Shopping XYZ

Distributed Systems – Project Proposal

IVANOV Danil, SANSONETTI Luigi, HÄNER Dominik, DRESCHER Lukas
ETH ID 1: 15-826-415, ETH ID 2: 14-803-688, ETH ID 3: 14-945-133 ETH ID 4: XX-XXX-XXX
divanov@student.ethz.ch, sluigi@student.ethz.ch, dhaener@student.ethz.ch, lukasd@student.ethz.ch

ABSTRACT

As of today, there is no open-source Android application that solves the consensus problem in a flat-share situation. As a result, users too often have to juggle between apps, trying to manually maintain an updated state of the situation on several different phones.

Shopping XYZ is an Android application to streamline the efficiency of grocery acquisition in a shared living accommodation. The main idea is to use the concepts of distributed computing to maintain an updated state of various lists across all devices in a cluster.

System overview: Implementation will be decentralised and distributed among the users. Each node will act as a client and a server, supplying the rest of the cluster with the required information for maintaining consensus.

Software and hardware used in this project: The application will be implemented on Android phones, using the Android SDK in conjunction with Java. The application requires internal storage, as well as internet access.

Expected deliveries: An Android application that solves the aforementioned problem, using distributed computing concepts.

1. INTRODUCTION

Manually updating a shared shopping list is an error-prone process that doesn't scale. The task constitutes a high cognitive load for humans, leading to mistakes and oversights during manual maintenance of the said list.

Shopping XYZ maintains the state of a shopping list distributed across several devices without relying on a central server.

However, by distributing this task, the consensus problem is to be taken into account ; it is important to make sure that all nodes agree upon the information, and to make sure that every change is communicated.

Each node implements a client/server architecture. The client listens to other servers for updates. The server communicates changes to other clients.

In order to maintain consensus over the shared list, a node notifies other when it modifies the list, and makes that all other nodes acknowledge the change. Upon connection to the network, the new node queries for the latest version of the said list.

Furthermore, not having a central server to rely on raises the question of how to create a closed network.

To build this platform, the following technological problem must be solved : "How to implement a distributed communication system over a network of Android devices ?"

2. SYSTEM OVERVIEW

Technical plan: Shopping XYZ requires the implementation of a distributed client/server logic among a network of android devices as well as the development of a UI design.

Any device can invite another one to its flat-share (network) by sending it an invite. The new device, upon entering the network, gets the set of all IP addresses of the other nodes, and broadcasts its own IP address to them. Every device keeps a set of the IP addresses of all the nodes in their network.

The application contains a service that would check whether the devices IP address changed. If it did, the application immediately broadcast its new address to the network, and all other nodes will update their IP addresses sets.

Upon performing a legal operation on an element in any list, the node responsible of this manipulation will immediately notify all other nodes of this manipulation.

Project design: The user interface will be similar to the following:

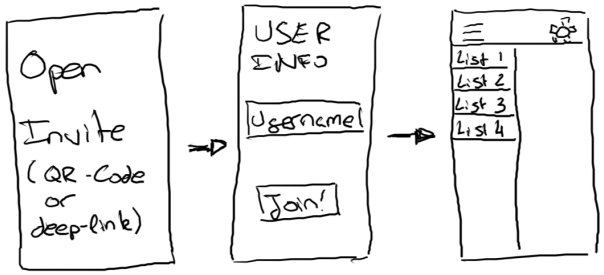

Figure 1: UI sketch

The protocol to join a network is as follows:

1. Node B sends an invite to node A via an external channel (messaging service or QR-code).
2. Node A acknowledges the invitation and sends its IP to B.
3. Node B sends the set of the IP addresses and their corresponding IDs of every node in the network.
4. Node A broadcasts its IP and ID to all the nodes in the network.
5. Every node updates their IP/ID map with the new entry.

(See Figure 2)

Upon joining a network, a node A accesses the public list, which is shared between every node, and creates

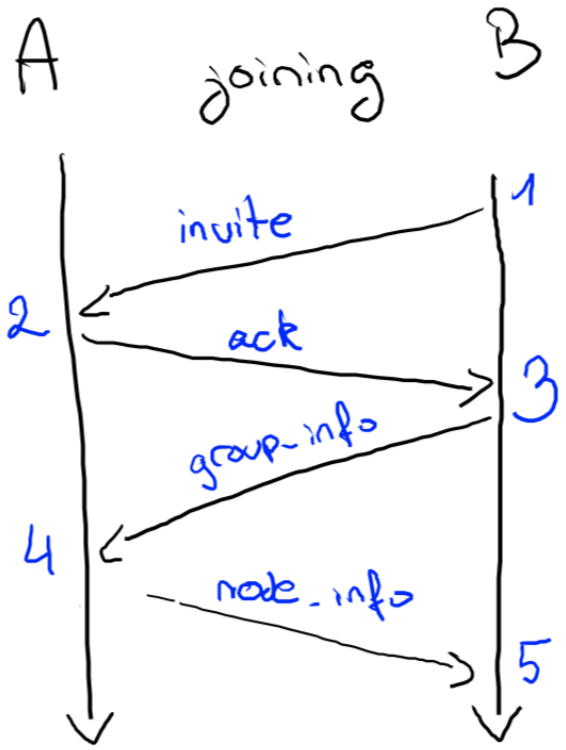

Figure 2: Joining protocol timeline

- a personal private list that will be known only to A,
- a personal public list that will be known to every node in the network.

