ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SS 2003 Roger Wattenhofer, Aaron Zollinger

Mobile Computing
Exercise 4

Assigned: May 8, 2003
Due: May 29, 2003

Instant Messenger

In this exercise you are going to use the previously developed layers to implement an instant messaging application. Basically, this application will present a list of terminals within transmission range. The user can choose one such neighbor with which to exchange messages, much like in one of the well-known instant messaging applications, such as ICQ or AOL Instant Messenger.

In order to allow for interaction of all of our implementations of the application, we have to define a message format and a number of message types. The message format for this exercise is based on the basic packet format defined in Exercise 2. In addition to that packet header—consisting of a sender and a receiver field—we introduce a type field. This field is formed by one byte placed directly after the sender/receiver header.

We define five different message types, each of which is identified by a byte value. For some of these messages, our protocol prescribes a certain application behavior upon receipt of such a message. The PING/PONG message pair is used for neighbor discovery. With the WHOAREYOU/IAM messages, users can identify themselves with names. The USERMESSAGE is employed to confer any kind of information to the destination user. The message types are summarized in the following table:

Message Type	Type Value	Data	Reaction upon Receipt
PING	0x01	none	send PONG message to sender of PING
PONG	0x02	none	none
WHOAREYOU	0x11	none	send IAM message to sender of WHOAREYOU
IAM	0x12	my name	none
USERMESSAGE	0x21	any kind	none

The last column of the above table only defines the reaction an application MUST implement. Of course all other message types have to be interpreted in a reasonable way. The first three message types do not use any data other than the type field. In the last two types the data segment (after the header) of the packet is also part of the message.

Since the application has to react directly to incoming messages, a `java.lang.Thread` dedicated to the handling of incoming messages should be used. In order to prevent blocking of this thread, it can spawn (although this is not mandatory) a new thread for the interpretation of each received message.

In order to acquire information about the neighboring terminals, the application has to emit PING messages. This can happen either on user demand or periodically. In any case, the neighborhood information collected from PONG responses has to be kept up to date in a suitable data structure. Messages of the remaining types can then be used to obtain user names and to transfer user messages.

A possible solution might offer a little graphical user interface, a command line application will however also be fine ;-). Finally, a word of caution: Your application will consist of several threads. Keep in mind that, as soon as two or more threads access a common data unit, they might come in each other's way. In order to prevent problems concerning the access of such a shared data unit, the threads require to be synchronized.