Ad hoc networking on mobile devices: an experimental study on synchronised audio playback

Jesse van den Ende
University of Amsterdam
11292954

Engel Hamer
University of Amsterdam
11256443

Lars van Hijfte
University of Amsterdam
11291680

Dennis Wind
University of Amsterdam
11299770

***Abstract*— With more and more handheld devices in use nowadays, the potential of a *mobile ad hoc network* increases. This paper aims to investigate the possibility of creating such a network that relies on a 'bring your own infrastructure' principle. In particular, we consider an implementation on Android devices. We emphasise several limitations of the platform. Finally, we propose a network structure using the Google Nearby Connections API that succeeds at constructing a network automatically, as well as sharing audio files among connected devices. A downside of this is that no specific connection medium can be chosen: Bluetooth or Wi-Fi Direct will be chosen by the API itself in a best-effort manner, depending on the connectivity.**

I. INTRODUCTION

Modern day computer networks rely strongly on *internet service providers* (ISPs) to facilitate a connection between end users. They charge a fee for their services, which all connected users have to pay. With the number of internet users surpassing 4 billion in 2018 [11], the influential power of these parties has become very large.

In this paper, we will investigate the possibility of creating a mobile peer to peer network without the interposition of any ISP. This will rely on a 'bring your own infrastructure' principle: the users themselves are responsible for providing connectivity. We will then experiment with this network for a specific use case: sharing and synchronised playback of audio files across multiple devices.

II. WIRELESS AD HOC NETWORKING

The network as outlined in the introduction is also known as an *ad hoc network* or *WANET* (wireless ad hoc network). The ad hoc property refers to the emergence of the network: no pre-existing network infrastructure is necessary. All connected devices (or: nodes) in the network can serve both as an end user as well as a router. The routing algorithm in use combined with the connectivity of the communicating parties dynamically determines the forwarding of data packets.

Naturally, an ad hoc network requires a significant density of its users in order to provide sufficient coverage. If the distance between two nodes becomes too large, the connection between them will break. In the case of insufficient redundant connections, this could likely result in the network becoming separated. Therefore it would be beneficial for the reliability of the network to have many nodes, and evenly distribute them around the coverage area. As the density of nodes increases, ad hoc networks should become increasingly reliable. However, this network might not have any control over the positioning of nodes. This would violate one of the main benefits of ad hoc networking, namely the freedom for nodes to dynamically reposition themselves. This information should be taken into consideration when deciding on the platform used to implement the WANET on.

Personal computers for once would be expected to result in a not very reliable network. They are often used only indoors and turned off when they are not actively in use. Additionally, many desktop computers are connected via ethernet and do not have a wireless adapter. All in all, this would make it difficult for a network to span across different towns or even blocks since no intermittent nodes are present to forward data packets across. Mobile devices, on the other hand, are carried around everywhere by their users. Besides, they are almost always on (operating in stand-by mode) which allows for a more persistent connection. With more and more smartphones in use nowadays (and over 1.42 billion sales in 2018 [9]), the construction of a MANET (*mobile* ad hoc network) becomes increasingly feasible. Whether it be indoors or out on the street, it is highly likely that there is some other device nearby to forward your data through.

In the following sections we will examine four different methods for constructing an ad hoc network: via *Bluetooth*, *Wi-Fi Ad Hoc*, *Wi-Fi Aware* and *Wi-Fi Direct*. For the sake of feasibility, we will focus on the usage of a homogeneous system, consisting only of mobile devices. In particular, we will perform a case study on the most widely used mobile platform of this moment: Android. According to its manufacturer Google, the platform has over 2 billion daily active users [7], and over 86.8% of handheld devices run Android [9]. For this reason, we have chosen to investigate the implementation of our ad hoc network on this operating system. Additionally, the android developer documentation is very extensive and contains chapters on both Bluetooth and Wi-Fi [4][24].

