Designing and Implementing a MANET *Network Service Interface* with Compact .NET on Pocket PC

Fabio De Rosa
Università di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica
Via Salaria 113 (2nd floor, lab C4)
I-00198 Roma, Italy
derosa@dis.uniroma1.it

Massimo Mecella
Università di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica
Via Salaria 113 (2nd floor, room 231)
I-00198 Roma, Italy
mecella@dis.uniroma1.it

ABSTRACT

Operators forming an ad hoc network (MANET) in emergency situations would benefit from software supporting their interaction. To date, however, development of such a coordination layer has required abstractions on the services and data provided by the lower network layers. In this paper we present the design and a possible implementation of the *Network Service Interface* [DeRosa03a] as a .NET Compact Framework component, coded in C#, to be run on PDAs with the Windows Mobile operating system. We chose Dynamic Source Routing (DSR) as the routing protocol supporting inter-device communication.

Keywords

Cooperative Work – Mobile Ad hoc Network – *Network Service Interface* – Object and Component Design – .NET Compact Framework – DSR.

1. INTRODUCTION

The widespread availability of network-enabled hand-held devices (e.g. PDAs with WiFi - the 802.11x-based standard) has made pervasive computing environment development an emerging reality. Mobile (or Multi-hop) Ad hoc NETworks (MANETs, [Agrawal03a]) are mobile device networks communicating with one another via wireless links without relying on an underlying infrastructure. This distinguishes them from other types of wireless networks, such as cell networks or infrastructure-based wireless networks. Each device in a MANET acts as an endpoint and as a router forwarding messages to devices within radio range. MANETs are a sound alternative to infrastructure-based networks whenever the infrastructure is lacking or unusable, such as in emergency situations.

.NET Technologies'2005 *conference proceedings,*
ISBN 80-86943-01-1

Operators acting in such emergency situations would benefit from software supporting their collaboration. Such a coordination layer would enable them to execute sets of activities (in sequence, concurrently, etc.) through specific applications (e.g. computer supported cooperative work - CSCW - tools [Grudin04a], workflow management applications [Leymann00a], etc.) running on hand-held devices, thus enabling cooperative processes to be run. All such applications typically require continuous inter-device connections (e.g. for data/information sharing, activity scheduling and coordination, etc.), but these are not generally guaranteed in MANETs.

We investigated a specific pervasive architecture, targeted at CSCW and workflow management applications constituting the coordination layer and able to maintain continuous connections among MANET devices.

As a typical example, consider the aftermath of an archeological disaster: following an earthquake, a team is equipped with mobile devices (laptops and PDAs) and sent to the affected area to evaluate the condition of archeological sites and buildings, with the goal of drawing a situation map to schedule rebuilding activities. A typical cooperative process to be enacted by the team would be that shown in Figure 1 (depicted as an UML Activity Diagram):

- the team leader has previously stored all area details (not included in the process), including a site map, list of the most important objects located in the site and previous reports/materials;
- the team is considered as an overall MANET, in which the team leader's device (requiring the most computing power, therefore usually a laptop) coordinates the other team members' devices, by providing suitable information (e.g. maps, sensitive objects, etc.) and assigning activities/tasks;

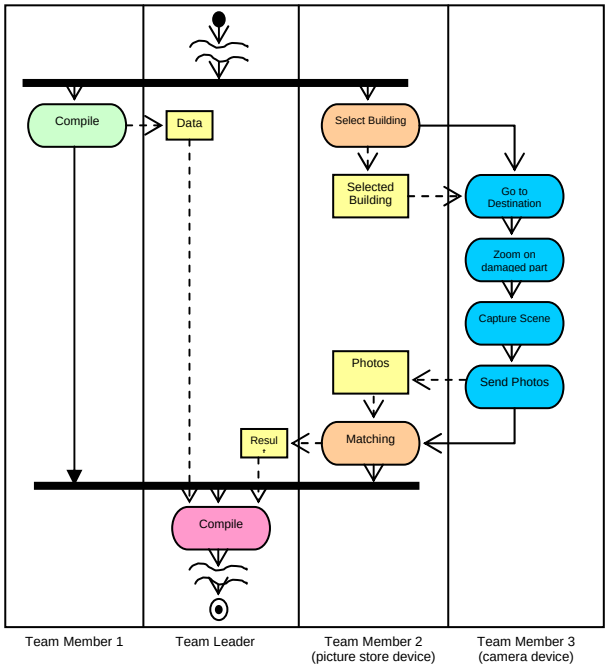

Figure 1. Cooperative process.

