 describes our vision for an alternative architecture and protocols specifically designed for the new ad hoc collaborative computing systems. Section 5 describes our work with a particular realization of that vision, Many-to-Many Invocation (M2MI) and the Many-to-Many Protocol (M2MP). Section 6 offers concluding remarks.

## 2 Ad Hoc Collaborative Computing

In an *ad hoc collaborative computing system*, multiple users with computing devices, as well as multiple stand-alone devices like printers, cameras, and sensors, all participate simultaneously (*collaborative*); and the various devices come and go and so are not configured to know about each other ahead of time (*ad hoc*). Examples of ad

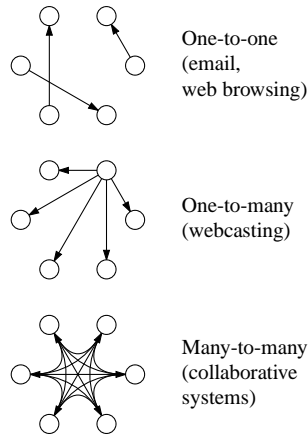


Figure 1: Communication patterns

hoc collaborative computing systems include:

- Multiuser applications: a chat session, a shared whiteboard, a group appointment scheduler, a file sharing application, or a multiplayer game.
- Applications that discover and use nearby networked services: a document printing application that finds printers wherever the user happens to be, or a surveillance application that displays images from nearby video cameras.
- Collaborative middleware systems like shared tuple spaces [1, 2].
- Sensor networks of all kinds, such as monitoring patients in a home health care setting, monitoring battlefield conditions, and environmental monitoring.

In many such collaborative systems, every device needs to talk to every other device. Every person's chat messages are displayed on every person's device; every person's calendar on every person's device is queried and updated with the next meeting time; every player's move is visible to every other player in the multiplayer game. In contrast to applications like email or web browsing (one-to-one communication) or webcasting (one-to-many communication), the collaborative systems envisioned here exhibit *many-to-many communication patterns* (Figure 1).

Many such collaborative systems are also intended to be used in a wireless proximal ad hoc networking environment. The devices in the system connect to each other using *wireless* networking technology such as IEEE 802.11 (wireless Ethernet). The devices are usually located in *proximity* to each other, around the same table or in the same room. Consequently, every device can hear every other device, and each transmitted message is immediately received by all the devices without needing to route the message through intermediate devices. In larger

settings like lecture halls and convention centers, distant devices communicate through wireless access points connected with a wired network rather than directly from device to device. Devices come and go as the system is running, and the devices do not know each others' identities beforehand; instead, the devices form *ad hoc* networks among themselves.

The computing devices on which collaborative systems run are often *small, battery powered* devices with limited memory sizes and CPU capacity. Unlike desktop computers, such devices cannot maintain constant network connections and cannot send large volumes of network traffic, because that would rapidly drain their batteries. To this end, collaborative systems can take advantage of the *broadcast* communication made possible by a wireless proximal network. A device can broadcast each message just once instead of sending a separate copy of each message individually to every other device, thus reducing both battery drain and network utilization.

### 3 The Current State

We discuss the current state of the art in adapting the Internet for ad hoc collaborative systems in four areas: transport protocols in wireless networks, ad hoc routing, ad hoc multicasting and broadcasting, and client-server architectures.

#### 3.1 Transport Protocols in Wireless Networks

TCP, the Internet's transport protocol, provides reliable data communication from end host to end host, acknowledging received packets and retransmitting unacknowledged packets. TCP also provides congestion control for the Internet. When a packet is unacknowledged, TCP assumes a router discarded the packet due to lack of buffer space and initiates congestion control procedures, greatly reducing the rate at which the host sends packets. While this strategy works well in the environment of immobile hosts and wired networks for which TCP was designed, this strategy works poorly in the ad hoc collaborative systems environment of mobile hosts and wireless networks. There, packets are mainly lost due to poor wireless transmission or due to changing connectivity as the hosts turn on, turn off, and move about, not due to congestion. In that case, initiating congestion control procedures leads to unnecessarily diminished throughput.

