Nearfly API – Conception and Implementation

Network-independent local peer-to-peer connections on mobile devices

Abstract

The present work deals with the conception and implementation of a library, which ensures the construction and communication of a local mobile network for Android-based systems regardless of the internet connection. The Nearby Connections API is selected as the underlying technology in the offline case and the MQTT API in the online case. The APIs are extended by adapters in order to be able to offer a uniform interface. For this purpose, a stateful, autonomously building star-shaped network is created using the Nearby Connections API, which allows MQTT-like connection establishment. For the data transfer of both APIs, the channel, priority, send buffer and an MQTT-side chunking mechanism for large data over 30 Kbytes are created.

Subsequently, the times required for establishing a connection as well as for sending data from both underlying technologies are measured and evaluated. The Nearby Connections when establishing the connection is poorly scaled compared to MQTT and already takes 7-11 seconds for the connection between two nodes. A network with six nodes takes about a minute, while MQTT can build the same network in under half a second. When transmitting data between two nodes, Nearby Connections has three times higher data rates than MQTT when the hotspot is active, while MQTT has better data rates with more than three participants. If, on the other hand, the hotspot is deactivated, data transfer is more than an order of magnitude slower.

Contents

1 Introduction

For a long time, smartphones no longer fulfill the primary purpose of making calls, but are mainly used for exchanging messages, browsing and other Internet-related activities [1]. On the other hand, there are also situations in which the exchange of information is desirable, but an internet connection is inappropriate or simply not possible. For example, if multimedia data is to be transferred decentrally for data protection reasons, no data volume is to be used up or urgent weather warnings are to be distributed in areas where the mobile network is (temporarily) unavailable due to a failure or a lack of infrastructure[3]. To implement these networking functionalities, existing technologies such as Bluetooth, Wifi-Direct or the Nearby Connections API published by Google in July 2017 are often used. While internet-based communication is possible through MQTT. The technologies are integrated into the project in the form of libraries that abstract complex issues and can be addressed through an API.

Within the scope of this work, the NearbyConnection and MQTT technology are to be analyzed and an Android middleware to be designed and implemented, which unifies the advantages of both underlying technologies in the form of a level based on both technologies and gives the Android app developer the option of a generic Use of the networked underlying offline and online solution (as in **Illustration 1**) offers.

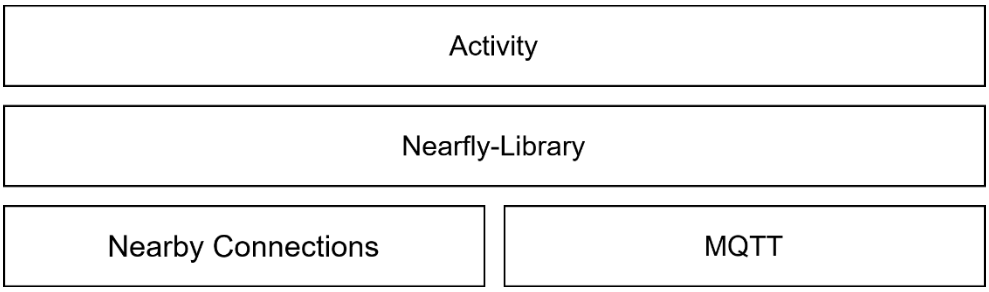


Illustration 1: Abstracted three-level representation of the Nearfly library

The Requirements analysis (Chapter 2) defines a messenger, a turn-based game and two direct-interactive games from which the requirements for the nearfly library are determined. The Nearfly library is supposed to transmit large multimedia data and small messages in near real time and with the lowest possible latency. The alignment of both underlying technologies requires that in Nearby Connections a strategy must first be selected for the network structure and the distribution of messages to all network participants, as is the case in MQTT.

In the chapter 3 (Software design), an MQTT and NearbyConnections adapter component is defined for extending the underlying technologies. It is also defined that the Nearfly library can be used by a NearflyClient object and that messages can be heard using the Nearfly listener. In the chapter4 (Connection establishment) it is shown that the Nearfly library requires seven mandatory and two optional authorizations, of which the authorizations for location and data access must be queried at runtime. By expanding the NearbyConnections API, a star-shaped network is implemented using a state machine. In chapter5 (Data transfer) defines two methods for sending data using the nearfly library. A distinction is made between data under 32 Kbytes and large binary data, which are broken down before transmission. Then the Evaluation (Chapter 6) that the establishment of a connection using Nearby Connections takes 7-11 seconds for the first node and does not scale well, while MQTT takes half a second. Nearby Connections, on the other hand, is faster when transmitted to small networks consisting of two nodes.

The Résumé shows that Nearby Connections is preferable for smaller networks consisting of two to three nodes, while MQTT should be used for larger networks. If there is no internet available, however, Nearby Connections can be used for up to seven nodes.

2 Requirements analysis

While the scenarios Messenger, Share Touchpoint Canvas, Bouncing Ball and Score Board Notepad use cases are defined in which different amounts of data have to be transferred, the functional requirements derive from the scenarios that the Nearfly library can prioritize small to large and only in the application context send visible messages. These should arrive with little delay, close to real time. If the connection is interrupted, the data must be held, and the connection established again.

The comparison of Nearby Connections and MQTT makes it clear that assimilating both technologies requires quite a large adaptation of both APIs. For example, MQTT connects to the server, while in Nearby Connections a network topology must first be selected for the network setup and a connection to each node must be established individually. MQTT also sends byte messages to all network participants, while byte, file and stream messages can be sent in Nearby Connections, but these initially only go to one node.

2.1 Scenarios and Features

The scenarios reveal the data rate from small to large amounts of data and long transmission times to near real time. While larger binary data are also transmitted with Messenger, touchpoints with the Shared Touchpoint Canvas must be sent very quickly (near real time). A practical extreme scenario for low latency is the bouncing ball, which requires a transmission frequency of at least 25 packets per participating user. As a turn-based game, the Score Board Notepad scenario must be fail-safe, the speed being in the range of seconds and only a small amount of data being transmitted. While the restricted-visibility message exchange feature ensures that the message exchange of different scenarios does not interfere.

2.1.1 Messenger

Marius is the Scrum Master of a five-person development team for mobile applications and is therefore responsible for the collaboration between the developers. He attaches great importance to data protection and would prefer not to transmit data over the Internet. To do this, he has created a group and asks his team members to join his group. Then he writes a greeting to everyone and asks if there are currently any problems. While Tommy is reading the message, Tommy runs to the toilet, which is a radio hole through the thick walls, and answers Marius that the coffee pads are empty. Shortly afterwards, Tommy sends a photo of the coffee pads, asking Marius to buy new ones.

The first scenario corresponds to that of a classic messenger app, which allows the development team to communicate offline. So, the user should be able to send text and multimedia. However, it is also conceivable to allow any binary data to be sent and thus, for example, to enable the data transfer of executable data. Furthermore, by creating and entering closed user groups (chat rooms), isolation within the app should be allowed, which reduces the visibility of dedicated messages to authorized users. In addition, sent messages should not be lost in the event of a disconnection, but should be kept until they can be sent.

table 1: Empirically determined average size range of multimedia data[1]

Data type	Average size in Mbytes	
	From	To
Documents (PDF)	0.1	1
Photos (JPG)	1 (4032x3024)	5 (4032x3024)
Screenshots (PNG)	0.4 - 3	3
Mobile Applications (APKS)	5	62.5

Since the development team mainly wants to transfer images or text via the smartphone, shows **table 1**the average data size of the multimedia data collected on three different smartphones. PDFs from one to a few pages in size have an average size of 60-800 KB, while photos or screenshots that are shot by smartphones sometimes depend heavily on the image detail and smartphone model and are between 1-4 MB in size. Since the team also develops Android applications and thus APKs have to be sent in the meantime as part of system tests, shows **table 1** continue the size range of Android applications[2]. The applications with 5 Mbytes[4] The Play Store has the fewest downloads on average, while most people have larger applications with an average of 62.5 Mbytes [4] download and own.

[1] The data, which are used to generate the values determined in the table, come from an LG G2 mini LTE, a Samsung Galaxy S7 Edge and a Samsung Galaxy S9. Exceptions are the values for the size range of the APKs.

[2] The average of all applications offered in the Play Store is 11.5 MB [4].

2.1.2 Shared touchpoint canvas

Tom, Beni and Jim sit in the office and draw together using their smartphones on a touchpoint-based virtual canvas. Since each user can set a maximum of 10 points of contact, which are quickly hidden again, Tom comes up with an idea. He asks his friends to draw a car with him. Each of the friends chooses an area of the car that they would like to draw, and Beni gives the start signal so that the points of contact are displayed as simultaneously as possible.