- team members are equipped with hand-held devices (PDAs), which allow them to run some operations but do not have much computing power. Such operations, possibly involving various hardware items (e.g. digital cameras, GPRS connections, computing power for image processing, main storage, etc.), are provided as software services to be coordinated. Team member 1 might compile some specific questionnaires (after a visual analysis of a building), to be analyzed by the team leader using specific software in order to schedule subsequent activities; team member 3 might take pictures of the damaged buildings, while team member 2 may be responsible for specific processing of previous and recent pictures (e.g. for initial identification of architectural anomalies).

In this case, it might be useful to match new pictures with previously stored images. The device holding the high-resolution camera must therefore be connected to the one containing the stored pictures.

But in a situation such as that shown in Figure 2, the movement of the operator/device equipped with the camera may result in its disconnection from the others.

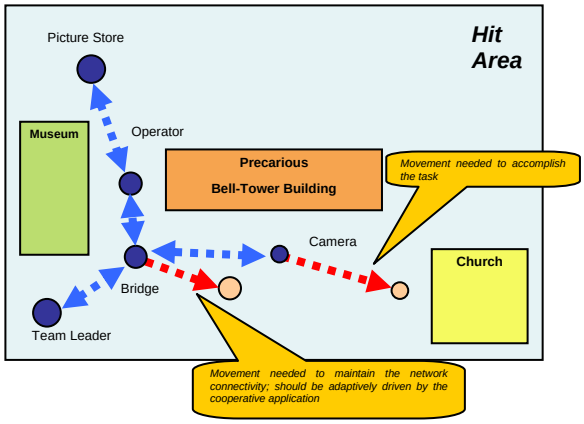

Figure 2. Critical situation and adaptive management.

A pervasive architecture should be able to predict such a situation, alert the coordination layer, and possibly have a "bridging" device (team member 4's device) to follow the operator/device moving out of range, maintaining the connection and ensuring a path between devices. In this way the coordination layer schedules the execution of new activities based on the prediction of a disconnection, as shown in Figure 3 (note the new activity for team member 4).

The process's adaptive change is centrally managed by the coordination layer, which has "global" knowledge of the status of all operators/devices and takes into account idle devices, operations that can be safely delayed, etc.

In recent years, research in the MANET area has focused on the development of appropriate routing protocols, methods for energy preservation, and other issues on the lower four ISO/OSI layers. Effective routing in ad hoc networks is still an actively-addressed open problem [Vaidya04a], with some interesting proposals presented in the literature (e.g. Dynamic Source Routing – DSR, Ad hoc On demand Distance Vector – AODV routing, Zone Routing Protocol - Z-RP, etc.).

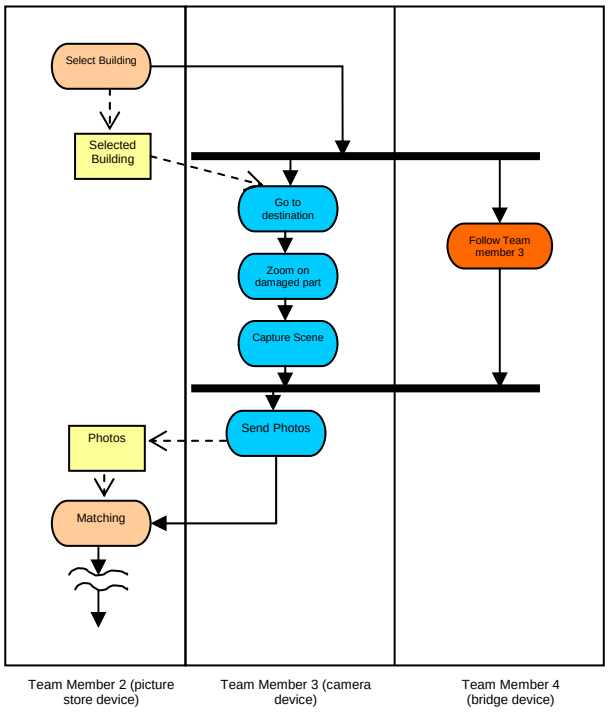

Figure 3. Modified process (details).

To date, development of application layer software (and thus of any information system for MANET), has required abstractions on the specific characteristics of the routing algorithms and, more generally, on the services and data provided by the lower network layers. [DeRosa03a] proposes a network service interface to be used as the basic layer on which to build application software, starting from the analysis and abstraction of current routing protocols.

