Universiteit van Amsterdam

Individual report Netcentric Computing

29 januari 2019

Student:
Wouter Loeve
11330198

Docent:
A. van Inge, E.H. Steffens

Cursus:
Netcentric Computing

1 Formulating the experiment

In the beginning we quickly agreed to build a network for transferring audio using wi-fi direct. We had the idea to stream the audio real time and synchronize it between devices. We thought it would be a good experiment to use an ad-hoc network structure with. However a similar project was done last year so we changed our plans a little bit to differentiate ourselves. We thought of a different test case with which you could share your music library with others in a self regulating ad-hoc network. In this network it would then be possible to request audio files.

2 Bluetooth versus Wi-fi direct

We thought it would be interesting to see how viable it was to send audio files from different devices in the network in terms of bandwidth, latency and robustness of the network. At the start we wanted to use wi-fi direct since it had the reputation of being faster and larger range than bluetooth. We changed this idea into a bluetooth network because by implementing an ad-hoc wi-fi direct network, one has to give up a lot of its advantages. Since you can't connect to multiple wi-fi networks at the same time by only using a single band. A kind of switching or time sharing would have to be implemented and latency would potentially increase. We justified the decision to switch to Bluetooth by choosing an Android application because it is natively possible in Bluetooth to create an ad-hoc network.

Bluetooth consumes less power than Wi-Fi but has the reputation to be slower and lower range. However it is also more interesting to develop for a mobile device because we believe wireless ad-hoc networking has a future in the mobile space. Nowadays, batteries are limiting factors in this space, so it would be more logical to settle with bluetooth as we also wanted to see the limit of what the technology can do. Streaming audio is relatively bandwidth heavy, so we suspected to be facing a lot of challenges to get this working properly on bluetooth.

3 Drafting the routing protocol

We started to work within two sub groups, the first group would work on building the bluetooth network while the second group would work on the discovering of audio files and the process of sending them.

When we started to see how the bluetooth protocol was implemented we started drafting a routing protocol. At first we thought of a system that would first find a route by broadcasting packets trough the network. In body of the packet, a list of nodes would be kept and when the packets arrived at the destination the shortest route would be taken. The destination device could then start streaming. This minimizes the amount of devices that would be under load both in terms of processing power as bandwidth to be able to handle more requests and reduce latency. We thought this would be good idea because we would only send small packets to discover routes and not save entire networks on a single device. It would also be a good way to find the fastest route, not only looking at the fewest nodes but also the fastest connection. It would also be a boon to the self-regulating aspect of the network, the network on the local device would not have to be updated every time a connection changed. The method would incur a latency penalty as we first had to send packets to the destination and then back to relay the fastest route. We thought that latency could take that penalty as bandwidth would be our greatest challenge.

We later found problems with our drafted protocol in the self-regulation of the network. If a connection or node would be terminated mid-transfer there could be problems in guaranteeing the full transfer of the message as a new route had to be found, again incurring that latency penalty. Ad-hoc networks are often moving dynamic networks while our protocol was better suited for a large network that would stay relatively static. We then conceded the idea of not saving the entire network on a device as it would be more logical to find the shortest path via Dijkstra's algorithm in order to keep latency low.

4 Connecting devices

By using the Bluetooth API for android we encountered some implementation problems. Connecting a single device using aforementioned API is straightforward and well documented. Automatically connecting Bluetooth devices to form a network is an entirely different story. First the device needs to find the Bluetooth-enabled device that runs the same service and then an ID needs to be matched. This UUID is supposed to be unique for each application using Bluetooth and it can be chosen by the developers. It is supposed to act as a security measure. The function that would match the device's UUID against the device it attempts to connect with didn't work for us. It returned values that were not well described in the API guides. Strangely enough it did work when attempting a full pair with the device and then matching the UUIDs. Using this to connect with all devices was our first implementation of a Service Discovery Protocol (SDP) for Android. However it was painfully slow and paired with every device it could find, so we continued to look for other options. This took some time since there weren't many useful guides or answers to be found on the web. Finally we managed to get it working by attempting to match the UUID of a single target device at a time. Apparently it was not possible to concurrently connect with multiple devices at once.

The SDP which was running serially and did not fully pair with every device was still slow, to counteract this we build in a cache which uses the paired devices saved in your android device to build up a recurring network more quickly.

Later we found another optimization in the SDP. When searching for nearby bluetooth devices, we noticed that the bandwidth of that device would be lower than when it was not searching for nearby devices. We build in an idle-mode which stops the discovery of other devices when it sending or receiving packets over the network. The device still stays discover-able to be able to establish a connection with devices that seek access to the ad-hoc network.