The collaborative scenario includes a canvas that can be painted by all users involved by tapping it. If a user touches the canvas, a colored touchpoint is created, which disappears after a certain time. Since everyone wants to draw together, the data must be received as soon as possible. The data packets are small and only contain the coordinates and the color of the contact points. All participants send their data and receive all data sent.

2.1.3 Bouncing ball

Mary, Robert and Frederik are in the classroom during the long break and try to balance a ball together by pivoting a virtual platform, which was randomly knocked at the beginning of the game. Since the behavior of the ball is the sum of the actions of all players, they all must compensate for the movements of the other players. After the ball has been successfully centered, it is subjected to another pulse. This is repeated a few times until the ball falls out of the platform after four minutes and the personal best time is displayed on all three devices.

In the third scenario, a lot of data is transferred within a very short time. To do this, the inclination data of all players must be recorded continuously, and the ball moved according to the sum of all inclination data recorded. At the same time, the game demands the lowest possible latency, so that the individual players can balance accordingly without reaction time and thereby guarantee a continuously synchronous gaming experience (equivalent to holding a physical object together).

2.1.4 Score board notepad

Steffan, Mark and Ricky play paper throwing together during their lunch break, and each round tries to hit the garbage can in the corner with ten crumpled paper balls. The one who was last is responsible for recording the points of the player who is currently on. Mark begins with the role of scribe while the other two players check whether Mark assigns the points correctly. Ricky starts and hits the garbage can. The ball hits the wall beforehand, so Mark only increments Ricky's score by one. Thereupon Ricky hits the garbage can without the ball touching anything and receives two points from Mark. Eight *more attempts later, Ricky's lap ends. Now it's Steffan's turn and his predecessor Ricky notes points. After 20 rounds,*

In turn-based games, data for controlling the game must be sent within dynamic time intervals. Both the reassignment of the scribe and the determination of the active player and the spectators must take place. The system must also ensure synchronization of the point table across all smartphones.

2.1.5 Exchange of messages with restricted visibility

Leonard, Mandy and Kevin are in the classroom and play the "bouncing ball" game during the long break, while the teachers in the next room prepare the upcoming lessons and send each other pictures and articles via the Nearfly Messenger app.

The feature for restricted message exchange shows that the data exchange between applications running in parallel is invisible to the other application. The transmissions of different devices should interfere as little as possible.

2.2 Functional and non-functional requirements

The established requirement matrix (table 2) shows the correlation of the requirements determined for the above scenarios. table 2 shows that the sending of text-based messages is required by all scenarios, e.g. for sending chat messages, status or coordinate data. In the Messenger scenario, this requirement is extended to include the sending of binary objects. Furthermore, due to the requested isolation for room and chat, the messenger scenario results in the requirement that different messages should be separable within the same application. In the event of an unintentional break in the connection, important messages, such as picture or text messages to be sent on the messenger or the points on the Score Board Notepad, should be kept and sent when the connection is re-established. Of course, this assumes that the connection is automatically re-established in the event of a connection failure. The games and animation scenarios also require near real-time latency and throughput, which allows smooth animation (at least 25 FPS) and direct-interactive control. In the bouncing ball scenario, important control data (such as game over and start game) must always be sent before the predominant coordinate data. Furthermore, there is a requirement for all scenarios that the exchange of messages between different applications that use the Nearfly library must not interfere.

Finally, both when using the Nearby Connections API and when using the MQTT API, the system should behave similarly in terms of connection establishment, sending and receiving of messages, so that both underlying technologies can be used for the same scenarios.

table 2: Requirement matrix of the nearfly library

Functional requirements	Messenger	Shared touchpoint canvas	Bouncing ball	Score board notepad
Send text-based messages	x	x	x	x
Send multimedia messages	x			
Separability of different messages within the same app	x			
Hold messages when disconnected	x			
Automatic connection establishment	x			
Near real-time (latency and throughput) for a smooth multiplayer game		x	x	
Prioritization of the message types			x	
Visibility of messages only within the same app	x	x	x	x
API of the underlying technologies is identical	x	x	x	x

Furthermore, a non-functional requirement stipulates that the library to be developed must run on all Android versions from API level 24 (Android 7.0).

2.3 Comparison of Nearby Connections and MQTT

Since the API should run with Nearby Connections as well as with MQTT, both technologies have to be compared in terms of usage and behavior. Nearby Connections[6] is described by the Nearby Connection Team as a peer-to-peer API with high bandwidth and low latency, which continuously offers encrypted and therefore secure data transfer between the connected nodes [6]. Connections are established using Bluetooth, Bluetooth Low Energy (BLE) and automatically created WiFi hotspot and thus ensure an almost completely automatic offline network. In order to make this possible, the API has the authorization to switch on both WiFi and Bluetooth and to return them to their initial state when exiting.

Before the actual connection is established, a network topology must be selected which also affects the maximum transmission rate in the network. Nearby Connections offers three different topologies (called strategies), as described in **Illustration 2**shown. The first is called the CLUSTER strategy (**Illustration 2**a) and allows any node in the network to accept any connection. The resulting NM topology only supports Bluetooth and only allows a small bandwidth. It should also be noted that the realistic maximum number of simultaneously connected Bluetooth devices is limited to 3-4[5]. The STAR strategy (**Illustration 2**b) on the other hand, a 1-N topology is made possible, which can use a WiFi hotspot in addition to Bluetooth and thus enables significantly more bandwidth. As soon as a network has been established, the API tries to upgrade the network to a WiFi hotspot[7]. The network can then accommodate up to 7 nodes. Finally there is the POINT_TO_POINT-Strategy (**Illustration 2**c), the behavior of which corresponds to the STAR strategy, with the difference that each node is only allowed one connection node.

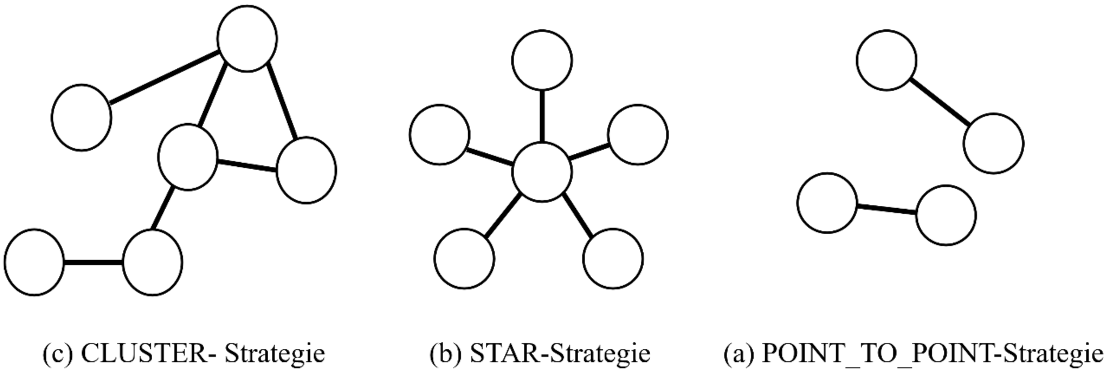


Illustration 2: Available network topologies (strategies) of the Nearby Conenctions API

The Nearby Connection API uses an advertising / discovery method to enable the devices to be recognized. Advertising is a passive state in which a device can be found, while discovering initiates an active search for advertisers, which significantly increases battery consumption. A device can assume either or both states at the same time[6]but this can lead to trashing [8th]. If a device in the discovering state sends a connection request, which is confirmed by the discoverer and advertiser, a symmetrical connection is established (see **Illustration 3**) initialized between both nodes. Authentication is not implemented by default, but can be implemented subsequently by the API user, for example as a popup request, which states the endpoint name of the device and only instantiates the connection after successful confirmation. If at least two devices are connected accordingly **Illustration 3**data can be exchanged by specifying a recipient in the form of an EndpointName. The data exchange is always encrypted[6]and is divided into 3 message types. On the one hand, these are byte messages, which are 32 KByte are limited and on the other hand the two unlimited data types file and stream[6]. If data is sent, the sender is informed of the output of the data fragments, while the receiver is informed of the receipt of the data fragments. The connection can be terminated by a disconnect.

