# Technical Report – Server

**Hardware and Setup**

The server in our project is only responsible for forwarding and, if the user is not connected, buffering the messages. It doesn’t store the user data in a permanent way and also cannot read the user messages, since the data is encrypted.

Because of these circumstances we decided on having a small, lightweight server. Therefore we used a Raspberry Pi Modell B [1]. The R-Pi runs an ARM 700 MHz processor and 512 MB of RAM. It is connected with an Ethernet cable to the network.

The R-Pi is running Raspbian, a linux distribution, adapted for Raspberry Pi devices. The whole installation is headless and accessed through SSH. This setup is very energy efficient.

Since we do not have access to a static Public IP, we have a client in the network, that updates a DNS entry [2] every minute and guarantees, that our server is always reachable over the same address.

**Software**

Because of the limited resources the hardware is offering. The server application also needs to be very lightweight. To avoid too much thread-switching overhead, to efficiently use the resources and offer good scalability, we decided on a Non-blocking I/O (NIO) [3][4] approach to design the server application. The core-application consists of two threads running in parallel. The server thread, which implements the NIO server and a worker thread which processes the incoming data and decides what to do with the messages. Based on the public header, the worker thread reads the sender, the recipient and the kind of message and then handles the message accordingly.

Two data structures are responsible to process the messages in a meaningful way. The first is a simple HashMap which stores the connected users and the corresponding sockets. If a user is not connected, then the message gets buffered in a user specific PriorityQueue. The time a user connects again, he can request all messages he missed when he was offline. Since the hardware has not too much memory available, the server does not guarantee to store all messages until the user connects again. In those cases, the user needs to rerequest the data from his friends.

Alongside the core-application, a monitor thread is running. The task of the monitor thread is to periodically check the connectivity and memory usage of the application. In case of failure and/or too less memory available, the monitor thread can command the server to drop the oldest messages or even to restart the whole server thread.

[1]http://www.raspberrypi.org/  
[2] freedns.afraid.org  
[3] <http://en.wikipedia.org/wiki/Non-blocking_I/O_%28Java%29>  
[4] <http://docs.oracle.com/javase/7/docs/api/java/nio/channels/package-summary.html>