5 Implementing the routing protocol

We encountered some minor concurrency problems on the device when the graph of the network would be updated using multiple messages at once. We solved this by using mutex locks and merging graphs instead of replacing them. To summarize, when a node would lose his connection with another node he would flood it over the network saying it was no longer available, the same would be done in case of newer nodes. The music library of said node would be send with this message so the library on every device would be up to date.

To route messages in a multi-hop way we send over objects of the P2P message class which contain fields similar to other message protocols such a TDP: source, destination, payload and also the message type. The types we have are handshake_network, update_network_structure, request_song, send_song and song_finished. The general way to request as song goes as follows: a device requests a song another node has in his play list (which is saved with the graph as is mentioned before). The sender then computes the shortest path using Dijkstra's algorithm to the nodes which have the music and takes the shortest one. In the third week we also started work on streaming music instead of sending the entire file and playing when it is done to reduce latency between requesting and playing.

Another problem we encountered is that on higher API level you cannot fetch your own MAC-address which is necessary to route packets as the source and destination fields use MAC-addresses. When a device connects to the network, he broadcasts his own node information including his music library through the network. What we found interesting is that you cannot find your own MAC address as an app but you can find that of others using bluetooth. We solved this problem by having your first connection send your own MAC-address to you.

6 Optimization's

When testing we noticed that when one device was streaming audio to another, no other messages would come trough, including network updates. To counteract this, we implemented a priority level to messages. For example, network updates have priority over an audio stream, causing the network message to be delivered to the destination in between an audio stream.

At first we implemented the streaming of audio files by loading the entire song into a byte array, dividing it into chunks and then sending it over the network one by one. We improved this by using an InputStream. With the InputStream it is possible to divide the byte array without having it fully in (main) memory. This should especially be helpful with large files. On the target device, whenever a new packet arrived it was loaded into a temporary file so it could be read from the built in Android MediaPlayer. These media players could be natively chained to create a stream. So we could play the song in its entirety without having it in main memory at any point as temporary files are not stored in main memory but written to the internal storage.

7 Testing

In the third week we started implementing some tests to see how our network performed. First we made sure the provided ad-hoc monitor worked but we thought that that was not enough for a scientific paper so we implemented some other measurements. One of these measurements is the time it takes before the music starts playing starting at the time of request. We wanted to test this metric with and without our optimisations as regards to the streaming of music, sending the full file and start playing afterward and streaming the music but only start playing when the full file has been transferred. This way we can test the benefit of streaming audio with as few changing variables as possible.

Another test is a bluetooth bandwidth stress test. By doing this, we can test the performance of our application as well as see a maximum capability of Bluetooth's bandwidth in regards to future works in ad-hoc networking.

8 Discussion

To conclude this report I want to discuss things that can be improved, optimized and tested more thoroughly. The tests we conducted: the stress test, comparison of streaming and non-streaming were not fully comprehensive as we tested in an environment which is open to interference of other bluetooth devices. This together with the different types of devices (ideally we would have liked to test with homogeneous devices) we tested with make for an experiment which is hard to compare against other, similar projects.

In section 3 of this report I explained that we had two versions of our routing protocol. The first version was rejected when we thought it compromised the latency to a large degree. This is purely theoretical and has no basis in empirical research. In future research we would like to test this alternative implementation along with other routing protocols to find the most optimal one for each situation. The networks we tested with where relatively small compared to immense networks such as the internet. If we get the chance to test alternate networking protocols, it makes sense to test them on a wide variety of configurations and scales, appealing to both lab-examples and real life scenario's. However this research is out of the scope of this proof of concept.

In order to improve performance when streaming multiple streams of the same song to multiple devices we would like to build in a cache of some sorts, so the load can be distributed over multiple devices.

The size of aforementioned audio streaming packets is arbitrarily chosen as 1/4th of a Megabyte. We would like to conduct testing to find the optimal size of this parameter. Having it too small creates a lot of overhead for the devices. Having the parameter set too large results in taking a long time before the packets arrive and start playing.
We believe it is also possible to build a feature to monitor the transfer speed and packet handling speed of the individual nodes and links between nodes. The information gathered from this monitoring could be used to update the weights of Dijkstra's algorithm. Things this information may encompass could range from CPU power of the device to the performance of the bluetooth adapter or even network load. This may result in a better network performance.

We are sure there are other improvements and optimization's possible to improve the performance of this application but these are the large ones we discussed over time when building this application.