Consequently, much work has been done on modifying TCP to perform better in a mobile wireless environment. Some schemes address the mobility issue and override congestion control by detecting whether packet losses are due to routing changes, either by explicit feedback from lower layers [3, 4, 5] or by inferring routing changes from out-of-order packet arrivals [6]. Other schemes address

the wireless issue and modify the transport layer algorithms when a wireless link is known to be in use [7, 8].

With this line of work, researchers are attempting to extend the Internet's (in particular, TCP's) concept of *reliable* data communications to the mobile wireless arena. However, trying to achieve reliable communications in an ad hoc collaborative system is ultimately futile, because devices enter and leave the system constantly. Any ad hoc collaborative application must build in, *at the application level*, the ability to "pump up" newly arrived devices with information they missed receiving while out of contact, and must build in the ability to accommodate the departure of a device as a normal occurrence rather than an exceptional situation. (See [9] for examples of such applications.) But if those abilities are built in at the application level, they will also compensate for loss of messages in the network. Therefore, it doesn't make sense to do a lot of processing at the network level to make network communication totally reliable. End-to-end reliability has to be built in at the application level [10].

### 3.2 Ad Hoc Routing

A considerable amount of work has been done on ad hoc network routing. This work has concentrated on how to make networking based on host addresses (such as IP addresses) work when hosts move around and do not stay attached to a fixed network segment. Mobile IP [11], for example, is a scheme where a host can move to a different location, obtain a temporary IP address there, and cause traffic sent to the host's permanent address to be forwarded to its temporary address. Many ad hoc routing algorithms have been developed to route messages from source to destination through a network of point-to-point connections where the hosts (including the routers) are mobile and thus the connections between hosts are constantly changing [12, 13, 14, 15, 16]. These routing algorithms tend to be complicated and to utilize substantial memory space, CPU time, and network bandwidth just to maintain the routing information, in addition to what the actual applications utilize.

These ad hoc routing algorithms extend the Internet's concept of point-to-point packet routing to the ad hoc networking arena. While sometimes this is necessary (as in an environmental sensor network deployed over a large geographic area), much of the time ad hoc packet routing is not necessary for a collaborative system. The devices can simply broadcast messages to each other (directly or via wireless access points), eliminating the ad hoc routing protocol's extra memory space, CPU time, network bandwidth, and battery power utilization.

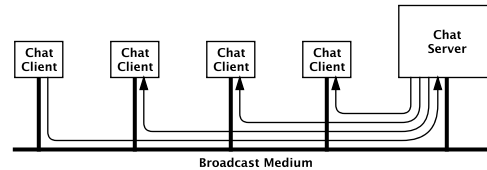


Figure 2: Client-Server Architectural Mismatch

### 3.3 Ad Hoc Multicasting and Broadcasting

Work has also been done on multicasting and broadcasting messages in an ad hoc network. Again, this work has focused on routing algorithms for delivering messages to certain specified hosts (multicast) or all hosts (broadcast) through a network of point-to-point connections, where the hosts are mobile and the connectivity changes constantly [17, 18, 19, 20]. Work has also focused on *reliable* multicast and broadcast algorithms which ensure either that all intended destinations receive each message (in the same order, for some algorithms), or that none do [21, 22, 23]. All these algorithms require memory space, CPU time, and network bandwidth to maintain group membership and to enforce reliable message delivery and ordering guarantees.

We have already remarked on the futility of trying to make point-to-point communication reliable in ad hoc collaborative systems. It is equally futile to try to make multicast or broadcast communication reliable in ad hoc collaborative systems. Basic broadcasting, in which most of the time a packet transmitted by one device is received by all the other devices but occasionally one or more devices fail to receive a packet, is sufficient. The application will then compensate for lost packets the same way it compensates for arriving and departing devices.

### 3.4 Client-Server Architecture

The Internet architecture and protocols are superbly suited for client-server systems: a server sits on a host waiting for connections, clients set up connections to the server, and all communication flows over these connections. Since the Internet is so pervasive, ad hoc collaborative systems often use a client-server architecture. However, that architecture is a poor match for an ad hoc collaborative system, leading to several negative consequences.