III. BLUETOOTH

Bluetooth is a widely used standard for pairing devices wirelessly, without the need for an internet connection. Almost every handheld device is equipped with a Bluetooth receiver nowadays, and therefore this standard has a lot of potential for short-range ad hoc networking. The chances that a device is compatible with the network are high.

A. Speed, power consumption and connection range

Most Bluetooth devices on the market are designed to replace short cables. The typical power consumption of such devices usually lies between 1mW and 10 mW [6]. Connections are maintained on a 2.4GHz frequency band, which is also crowded with home appliances such as microwaves. Bluetooth, therefore, suffers from interference by many sources. Combined with the low energy consumption this imposes severe limitations on the connection speed.

The latest Bluetooth standards, from Bluetooth 3.0 onwards, promise theoretical data transfer rates of 24 Mbit/s maximum [16]. The theoretical range of a Bluetooth connection tops around 100 meters, but in practice file sharing is only possible when communicating parties are within 5 to 10 meters distance from one another. Additionally, the closer a receiver is to the Bluetooth transmitter, the higher the file transfer rate will be.

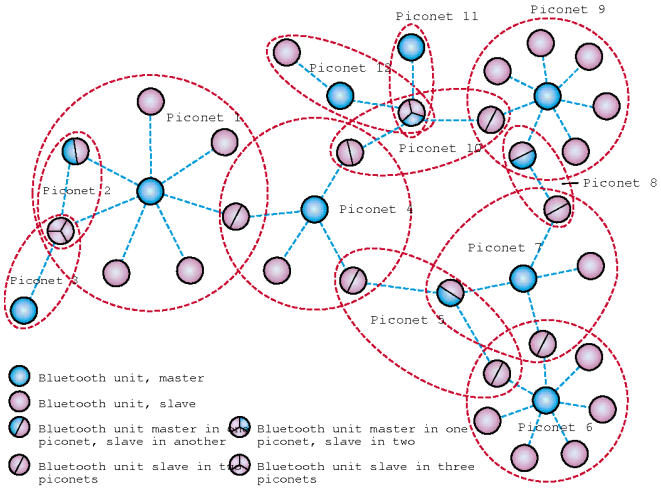

Fig. 1. Bluetooth devices in a scatternet formation. [12]

B. Connection structure

In addition to the radio interface, Bluetooth also defines a communication protocol. This includes peer discovery as well as advertisement of services. All connections are operated in a *master-slave* structure. Up to seven slaves can be connected to a single master, creating a host-client network also known as a *piconet*. A slave can be a part of multiple piconets, thereby creating a *scatternet* as depicted in Figure 1 [12]. This kind of network supports multihop wireless traffic. A Bluetooth scatternet can be used as an ad hoc network: there is no requirement for any pre-existing infrastructure. On top of that, a connection between any two nodes can be made through others. This is because nodes can also forward data packets addressed to other nodes in the network.

IV. WI-FI

Wi-Fi is a technology for wireless local area networking, listed in the IEEE 802 family of protocols as IEEE 802.11. It is widely used both in home as well as enterprise networks. Like Bluetooth, every modern smartphone has a Wi-Fi transmitter built-in. Unlike Bluetooth, Wi-Fi is used mostly in the context of *local area networking* (LAN). Modern routers often have a *Wi-Fi access point* (AP) built-in to facilitate a wireless internet connection to their users.

A. Speed, power consumption and connection range

The current official standard for Wi-Fi is IEEE 802.11n [10]. This version operates on two frequency bands: 2.4GHz and 5GHz. The first of these is shared with Bluetooth and many other devices. The latter on the other hand is only available upon licensing, thereby suffering much less from interference. Additionally, the 5GHz band offers a higher data rate and usually has fewer devices using this frequency. This does come at the cost of higher power consumption: Wi-Fi's typical output power ranges between 30mW to 100 mW [6].

According to the IEEE 802.11n specification, Wi-Fi should provide theoretical data rates of at most 300Mbps on 2.4GHz and even 900Mbps on 5GHz bands. However, in practice speeds peak at 150Mbps and 450Mbps respectively [19].