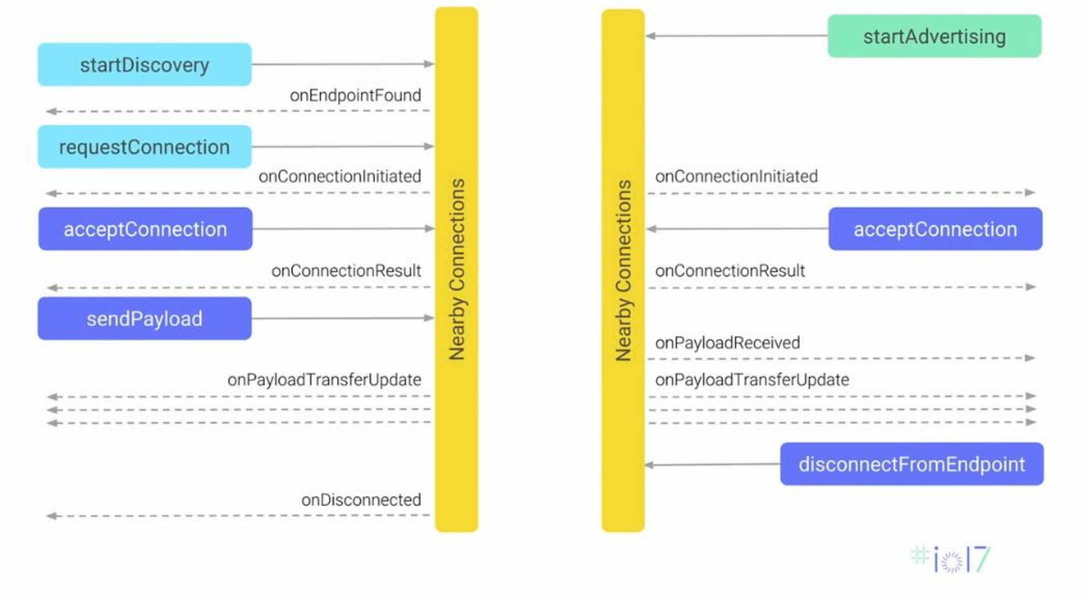


Illustration 3: Nearby Connections API message history [9]

MQTT[10] however, in the OASIS specification [10]described as a lightweight, easy-to-use client server pubish / subscribe message protocol that requires little bandwidth and leaves a small code footprint. In contrast to Nearby Connections, MQTT in practice requires a TCP connection to the server (broker)[11], which managed the connections and messaging and decoupled the clients from each other so that they are never directly connected to each other. The MQTT specification also allows other protocols that allow an orderly, lossless, bi-directional connection[10].

Once a client has established the connection to the broker, it can publish messages and subscribe to certain topics (e.g. "test / topic") [11] A topic means a UTF-8 encoded string which can be subdivided hierarchically by a slash ('/' U + 002F), similar to a tree structure, e.g. "tree / branch" including "tree / branch / leaf"[10]. The messages must always be a topic, as well as a byte payload under 256 Mbytes[12] so that the broker can forward the messages to subscribed clients.

The MQTT specification does not provide for any encryption of the messages in order to keep the protocol as simple as possible [10], however, this is supported in many implementations, as is the Paho client library to be used.

In addition, messages receive a Quality-Of-Service (QoS), which can be used to set whether a message should be received at most once (zero), at least once (one) or exactly once (two) [10]. Higher QoS levels result in more overhead. And a retain flag, which ensures that the server saves the last message and then sends it to all subscribers of the associated topic if the subscription is successful[10].

table 3: Overview of differences between Nearby Connections and MQTT [6]

technology	MQTT	Nearby Connections
Network topology	Server / client	Cluster, star or point-to-point
connection	Persistent to the server	Persistent to the individual peers
Protocol used	TCP / IP	Bluetooth, BLE, WiFi hotspot
Data rate (theoretically)	WiFi: max. 250 Mbit / s LTE: max. 300 Mbit / s	Bluetooth 3.0: max. 24 Mbit / s WiFi: max. 250 Mbit / s
latency	32-41 ms	Bluetooth: 6.47 ms WiFi: 14-22 ms
Exchange of messages	Publish (1: N)	Bidirectional one-way (1: 1)
Authentication	Name and password field	Connection request to user (Optionally implementable)
Message encryption	TLS possible	continuous
Payload type	byte	Byte, stream or file
addressee	Topic subscriber	EndpointName
Max packet size	Maximum of 256 MB (adjustable by the broker)	Bytes: 32 Kbytes Files & Stream: unlimited

The comparison of the two technologies with regard to the data rate also makes sense, since Nearby Connections, unlike MQTT, does not always use an Internet protocol, but as a hybrid technology initially communicates via Bluetooth and only changes when the opportunity arises. Basically, all chips from Bluetooth version 3.0 create an additional high-speed channel through a combination of Bluetooth and WLAN transmission technology, which allows a theoretical maximum transmission speed of 24 Mbit / s, e.g. for transmitting photos, videos or music[13]. From version 4.0, Bluetooth Low Energy (BLE) is also integrated in the chips, which consumes significantly less power, but only has a maximum of 220 Kbit / s and a limited range of 10 meters[14]. With version 4.2 the BLE data rate is increased by a factor of 2.5 compared to the previous version and the transmission is more secure, while with Bluetooth 5.0[15] Depending on the requirements, a doubled gross throughput of 2 Mbit / s or a range of up to 200 m can theoretically be achieved with BLE.

However, the actual data rate is up, like Evan Fallis and Petros Spachos[16]by testing four different smartphones with Bluetooth 4.0 and 4.1 shows with 1 Mbit / s far below the theoretical figures. Furthermore, the measurements of these two show that the WiFi transmission via smartphone with an average data rate of 28.95 Mbit / s and extremes of 25.7 Mbit / s, as well as 37.8 Mbit / s in the client-side transfer of TCP packets[16], the theoretical specification of up to 250 Mbit / s is far from. If, on the other hand, a smartphone is in the LTE network, it achieves an average of 42 Mbit / s by using 4G LTE Advanced Technology[17] (with a theoretical specification of 300 Mbit / s).

While the theoretical information can only be achieved in a controlled laboratory environment, the actual data rates show the range in which the values should move during transmission using the underlying technologies. When determining the actual data rates, the drastic difference of factor 29 (1Mbit / s to 28.95Mbit / s) between Bluetooth and WiFi is noticeable. If a transmission is transmitted without an activated WiFi hotspot, it should be noticeably slower.

With regard to the latencies, a small Java app, an MQTT client and a set QoS of one can measure an average latency of 32-41 ms, while for a QoS of zero the latency is half as high. A Ping Tool Android app shows that the WiFi latency between a Samsung S9 and a local Tomcat server is on average in the range of 14 to 22 ms. It should be noted that MQTT itself has little latency with a QoS of zero and most of the latencies are caused by WiFi. On the other hand, Shah, Nachman and Wan measured the latency of Bluetooth when transmitting 10-byte data packets to around 6.47 ms[18]at a distance of 1 m between transmitter and receiver. Here, too, the measured values allow a first impression of the various latencies of the technologies used and show which latencies should at least be expected.

3 Software design

The Nearfly library should be accessible via the Nearfly API, which can be used by instantiating the Nearfly client. In order to react to certain events on the part of the Nearfly library, such as the receipt of messages or status changes (e.g. connected / disconnected), any observer (called NearflyListener) should be able to be registered and deregistered with the NearflyClient.

The NearflyClient (as in Illustration 4) the underlying NearbyConnections and MQTT APIs should be used via dedicated adapters, which forward the requests to the respective APIs by delegation and the NearflyClientTarget (as in Illustration 4) implement, address. The NearflyClientTarget interface defines the methods, such as sending messages, which the NearflyClient needs.

The functionalities based on the Nearby Connections (NeCon) API should be in two different components, as in Illustration 4 shown to be outsourced. While the NeConEssentials component offers the basic functionalities for a correctly functioning Nearby Connections API, such as setting the endpoints in the advertising / discovering state or managing discovered, accepted and pending connections. Should the NeConAdapter component mainly offer the functionalities that are specifically required for the Nearfly library. This includes the autonomous construction of a local peer-to-peer network from the nearby nodes, as well as the functionalities necessary for data transmission. Between the NeConAdapter and the NeConEssentials component there should be a dependency of a base / child class.

In order to react to incoming messages, the NeConEssentials component should continue to have a dependency on the PayloadReceiver component, which is an extension of the primitive PayloadCallback listener offered by Nearby Connections API.

The use of the MQTT API should also be distributed over two components. While the MqttMessaging component is the one that ensures the functioning of MQTT, the functions required for the Nearfly library, such as the sending of multimedia, are implemented in the MqttAdapter component. Incoming messages, occurring errors and changed connection status are communicated by the MqttMessaging component to the MqttAdapter component via the respective listener.

According to the communication between the Nearfly library and the activity, the adapter classes should signal the receipt of messages or status changes to the NearflyClient by using dedicated interface components.

