**Bluetooth Notepad**

Documentation for Distributed Systems 2018 course work by group Töttöröö

Niko Lehto – 2374145 – [Mikko.N.Lehto@student.oulu.fi](mailto:Mikko.N.Lehto@student.oulu.fi)  
Timo Mattila – 2255455 – [Timo.Mattila@student.oulu.fi](mailto:Timo.Mattila@student.oulu.fi)  
Arttu Niemelä – 2255484 – [Arttu.Niemela@student.oulu.fi](mailto:Arttu.Niemela@student.oulu.fi)

**Introduction**

Bluetooth Notepad works as a distributed, collaborative notepad. Users can edit a single document together on separate machines, being connected via Bluetooth. The program presents a simple notepad interface. One user starts the program and creates a group by giving it a name and a password. Other users may join this group with the name and password. Once connected, edits made by any user in the group are shared with the others, keeping the document synchronized.

**Architecture**

Nodes in the system are essentially any machines with Bluetooth and ability to run the program. Nodes are separated into a host and clients. Host is usually the first node on which a document is created. Clients are nodes that join an existing host. The host and clients form an ad hoc Bluetooth network. Clients are only connected to other clients through the host. Therefore the host acts as a coordinator for all messaging. Bluetooth stack should support at least 8 node network. However, the program has been tested with maximum of 3 nodes (1 host, 2 clients).

**Processes**

The systems heavily utilizes threading. The program is divided into 3 classes: ‘text editor’, ‘text window’ and ‘Harald’. Text editor is the users’ entry point to the program and defines part of the user interface e.g. buttons. Text window controls the user’s written input and displaying of text in the editor. All Bluetooth communication is implemented in Harald. Edits and other messages are exchanged in queues between the text window and Harald. Text editor and window run on the main thread. Harald runs a parallel thread which handles sending messages and exchanging data with the text window. If the program is ran in client mode, an additional thread is started when connecting to a host. This thread is also used to receive messages from the host. If the program is ran in host mode, the host continuously runs an advertising thread. When a client joins, the host starts individual receiving thread for each client. Additionally host is continuously checking possible other hosts in another thread. In total a client runs 3 threads. Host runs at least 5, depending on the number of clients.

**Communication**

Nodes (users) are connected via ad hoc Bluetooth network. One node acts as the host. Other nodes connect to the host as clients. All messages (edit proposals) from clients go through host. Host orders the edits and broadcasts them to clients. Most messages are edits done by users. Possible edits include inserting and deleting letters, or copying and pasting larger parts. Other messages are access and synchronization requests. Messages are sent in JSON format, which includes the following data:

* source: the source function of the message
* message: letters or text to insert
* from: coordinates on the notepad from which to start an insert or a delete
* to: coordinates to which to end an insert or a delete
* type: type of message, e.g. insert, delete, access or synchronization request
* order: synchronized order of the message

The program also supports long messages by splitting them into multiple parts and sending them one by one. In such case the messages also include a header, in which the length of the message is stated. With the length receiving nodes know to wait for the rests of the message and put it together.

**Naming**

Hosts advertise their available service with Universally Unique Identifier (UUID) specific to the application. Clients recognize available services by this UUID. Hosts also advertise their group’s name defined by the user. Clients search available services for the group they wish to join. Therefore clients are able to find the correct Bluetooth Notepad group by the UUID + group’s name. In the special case there are multiple Host’s advertising with the same group name, the clients choose the one which was started earlier. For this, hosts include their start time as UTC timestamp in their advertisement.

**Synchronization**

Synchronization is done by having the host act as a coordinator. When a user edits a document, the edit first goes through host’s ordering process. Host decides the order (an incrementing integer) of the edits and broadcasts these to clients. With the order, client know if they receive the messages in correct order, or if a message has been lost. If a client notice the message order has been broken, a synchronization request is sent to host. Host responds to this by sending the whole current text as response, with the current number of order. Clients replace the text on their end with the received one and continue from the received order number.

**Consistency & Fault tolerance**

Each node periodically caches the current text onto their local drive. With the cache, editing can continue where it was left if the program is closed. However, cache has more important function to provide consistency if host goes offline. As the text is synchronized between each node, so the cache stays consistent. If connection to host node is lost, the whole network is down. In this scenario each client starts to host by themselves. The nodes advertise their service, with a timestamp of when they started hosting. At the same time the nodes also scan for other hosting nodes. If there are other hosting nodes, a scanning node compares the starting times and joins the earliest host. Thus all the nodes eventually cascade into being connected through one host again.

**Security**

An access control scheme is in place. A password is required to join any group. After finding correct group, a connection between a host and client is established. Before any other messages are passed between this specific connection, client sends the password given by the user. If the password matches the one for the group of the host, connection is continued and messaging may begin. Otherwise host closes the connection to this client and client notifies the user of this.

**Reflection**

The project really touched many of the subjects presented in course material. Threading became very important to run Bluetooth network and the editor concurrently. The ad hoc nature of the network forced to think hard how to implement coordination, fault tolerance etc. since communications are lost completely with the host going down. Also the used Bluetooth library was a bit lacking for Windows and many unpredicted problems were encountered. Developing a Bluetooth emulator might have actually saved time in the end, since testing was very time consuming. In any case, project was very much fun to work with, especially with the Bluetooth giving a sort of miniature complete network to play with.

The project should give right to 5 extra points:

* *Fault tolerance and recovery*: system is able to automatically reform connection
* *Consistency and replication*: with caching, system is able to pick up where editing was left
* *Naming*: other nodes are found using Bluetooth’s discovery mechanism / scanning
* *Security*: password is required to join a group
* *Distributed synchronization*: order of events is decided by the host and clients follow this order