NearTweet: Location-aware Microblogging Service

Project Assignment for Mobile Computing

2012/2013

MEIC/MERC

1. Introduction

The goal of this project is to build NearTweet, a microblogging service for crowds. From a user point of view, NearTweet consists of an application that users install on their mobile devices and enables them to share content within a crowd. Communication is possible either by relying on some server or by leveraging the devices' close-range wireless capability to create a *spontaneous network*, over which users can broadcast messages. Spontaneous networks could form in relatively dense concentrations of people, both in open or closed spaces, such as stadiums, pavilions, malls, traffic jams, concerts, and restaurants. Although such gatherings are normally formed by anonymous and unrelated individuals, the context of the environment could prompt them to communicate with each other. For example, in a traffic jam, one could wish to ask the crowd why the traffic is stuck; in a canteen, poll the population for the best dish; in a concert, advertise events of his own band; and on a stadium, spread the word on some tickets left to be sold. NearTweet could support such use cases as well as many others of similar nature.

NearTweet is the CMov course's project proposal. Students are expected to design, implement, and evaluate NearTweet on the Android platform. In the rest of this document, we describe NearTweet in more detail and provide additional instructions to realize the project. Keep in mind that NearTweet is loosely specified so that students have the room to further extend its functionality and make their own design decisions.

2. NearTweet Overview

Figure 1 illustrates the software stack of a mobile device running NearTweet.

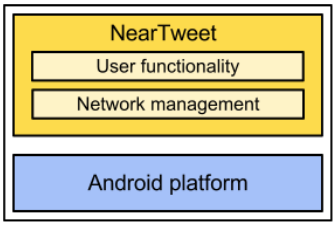

Figure 1: Software stack of a mobile device running NearTweet

NearTweet is an Andoid application whose functionality enables users to communicate on spontaneous networks. NearTweet must also manage the underlying network in order for the communication to take place. In the following sections, we describe the functionality of the application, and characterize the network architecture.

3. Functionality

NearTweet allows users to exchange information within the spatio-temporal context of crowds by providing the functionality described below. Optionally, students can develop additional features of their liking on top of NearTweet's basic ability for exchanging messages.

Tweet: The tweet operation enables users to broadcast a message on the network. Users are notified of the messages sent by the other elements of the crowd. The maximum size of a message is 160 characters. Each user can configure NearTweet with a pseudonym of his choice. This pseudonym is the user's ID in the network.

Reply: A user must be able to respond to the messages he receives from the other users. The user can either reply to all the network participants or to the sender only. NearTweet must display the messages to users so as to reflect the thread of each conversation.

Retweet: The messages propagated and received on a spontaneous network can be forwarded to an online service using the retweet operation. First, users configure NearTweet with the online services on which they have an account (e.g., Twitter, Facebook). Then, retweet lets users choose which of these services should a message be sent to. NearTweet must support at least one online service.

Poll: This feature enables users to poll the crowd about a particular issue (e.g., to ask what's the best dish of the restaurant). The poll operation allows the user to define his query (a ballot), consisting of a message body and a set of options. NearTweet broadcasts the query and lets other members cast a vote by picking an option. At the user end, NearTweet aggregates the responses and displays the outcome of the voting to the user.

Multimedia sharing: In addition to text, NearTweet allows for the sharing of multimedia content, namely photos. These files can be sent as attachments of NearTweet messages. Users can include attachments from at least of three possible sources: i) an online service (based on a url), ii) from Android's clipboard, or iii) from another application. The multimedia content can be visualized in NearTweet or using an external application.

Sensor data sharing: Users can also share data that stems from the sensors of their devices. For example, a user could take a picture using the local camera and share it with the crowd, or read his absolute location from the phone's GPS and send it out. At least one sensor must be supported.

Spam detection and eradication: To alleviate the problem of unwanted content dissemination (spam), NearTweet must provide a mechanism for detection and eradication of spammers. Whenever a spammer broadcasts a message to the network, users can flag that particular message as spam. As soon as the number of accusations reaches a pre-defined value, NearTweet bans the spammer from the network, and prevents him from sending more messages. Students are free to propose and develop smarter spam mitigation techniques.

4. Network Management

4.1. Network Model

NearTweet focuses mostly on spontaneous networks, which emerge randomly whenever a crowd's mobile devices interact. NearTweet can also make use of devices' connectivity to the Internet to enable interaction with online services (e.g., for retweet). NearTweet manages the spontaneous networks and bridges them with online services.

To better clarify the concept of spontaneous network, consider the scenarios depicted on Figure 1 and Figure 2. A spontaneous network consists of a communication network that can be established between closely located mobile devices, such as the phones of people that are stuck in traffic (see Figure 1) or dining in a restaurant (see Figure 2). The network comprises the devices that can form a fully connected graph. The connectivity graph could change as

the users move around, and network partitions could also occur. For example, if user D of the restaurant scenario (see Figure 2) leaves his place to visit the bar, user E could not communicate with the rest of the network and vice-versa. To differentiate temporary from permanent (e.g., users D and E finish their meal and leave the restaurant) partitions, each partition forms an independent network after a predefined time threshold has elapsed. Over time, the membership of nodes that belong to the network can change as new users approach and leave the network.