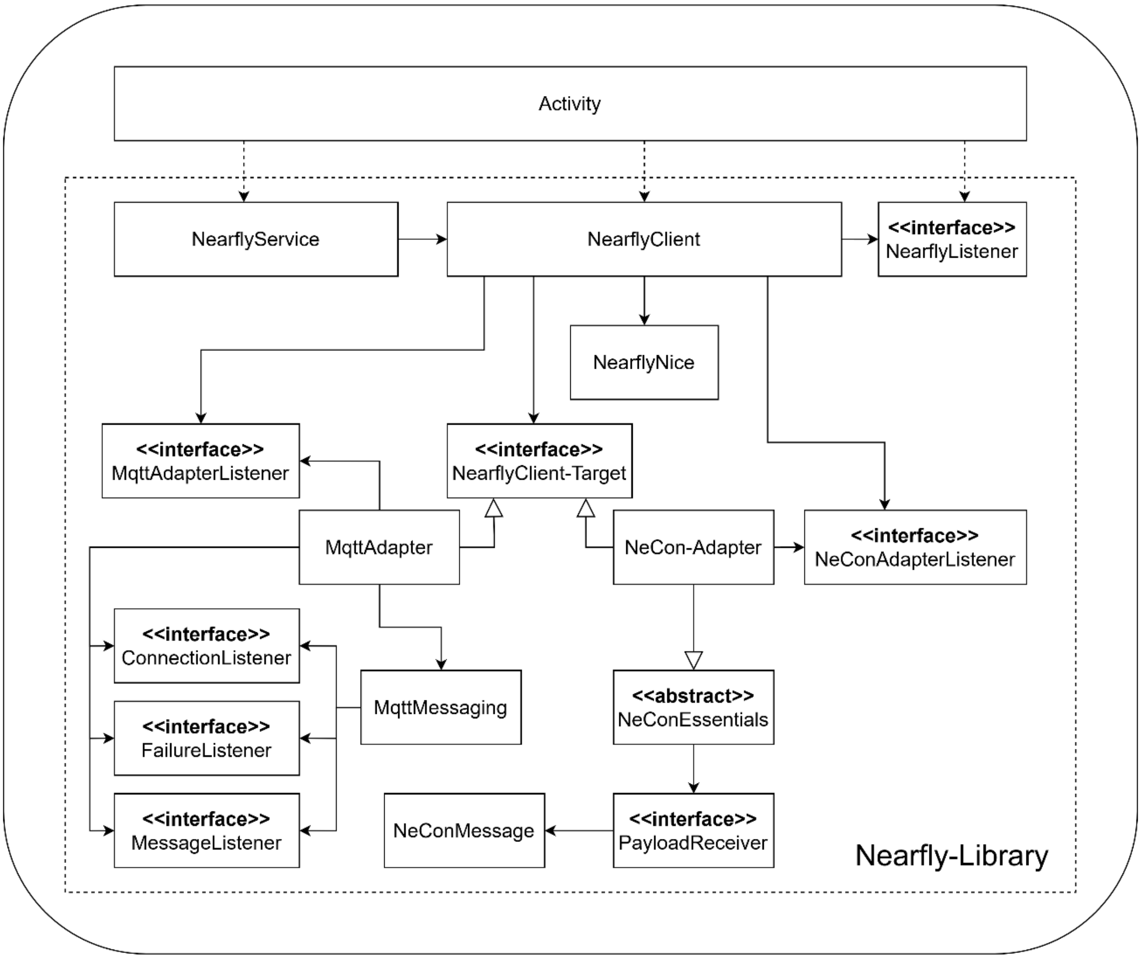

Activity
NearflyService
NearflyClient
<<interface>> NearflyListener
NearflyNice
<<interface>> MqttAdapterListener
<<interface>> NearflyClient-Target
MqttAdapter
NeCon-Adapter
<<interface>> NeConAdapterListener
<<interface>> ConnectionListener
MqttMessaging
<<abstract>> NeConEssentials
<<interface>> FailureListener
<<interface>> MessageListener
NeConMessage
<<interface>> PayloadReceiver
Nearfly-Library

Illustration 4: System architecture of the Nearfly API

In addition, the Nearfly API is to be used via an Android service (called NearflyService), which itself is a wrapper class (as in Illustration 4) which NearflyClient is holding can be used. Addressing the Nearfly API via the NearflyService should be the method that is preferred in this work and has the advantage that the service, after being started once, runs as long as the actual app runs in the background. If the activity is changed within the same application, the connection to the network remains and the new activity can also use the NearflyAPI. If they want to react to events, they can also register a NearflyListener.

4 Connection establishment

The mandatory authorizations show that the MQTT only needs one authorization for Internet access, while the Nearby Connections API requires a total of nine authorizations for Bluetooth, Internet, location and data access. Since location and data access can potentially cause a lot of damage, these authorizations must be queried separately at runtime. Creating a visibility restriction ensures that a network build-up takes place between devices with different applications that use the Nearfly API. The choice of topology shows that between overlay queue and star network, the star network is chosen in order to use the WiFi hotspot and thus have more bandwidth for data transmission within the network.

The creation of an autonomous peer-to-peer network shows that the network construction process is state-based, and, after a root-finding phase, the most powerful device is seen by all nodes as the root to which all other nodes connect.

4.1 Mandatory permissions

Apps in Android need static and dynamic authorization for all operations that can adversely affect other apps, the operating system or the user [19]. This includes reading out all data content outside of your own application context, as is required for Nearby Connections. The authorizations are divided into three risk groups (called protection level). Normal permissions can be entered in the manifest and are then requested once when installing the app. Signature authorizations, on the other hand, do not have to be requested explicitly if they have already been granted to an app with the same signature. Finally, there are dangerous authorizations, such as reading user contacts[19]. As of Android 6.0 (API level 23), these must be explicitly queried by the application by implementing a query at runtime[19]. While MQTT only has normal authorization to open Internet Sockets (as in **table 4**), Nearby Connections including the ability to send binary data requires a total of eight different authorizations. The normal authorizations include two authorizations to disclose and pair devices via Bluetooth (as in **table 4**) and two authorizations to read out WiFi information and change the WiFi connectivity status used for the hotspot (as in **table 4**). The remaining four mandatory authorizations belong to the Dangerous protection level and include two authorizations for querying the approximate (FINE_LOCATION in **table 4**), as well as the exact location (COARSE_LOCATION in **table 4**) of the device and two authorizations for reading and writing data outside of the application-specific directory. It should be noted that the read authorization is automatically granted when the write authorization is granted. This also applies to the creation of the FINE_LOCATION authorization, which includes access to the service that can be used by

COARSE_LOCATION. This shows that the Nearfly library requires a total of nine authorizations, four of which have to be queried at runtime, but are only displayed as two queries to the end user due to the dependencies mentioned.

table 4: Mandatory permissions to use the Nearfly library

	INTERNET	BLUETOOTH	BLUETOOTH_ADMIN	ACCESS_WIFI_STATE	CHANGE_WIFI_STATE	READ_EXTERNAL_STORAGE	WRITE_EXTERNAL_STORAGE	ACCESS_MEDIA_LOCATION	ACCESS_COARSE_LOCATION
Nearby Connections		X	X	X	X	(X)	(X)	X	X
MQTT	X								

(X) optional □ Normal permission ■ Dangerous permission

4.2 Visibility restriction

Before establishing the connection, the developer must decide on the technology to be used (by setting the ConnectionMode). If the developer has previously decided to use the NearflyService, the method for initializing the connection establishment may only be called after the service has been successfully bound, as the service instance is only transferred with the binding.

In accordance with the requirement of restricting the visibility of the messages outside the same application, a text-based identifier that is unique for the application (called a room string) should be specified before the connection. Applications that use the same room string can communicate with one another. Technically, this is implemented by MQTT by entering the room string as a top-level topic when subscribing and publishing. Nearby Connections offers a so-called SERVICE_ID for this. This is specified when advertising or discovering is started and is used from now on to restrict the visibility of nodes with different SERVICE_IDs. After entering the room string, the connection can now be established.