In this paper we present the design and a possible implementation of the *Network Service Interface* [DeRosa03a] layer as a .NET Compact Framework component, coded in C#, to be run on PDAs with the Windows Mobile operating system. Dynamic Source Routing was chosen as the routing protocol supporting inter-device communication . To our knowledge, this is the first effective implementation of a MANET routing protocol for PDAs (which are mainly Windows-based); current research and the commercial tools available are targeted only at laptops running Linux.

The paper is organized as follows: in Section 2, the workflow architecture constituting the reference framework for cooperative work on MANET is described; this provides the overall framework for the results presented in this paper. In Section 3 we show the design of the *Network Service Interface* layer, while in Section 4 we report the results of *NSI* component testing experiments. In Section 5 an example of Windows Mobile application –*MANET-Chat* – is described, to show the use of the *NSI* component. Finally in Section 6 we report our conclusions and future work.

2. WORKFLOW ARCHITECTURE

Figure 4 shows the architecture supporting cooperative work on MANETs. The various MANET devices are equipped with some wireless network interfaces and specific hardware for calculating distances from neighboring devices (*Wireless Stack* in the figure), while the *Network Service Interface* (NSI) provides the upper layers with the basic services for sending and receiving messages (through multi-hop paths) to/from other devices, by abstracting the specific routing protocols.

Services (i.e. specific applications supporting the device users' tasks [1]) are accessible to other devices and can be coordinated and composed in a cooperative process. In contrast, the coordinator device presents the *Predictive Layer* on top of the *Network Service Interface*, signaling any probable disconnection to the upper *Coordination Layer*. The *Predictive Layer* implements a probabilistic technique [DeRosa05a] which can predict if all devices will still be connected in the successive moment. At a given time instant t_i in which all devices are connected, the coordinator device collects all device distance information and builds a next connection graph, i.e. the most likely graph at the next time instant t_{i+1}, in which the predicted connected and disconnected devices are highlighted. In the interval [t_i, t_{i+1}], the coordinator layer enacts the appropriate actions to enable all devices to be still connected at t_{i+1}. In predicting at t_i the next connection graph, the technique considers not only the current situation, but also recent situations and predictions (i.e., at t_{i-1}, t_{i-2}, etc.), specifically considering distances calculated in the recent past. Thus, although the pervasive architecture guarantees that constant connection of all devices, MANET's evolution is considered as it would be in a "free" scenario (i.e. without remedial actions by the coordination layer) when predicting the future situation. The

[1] Some of these services are applications that do not require human intervention (e.g. an image processing utility), whereas others act as proxies in front of human actors (e.g. the service for instructing human actors to follow a peer is a simple GUI that alerts the human operator by displaying a pop-up window and emitting a signal).

reasonable assumption is that if two devices have the tendency to go out of radio range if left "free", and are thus connected through the coordinator's remedial actions, then this influences the subsequent connection probability. The predictive layer therefore calculates a probable distance $S^{t+1}(i,j)_{p(i,j)}$ (see equation 1) at time t_{i+1} between each pair of MANET devices i, j, , taking into account previous real distances h (distance history) between devices, each with a different weight (α_k/c with $\alpha_k = k$ and $c = \sum_{k=1}^{h} \alpha_k$),as more importance is given to recent movements (h is the dimension of the predictive algorithm temporal window).

$$S_{p_{(i,j)}}^{(t+1)} = \frac{\sum_{k=1}^{h} \alpha_k S_{(i,j)}^{t-(h-k)}}{\sum_{k=1}^{h} \alpha_k} = \sum_{k=1}^{h} \left(\frac{\alpha_k}{c}\right) S_{(i,j)}^{t-(h-k)}$$

Equation 1. Predicted distance between two MANET devices *i, j*.

Starting from these predicted distances and by considering the maximum communication range (S_{dev}) of the wireless technology utilized (e.g. approximately 100 m if the device uses IEEE 802.11b), the predictive layer estimates the probability that a pair of MANET devices (i, j) is still within radio range at the next instant t_{i+1} (equation 2).

$$P_{(i,j)}^{(t+1)} = \begin{cases} \frac{|S_{dev} - S_{p_{(i,j)}}^{(t+1)}|}{S_{dev}} & S_{p_{(i,j)}}^{(t+1)} \leq S_{dev} \\ 0 & S_{p_{(i,j)}}^{(t+1)} > S_{dev} \end{cases}$$

Equation 2. The Probability that a couple of MANET devices *i, j* being still in the radio range at the next instant t_{i+1}.