B. Connection structure

There are three possible approaches to constructing an ad hoc peer to peer network network via Wi-Fi. By using *Wi-Fi Ad Hoc*, *Wi-Fi Neighbor Awareness Networking* or *Wi-Fi Direct* [18]:

- **Wi-Fi Ad Hoc** [10], defined in IEEE 802.11, offers an alternative to Wi-Fi's standard infrastructure mode. It allows for the creation of a network without a preexisting infrastructure. This is achieved by making each node responsible for both sending and retrieving as well as forwarding data. Furthermore, the network is passive. Because of this, participating devices do not have to create a connection. Every device that is advertising the same ID can be used as a node in the network.
- **Wi-Fi Neighbor Awareness Networking** [20], or *Wi-Fi Aware* for short, is a standard that uses a publish and subscription model. Every participating device can either publish to a service or subscribe to one. Publishing to a service will send data to all devices that are subscribed to it.
- **Wi-Fi Direct** [23], or Wi-Fi P2P, allows for the construction of a peer to peer network by letting a device itself act as an access point. Other devices can connect to the network through a handshake protocol. This only requires acceptation of a connection request via a prompt. Alternatively, devices can connect to the network as if it were any generic Wi-Fi access point. This method is referred to as *legacy mode* [22].

V. CHOOSING A MEDIUM

When deciding on which type of medium should be used for the network, we take into consideration the properties of Bluetooth and Wi-Fi as outlined in the previous sections.

Connection speed, power consumption and connection range are among the most important factors here:

- Depending on the bit rate of the encoding, audio files can grow to significant size. An average mp3 file of 3 minutes long at 128kbps takes up 2.88MB. In order to allow users to share and forward files of this size, the **connection speed** should be sufficiently high.
- Since mobile devices have only relatively small batteries, **power consumption** should be minimised.
- To provide a reliable connection the chances of disconnecting from the network should be minimised. A large **connection range** should result in increased reliability here.

Bluetooth provides significantly lower data transfer rates than Wi-Fi. The theoretical difference is by factor 35. Additionally, the typical range for Bluetooth connections is approximately ten times lower than for Wi-Fi [6]. This does come at the cost of increased power usage. However, increased power usage does not obstruct the functionality of the application. The battery drain per byte sent is better for Wi-Fi than for Bluetooth [13]. Furthermore the benefits of Wi-Fi are much in favour of network reliability here. Therefore Wi-Fi will be the medium of choice for the implementation network.

Though it has been promised to be implemented ever since 2008, Wi-Fi Ad Hoc mode is not currently supported on Android devices [17]. The functionality can only be accessed by *rooting* the phone and changing the kernel [8]. This operation voids the warranty of most devices. Furthermore, it could result in permanent damage (or '*bricking*' of the device). We have therefore decided not to proceed with this method. Wi-Fi Aware is supported by Android, however only from version 8.0 onwards [21]. As of this writing only 21.5% of all devices meet this requirement [2]. Actual compatibility shares are expected to be even lower, since many manufacturers have not enabled this feature. For this reason we will not take this into further consideration, leaving Wi-Fi Direct as the only remaining mode for further research.

VI. WI-FI DIRECT

Wi-Fi Direct uses a *single-host, many-clients* structure. This means that every node can only be either a host or a client, but not both. Additionally, a client is limited to maintain only one connection to a single host. There can only be one active link between any two nodes, and all network traffic will go through the host node. This single point of failure makes the network rather vulnerable: it completely collapses once the host disconnects. In the rest of this section we propose five different strategies to work around this problem and construct an ad hoc network using Wi-Fi Direct. Two of these use a standard Wi-Fi Direct connection. The other three use the legacy connection mode.