4.3 Choice of topology

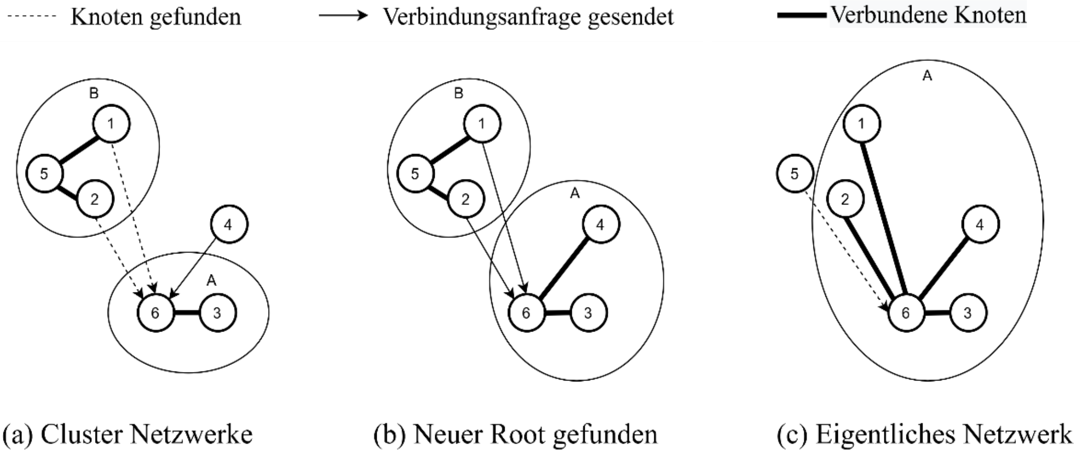
Due to the server / client architecture, MQTT only needs to call the correctly parameterized Connect method, which creates an indirect connection to all other broker clients (directly only to the server). The situation is different with Nearby Connections, which as a peer-to-peer API call for the successive establishment of a network through requests from the nodes to be connected. Two ways have been found to address these problems. The first option, which has already been developed internally by the NearbyConnections team, and up to about 30 devices[20] can connect, includes the creation of an overlay queue in which header and tail advertise and discovers at the same time *[20]*. A disconnected device is seen as a queue of length one, while the header and tail are not allowed to connect to each other[20]. However, this means that the CLUSTER topology must be selected, which, as already mentioned, is limited to Bluetooth and thus leads to a comparatively lower data transfer. In order to be able to use a WiFi hotspot, a star-shaped network must be used. This has the advantage of broadband, stronger data transmission that works particularly well with a low number of participants (three to four). However, this second option also has weaknesses. On the one hand, the central node (root node) is more heavily loaded and the load is no longer distributed. On the other hand, the root node, as the connecting unit of all network participants, forms a single point of failure. If the root node falls away, the network has to be rebuilt. It should also be noted that the network, as already known, despite WiFi hotspot is limited to seven participants by Bluetooth. To make use of the WiFi hotspot, a star-shaped Network topology used.

4.4 Creation of the autonomous peer-to-peer network

Since each device is shown the endpoint name of all discovered advertisers during the discovery process, the advertiser must decide via the endpoint name which device can serve as the root node. For this purpose, the choice of the endpoint should be appropriate (1) from the maximum CPU frequency in megahertz () of the device and the difference between the maximum long value and the current time of day, whereby the constant and the Unix time are in milliseconds. This is used to evaluate the respective device.$f_{CPU}MAX_{Long}$ $T_{curr}MAX_{Long}2^{63}-1T_{curr}f_{CPU}$This is to ensure that more powerful devices that join the network later can no longer serve as root. is a constant gain factor of 20, which was determined empirically and determines the weighting of the frequency.$MAX_{Long} - T_{curr}Wf_{CPU}$

$$MAX_{Long} - T_{curr} + f_{CPU} \cdot W \qquad (1\quad)$$

The in Illustration 4 The NeConAdapter shown uses a state machine with six states to establish the connection (see **Illustration 6**). While the NeConAdapter is not being used by the NearflyClient, it should remain in the STANDBY state, otherwise it changes to the FINDROOT state. During the FINDROOT state, both advertising and discovering are activated. In this phase, each node checks when it detects (onEndpointDiscovered) another node by comparing the endpoint names to see whether it is smaller. If this condition is met, it changes to the NODE state and at the same time deactivates advertising so that it is no longer visible to others. At the same time, it sends a connection request to the node discovered with the largest endpoint name.



(a) Cluster Netzwerke

(b) Neuer Root gefunden

(c) Eigentliches Netzwerk

Illustration 5: Network building in cluster formation

In the multi-node network, it can happen that a few nodes initially clusters (such as cluster B in Illustration 5b) form and subsequently to the root (Illustration 5c) connect. Meanwhile, other nodes are building according to node 4 in Illustration 5a connects to the actual root. Empirical evidence shows that an initial root-finding phase, in which the in Illustration 5 situation is avoided, but leads to longer connection times in both small and large networks.

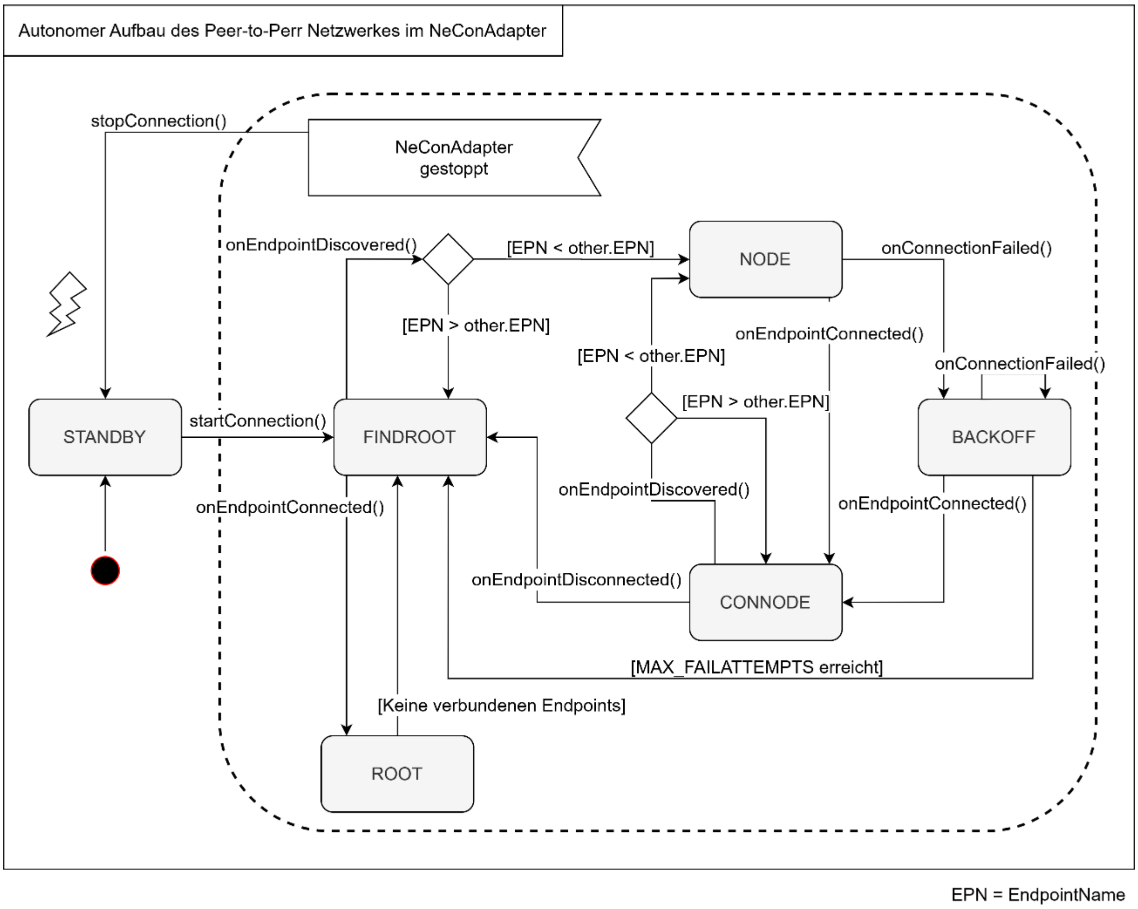


Illustration 6: State diagram of the connection establishment of the NeCon adapter

If a node A notices an incoming connection request (onConnectionInstantiated) from node B and it is not in the process of establishing a connection with another node C, the latter accepts it. After the connection has been successfully established (onEndpointConnection), node A changes to the ROOT state and deactivates discovering because it now wants to be found (as the root node). At the same time, node B changes from the NODE state to the CONNODE state. On the other hand, if node A is currently in the process of establishing a connection, the connection request from node B is rejected (onConnectionFailed). The requesting node B (in the NODE state) then goes into the BACKOFF state and waits a random time (BACKOFF_TIME) within a given time window until it then sends a new connection request. Since the establishment of the connection can also fail if the root node can no longer be reached due to a network exit (peer churn), node B changes to the FINDROOT state after MAX_FAILATTEMPTS failed requests and deletes its cache (including pending endpoints). If all nodes have connected to the root network, the network exists.