Consider an example of an ad hoc collaborative system, a chat application, running over a broadcast communication medium like a wireless Ethernet or a wired LAN segment (Figure 2). Each chat client device first sets up a connection to the chat server. This means the device either must be preconfigured with the server's IP address or host name (in which case a DNS server is also needed), which is problematic in an ad hoc network; or the device must carry out some kind of discovery protocol to find the

chat server, which uses time, network bandwidth, and battery power for a task that is peripheral to the application’s main purpose, chatting. When one device has a chat message, it sends the message to the chat server, which then sends a copy to all the devices. This leads to extremely inefficient utilization of the broadcast medium — at the network level, the chat message is broadcast multiple times when once would have sufficed, and the more devices that join the system, the greater the inefficiency. Worse yet, if the central server crashes or loses its network connection, the entire system grinds to a halt, even though the devices are perfectly capable of communicating directly with each other.

## 4 The Vision

Essentially, three things make the Internet architecture poorly suited for ad hoc collaborative systems:

**Address-based messaging.** You can’t even begin to communicate on the Internet until you have your own IP address and you know the other party’s IP address. Even the Internet’s concept of group communication, IP multicasting, boils down to communicating with an IP (multicast) address. But to a device participating in an ad hoc collaborative system, what’s important is obtaining the services of the system, not finding its own address or the addresses of the other devices. Getting all the network addresses set up uses extra memory space, CPU time, network bandwidth, and battery power on a task that is peripheral to the system’s purpose. It would be better if the devices could simply use each other’s *services* without needing to know each other’s *addresses*.

**One-to-one communication.** The Internet revolves around the TCP connection, which allows exactly two parties to communicate. But ad hoc collaborative systems typically exhibit *many-to-many* communication patterns, not one-to-one. Emulating many-to-many communication over one-to-one connections uses extra memory space, CPU time, network bandwidth, and battery power on a task that is peripheral to the system’s purpose.

**Central servers.** Because in the Internet the only thing you can communicate with is an IP address, each network application needs a server to listen to an IP address. Since only one host at a time can listen to an IP address, the server has to be a *central* server to which all other devices in the application connect. As discussed previously, using a central server in an ad hoc collaborative system leads to inefficient usage of the broadcast network and unnecessary loss of the system’s functionality upon an outage.

Unless these characteristics are changed, the Internet will continue to be unsuitable for ad hoc collaborative systems. Yet little of the current work is aimed at altering these characteristics.

To better suit the needs of ad hoc collaborative systems,

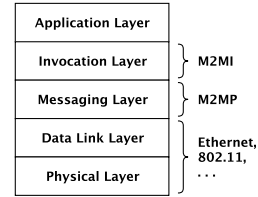


Figure 3: Network architecture for ad hoc collaborative systems

we envision new architectures, protocols, and middleware whose characteristics are the opposite of the Internet’s:

**No central servers.** The system is designed assuming that devices arrive and depart regularly, so no one device can act as a central server. Instead, all the devices — whichever ones happen to be present in the changing set of proximal devices — act in concert to run the system. Powerful middleware support makes it easy to write applications of this kind.

**Broadcast communication.** The network protocols revolve around broadcast messaging rather than one-to-one communication, taking advantage of proximal networks’ inherent broadcast ability. The serverless applications compensate for network message loss as they do for device arrivals and departures, leading to simpler broadcast protocols at the network level.

**Content-based messaging.** Rather than sending a request for service to a specific address, a device simply broadcasts each request for service. Devices receive messages based on the messages’ contents — that is, based on whether the message contains a request the device is able to serve. Only those devices capable of responding to a message do in fact receive and respond to the message. Network protocol software and, eventually, network hardware supports filtering of messages based on content.

Figure 3 shows the new layered network architecture we envision. The bottommost Physical and Data Link Layers are the same as in the Internet, providing a broadcast medium; examples include wired Ethernet and wireless Ethernet. Next comes a Messaging Layer that broadcasts each outgoing message to all devices in the proximal network and allows each device to filter incoming messages based on their contents. An Invocation Layer sits on top of that, providing a distributed object middleware system like CORBA or Java RMI but with the ability to broadcast each remote method invocation to multiple objects. Finally, at the Application Layer, ad hoc collaborative systems are built using the Invocation Layer’s broadcast method invocation ability.

## 5 The Realization

We are exploring one possible realization of the envisioned new Messaging Layer and Invocation Layer by developing a new broadcast messaging protocol, the Many-

to-Many Protocol (M2MP), and a new distributed object middleware, Many-to-Many Invocation (M2MI). A fuller description of M2MP and M2MI is found in [9].