Suppose node A is trying to reconnect to the network. The protocol is as follows:

1. A broadcasts a reconnection request message to all the nodes in the network.
2. Every node getting this message acknowledges it.
3. Let B be the first node to acknowledge A's message. A queries B for its set of IPs and IDs.
4. A updates its set of IPs and IDs, and broadcasts it.

In the case where every node is disconnected (i.e.: A gets no acknowledgement message to its reconnection request), the network has to be created again. The joining protocol is therefore repeated for all the disconnected nodes when they reconnect in a way that will save the existing data.

When a node A tries to apply a modification with a parameter to an item of a list, it simply broadcasts the following message:

A's ID	list	modif.	value	item ID	previous value

Figure 3: Modification message content

where the item ID is the hash of the item's value.

The modification can be either "add", "remove", "update", "makePublic" or "makePrivate".

- Add: takes a list ID L, a value V and an ID I as parameters, and adds the item of ID I and value V to L.
- Remove: takes a list ID L and an ID I as parameters, and removes the item of ID I of the list L.
- Update: Takes a list ID L, a value V and an ID I as parameters, and updates the value of the element of ID I of list L to V.
- makePublic: Takes an ID I, removes the element of ID I of the node's personal private list, and adds it to the node's personal public list.
- makePrivate: Takes an ID I, removes the element of ID I of the node's personal public list, and adds it to the node's personal private list.

Expected issues: All nodes disconnecting from the network would require the network to be rebuilt. This problem is rather minor, given that

a) In small networks, rebuilding is fast,

b) The larger the network, the greater the chances of at least one node being connected.

If two nodes propose changes at the same time, then the following happens:

- Add and Update: The user is asked to resolve the conflict manually.
- Remove: The item is simply removed.
- makePublic and makePrivate: only the node in question can perform those operations on its lists.

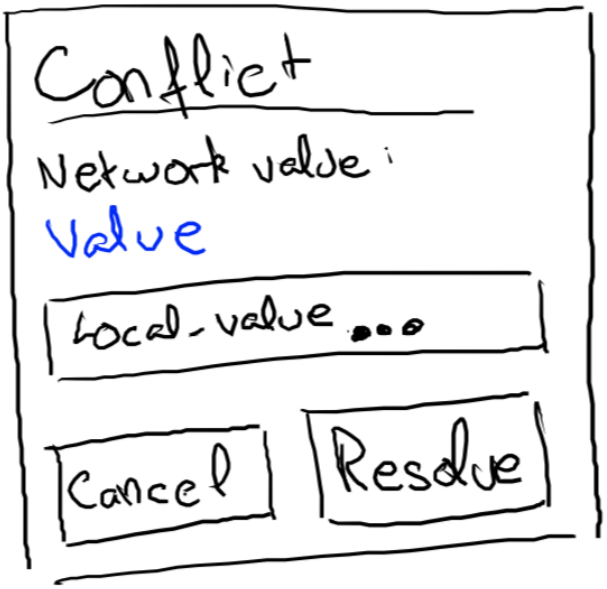

Figure 4: Conflict solving prompt

Figure 4 shows what the user will be prompted to solve a conflict. The user can modify the *local_value* text field to resolve the conflict, or cancel to accept the network value.

3. REQUIREMENTS

A basic Android phone with internet connectivity is required.

Given that several lists will be stored on every device, some storage space is required as well (a few MB).

4. WORK PACKAGES

Breakdown of the work to subtasks to meet the project requirements.

Joining protocol: Implement the protocol for a node to join a network.

Communication protocol: Implement the protocol to allow nodes to communicate.

List manipulation: Implement list logic (modification operations, UI communication).

UI design: Implement UI design.

Presentation: Prepare the presentation slides, project logo, and demo.

5. MILESTONES

5.1 Division of tasks

Joining protocol: Danil, Luigi.

Communication protocol: Lukas, Dominik.

List manipulation: Danil, Luigi.

UI design: Lukas, Dominik.

Presentation: Everybody.

5.2 Schedule

26/11: Joining protocol implemented.

03/12: Communication protocol and list logic implemented.

10/12: UI design implemented.

15/12: Presentation and logo ready.

17/12: Project completed, functioning demo.