If a new device wants to join the existing network, it only sees the root node, which is the only one that has activated advertising and tries to connect to it as above. If a node wants to join the network and has a higher endpoint name than the current root due to the set weighting, all network participants gradually connect to it, followed by the old root. After

60 seconds have elapsed, all nodes in the CONNODE state deactivate discovering in order to save the battery. If the NearflyClient disconnects the NeConAdapter, the node returns to the STANDBY state. Advertising and discovery are deactivated, and existing connections are disconnected and cleaned up.W

5 Data transfer

Before the data transfer, the recipient must subscribe to the sender's target channel using the subscribe method, which registers the recipient and the associated channel, depending on the connection mode set, with the central component (broker or root).

The actual data transmission of the Nearfly Client is parameterized by a channel, a priority and a flag for buffering the messages. Data under 30 Kbytes can be sent using the pubIt method, while larger data must be sent using the pubFile method.

According to the current ConnectionMode, the Nearfly API calls the respective adapter component, which carries out a fragmented data transfer using the NeCon message protocol with different fields. The fields mainly contain the file extension and identifier and are used by the recipient to compose the file in the standard download directory. The NeConAdapter sends, limited by Nearby Connections, the pure file and the information for composing the file separately, while the MqttAdapter sends the file and related information as one message.

5.1 Subscribe with Nearfly

In order to meet the requirement that different messages can be separated within the same application, at least one channel must be subscribed to before the first message exchange.

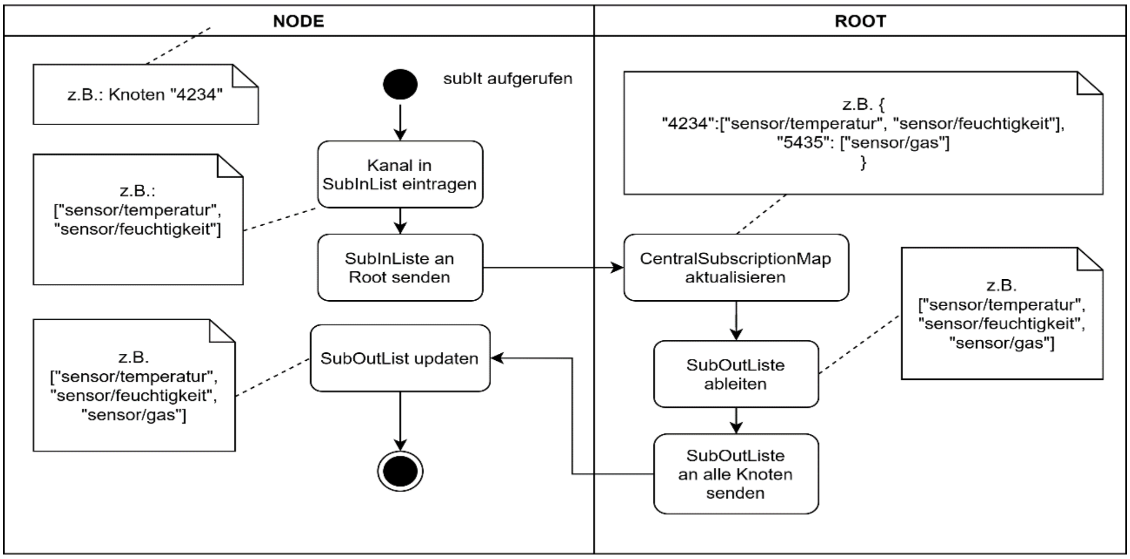


Illustration 7: Subscribe process with the NeConAdapter

Since the nearfly channel is the same as the MQTT topic, it is sufficient to call up the dedicated subscribe method on the part of the Mqtt adapter with a topic (room / channel), which is formed from the room string and the channel string. A NeConControlMessage is implemented for this purpose on the part of the NeCon adapter. This is defined by a CMD and a subList field and is used to send the subscribe lists of the individual nodes. The CMD field is intended to facilitate future extensions and currently only has the value CMD_SUB. Each node has two lists. The SubInList contains the channels to which the owner of the list has subscribed and is sent to the root by the nodes when new channels are subscribed. The SubOutList contains the channels to which all other network participants have subscribed and is sent by the root, to inform the nodes in the network which channels they want to receive. In addition, the root node has a CentralSubscriptionMap, which shows the network participants and the associated subscribed channels (as in Illustration 7) is linked and used to forward the received messages. If a NODE subscribes to a channel, the corresponding Illustration 7. First the channel string is entered in the SubInList and sent to the root. This uses the received list to update the entry of the sender node in the CentralSubscriptionMap and then creates a SubOutList (as in Illustration 7), which is sent to each network node and is used to update those SubOutLists. As soon as the connection to a node is disconnected, the root deletes the node's entry from the CentralSubscriptionMap, updates the SubOutList and sends it again to all nodes.

If at least one channel is subscribed to, all messages addressing the same channel can be responded to after a NearflyListener has been transferred. Conversely, unsubscribed or unsubscribed channels are ignored. The NearflyListener differentiates between three message types. On the one hand, status events (onLogMessage) of the adapter components, such as the status changes on the part of the NeCon adapter or the successful connection establishment, can be distinguished. On the other hand, it can react to multimedia messages (onFile) and binary messages (onMessage).

5.2 Send with Nearfly

If there is a connection to the network, data can be transferred using one of the two methods offered by the NearflyClient (pubIt and pubFile). While smaller byte data of up to 30 KBytes (MAX_PUBIT_MESSAGE_SIZE) can be sent by pubIt, larger multimedia binary data must be transferred using the pubFile method. The border arises from the already from the section2.3 known limitation for byte messages of the Nearby Connections API of 32 KBytes and a generous tolerance to allow for larger future extensions of the NeCon protocol (which is described in the following section 5.3is discussed) to compensate. In addition to the mandatory specification of the receiving channel, the methods can be used with a priority level from -20 (lowest priority) to 19 (highest possible prioritization), based on the nice level in the Unix system[21], as well as a retain flag can be parameterized.

A set retain flag means that messages can be "sent" even when there is no connection. For this purpose, these are first placed in a priority-based send buffer and are only sent when there is a connection to the underlying technology and the connection is not currently being changedbecomes. If the retain flag is not set, the transmission process is aborted in the connectionless state and the user is informed of this with a return value.

Since MQTT guarantees that the order of the messages for messages with the same QoS is observed[10] and the NearbyConnection API guarantees the same payload types (bytes and files) [6], the implementation of a prioritization on the send side is sufficient.

5.3 Realization with NearbyConnections

While Bytes messages are sent and received ad hoc by the Nearby Connections API, file messages are different. These are automatically chunked and transferred piece by piece. The Nearby Connection API creates the received file piece by piece in a self-created "Nearby" folder in the standard download directory. This explains the necessary dangerous authorization for reading and writing data outside the application-specific directory. Because although the Nearby Connections API is through the use of Google Play Services[3] himself already has these authorizations to edit external data [22], the activity that finally wants to open, rename or delete the received file needs access to this directory.

However, since the stored data is named according to its identifier (e.g. -7808006164344755168), the end user cannot open this file due to the missing file extension and experience shows that restoring the file extension using the MIME type is not always successful, when sending a file -Message can also be sent a message with the meta data. This contains the file extension and the file identifier, which is used for the later assignment of the file and will be referred to below as NeConFileInformation. In addition, this contains a text attachment (textAttachment), which allows the later developer to attach an additional string (e.g. for further metadata).

[3] The Nearby Connections data is stored in an external directory, as Nearby Connections runs in a separate process and therefore does not have access to the application-specific directory of the application that uses the API.