## 5.1 M2MP

Intended particularly for the wireless proximal ad hoc networking environment, M2MP's design is based on these assumptions:

**There are no device addresses.** Consequently, devices can enter and leave the network in an ad hoc fashion without having to do network configuration.

**Messages are broadcast to all devices.** Since wireless radio transmissions are inherently broadcast within a certain proximal area, at the radio level it's just as easy to deliver a message to all devices as to one device.

**A message's relevancy is determined by its contents.** A device decides which incoming messages to process by examining the initial bytes of each message.

**Message delivery is mostly reliable.** Most of the time, a message broadcast by one device is received by all the other devices. However, on rare occasions a message broadcast by one device is not received by some or all of the other devices.

When an application on one device sends an M2MP message, the application writes a stream of bytes with the message's contents to the M2MP layer. The M2MP layer breaks the byte stream into a sequence of packets and broadcasts each packet. At the receiving end, the M2MP layer provides a byte stream for each incoming message which the application reads. The M2MP layer feeds each incoming packet's contents into the corresponding message's byte stream. If any packet of a message fails to arrive within a certain timeout after the previous packet, the M2MP layer just aborts the whole message, which greatly simplifies the M2MP protocol software. However, messages are seldom aborted because the underlying network is assumed to be mostly reliable.

To receive incoming messages, an application must register a *message filter*, a fixed byte string, with the M2MP layer. If an incoming message's initial bytes match a registered message filter, the M2MP layer passes the message up to the application, otherwise the M2MP layer discards the message. An application that uses M2MP, such as M2MI, designs its M2MP messages to weed out irrelevant messages using M2MP's message filters.

## 5.2 M2MI

Just as RMI provides an object oriented abstraction of point-to-point messaging, M2MI provides an object oriented abstraction of broadcast messaging. M2MI lets an application invoke a method declared in an interface and have the method invocation broadcast to multiple objects,

all of which execute the method. To do such an invocation, the application needs some kind of "reference" upon which to call the method. In M2MI, a reference is called a *handle*, and there are three varieties, omnihandles, multihandles, and unihandles.

### 5.2.1 Omnihandles

An *omnihandle* for an interface stands for "every object out there that implements this interface." An application can ask the M2MI layer to create an omnihandle for a certain target interface *X*. Thereafter, calling method `foo` on the omnihandle for interface *X* means, "Every object out there that implements interface *X*, perform method `foo`." The method is actually performed by whichever objects implementing interface *X* exist at the time the method is *invoked* on the omnihandle. Thus, different objects could respond to an omnihandle invocation at different times.

To receive invocations on a target interface, an application creates an object that implements the interface and *exports* the object to the M2MI layer. Thereafter, whenever anyone calls a method on an omnihandle for the interface, the M2MI layer will call that method on the exported object.

The target objects invoked by an M2MI method call need not reside in the same process or device as the calling object. As long as the target objects are in range to receive a broadcast from the calling object over the network, the M2MI layer will find and invoke the target objects.

As an example of an M2MI-based collaborative system, consider a chat application. Whenever one user types a message on her device, the message appears on all the devices in the proximal network. The target interface is:

```
interface Chat {
    void putMessage (String msg);
}
```

Each device creates an object that is an instance of a class implementing interface `Chat`, say class `ChatImpl`, and exports that object to the M2MI layer:

```
Chat chat = new ChatImpl();
M2MI.export (chat, Chat.class);
```

Each device also creates an omnihandle for the target interface:

```
Chat allChats =
    (Chat) M2MI.getOmnihandle
        (Chat.class);
```

When the user types a chat message, like `Hello`, the device executes:

```
allChats.putMessage ("Hello");
```

The M2MI layer broadcasts this omnihandle invocation to all the chat objects on all the devices. Executing its `putMessage` method, each chat object (including the one on the user's own device) displays the message passed in as the argument. Figure 4 shows a sequence of M2MI

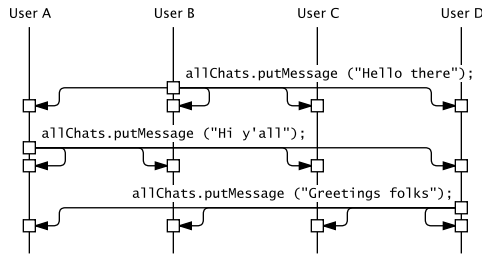


Figure 4: M2MI calls for a chat application

calls that might occur when four instances of this chat application run in four nearby devices.