A. Standard connection

1) Sequential connections: The first proposal is to sequentially establish a connection with all neighbouring nodes a node can reach. After receiving a message through this connection, the neighbouring node will disconnect itself and forward the message to all its own neighbours using the same method. This capitalises the quick reconnect functionality between two previously paired nodes.

To discover the neighbours of a certain node in the network, peer to peer discovery can be used (`WifiP2pManager.discoverPeers` [26]). Once the neighbours have been found, a node can proceed to connect with another node using `WifiP2pManager.connect` [26]. This strategy exposes a peculiar quirk of Wi-Fi Direct as it involves disconnecting from a peer. According to its documentation `WifiP2pManager.cancelconnect` [26] should be used for disconnecting, but during our experiments we noticed that the connection persisted upon calling this method. Fortunately, `WifiP2pManager.removeGroup` [26] disconnects from the current group without removal, contrary to what the documentation states.

The problem we found with this strategy is that the peer list of a node is lost after disconnecting. This significantly increases setup time, as finding peers took the most time in this setup. Furthermore, this strategy also turned out to be most unreliable: at times, no peers could be found. The *Oneplus 6* was particularly incapable of this, never having found any peers at all. For these reasons, research into this strategy was discontinued.

2) Forcing group ownership: This second proposal is an alteration on the first. It forces the sender of a message to become the group owner, by first creating a group using `WifiP2pManager.createGroup` [26]. This creates a group to which all neighbouring nodes can be connected. The unreliable peer discovery as mentioned in the previous subsection remains here, however discovery of neighbours now only needs to take place once per session.

A major problem with this strategy is that every newly created group displays a prompt on all receiving devices, asking permission to connect to it. Ever since the Android hidden API was blocked in 2018 [3], this prompt cannot be automatically accepted anymore without rooting the device. As a result, this strategy requires a user to manually accept a request prompt before connecting to other nodes. This prompt also needs to be answered before the connection times out. Due to the fact that it requires too much user interaction we cancelled further research into this strategy.

B. Legacy mode

Using legacy mode, we propose three more methods that all work by creating a Wi-Fi Direct group (`WifiP2pManager.createGroup` [26]). In order to let legacy devices access this group, each node treats such a group like a regular Wi-Fi access point with an SSID and password. These credentials are randomly generated. To communicate them to the other devices, a Wi-Fi Direct *local p2p service* is created with as instance name: "`SSID:password:ipAddress:port`". This can be found by the other devices using

`WifiP2pManager.discoverServices` [26], after which `WifiManager.enableNetwork` [25] is used to connect to that access point.

3) Broadcasting for every message: In the third proposal a node starts broadcasting once it wants to send a message. When a different node detects its service they can connect to the created access point. After establishing a connection the node receives any data the broadcaster wants to share.

4) Permanent broadcasting: The fourth proposal works by letting every node permanently broadcast their local p2p service. If a node wants to start sending data, it first stops broadcasting itself. After that the node sequentially connects and disconnects to all other services it detects. Once other nodes receive the data, they will forward it using this same mechanism.

Both the third and fourth proposal stranded on the very same problem. Like peer discovery, service discovery exhibits rather inconsistent behaviour. This is mostly noticeable when using devices from different manufacturers. These all have to implement the `WifiP2pServiceRequest` themselves, resulting in slightly different protocols. The *LG G4* for example can only find one service per `WifiP2pServiceRequest` [27], whereas the *Oneplus One* found multiple services per `WifiP2pServiceRequest` [27]. The *Samsung Galaxy S6* stops service discovery once it has found a single service, and will remain inactive until it is manually restarted. Another impediment found on the *LG G4* is that restarting active service discovery blocks it. We discontinued both proposals because these inconsistencies highly obstruct development.

5) Static self-repairing network: In an effort to create a workaround for the problems encountered in the previous four proposals, we invented a fifth strategy. It minimises the need for service discovery and works using a host-client model. At first each device starts looking for a service. If none is found after a given timeout a device will start its own service. It then becomes the host node of the network and starts broadcasting the service for others to discover. Once a service is found a device immediately connects to it. This creates a static network that can repair itself by restarting the construction strategy.