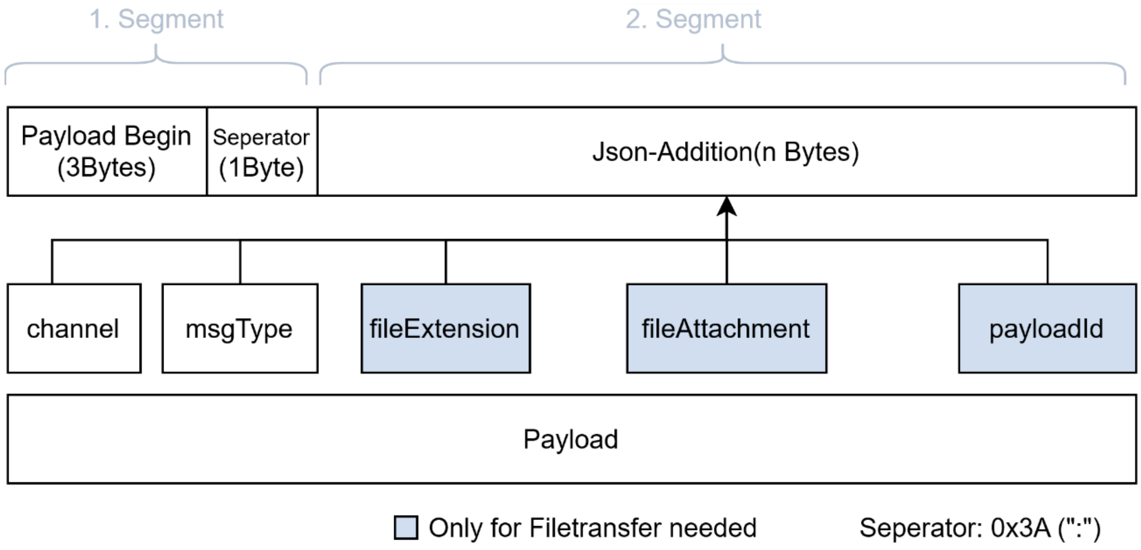


Illustration 8: NeCon message protocol

If the NearflyClient now calls the pubFile method, the NeConAdapter sends the file and also the NeConFile information, while calling the pubIt method sends a NeConBytesMessage. For both NeCon messages, the in Illustration 8 NeCon message protocol shown is used. To enable a dynamic header length, the header is divided into two segments. The first segment has a fixed byte length and is obtained from a UFT-8 encoded string. It contains the field for the beginning of the Bytes payload, which is initially limited to 3 bytes and thus allows a maximum header length of 999 bytes. Each string element needs exactly one byte, as this is guaranteed for UTF-8 encoded characters, which can be found in the ASCII table. Other characters, on the other hand, can take up to 4 bytes. The size of the second segment is dynamic and contains a JSON object in UTF-8 string format, which currently only receives the channel for NeConBytes messages. In the case of NeConFile information,

Receives the NearbyConnection API (according to **Illustration 9**) a payload header, the PayloadReceiver checks whether it is of the File or Bytes payload type. If this is a Bytes message (not to be confused with the similarly named NeConByte message), the NeCon message type is identified. The message is transferred to the NeCon client (onMessage). This checks whether the corresponding channel of the message is subscribed and forwards it to the NearflyService if it is successful. If this is not the case, the message is discarded. If the current node is declared as the root node, the message is also sent to all connected nodes except the sender of the message. The NearflyListener can ultimately be used to respond to the message. A NeConFileInformation message, on the other hand, is initially cached, at the same time it is checked Whether the payload header of the file message with the same identifier has already arrived and is in the CompletedFilePayload buffer. If this is not the case, the file is passively awaited.

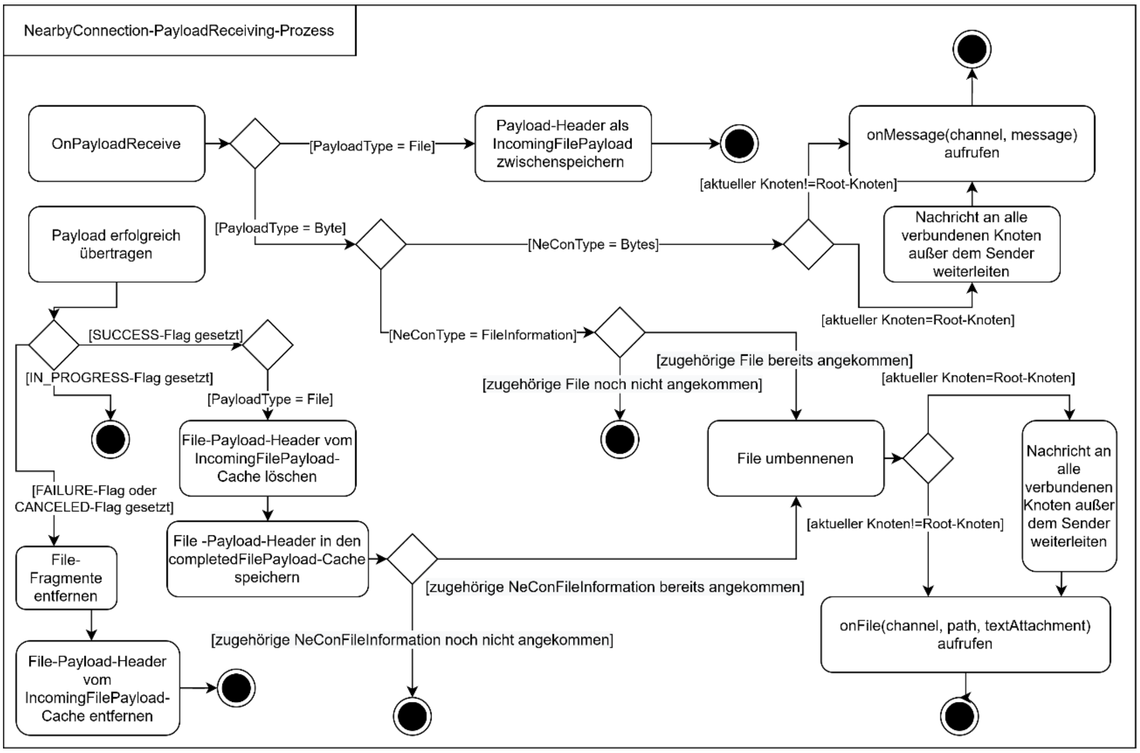

NearbyConnection-PayloadReceiving-Prozess
OnPayloadReceive
[PayloadType = File]
Payload-Header als IncomingFilePayload zwischenspeichern
onMessage(channel, message) aufrufen
[aktueller Knoten!=Root-Knoten]
Nachricht an alle verbundenen Knoten außer dem Sender weiterleiten
Payload erfolgreich übertragen
[PayloadType = Byte]
[NeConType = Bytes]
[aktueller Knoten=Root-Knoten]
[NeConType = FileInformation]
[SUCCESS-Flag gesetzt]
[IN_PROGRESS-Flag gesetzt]
[PayloadType = File]
[zugehörige File bereits angekommen]
[zugehörige File noch nicht angekommen]
[aktueller Knoten=Root-Knoten]
File-Payload-Header vom IncomingFilePayload-Cache löschen
File umbennenen
Nachricht an alle verbundenen Knoten außer dem Sender weiterleiten
[FAILURE-Flag oder CANCELED-Flag gesetzt]
[aktueller Knoten!=Root-Knoten]
File -Payload-Header in den completedFilePayload-Cache speichern
File-Fragmente entfernen
[zugehörige NeConFileInformation bereits angekommen]
onFile(channel, path, textAttachment) aufrufen
[zugehörige NeConFileInformation noch nicht angekommen]
File-Payload-Header vom IncomingFilePayload-Cache entfernen

Illustration 9: Process flow when receiving a message from the NeCon client

If the payload header of a file message arrives (onPayloadReceive), this signals the beginning of the transmission of a file message. The payload header (which contains the identifier) is then temporarily stored as IncomingFilePayload in order to note the ongoing data transfer. During the transfer, a payload header (OnPayloadTransferUpdate) with the IN_PROGRESS flag set is received for each transferred file fragment, by means of which the number of transferred bytes can be viewed. If an error occurs, this is indicated by a set FAILURE or CANCELED flag. The initially cached payload header, including NeConFileInformation and the fragmented file in the download directory must then be deleted. If the file has been completely transferred, the Payoad header contains a set SUCCESS flag. The noted header is now removed from the IncomingFilePayload cache and pushed into the CompletedFilePayload cache. If the NeConFileInformation message is now also in the buffer, the process for renaming the file is initialized.

If the current node acts as the root node, the received file is also forwarded to all connected nodes. If the receiving process has been successfully completed, the payload header and the NeConFileInformation are deleted from their cache and the onFile method is called with the channel, the file path and the textAttachment.

5.4 Realization with MQTT

Since the nearfly channel is based on the MQTT topic, it is sufficient for the Mqtt adapter to call up the dedicated subscribe method with a topic (room / channel), which is formed from the room string and the channel string. To enable the transmission of multimedia data using the MQTT API equivalent to the Nearby Connections API, sent data should also be stored in the standard download directory in the "Nearby" folder after arrival. In addition to the file to be transmitted, the file extension, a text and the message type must also be sent here.