Note that the M2MI-based chat application does not need to find and connect to a central chat server. Neither does the application need to know which other devices are part of the chat session or connect to them. The user’s device simply shows up and starts broadcasting `putMessage` invocations. This shows how M2MI simplifies the development and deployment of ad hoc collaborative systems.

### 5.2.2 Multihandles

A *multihandle* for an interface stands for “one particular set of objects out there that implement this interface.” A multihandle refers only to those objects that have been explicitly *attached* to the multihandle. Calling method `foo` on a multihandle means, “The particular object or objects attached to this multihandle, perform method `foo`.” As with an omnihandle, the target objects for a multihandle invocation need not reside in the same process or device as the calling object or each other. One process can create a multihandle, attach objects to it, and send it to another process or processes; then the destination processes can attach their own objects to the multihandle.

Let us add a feature to the chat application: multiple independent simultaneous chat sessions. The user can discover which chat sessions are out there and participate in one of them, or the user can start a new chat session. The user then sees only the messages sent to that chat session, not all the other chat sessions.

To see only the messages for a particular chat session, each user’s device has a chat object implementing interface `Chat` as before. Now, however, there is a *multihandle* for interface `Chat` for each separate chat session. To participate in a particular chat session, the device attaches its chat object to the corresponding multihandle. When the user types a line of text, the device invokes `putMessage` on the chat session’s multihandle. Since the invocation is performed on a multihandle instead of an omnihandle, only those chat objects that have been explicitly attached to the multihandle — that is, only those

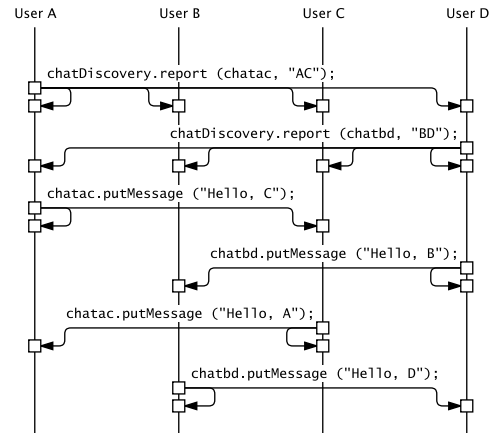


Figure 5: M2MI calls for multiple chat sessions

devices participating in the chat session — will execute the method.

To discover which chat sessions are out there, a new interface is used:

```
interface ChatDiscovery {
    void report (Chat session);
}
```

The device exports a chat discovery object implementing interface `ChatDiscovery`. Each device with an active chat session periodically invokes `report` on an omnihandle for interface `ChatDiscovery`, passing in the multihandle for the chat session. Processing each `report` invocation, the chat discovery object collects the chat sessions in a list and displays them for the user to choose.

If the user decides to participate in an existing chat session, the device obtains the corresponding chat session multihandle from the list and attaches the chat object to that multihandle. If the user decides to start a new chat session, the device creates a new chat session multihandle and attaches the chat object to the new multihandle.

Figure 5 shows a sequence of M2MI calls that might occur when four instances of this chat application run in four nearby devices. The chat session multihandles are named `chatac` and `chatbd`. The omnihandle for interface `ChatDiscovery` is named `chatDiscovery`.

### 5.2.3 Unihandles

A *unihandle* for an interface stands for “one particular object out there that implements this interface.” An application can export an object and obtain a unihandle attached to that object. Calling method `foo` on the unihandle means, “The particular object out there attached to this unihandle, perform method `foo`.” As with an omnihandle or multihandle, the target object for a unihandle

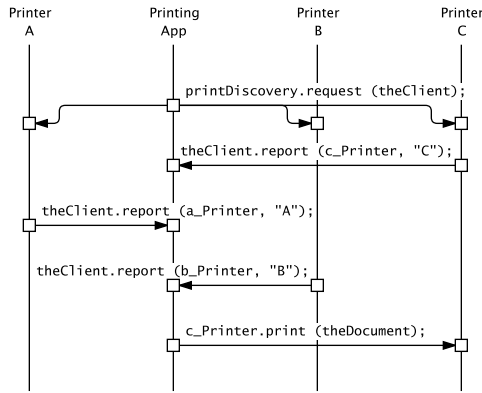


Figure 6: M2MI invocations for a print service

invocation need not reside in the same process or device as the calling object.