These probabilities are used to build a square probability matrix $|E|$ x $|E|$ ($|E|$ = number of MANET mobile devices) M = (m_{ij}), in which $m_{ij} = P^{(t+1)}{}_{(i,j)}$ (equation 3). This matrix is used to build the subsequent connection graph: the set of graph nodes is $E = \{e_1, \ldots, e_m\}$ and the set of graph arcs is $A = \{(i, j) \mid m_{ij} = P_{(i,j)} \geq \beta\}$, where $0 \leq \beta \leq 1$ represents a probability threshold. The value of β depends on the type of situation, but is normally $\geq$ ½.

$$\begin{pmatrix} 1 & P_{(1,2)} & \cdots & P_{(1,m)} \\ P_{(2,1)} & 1 & \cdots & P_{(2,m)} \\ \vdots & \vdots & \ddots & \vdots \\ P_{(m,1)} & P_{(m,2)} & \cdots & 1 \end{pmatrix}$$

Equation 3. The square probability matrix.

The strategy of the algorithm used in the *Predictive Layer* component is therefore to find the connected components in the subsequent connection graph (using the SUB CCDFSG procedure), and verify if two devices e_i and e_j, belong to the same connected component (the TEST CONNECTION procedure); if so, then *they* will still communicate in the subsequent instant and if not, they will lose their connection. After building the matrix M = (m_{ij}), it is therefore possible to verify which devices are directly (one hop) or indirectly (multi hop) connected to all other devices, and thus let the coordinator decide whether or not to take actions to maintain connection between the involved devices. The predictive algorithm is reported below:

```
PROGRAM MGR(Comps[m])
1.  numcomps ← 0
2.  for i ← 0 to (m - 1)
3.      do if Comps[i] = 0
4.          then numcomps ← numcomps + 1
5.               CCDFSG(M, i, numcomps, Comps[])
6.  return Comps[]

SUB CCDFSG(M, i, numcomps, Comps[m])
1.  Comps[i] ← numcomps
2.  for each M[i, j] ≥ Beta
3.      do if Comps[j] = 0
4.          then CCDFSG(M, j, numcomps, Comps[])
5.  return NIL

1.  PROGRAM TEST CONNECTION(i, j, Comps[m])
2.  if Comps[i] = Comps[j]
3.     then TEST ← true
4.     else TEST ←false
5.  return TEST
```

The coordination layer manages situations when a peer is about to disconnect (e.g. by instructing a specific device to "Follow Peer X"). For example, if the coordination layer realizes a workflow

management system, then the coordination layer may restructure the workflow schema on the basis of the current prediction.

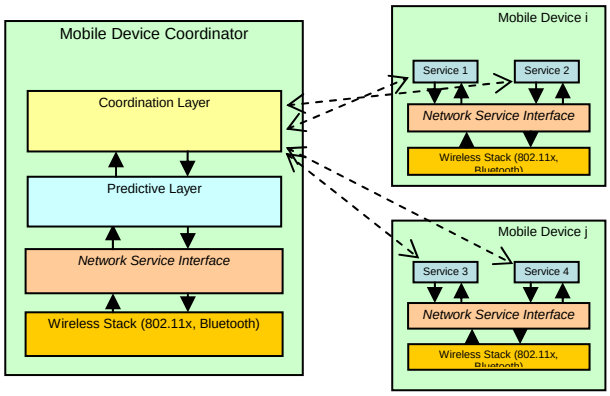

Figure 4. Proposed Architecture for supporting cooperative work on MANETs.

3. *NSI* COMPONENT DESIGN

Figure 5 reports the *Network Service Interface* API [DeRosa03a], which provides the following operations to the upper layers:

- *bind()*, which enables applications running on the same device to be bound to the MANET network layer;
- *send()*, which sends messages to a peer and reports the success or failure of data transmission;
- *receive()*, which receives messages from peers in the MANET;
- *isLinked()*, which reports whether a given peer is present in the MANET at that time;
- *close()*, which closes the MANET socket related to a specific application;
- *release()*, which releases all resources locked by a specific MANET socket.