This strategy does have a drawback: the maximum physical size of the network is bound by the maximum connection range of the host. To bypass this limitation, a scatternet like in the Bluetooth example from Figure 1 could be implemented. This is theoretically possible since a node can be both a Wi-Fi Direct group owner (access point) and a client connected to an access point simultaneously. Nevertheless, this structure is not officially supported and might therefore be incompatible with some devices. Though the connections are possible, each Wi-Fi Direct group owner gets the same IP address when creating a group. As a result nodes would not be able to connect to the socket of their parent in a scatternet topology, since they share the same IP address. We therefore abandoned this strategy for constructing a scatternet as well. Since all of the proposed strategies were proven to be unfeasible using the native `WiFiP2pManager`, a different approach to solving the research problem is required.

VII. Google Nearby Connections API

The Google Nearby Connections API aims to offer peer to peer networking functionality focused on seamless connections [15]. Due to the fact that it is under development by Google, it can use functionality of the device's hardware that normal developers cannot. This resolves many of the problems described in the previous section. As an addition, users are not prompted to turn on Bluetooth or Wi-Fi. These features are automatically enabled as they are required, which allows for making connections without any user interaction. The API facilitates callbacks for advertising, discovery and connections between compatible devices in an offline peer to peer manner. The connections are established via different mediums under the hood: using Bluetooth, Bluetooth low energy and Wi-Fi Direct *legacy mode*. The choice of medium is handled by the API itself, depending on both connectivity and the topology of the network.

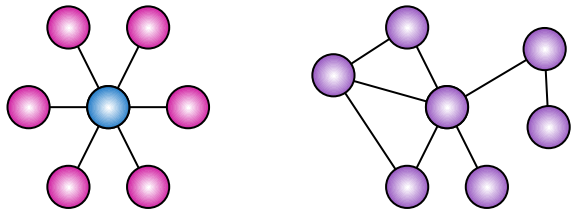

Fig. 2. `P2P_STAR` (left) and `P2P_CLUSTER` (right) topologies from Google Nearby Connections API.

As for the network topology, Nearby Connections supports two different strategies as depicted in Figure 2:

- `P2P_CLUSTER` allows nodes to maintain *M* outgoing and *N* incoming connections simultaneously, creating a cluster-shaped topology. It can be used to create a scatternet consisting of multiple clusters. This strategy only uses Bluetooth.
- `P2P_STAR` operates in a host-client manner. Every node in the network can either be a host (maintaining *N* incoming connections) or a client (maintaining a single outgoing connection). This strategy does not allow nodes to fulfil both roles at the same time. In contrast to the previous strategy, it does *sometimes* make use of Wi-Fi Direct. This is done based on the connectivity, in a best-effort fashion.

In a previous section Wi-Fi was chosen in favour of Bluetooth, mainly for its data transfer rates. Therefore, the `P2P_STAR` topology best suits the needs of the application.

A. Establishing the connection

In order to initiate a connection, every device running the application first starts advertising and discovering simultaneously. As soon as a device has established a connection with another device, both of them will stop discovering. Mostly because this is a rather costly process. Advertising on the other hand is relatively low on resources, and therefore

remains active at all times. This allows any new devices to connect to the network without any user interaction.

As soon as a device has lost its connection to the network, the `onDisconnected` callback of the connection manager is executed. This has been configured to re-enable peer discovery, and thereby automatically re-establishes a connection to the network like before. If the host loses the connection, the network will collapse. However, since every node follows the same procedure, negotiation on a new host will start immediately. This allows the network to rebuild itself.

B. Decomposing Google Nearby Connections

When conducting research using an API an important question to ask is what underlying standards are used. As mentioned before, the Google Nearby Connections API operates either on Bluetooth or Wi-Fi Direct. An interesting follow up question is how Wi-Fi Direct is implemented within the API in order to establish the network connection.

