University of Amsterdam

Personal document

Creating COMRAD

January 29, 2019

Student:
Okke van Eck
11302968

Lecturer:
drs. A. (Toto) van Inge

Course:
Netcentric Computing

Course code:
5062NECO6Y

1 Note to reader

During the project, I will also be inducted for a board year at **via**. Therefor, I will also do some work as preparation for the upcoming year. In consultation with my team members, I will work some days of the week till 3PM and work in the weekends. They will not work in the weekends and thus our efforts will be equal. Since I will work relatively more on my own, we are going to give me the tasks that can be worked out best separate from the rest of the project.

2 Logs

2.1 Starting up

We started our project with a brainstorm for the subject. Several possibilities were discussed regarding the creation of an application with music. We all love music and noticed that there are a lot of different takes on a music app. To create an ad hoc network, we searched for different ways of actually setting up a wireless network. The first thing that came up was Bluetooth. However, we also looked up other ways on the internet and found out about Wifi Direct. Wifi Direct is much faster than Bluetooth and has more range. Therefor, we decided to go with Wifi Direct for our first take on creating an ad hoc network.

With Wifi Direct, we can make an PC based application. This appealed more to us than mobile development, since you don't have the hassle of setting up Android Studio and getting your test devices to work. Since our application will be a proof of concept, we do not have to think about the difference in usability between an mobile app and an PC application. If our idea turns out to be more suitable for mobile, it does not matter if it works on a laptops. We looked up different ways of creating PC based applications and found out that JavaFX is easy to use and can create good functional applications.

2.2 Working out the idea

When working out different ideas regarding the application, there is one idea that sticks with us. We want to work out an application that analyses the optimization of sharing files over an ad hoc network. It would be interesting to see the robustness of a wireless connection when sharing big files. With big files, we need to make as efficient use of the available bandwidth as possible. We also need to find ways of decreasing the latency between the request of a file and receiving it. We are going to do several tests to see what does and does not improve the bandwidth and/or latency.

Another issue with ad hoc networks is that it is serverless. Thus if a device requests a song and the sending device disconnects, we do not have a valid source and the requested node fails. It could be that another node in the network has the same file. Therefor, it would be interesting to see if we can find a way of rearranging the network in such a way that the receiving node does not or barely notices anything of the disconnection of the current source. We can also look at a way of caching songs, so there is never a single device a song is solely stored on. Maybe there are other forms of redundancy that we can find and are more effective.

We think that sending music over a network is exactly the way to test these aspects of ad hoc networks. Music files can be very large and take up lots of bandwidth. Therefor, we need to come up with an efficient way of distributing them through the network. With these big files, it may also need to be necessary to look for congestion avoidance methods.

The current idea is to make a self regulating network that scrapes music of a device and make public to the other devices (*nodes*) in the network. The final product will be a proof of concept, hence we will focus the most on the theory behind the sharing/streaming of music via ad hoc networks. To accomplish such a product, we at least need to implement the following parts:

- Make connection between devices possible
- Make discovery of nodes possible
- Find a way to scrape the music of a device
- Setup a discovery table for the music distribution in the network
- Make the network self regulating
- Create a data structure for sending songs across the network
- Make a system for requesting songs
- Find a way to upload/download/buffer/stream music
- Create a pathfinding algorithm for nodes to find each other
- Find a way of playing music on a device

2.3 Wifi Direct vs Bluetooth

We have looked at different ways of implementing Wifi Direct and came to the conclusion that it is going to be very difficult for us to create the network we are imagining. Wifi Direct requires a group owner that manages the communications like an access point. That means that if we want to create a self regulating network, we need to actively setup multiple group owners, which is not efficient. It was also difficult to get our PCs connected via Wifi Direct. Therefor, we decided to switch to Bluetooth instead. Bluetooth is a bit slower than Wifi

Direct and has a smaller range. This, however, is a good thing for the research. By using a less powerful network technology, we need to be more efficient with our way of sharing the songs. Since we switched to Bluetooth, we have also decided to create an android app instead of a PC application. We know that the interfaces for Bluetooth on android devices is very well documented, which will hopefully give us more time to spend on the research.

