ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Friedemann Mattern, Matthias Kovatsch
Distributed Systems Assignments HS 2012

Assignment 3

Start:	26 October 2012
End:	09 November 2012

Objectives

In this assignment, you will develop an n-person mobile chat application that preserves the causality and temporal ordering of messages in spite of the unreliability of the underlying UDP protocol. You will implement two concepts to determine the order of events in a distributed system that you have learnt in the lecture:

Lamport Timestamps   represent a simple algorithm to partially order distributed events. The rules that this algorithm follows were determined by L. Lamport[1]. Distributed processes that implement Lamport Timestamps satisfy the so-called *clock consistency condition*: If event A comes before event B, then event A's logical clock comes before event B's. Therefore, if event A's logical clock comes before event B's logical clock, then A may have happened before or at the same time as B, but did not happen after B.

Vector Clocks   represent an extension of Lamport Timestamps in that they guarantee the *strong clock consistency condition* which (additionally to the clock consistency condition) dictates that if one event's clock comes before another's, then that event comes before the other, i.e., it is a two-way condition. This is achieved by holding a vector of n logical clocks in each process (where n is the number of processes) and include these values in all inter-process messages.

To support this assignment, we have two servers running:

- **vslab.inf.ethz.ch:3999** provides a capitalization service – it returns every incoming message written in capital letters. This server may be used to test UDP-based communication. It also replies on port `Client:3999`.

- **vslab.inf.ethz.ch:4000** provides the chat service by distributing all messages received from registered clients to all other registered clients. The server randomly delays messages to simulate communication unreliability inherent in real systems. While this server is listening for incoming datagrams (registration commands or chat messages) on its port `Server:4000`, it responds to registration messages to `Client:4000`, and it is distributing the chat messages to `Client:4001` of the registered clients.

With this assignment you can gain 10 points out of the total 45. The exercises marked with a ⊙ are necessary to meet the minimum requirements ("save point"). Please use *VS_G**_A3_{TaskNumber}* and *ch.ethz.inf.vs.android.g**.a3* as Eclipse project/package names.

---

[1]Leslie Lamport - Time, clocks, and the ordering of events in a distributed system; ACM Communications Magazine, volume 21, issue 7, July 1978

![ETH]

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Friedemann Mattern, Matthias Kovatsch
Distributed Systems Assignments HS 2012

1 Getting Familiar with Datagrams

To familiarize yourself with the sending and receiving of UDP messages, create an Android application that provides a capitalizing service to its user while relying on the server at `vslab.inf.ethz.ch:3999`. This application has only demonstration purposes and does not need to be submitted, however, you can reuse parts of it later.

1. Create a new application and setup the UI to enable the user to enter and submit a text message and also display the server's answer.

2. Use UDP sockets `DatagramSocket(int port)` for communication with the server.

Figure 1: The setup of the capitalization service

2 Starting the Conversation – Lamport Timestamps (4 Points, ⊙)

In this task, you will create a chat application that leverages the communication server at `vslab.inf.ethz.ch:4000`. The application will use Lamport Timestamps to delay the delivery of messages to the user if necessary. Create a program that enables the user to choose a user name, register with the server and start chatting with other clients using the guidelines defined in the accompanying slides that are provided at `http://vs.inf.ethz.ch/edu/vs/android/`.
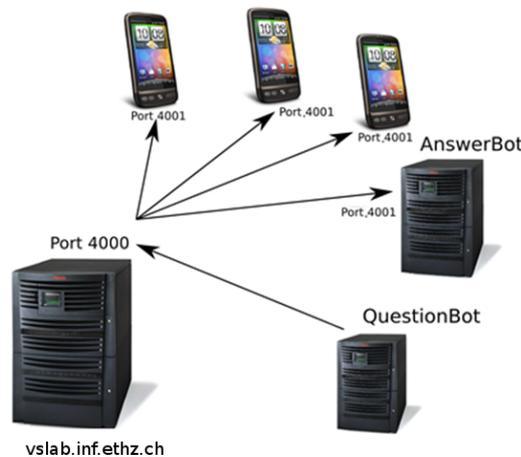

Figure 2: Overview of the chat system

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Friedemann Mattern, Matthias Kovatsch
Distributed Systems Assignments HS 2012

1. Create a new application and configure the UI to provide a server registration/deregistration button, to display incoming messages, and to let the user send chat messages. For displaying chat messages, you can, for instance, use the Android `ArrayAdapter` that provides an implementation of an `Adapter` and uses an array of arbitrary objects to manage a `ListView`.