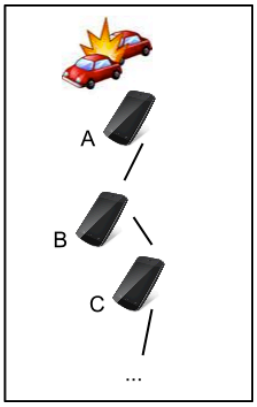

Figure 2: Traffic jam scenario

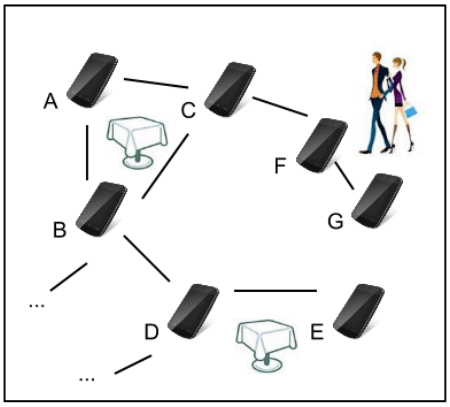

Figure 3: Restaurant scenario

We leave additional issues to be specified by the students. For example, when two devices of independent networks approach each other, what happens? Should they merge into a single network? Should both networks be preserved? Students are free to define the semantics they find more reasonable and must properly justify their choices.

4.2 Network Architecture

Implementing the network model just described raises some challenges, such as defining the network boundaries, handling churn (participants entering and leaving), and routing messages. To address these challenges, in the first phase, students start off by adopting a client-server architecture, which is simpler to implement. Later, students implement a fully decentralized architecture, which is required for reaching the maximum score. NearTweet must be implemented in Java, and the communication done over TCP/IP. WiFi should be used for wireless communication.

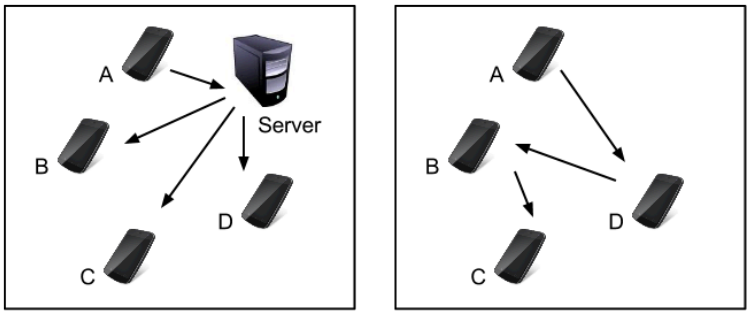

Figure 4: Message broadcast by node A on client-server (left) and decentralized (right) architectures.

Client-server: In a client-server architecture, a spontaneous network forms around a stationary server, which is responsible for managing the network. The devices play the role of clients; they connect wirelessly to the server in order to be part of the network formed around the server. Servers will be placed in areas where spontaneous networks are expected to form, e.g., at road intersections, or restaurant halls. Since a server can manage the entire state of his network and provide for the connectivity between devices, it is relatively easy to solve the technical issues related to the device membership management and communication support. This architecture, however, is very limited, because spontaneous networks cannot be formed in spaces where servers are not deployed.

Decentralized: A decentralized architecture aims to address the limitations of client-server. Instead of requiring the presence of a server to coordinate the network, the coordination takes place entirely between devices. As a result, spontaneous networks can be formed anywhere. The price to pay for this versatility is complexity. There are two main challenges: i) keeping track of a network's members as they roam around, and ii) disseminating messages on the network. Since the network state is no longer centralized in an omniscient server, additional coordination effort between devices is required in order to keep the network state consistent, namely the current set of network participants. Communication is also more complex, and the devices must implement dedicated protocols for message broadcasting. Students are free to design and implement the distributed protocols they deem more adequate.

5. Implementation

The project must be implemented using the Android Framework. Students can use third-party support libraries, as long as this does not provide the solution but rather contributes to it. All used software should be referenced appropriately in the report.

For testing the prototypes, a few Android tablets are available for student use in addition to the Android emulator. The tablets are Samsung Galaxy Tab, running Android 2.3 and Asus eeePad Transformer, running Android 3.2. To use the tablets, students must make a reservation. Each lease will last for one day, after which the device must be returned. A group may only make another reservation after returning the device. Each group may only request one device at a time. Details of the reservation, pick up and delivery process will be placed in the course's website.

7. Evaluation Criteria

The projects will be graded based on multiple aspects. The most important ones are: the functionality implemented, the type of architecture that is supported (client-server or decentralized), and the design decisions taken by the students. The adherence to the Android architectural model will also be evaluated, i.e., application decomposition into loosely coupled components (Activities, Services, Broadcast Receivers, Content Providers).

8. Grading Process

The grading process includes four stages: checkpoint, final delivery, visualization, and discussion.

- **April, 8th to 12th: Checkpoint** – The checkpoint consists of a presentation of a first version of the project. Additional details on what is expected to be shown will be provided on the course's website. The checkpoint will account for 15% of the grade, but only if its grade is higher than that of the final delivery. The checkpoint will take place in the lab classes.

- **May, 11th, 17:00: Final delivery** – For the final delivery students are expected to hand in the source code of the project and a report. The report should clearly state what was and wasn't done. It is limited to 5 pages (excluding cover). The cover should clearly indicate the group number and the name and number of each of its elements. A template for the report is provided in multiple formats on the course website. The delivery of the code and final report is done via Fenix.

- **May, 13th to 17th: Visualization** – Each group will have 20 minutes to present the developed application. Each group should design the set of experiments to be presented in the visualization. These experiments should be carefully crafted in order to demonstrate that the application works as expected.

- **May, 20th to 24th: Discussion** – The discussion will take 30 minutes per group. The final grade awarded to each student will depend on his performance in the oral discussion and may vary within each group.

Good Luck!