2.4 Scraping music from devices

I have started on searching for a way of scraping music of devices. The first thing to do with an android app is to ask the user for permission. It was a bit of searching, but I have found a way of storing the users permission for the rest of the time the app is installed. That way we do not have to ask for permission every time the app starts up. With permission, we now need to find a way of scraping the music. I have used a ContentResolver to get the URI's of the songs. With the URI's, we can create a Cursor object that loops through the found URI's. Then we can fetch all the meta data per song and create Song objects. To be able to create those Song objects, we first need a data structure and class. Therefor, I have created a Song class with the needed getters and setters to store the meta data of a song. We will be storing title, artist, location on the device and size of the songs. The MAC address of the owner will also be stored in a Song object as it is needed for testing.

2.5 Songs

2.5.1 Playing on device

Playing the music on a device is also something I looked at. First, I created a view that lists all the songs found on the device. This view uses the *.toString* method of the previously generated Song objects. The idea is to start a media player when a song has been tapped on a device. The android media player has support for playing music from local files. With in mind that we will receive songs over the network, the current approach is to store songs in a temporary file. The media player uses a FileOutputStream to load the song, which is than prepared via a FileInputStream. The problem I had was to play the music while loading songs. To let them work in parallel, I had to prepare in async and create a listener on completion. When writing to the temporary file, I had to convert the music to a byte array. This was a bit of an hassle, since I could not find great documentation on how to do it. In the end I figured something out by creating a new BufferedInputStream on top of a FileInputStream and writing the file bytes to the BufferedInputStream.

2.5.2 Songs over the network

The next step was sending songs over the network. Other members of the team have already created a messaging system which uses headers to tell the nodes where which packet needs to go. I used this as the basis for sending songs. The first thing I came across was that our current system only works for plain messages. I needed to be able to send objects (songs) across the network. Therefor, I did a rework of the setup of the sockets so we now create and handle via an ObjectInputStream and an ObjectOutputStream instead of an InputStream and an OutputStream. The sockets still uses a normal Input- and OutputStream to transfer the data, since this is the only way Bluetooth can transfer Serializable data. Hence, the new way is an extra layer on top of the socket streams.

Next up, I started working on the addition of Split Packets to our program. Until now we have only send whole songs as one across the network. A node has to wait for the whole song to arrive before it can start playing. Split Packets is a way of sending the songs split up into multiple parts, but still only play the song when all the parts have arrived. The first step is splitting up the song and sending it buffered to the receiving node where the node puts everything together and plays the song. This turned out to be more work than

expected, since our code did not support the sending and receiving parts of songs at all. I reworked the framework for messages to support this feature. To keep things simple, I just saved the songs in memory. This turned out to be an issue when sending large songs over the network. Therefor, we decided to just save the song parts on the device and let the media player run along all parts. The problem is when to load which packet. Therefor, I have created a SongPacket class which contains an ID and the size of a packet. This is enough for the program to load the packets in the right place in the buffer. When all packets are loaded in the buffer, the media player starts and the device plays the song.

There are a couple of downsides with this approach. A big downside is that the requesting node has to wait for the whole song to be transmitted across the network. Moreover, if there is a node in between, the song needs to be fully transmitted twice, which doubles the waiting time. This increases for every extra hop in the network.

Another downside is that the music is stored fully in memory. This is not a problem for regular sized songs (up to 10MB). However, larger files (with for example lossless compression) can cause nodes to run out of memory when receiving such songs.

2.5.3 Active Streaming

With active streaming, a song is split up in parts of a specified size and send after each other across the network. The media player can then play the first part of the song while it receives the other parts. Sending the song fragments is done via a filestream. Because there was no documentation of a MediaPlayer having streaming capabilities, AudioTrack was researched, which does have a streaming mode. However, AudioTrack does not provide automatic decoding of the music stream, while MediaPlayer does. Since we are supporting a wide range of music files, we decided it was too much effort to implement decoders for AudioTrack properly. Instead, MediaPlayer supports a feature where it can start a MediaPlayer right after the previous one completed. In order to stream the music, we simply chained a lot of MediaPlayers together, each playing the small fragment of a song that is encoded in the received SongPacket. SongPackets arriving in the wrong order is no problem, since Bluetooth is very sequential. In other words, the packets arrive at the exact order they were sent, and the MediaPlayers can be played right after each other.

There are some small drawbacks to this way of playing the music. Some music encodings have headers and footers. Since the packets are played in isolation, the MediaPlayer is unable to understand the file, and therefore unable to play it. Luckily, in most major file formats this does not seem to be the case.

Another drawback is that starting a new MediaPlayer can sometimes be delayed slightly, so it can in some scenarios cause a slight cut in the sound. In most cases this is unnoticeable, but if the device is busy or does a lot of computation, it may become more apparent.