As an example of an M2MI-based system involving service discovery, consider printing. To print a document from a mobile device, the user must discover the nearby printers and print the document on one selected printer. Printer discovery is a two-step process: the user broadcasts a printer discovery request, then each printer sends its own unihandle back to the user. To print the document, the user does an invocation on the selected printer's unihandle.

Specifically, each printer has a print service object that implements this interface:

```
interface PrintService {
    void request (PrintClient client);
    void print (Document doc);
}
```

The printer exports its print service object to the M2MI layer and gets a unihandle attached to the object. The printer is now prepared to respond to printer discovery requests and to print documents.

Each user has a print client object that implements this interface:

```
interface PrintClient {
    void report (PrintService printer);
}
```

The client printing application exports a print client object implementing interface `PrintClient` to the M2MI layer and gets a unihandle attached to the object. The application also gets an omnihandle for interface `PrintService`. The application is now prepared to make print discovery requests and process print discovery reports.

Figure 6 shows the sequence of M2MI invocations that occur when the document printing application goes to print a document with three printers nearby. The application first calls

```
printService.request (theClient);
```

on the `PrintService` omnihandle, passing in the unihandle to its own print client object. Since it is invoked on an omnihandle, this call goes to all the printers. The application now waits for print discovery reports.

Each printer's request method calls

```
theClient.report (thePrinter);
```

passing in the unihandle to the printer's print service object. After executing all the `report` invocations, the printing application knows which printers are available and has a unihandle for submitting jobs to each printer.

Finally, after asking the user to select one of the printers, the application calls

```
c_Printer.print (theDocument);
```

where `c_Printer` is the selected printer's unihandle as previously passed to the `report` method. Since it is invoked on a unihandle, this call goes just to the selected printer, not the other printers. The printer proceeds to print the document passed to the `print` method.

Clearly, this pattern of broadcast discovery request – discovery responses – service usage can apply to any service, not just printing. It is even possible to define a *generic* service discovery interface that can be used to find objects that implement *any* interface, the desired interface being specified as an argument of the discovery method.

For a fuller discussion of the M2MI paradigm and additional examples of M2MI-based applications, see [9].

### 5.3 Status and Plans

We have developed initial prototypes of M2MP and M2MI in Java. The source code and documentation are available [24]. The prototypes have been tested on desktop hosts running Solaris and Linux with a wired Ethernet network. We plan to migrate M2MP and M2MI to portable computing devices with wireless Ethernet connectivity, such as laptops, tablet PCs, and PDAs.

The prototype M2MP implementation uses UDP datagrams sent to an IP multicast address to achieve network broadcasting. Work is underway to develop an alternative M2MP implementation that hooks up to the Ethernet data link layer directly, without incurring the undesirable overhead of going through the Internet protocol stack. Ultimately we plan to migrate M2MP into the Linux kernel.

Initial prototypes of several ad hoc collaborative applications, including chat, IM, whiteboard, calendar, file sharing, and tuple space, have been constructed using M2MI. Work is in progress on a collaborative information retrieval system and a shared bulletin board system similar to a tuple space. From our initial investigations we are getting an inkling of a general paradigm for building collaborative systems using M2MI. We plan to continue gaining experience building M2MI-based systems, identifying design patterns for M2MI-based systems, and codifying the M2MI paradigm.

Work is underway to add security to M2MI-based systems. Achieving security in an ad hoc environment is challenging because most existing security approaches rely on central servers. We are investigating serverless group authentication and group key exchange protocols, and we are modifying the M2MI layer to add encryption and authentication of M2MI calls.

## 6 Conclusion

According to Jesus, “No one after drinking old wine wants the new, for he says, ‘The old is better.’ ” Many networking researchers and developers prefer to stick with the old wine of the Internet architecture and protocols, reasoning that what worked well in the past will continue to work well in the future. We, however, feel it is time to embrace the new wine and start exploring alternatives that are better suited for ad hoc collaborative applications — new architectures, protocols, and middleware with content-based messaging, broadcast communication, and no central servers. M2MI and M2MP are a first step in that direction.