Figure 5 also shows the realization and dependency relationships among the *NSI*, the *MANETServices* component, and a generic Client Application running on Pocket PC. Client Applications may be stand alone (e.g. chats, electronic agendas, etc.) or other components using the *NSI* to communicate with other network peers, and *MANETServices* implements the MANET Network layer, enabling communication among MANET mobile devices. The *MANETServices* component and its constituent packages are described below.

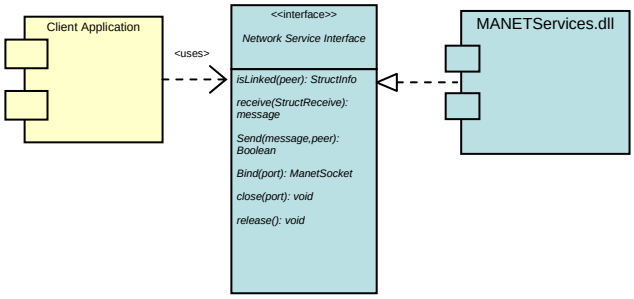

Figure 5. The *NSI* with realization and dependency relationships.

The *MANETServices* component consists of two main packages: the *MANETService* package and the *RoutingProtocol* package (Fig. 6).

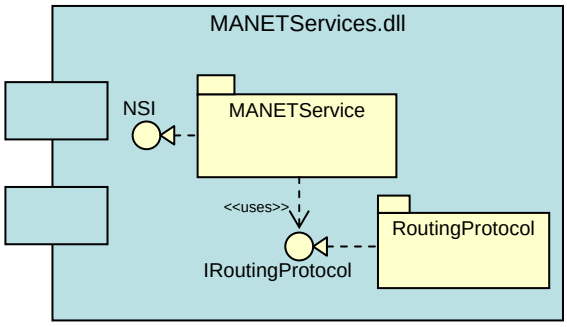

Figure 6. The *MANETServices* component and its constituting packages.

The *MANETService* package contains all interfaces and classes implementing the *NSI* API. The *RoutingProtocol* package includes all interfaces and classes implementing the specific MANET routing protocol: e.g. in our case, the classes and interfaces implementing the DSR routing protocol are collocated in the *RoutingProtocolDSR* package, a *RoutingProtocol* sub-package. It was decided to have two packages linked by the *IRoutingProtocol* interface (a common interface for all MANET routing algorithms) in order to keep the *MANETServices* component as modular as possible. In fact, by separating the routing algorithm logic from the MANET network management, the NSI is kept independent of the routing protocol utilized. For example, to use AODV routing protocol rather than DSR protocol, it is only necessary to implement the AODV algorithm (e.g. by producing a *RoutingProtocolAODV* sub-package), by implementing the *IRoutingProtocol* interface, and configure the MANET network layer context (by setting up specific component properties). This requires no change to the MANET management,

nor, in consequence, to the client application source code. This is a typical application of the "Strategy" pattern presented in [Gamma94a], in which *ConcreteStrategies* are the classes realizing the MANET routing protocols (see Figure 7).

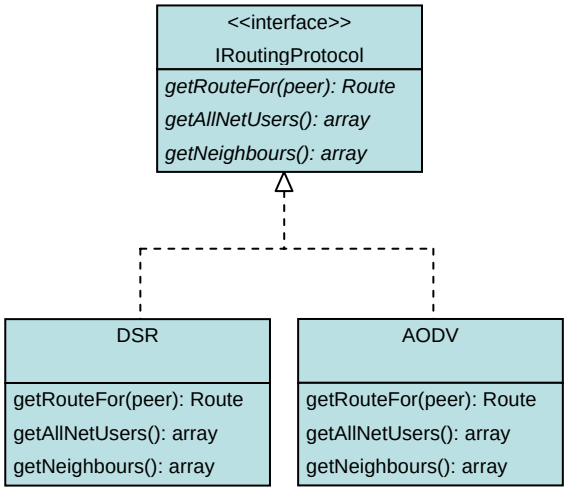

Figure 7. The *Strategy* pattern for MANET routing algorithms.

MANETService Package

The main classes constituting the *MANETService* package and realizing the *Network Service Interface* are *MANETManager* and *MANETSocket* (see Figure 8).