2. Provide the user with the ability to enter a username before registering with the server.

3. Add functionality to allow the user sending messages to the server via its port `Server:4000`. The registration and deregistration buttons should also make use of this function. The server will reply to registration commands to the client's `Client:4000` port, so also add functionality to receive these replies. Also make use of the `setSoTimeout(int timeout)` function. The server will return the assigned username, the assigned vector index and the current time vector. Ignore the information on the assigned index, this will be needed for task 3.

4. Familiarize yourself with Java Threads. Implement a Thread that listens for incoming messages (on port `Client:4001`) in the background. For cross-thread information exchange, you will most probably want to use the `Handler` class. Remember to keep your `onCreate()` method as clean as possible and to use Threads for delegating potentially long-running tasks.

5. Test your listener thread. If you are using the emulator, remember to set port redirects. Depending on how fast you have progressed with the assignment, there will already be lots of communication going on. At the least `QuestionBot` and `AnswerBot` will be bringing the chatroom to life.

6. Enhance your listener Thread: Use Lamport Timestamps (index 0 in time vector) to delay the displaying of incoming messages if their timestamp indicates that they should not yet be delivered to the user. To do this, create a method `isDeliverable(...)` that explicitly inspects the timestamp of every incoming message and decides whether or not to delay its delivery. As soon as this function returns `true` for a message, it shall be delivered to the user.

7. Design your application such that it can handle multiple incoming messages simultaneously.

3 Overcoming the Desequencer – Vector Clocks (4 Points)

For this task, create a new application that implements the same functionality as the one developed in task 2 but uses Vector Clocks for determining the order of messages in the distributed system. Your application should also be able to handle that clients are dynamically joining and leaving the chat.

1. Create a new application and setup the UI and functionality as before.

2. Instead of Lamport Timestamps, use Vector Clocks to determine the order of messages and display them to the user accordingly. Thus, you now have to parse and make use of the index number that the server assigns to your application during registration.

3. Create the method `isDeliverable(...)` that explicitly inspects the vectorial timestamps of incoming messages and decides whether or not to delay the delivery of a message. As soon as this function returns `true` for a message, the message shall be delivered to the user.

4. Design your application such that it can handle multiple incoming messages simultaneously.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Friedemann Mattern, Matthias Kovatsch
Distributed Systems Assignments HS 2012

4 Report (2 Points, ⊙)

As part of the assignments, you should produce a short report **(1-2 pages)** on the design and implementation issues of tasks 2 and 3 and motivate any choices you have made during the process. You can find a template for your report on the course website. Include a thorough discussion of issues and considerations related to Vector Clocks in your report. Specifically, answer these questions:

- What are the main advantages of using Vector Clocks instead of Lamport Time?

- When *exactly* are two Vector Clocks causally dependent?

- We decided in the exercise that we would not let our applications trigger a tick when receiving a message. What would be the implications of ticking on receive?

- Does a clock tick happen before or after the sending of a message. What are the implications of changing this?

- Read and assess the paper Tobias Landes - Dynamic Vector Clocks for Consistent Ordering of Events in Dynamic Distributed Applications[2] that gives a good overview on the discussed methods. In particular, which problem of vector clocks is solved in the paper?

Deliverables

The following two deliverables have to be submitted by **09:00am, 9. November 2012**:

- **code.zip** You should create a zip file containing the Eclipse projects created in this assignment. The projects should have been tested both on the mobile phone and on the emulator. The code must compile on our machines as well, so always use relative paths if you add external libraries to your project. Do not forget to include those libraries in the zip file. Please use UTF-8 encoding for your documents and avoid special characters like umlauts.

- **report.pdf** The report in **pdf** format.

Submission

Report and code must be uploaded through:

```
https://www.vs.inf.ethz.ch/edu/vs/submissions/
```

The group leader can upload the files, and other group members have to verify in the online system that they agree with the submission. Use your `nethz` accounts to log in. The submission script will not allow you to submit any part of this exercise after the deadline. However, you can re-submit as many times as you like until that.

---

[2]http://vs.inf.ethz.ch/edu/vs/exercises/DVC_Landes.pdf