## 7 Acknowledgments

This research was supported by grants from Sun Microsystems and Xerox Corporation.

## References

- [1] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
- [2] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A middleware for physical and logical mobility. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'01)*, pages 524–533, April 2001.
- [3] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback based scheme for improving TCP performance in ad-hoc wireless networks. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS '98)*, pages 472–479, May 1998.
- [4] D. Kim, C.-K. Toh, and Y. Choi. TCP-BuS: Improving TCP performance in wireless ad hoc networks. *Journal of Communications and Networks*, 3(2), June 2001. <http://jcn.or.kr/>.
- [5] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 19(7):1300–1315, July 2001.
- [6] F. Wang and Y. Zhang. Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC 2002)*, June 2002.
- [7] A. Bakne and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS '95)*, pages 136–143, 1995.
- [8] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, December 1997.
- [9] A. Kaminsky and H.-P. Bischof. Many-to-Many Invocation: A new object oriented paradigm for ad hoc collaborative systems. In *17th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 2002), Onward! Track*, November 2002. <http://www.cs.rit.edu/~anhinga/publications/m2mi20020716.pdf>.
- [10] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 44–57, December 1993.
- [11] Internet Engineering Task Force. IP Routing for Wireless/Mobile Hosts (mobileip) Working Group. <http://www.ietf.org/html.charters/mobileip-charter.html>.
- [12] C. E. Perkins and P. Bhagwat. DSDV routing over a multihop wireless network of mobile computers. In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, pages 183–206. Kluwer Academic Publishers, 1996.
- [13] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: the Dynamic Source Routing protocol for multihop wireless ad hoc networks. In Charles E. Perkins, editor, *Ad Hoc Networking*, pages 139–172. Addison-Wesley, 2001.
- [14] C. E. Perkins and E. M. Royer. The ad hoc on-demand distance-vector protocol. In Charles E. Perkins, editor, *Ad Hoc Networking*, pages 173–219. Addison-Wesley, 2001.
- [15] Z. J. Haas and M. R. Pearlman. ZRP: a hybrid framework for routing in ad hoc networks. In Charles E. Perkins, editor, *Ad Hoc Networking*, pages 221–253. Addison-Wesley, 2001.
- [16] J. J. Garcia-Luna-Aceves and M. Spohn. Transmission-efficient routing in wireless networks using link-state information. *Mobile Networks and Applications*, 6(3):223–238, June 2001.
- [17] S. Basagni, D. Bruschi, and I. Chlamtac. A mobility-transparent deterministic broadcast mechanism for ad hoc networks. *IEEE/ACM Transactions on Networking*, 7(6):799–807, December 1999.
- [18] S.-Y. Ni, Y.C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 151–162, August 1999.
- [19] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. Algorithms for energy-efficient multicasting in static ad hoc wireless networks. *Mobile Networks and Applications*, 6(3):251–263, June 2001.
- [20] W. Lee, S.-J. Su and M. Gerla. Wireless ad hoc multicast routing with mobility prediction. *Mobile Networks and Applications*, 6(4):351–360, August 2001.
- [21] E. Pagani and G. P. Rossi. Reliable broadcast in mobile multihop packet networks. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '97)*, pages 34–42, September 1997.
- [22] E. Pagani and G. P. Rossi. Providing reliable and fault tolerant broadcast delivery in mobile ad-hoc networks. *Mobile Networks and Applications*, 4(3):175–192, October 1999.
- [23] D. M. Chiu, M. Kadansky, J. Provino, J. Wesley, H.-P. Bischof, and H. Zhu. A congestion control algorithm for tree-based reliable multicast protocols. Technical Report TR-2001-97, Sun Microsystems, June 2001. [http://research.sun.com/nova/cgi-bin/smli\\_tr-2001-97.pdf](http://research.sun.com/nova/cgi-bin/smli_tr-2001-97.pdf).
- [24] A. Kaminsky. Many-to-Many Invocation Library. <http://www.cs.rit.edu/~anhinga/m2mi.shtml>.