The fact that all but one device in a Nearby Connections network are connected to a Wi-Fi Direct group via an access point tells us that these connections are established using Wi-Fi Direct legacy mode. Unlike in the experiments with our proposed strategies using this mode, the random SSID and password of the Nearby Connections group were shared with other clients without creating a service. This means that they are shared by some other method. This can only be in one of two ways.

- **Credentials could be shared via Bluetooth**, since the Nearby Connections API can use both Wi-Fi and Bluetooth. However, this method is unlikely since the choice of the medium is made based on which provides the best connectivity at the moment. It would be illogical to still use Bluetooth after deciding to connect via Wi-Fi.
- The more likely option is that the SSID is found by the `WifiManager` [25] and **the password is set to a *static string***. This could be done by the Android hidden API, which the Google Nearby Connections API has access to whereas regular developers do not.

VIII. Implementation Experiment

In order to perform an analysis on mobile ad hoc networking possibilities, we developed an application that incorporates an ad hoc network. The aim of this application is sharing and simultaneous playback of audio files among peers. This is done by maintaining one synchronised playlist across the entire network, where the music files will be played in the same order. Every device can add songs to the playlist. Upon addition, an audio file will be forwarded across the whole network using the data transfer protocol defined below. Once a song is added to the shared playlist it will go into the priority queue where it will get precedence over previously played songs. After the first time played, it will be moved to the repeat queue. From there all songs will be repeated until the application is closed and every device is disconnected from the network.

A. Exchanging data over the network

Once the network is constructed, every connected device can share an *mp3* audio file. Preceding every file stream should be an application layer header containing metadata used by the application. This header is sent separately from the actual file, and contains the fields depicted in Table 1.

TABLE I
APPLICATION LAYER HEADER

Type	Name	Usage
`long`	`payloadId`	Identifier of upcoming file payload
`String`	`sender`	Sender device name
`long`	`timeStamp`	Moment of adding to queue
`String`	`fileSource`	Local file path
`String`	`title`	Song metadata
`String`	`artist`	Song metadata

The payload Id in the header is the value retrieved from converting the audio file into a file `Payload` by the Nearby Connections API. After this conversion, the file can be transmitted to the peers in the network. Algorithm 1 below illustrates this behaviour.

Algorithm 1 Queue synchronisation - Sender

```
1: procedure SEND(song)    ▷ Sends a song to all peers
2:     generate payload from song
3:     generate header from payload
4:     send header to all peers
5:     send payload to all peers
6: end procedure
```

It is important to note here that sending is handled by the API using the underlying Bluetooth or Wi-Fi Direct connection. Since the header and file itself are different types of `Payload` objects, the order of sending is not guaranteed [15]. Therefore, the receiver has to implement its own logic for combining headers and files. This requires two global mapped arrays, functioning like dictionaries:

- `files` mapping payload Ids to incomplete file payloads.
- `headers` mapping payload Ids to headers.

As soon as the sender starts sending a payload, the `onPayloadReceived` callback is triggered (denoted by `PR`) [15]. For headers, all data is transmitted at once. No updates follow. For files on the other hand, the `onPayloadTransferUpdate` callback is triggered after every chunk of data (denoted by `PU`) [15]. Algorithm 2 illustrates the behaviour of both callbacks.

B. Broadcasts

In order to share a file queue with every peer in the network, messages are sent as broadcasts. As soon as the host device receives both the header as well as the corresponding file from one of its peers, it will then forward them to all other devices except for the sender itself. This way, the queues of all connected devices will be the same. Due to the

Algorithm 2 Queue synchronisation - Receiver

```
1: procedure PR(sender Id, payload)                ▷ Initial
2:     if payload is a header then
3:         add to headers with payloadId as key
4:         if files contains payloadId key then
5:             combine header data and audio file
6:             add to queue, sorted by timeStamp
7:         end if
8:     end if
9:     if payload is a file then
10:        add to files with payloadId as key
11:    end if
12: end procedure
13:
14: procedure PU(sender Id, payload)               ▷ Update
15:    if payload completed successful then
16:        if headers contains payloadId key then
17:            combine header data and audio file
18:            add to queue, sorted by timeStamp
19:        end if
20:    end if
21: end procedure
```

star topology of the network, no routing loops can emerge by these broadcasts.