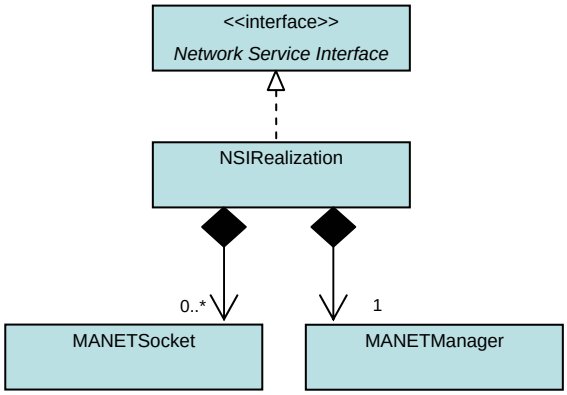

Figure 8. Main Classes constituting the *MANETService* package.

As with a file system manager, a window manager, or a printer spooler, the *MANETManager* class manages and controls concurrent access to the MANET network layer of client applications running on the same mobile device, specifically managing access to shared information of the routing protocol used, e.g. neighbor list, routing tables if any, etc. At run time there is therefore only one *MANETManager* class instance which has strict control over how and when client applications access the *NSI*. The unique manager object maintains a list of opened *MANETSocket* objects for each application, which obtain shared information through synchronized methods. For these reasons we adopted the *Singleton* pattern [Gamma94a] for the MANET communication layer as a design solution, where the singleton class is our *MANETManager* class (Figure 9).

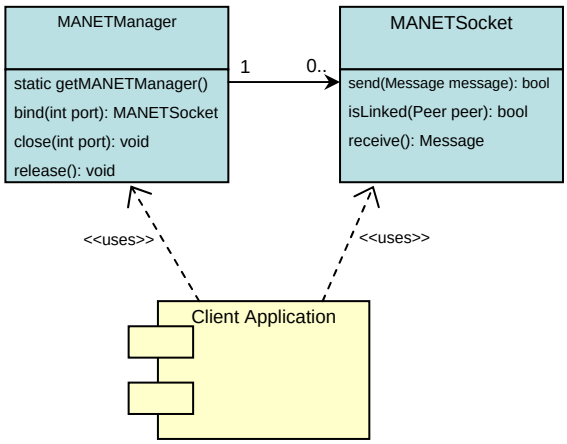

Figure 9. The *Singleton* pattern for MANET connection manager.

4. *NSI* IMPLEMENTATION AND TESTING

We implemented *NSI* as a .NET Compact Framework component, coded in C#, to be run on both PDAs, with the Windows Mobile operating system, and laptops (or any desktop) with the Windows operating system desktop version. The Dynamic Source Routing protocol [Johnson94a], specifically optimized for route caching [Vaidya04a], was implemented to support inter-device communication. To our knowledge, this is the first effective implementation of a MANET routing protocol for PDAs (which are mainly Windows-based), as current research and the available commercial tools are targeted only at supporting laptops running Linux.

For our experiments we deployed the *NSI* component on several kinds of PDA devices, with:

- IPAQ 5550 and IPAQ 5540 with 450 MHz processors and 128 MB RAM,

and on:

- desktops with 3 GHz processors and 1 GB RAM;
- laptops with 2.8 GHz processors and 512 MB RAM.

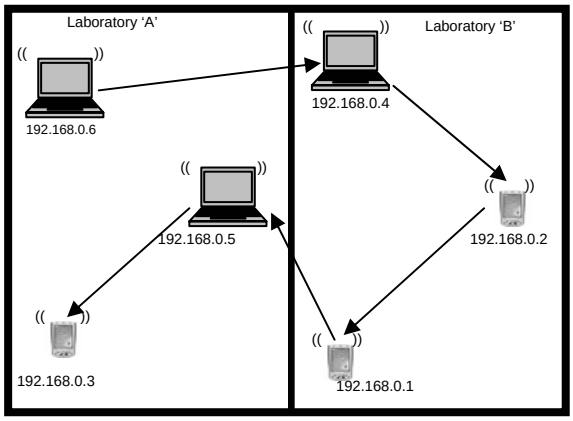

Figure 10. Experiment environments: laptops, desktops, and PDAs placed in adjacent rooms and constituting an unique MANET.

We deployed heterogeneous devices in order to better test the *NSI* component and verify its performance on devices with different hw/sw. It is easy to predict that when a laptop or a desktop forwards packets to PDAs, throughput is limited by their different clock speeds. One of our goals was to establish how this affects the routing protocol performance (in our case the DSR protocol performance).

The experiments were conducted indoors, with the devices placed in several adjacent rooms to form a single MANET, thus using the walls as separators to simulate obstacles (see Figure 10).

