Netcentric computing Personal rapport

Xavier, van Dommelen
11321458

January 2019

1 Start of project

We started with brainstorming about what we should do for the project. After thinking for a while we came up with a plan to analyse if it was possible to create and optimise a viable Bluetooth ad-hoc network. The test this we decided on creating an Android application that would create the network, with the purpose of audio file sharing and playing.

After we came up with the idea we started working out some details like the project plan. After finishing the project we started doing some research on what software/programming languages and frameworks we could use. In the first place we started looking into Wi-Fi Direct. Wi-Fi Direct could be a good choice, since it has a larger transfer speed and a larger connection range. The faster connection speed would be an especially useful, since songs are most of the time a few Mega Bytes.

Since we planned to execute this idea as an application in Android we started looking into the software needed and especially in how to use Wi-Fi Direct for this use-case. After looking more into the documentation we found that when using Wi-Fi Direct each *client* device can only connect with a *server* device, only one connection for each client. This new knowledge made us switch gears and switch over to Bluetooth. The reason for this was that when using Wi-Fi Direct our network would be extremely unstable in case of a server disconnection, since all the clients would also drop out of the network and need to be reconnected to the network. Bluetooth doesn't have this problem and is able to support multiple connections for each device.

Now that Bluetooth was chosen we started looking into methods how to use Bluetooth to automatically connect with multiple devices that are using our application. We saw that we would be able to use UUID's for such a use case. After struggling for a while we had a working implementation of the *Hail Marry* algorithm. This let each device automatically connect with each device that had Bluetooth enabled and then try to set up a connection with our specified UUID.

The downside of this method is that each device that runs this algorithm will pair with each device, independent if the connection was successful. This would lead that our idea of using already paired devices as cache to connect already known devices faster would be impossible. Then we went back to looking if it was possible to request the UUID of a device before pairing the devices. This was possible, but it could only execute this checking procedure one device at a time. This checking algorithm was not as fast as we hoped, around one minute, but the caching worked and was fast. So we decided to stick with this method.

2 Network topology

Once it was possible to create multiple connections automatically we started working on the network structure. For the network structure we decided that each device, also known as a node, will know how the entire network structure (graph). The reason for this is that for the application to work we need to know what songs are available in the graph. To make this information available across all nodes we need to send such a packet containing the information surrounding the song and the origin of the node. This packet alone would cost network traffic and thus we decided to add the information surrounding the known graph, since this would prevent sending later on more packets to find out the network structure.

For the discovery of the network on establishing a connection we implemented two important steps, the first one being the handshake and a sequence step that floods the rest of the two remaining networks.
The handshake starts immediately after a connection establishes between two devices. During the handshake each of the devices will send its own complete graph to the other device. Each device will then first calculate the difference between its own graph and the newly received graph. This difference in nodes and edges will then be added to its own graph and the difference will then be flooded over the network, from each of the two devices.

This *flooded* packet will be marked as a flooded type and each time a device receives such a broadcast it will apply the corresponding update to its network. For the flooding to work each flood message will contain the source node MAC-address that started the flood and a unique number that increases each time a node starts another flood. Each node will then have a set of MAC-addresses / nodes, from which it received, the source node, and the corresponding counters that it has received from that source. On receiving such a flood message the device will check if the node with corresponding counter has already been received once. If it hasn't received that specific flood message, it will send this flood message to all its other peers so that the flood of the message continues.

For sending a packet from node A to node B we used the Dijkstra Algorithm.

We set all the edges to an equal weight, so that the algorithm will calculate a path with the least hops. We didn't use Dijkstra to only calculate the shortest path, but also to calculate if nodes and paths needed to be removed from the graph. Since our Dijkstra algorithm will calculate every shortest path to each reachable node we are able to take the difference from the current known nodes and see if nodes are missing and if so remove all corresponding nodes and edges.

Once a connection between two nodes breaks a flood message will be sent across the networks that the edge between those two devices needs to be deleted. Once a device receives such a message it will delete locally that edge and run Dijkstra to update all the shortest paths and to see if more edges and nodes need to be removed.

3 The music-album network

Now each device will know the layout of the network after it has established a connection with the network. In the first place the nodes only contained the MAC-addresses of each device, but now each node will also contain every song that every devices has. This information will contain the song name, the song artist, the song size in bytes and the path to the audio file in the corresponding device.

Now that we know every song every node has we are able to request a song. This will be done by sending a *request-song* message type to the corresponding node using Dijkstra to find the closest corresponding song. As explained earlier each node has already run Dijkstra to find every shortest path. This shortest paths can now be used to find out to which node this request song needs to be forwarded to. The forwarding of such a message will be done until it reaches its final location. Once this request has reached the node that has the song it will start sending the whole song to the node that originally has requested the song using the *send-song* message type. Once this song has reached it's final destination through the network the device will start playing the song.

4 Bluetooth MAC-address

In the previous chapters it was mentioned that the MAC-address was used for linking a node to a device. During the development of the application we encountered several problems where the device was not able to request its own Bluetooth MAC-address, since google has blocked this due to privacy reasons. In some lower API levels we encountered that there were some workarounds to still get the devices MAC-address. The problem however was that starting from API-level 26 these options didn't work anymore. To solve this we implemented an additional step in the handshake. On connection with another device, the device will be able to see the other devices MAC-address. This behaviour was then used to see during the handshake if the other device didn't know what it's own MAC-address was in the received graph. When this would happen it will now overwrite the corresponding node, the node with the unknown MAC-address, with the correct MAC-address in its graph and send an update-graph message back to the other device. This other device will then overwrite its own MAC-address with the correct one and update its own graph.

5 Network monitor

Once the basic layout of the network was set and it was possible to request and send music a ad-hoc network monitor was implemented. This monitor would make it possible to see how the network would look like in a graph, network traffic and CPU-usage. This data could be useful for later testing so it was decided to build it into the application.

6 Testing and improvements

After the implementation of the previously mentioned ad-hoc network monitor it was time to search for optimizations for the experiment. The first step of the research was to see if it was possible to create a viable Bluetooth ad-hoc network, which we encountered to be possible.

The current implementation had just one flaw, that was the time it took before playing an audio file over multiple hops. In the current state each device needs to receive the complete song before able to forward the song to the next node or play the song. It was encountered that the time before playing with only one hop was still acceptable, but over multiple hops this time would grow exponential. To solve this problem it was decided to look into the possibilities of streaming the audio file and play the song while it's still being received.

Before implementing the streaming method we encountered another possible optimisation. During the tests that would calculate the time that was needed for requesting a song until the moment that a song was *playable* we encountered some outliers. These outliers could be explained by the current Bluetooth connection protocol. This protocol would every two minutes try to discover devices, connect with paired devices, request the UUID of every discovered devices and then connect with a device that has a matching UUID. In this process especially the requesting of UUID's was very bandwidth heavy. If a device is not sending or receiving any data this doesn't form a large problem, but it becomes noticeable once a large amount of data is send, like a song, when this protocol is running. To solve this an idle protocol was implemented. This protocol would prevent the Bluetooth protocol of running once it was sending or receiving data.

7 Final words

A lot of time has passed so far and the end of the research project is approaching. The remaining time was used for further testing, patching the application and the final paper. In the past few weeks we were able to test if it was possible to create a Bluetooth ad-hoc network with the purpose of sharing audio files. We were able to experiment with several techniques, encounter downsides of some implementations and find several solutions to further optimise the network structure.