C. Time synchronisation

Once the network has been set up, the song which will be played first can be chosen by any connected device. After this device selects a song, it will start playing it straight away, and simultaneously share it among the network. The timestamp (in milliseconds) included in the song headers is used for maintaining the the playlist sorted in increasing order. Additionally, the first part of the song that was required for the file transmission itself can be skipped by computing the difference between the current time and the timestamp.

As the size of an ad hoc network increases, the time required to spread a file across it also increases linearly. This results in a delay which should be taken into account. After a certain size the required transfer time will be higher than the duration of the song itself. In such a case, the sending device will already proceed with playing the next song, disturbing the time synchronisation of the playlist. The application can successfully maintain a shared playlist across the network, but the exact ordering of it might shuffle for this reason.

IX. DISCUSSION

A. Conclusion

There are many available mediums for interconnecting devices directly without the need for a networking infrastructure. Some of these can be used for ad hoc networking on the Android operating system. Despite the fact that developers cannot gain sufficient control over the Wi-Fi adapters, it is still possible to construct an ad hoc network using Wi-Fi Direct. We proposed an implementation of an ad hoc network on mobile devices, using the Google Nearby Connections API. The resulting application succeeds at constructing a network automatically, as well as sharing files among connected devices. A downside of this, is that no specific connection medium can be chosen: Bluetooth or Wi-Fi Direct will be chosen by the API itself in a best-effort manner, depending on the connectivity.

B. Future work

As described in section VII, a device will automatically reconnect to the network once the connection is lost. However, since there is no fallback mechanism in place, all files transmitted during the downtime of a node cannot be recovered. This causes queues of reconnected peers to get out of sync. Future research on recovery of missed files could improve the reliability of the application.

To avoid queue shuffling issues between connected devices as described in the previous section, an alteration to the file sharing system could be made. Currently, an entire file is shared at once, requiring all devices to delay playing it until it has been fully received. Changing the sharing system into a *stream*, where the music is shared in smaller chunks meant for immediate playback, could minimise these synchronisation issues. This seems like a more user-friendly experience, rather than waiting for the entire file transmission before the music can start playing.

Additionally, minor clock differences between devices cause the playback of audio files to get noticably out of sync. If one desires more accurate simultaneous playback across multiple devices, *network clock synchronisation* could be implemented, using for example the *Network Time Protocol* (NTP) [5][14]. This ensures that all connected devices maintain the same calibrated time, usually differing no more than a few dozen milliseconds from the host.

As a replacement for the Google Nearby Connections API, which sometimes uses Bluetooth rather than the faster Wi-Fi Direct connection, the Wi-Fi Aware protocol could be researched further. This could greatly improve the robustness and flexibility of the network.

REFERENCES

[1] *AdHoc-Monitor*. URL: https://github.com/ERLKDev/AdHoc-Monitor.

[2] *Android distribution dashboard*. URL: https://developer.android.com/about/dashboards/.

[3] *Android Hidden API*. URL: https://developer.android.com/about/versions/pie/restrictions-non-sdk-interfaces.

[4] *Bluetooth overview — Android Developers*. URL: https://developer.android.com/guide/topics/connectivity/bluetooth.

[5] H. Cho et al. "Precision Time Synchronization Using IEEE 1588 for Wireless Sensor Networks". In: *2009 International Conference on Computational Science and Engineering*. Vol. 2. Aug. 2009, pp. 579–586. DOI: 10.1109/CSE.2009.264.