Two kinds of test were conducted on the *NSI* component. The first was to fine-tune various component parameters such as packet size. The maximum time spent in discovering a node route, plus the time spent in sending data (message) to destination node (i.e. the total time spent for the complete execution of the `getRouteFor(peer): Route` and the `send(Message message): bool` methods – see Section 3 *IRoutingProtocol* interface and *MANETSocket* class) was chosen as the validating parameter, and 256, 512, 768, and 1024 bytes were selected as instance values for packet size. Results showed that 1024 byte packages were a good compromise between the time spent in sending the message and its size. 512, 768, and 1024 byte packages take almost the same time (Figure 11). In fact with messages of this size, most time is spent in discovering the route to the destination node, while the data transmitting time is relatively small (requiring four packets per message at most). Messages over 1024 bytes must be split into more packets, thus requiring more time to send the message from one hop to another. In this case, the node mobility means that connection failures are quite likely, necessitating a great deal of packet retransmission (this also explains why the time increases when the message size exceeds 1024 bytes).

The second test focused on measuring component soundness and reliability. The main goal was to verify the capacity of connection servers to accept and satisfy incoming packet requests from neighbors, especially when running on PDA devices. This was achieved by producing high packet traffic in the network to provoke frequent full server connection queue exceptions and thus packet retransmissions.

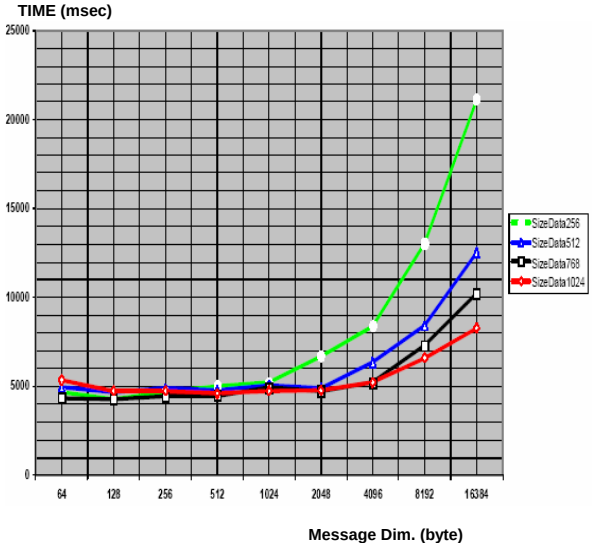

Figure 11. Experiment results. X axis represents message size in bytes, while Y axis represents the spent time to send message with 256, 512, 768, and 1024 bytes packet size, resp.

Packet traffic was generated by decomposing fixed size messages (i.e., 1, 2, 4 and 8 MB), into 1024 byte packets and straining the MANET hosts. Table 1 shows the results of our experiments. The time (in seconds) to send the whole message in MANET with 3 and 6 hosts is reported for each message size . As can be seen, the time spent in sending a message increases with its size, due to the higher number of packet retransmissions, principally caused by the greater number of full server

connection queue exceptions. The different results obtained in the two cases considered are because there are more alternative routes to the destination node for MANET with 6 hosts than for 3 hosts, thus decreasing the number of connection requests to each host.

Message Dimension	Time for 3 hosts (in sec.)	Time for 6 hosts (in sec.)
1 MB	14	323
2 MB	90	624
4 MB	438	1800
8 MB	840	2400

Table 1. Experiment results obtained for testing the component soundness and reliability.

Figure 12. MANET chat application used for testing the *MANETService* component.

5. USING THE *NSI* COMPONENT

In this section we report an example of Windows Mobile application –*MANET-Chat* –implemented to show the use of the *NSI* component (see Figure 12).

MANET-Chat is a simple chat application that may be run independently on PDA and laptop/desktop devices and on top of a MANET network. It uses the *NSI* component as MANET network layer to send and receive messages to and from other devices.