[6] E. Ferro and F. Potorti. “Bluetooth and Wi-Fi wireless protocols: a survey and a comparison”. In: *IEEE Wireless Communications* 12.1 (Feb. 2005), pp. 12–26. ISSN: 1536-1284. DOI: 10.1109/MWC.2005.1404569.

[7] Google. *Thanks to developers and our partners around the world, there are now more than 2 billion monthly active Android devices. io17*. May 2017. URL: https://twitter.com/Google/status/864890655906070529.

[8] “http://thinktube.com/index.php/tech-en/android/wifi-ibss”. In: ().

[9] *IDC - Smartphone Market Share - OS*. URL: https://www.idc.com/promo/smartphone-market-share/os.

[10] “IEEE Standard for Information technology–Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput”. In: *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)* (Oct. 2009), pp. 1–565. DOI: 10.1109/IEEESTD.2009.5307322.

[11] *Internet Usage Statistics 2018*. URL: https://www.internetworldstats.com/stats.htm.

[12] P. Johansson et al. “Bluetooth: an enabler for personal area networking”. In: *IEEE Network* 15.5 (Sept. 2001), pp. 28–37. ISSN: 0890-8044. DOI: 10.1109/65.953231.

[13] G. Kalic, I. Bojic, and M. Kusek. “Energy consumption in android phones when using wireless communication technologies”. In: *2012 Proceedings of the 35th International Convention MIPRO*. May 2012, pp. 754–759.

[14] H. Melvin and P. Corcoran. “Playback Synchronization Techniques for Networked Home Appliances”. In: *2007 Digest of Technical Papers International Conference on Consumer Electronics*. Jan. 2007, pp. 1–2. DOI: 10.1109/ICCE.2007.341436.

[15] *Nearby Connections API*. URL: https://developers.google.com/nearby/connections/overview.

[16] Bluetooth SIG. *Bluetooth Core Specification*. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification.

[17] *Support Wi-Fi ad hoc networking*. URL: https://issuetracker.google.com/issues/36904180.

[18] *Thali project*. URL: http://thaliproject.org/AndroidP2P/.

[19] *Which frequency should you use, 2.4Ghz or 5Ghz?* URL: https://www.centurylink.com/home/help/internet/wireless/which-frequency-should-you-use.html.

[20] *Wi-Fi Aware*. URL: https://www.wi-fi.org/discover-wi-fi/wi-fi-aware.

[21] *Wi-Fi Aware Android documentation*. URL: https://developer.android.com/guide/topics/connectivity/wifi-aware.

[22] *Wi-Fi Direct, documentation*. URL: https://developer.android.com/training/connect-devices-wirelessly/Wi-Fi-direct.

[23] *Wi-Fi gets personal: Groundbreaking Wi-Fi Direct launches today*. URL: https://www.wi-fi.org/news-events/newsroom/wi-fi-gets-personal-groundbreaking-wi-fi-direct-launches-today.

[24] *Wi-Fi peer-to-peer overview — Android Developers*. URL: https://developer.android.com/guide/topics/connectivity/wifip2p.

[25] *WifiManager*. URL: https://developer.android.com/reference/android/net/wifi/WifiManager.

[26] *WifiP2pManager*. URL: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html.

[27] *WifiP2pServiceRequest*. URL: https://developer.android.com/reference/android/net/wifi/p2p/nsd/WifiP2pServiceRequest.

APPENDIX

AD HOC MONITOR

In an attempt to perform more analysis on the network itself, we considered using ad hoc monitor software, like ERLKDev/AdHoc-Monitor [1]. Unfortunately these monitors were incompatible with the Google Nearby Connections network setup. This is due to the use of Wi-Fi Direct legacy connection mode: connected devices can not be connected to any different network in which the monitor resides. The host node could be monitored using this tool as it is not connected to a different Wi-Fi access point, but only monitoring a part of the network seems unhelpful. Especially considering the host node might disconnect and reconnected as a client. Creating a monitor for this type of network would provide a useful follow up project as this would allow better monitoring of a network like we propose in this paper.