The main class application is the `Form` class of the `System.Windows.Forms` package. It includes: a `TextBox` object to enter the text message; a `ComboBox` object to select the list of message destinations; the `isLinked`, `send`, and `close` button objects to verify if a device is linked to the network, send the message, and unbind the application from the *NSI* component. Finally, the bigger `TextBox` object is used to show incoming messages from other network devices. The packages needed by the chat application are reported below. The `MANETService` package and the `MANETService.Utility` package contain all classes implementing the *NSI* component.

```
/* MANET chat application */

using System;
using System.IO;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Threading;
using System.Data;
using System.Text;
using System.Net;

/* using the MANETService package */

using MANETService;
using MANETService.Utility;


public class Form1 :
       System.Windows.Forms.Form
{
       …
       private MANETManager
              manager = null;
       private MANETSocket ms1 = null;
       private Thread listener = null;
       …
}
```

In the class constructor, variables are initialized with the instance of the `MANETManager` class and the `MANETSocket` object assigned to the application by a binding operation.

```
public Form1()
{      InitializeComponent();

/* Getting the unique MANETManager
instance */

       manager =
       MANETManager.getMANETManager()
;

/* Binding application on port 50 */

       ms1 = manager.bind(50);
       listener = new Thread(new
       ThreadStart(this));
       listener.Start();
       …
}
```

The send, isLinked, and close application buttons use the *NSI* component's send, isLinked, and close methods. The receive method is called by a thread object listening on a specific port.

6. CONCLUSION AND FUTURE WORK

In this paper we presented the design and a possible implementation of the *Network Service Interface* layer as a .NET Compact Framework component, coded in C#, to be run on PDAs with the Windows Mobile operating system; we chose Dynamic Source Routing as the routing protocol supporting inter-device communication.

The layer prototype is available at: `http://www.dis.uniroma1.it/pub/~mecella/projects/MobiDIS/`.

We reported a set of *NSI* component tests and their results. Finally, we described an example of Windows Mobile application –*MANET-Chat* – in order to show the use of the *NSI* component.

Future work will involve the development of the predictive layer on top of the *NSI* component in the .NET environment, using the probabilistic technique presented in [DeRosa05a].

```
/* Using the NSI send() method */

public void send(){
       this.textBox3.Text = "";
       string nameDest =
       this.comboBox1.Text;
       string message =
       this.textBox1.Text;
       byte[] message_b=
       Encoding.UTF8.GetBytes(message);

       Boolean boo =
       ms1.send(nameDest,50,message_b);

       this.textBox3.Text =
       boo.ToString();
}// End the send method

…

/* Using the NSI receive() method */

public void receive(){
       …
StructReceive sr = new
StructReceive(50);
StructReceive result = null;
while(breaking)
{
       result = ms1.receive(sr);

       if(result != null){
              string[] message =
              result.getMessage();
              …
              break;
       }
}
…
}// End the receive method

…
/* Using the isLinked() method */

public void isLinked(){
       string nameDest =
       this.comboBox1.Text;
       Boolean bo =
       ms1.isLinked(nameDest);
       textBox5.Text =
       bo.ToString();
}
/* The close method to unbind the
MANET chat application */

public void close(object sender,
System.EventArgs e)
{
       MANETManager.close(50);
       …
       MANETManager.release();
       …
}
```

7. ADDITIONAL AUTHORS

Fiammetta Pascucci and Piergiorgio Faraglia, undergraduates of the Faculty of Computer Engineering, University of Rome “La Sapienza”.

8. REFERENCES

[Agrawal03a] Agrawal, D. P., and Zeng, Q. A. , “Introduction to Wireless and Mobile Systems”, Thomson Brooks/Cole, 2003.

[Grudin04a] Grudin, J., “Computer-Supported Cooperative Work: History and Focus”, IEEE Computer 27(5): 19-26, 1994.

[Vaidya04a] Vaidya, N. H., “Mobile Ad Hoc Networks: Routing, MAC and Transport Issues” Tutorial on Mobile Ad Hoc Networks, http://www.crhc.uiuc.edu/nhv, University of Illinois at Urbana-Champaign, USA, July 2004.

[DeRosa03a] De Rosa, F., Di Martino, V., Paglione, L., and Mecella, M., “Mobile Adaptive Information Systems on MANET: What We Need as Basic Layer?”. In Proceedings of the 1st IEEE Workshop on Multichannel and Mobile Information Systems (MMIS’03), Rome, Italy, 2003.

[DeRosa05a] De Rosa, F., Malizia, A., and Mecella, M., “Disconnection Prediction in Mobile Ad hoc Networks for Supporting Cooperative Work”. IEEE Pervasive Computing, 2005, to appear.

[Gamma94a] Gamma, E., Helm, R., Johnson,R, and Vlissides, J., “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley Professional Computing Series, 1994.

[Johnson94a] Johnson, D., and Maltz, D. A., “Dynamic source routing in ad hoc wireless networks," in Mobile Computing (T. Imielinski and H. Korth, eds.), Kluwere Academic Publishers, 1994.