An uncomplicated and therefore often chosen option to solve this problem is to send the entire message as a string in JSON format. The binary multimedia file is encoded into a string using the Base64 encoding process and decoded again when it is received. However, Base64 encoding ensures that three byte octets are encoded into four character octets and the data is then paddingto a multiple of four for it, which is 1.37 times that for all binary data in the coded state [23]the original data size is required. In order to avoid this overhead, another option should therefore be chosen. This results from the use of the NeCon message protocol (Illustration 8), which separates the string from the payload bytes with the dynamic string header, which can be easily modified by the JSON segment.

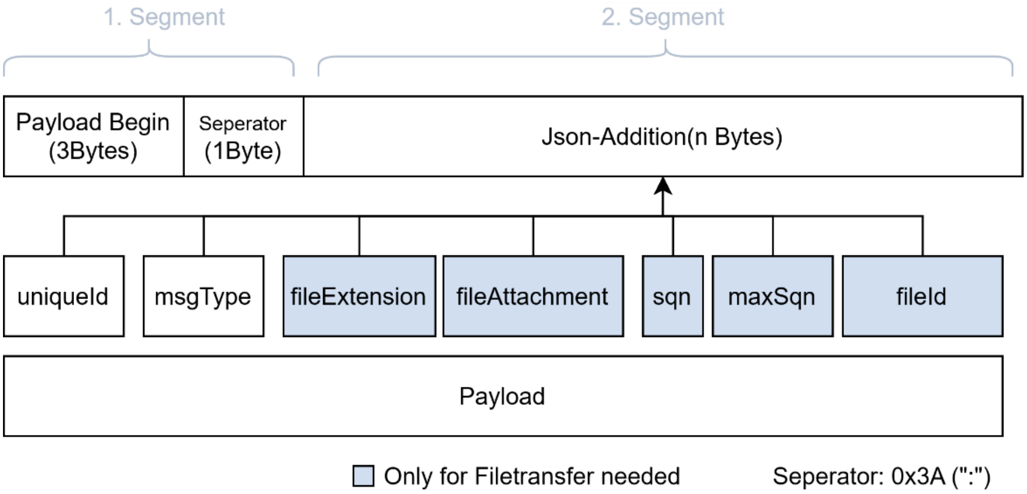


Illustration 10: Modified NeCon message protocol for message transmission by MQTT

Because the Nearby Connections API does not specify a maximum data size for multimedia data and the MQTT protocol a maximum payload size of 256 Mbytes [12] provides, the MQTT adapter becomes a chunking mechanism[4], which breaks down packets from 2 Mbytes (MAC_CHUNK_SIZE). The threshold value is chosen so large that a

[4] It would also be conceivable to use chunking to enable prioritization. In order to achieve the same behavior on the part of both adapters, such a file prioritization through chunking would also have to be implemented with the NeConAdapter. This would slow down the data transfer process and should therefore not be implemented.

threshold value that is too small, for example 30KByte for a 5MByte picture, would initially flood the broker with 167 chunks and thus greatly slow down MQTT. To send the data, a sequence number (sqn), the maximum number of fragments to be sent (maxSqn) and an identifier (fileId) are added to the JSON segment. The fileId serves to distinguish different multimedia fragments on the recipient side and is identical for each fragment of the same file. The sequence number is used in order to be able to recognize and ignore duplicate received file fragments, which can appear due to the set QoS of one. This results in in **Illustration 10**JSON segment shown. If a multimedia file larger than MAC_CHUNK_SIZE is to be sent, the sender generates a random five-digit fileId and sends the file fragments, which are then cached by the recipient. If the recipient receives the file fragment with the sequence number (maxSqn-1), the file is reconstructed, and the cached file fragments are deleted. The onFile method of the NearflyService is then triggered.

In order to enable bytes to be transferred, the JSON segment also receives the message type (msgType) and a uniqueId (as in **Illustration 10** shown), which is intended to prevent the sender from receiving its own messages and, first of all, to adapt the behavior with regard to receiving both ConnectionModes is used.

6 Evaluation

The tests for measuring the times for the connection establishment show that the connection establishment using the NeCon adapter between two nodes with an average of 7.5 to 11 seconds and an average of 60 seconds for establishing a network with six nodes scales significantly worse than the MqttAdapter, which is used for establishing a network from five network participants only needed an average of 454 milliseconds. The times measured for data transfer, on the other hand, show that the latency in both adapter classes is the same at around 150 milliseconds, but the NeConAdapter can transfer data about three times faster when the WiFi hotspot is active, while data transfer of the same files with deactivated WiFi Hotspot slowed down by a factor of about 27.

In Sample apps with Nearfly shows that all requirements have been met, whereby the near real-time capability required for the Bouncingball App is only met when using the NeCon adapter in a network with two nodes. Furthermore, the shared touchpoint canvas shows that the MqttAdapter also scales better than the NeConAdapter in relation to data transmissionand that both adapters behave roughly the same in a network with three nodes. Applications with little data traffic, such as the Score Board Notepad, prove to be well suited for both adapters if precautions are taken for possible intermittent connection crashes.

6.1 Test equipment used

For data collection, accordingly table 5the Motorola G5, G6, as well as the Samsung Galaxy S7 edge, the S9 and the LG G2 mini LTE are used. All smartphones support the same WiFi protocols but differ both in the Android versions and in some cases in the built-in Bluetooth chips, which are essential for establishing a connection in the NeCon ConnectionMode. To the in table 5 called smartphones, a Samsung Galaxy s5 mini, which was also available and had Bluetooth 4.0, had to be excluded as a test object because it frequently refused connections and this led to very long connection times.

table 5: Smartphones used for the evaluation

Smartphone	Android version	WiFi 802.11	Bluetooth
Motorola Moto G6	9.0	a / b / g / n	4.2
Motorola Moto G5	8.1	a / b / g / n	4.2
Samsung Galaxy S9	10	a / b / g / n	5.0

Samsung Galaxy S7 edge	7.0	a / b / g / n	4.2
LG G2 mini LTE	7.1.2	a / b / g / n	4.0
Samsung S5 mini	7.1.1	a / b / g / n	4.0

6.2 Times for establishing a connection

First, the times are measured which are required in the NeCon ConnectionMode to set up a two-node network. This has the advantage that a better insight into the required times of an individual node can be obtained. The test to establish a connection runs automatically, whereby the time required to reach the states (NODE, CONNODE) is logged by the system. If a connection has been successfully established by the last node, it changes to STANDBY status after five seconds and thus disconnects. By switching to STANDBY-State it is ensured that all previously found nodes are deleted from the cache and the initial root-finding phase has to be continued again. Another five seconds later, which serve as a tolerance time for recognizing the disconnected connection, the non-root node returns to the FINDROOT state in order to participate again in the network setup. Figure to **Illustration 14** represent the time required to set up this smallest network in 50 experiments with different device combinations.

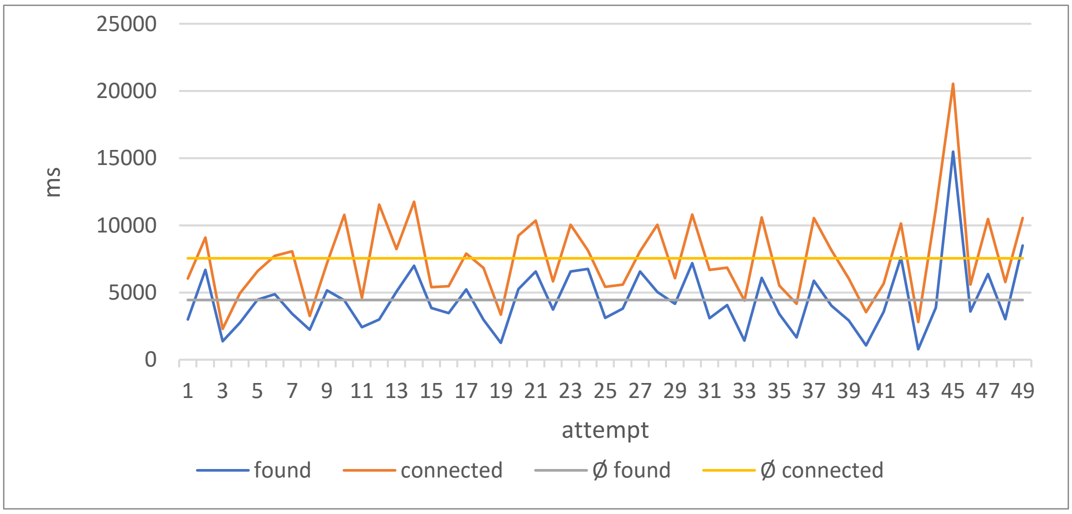


Illustration 11: Times required to establish a connection between G5 and S9

In **Illustration 11**the G5 finds the S9 after an average of 4445 milliseconds, while a connection ultimately takes place after an average of 7548 milliseconds. The connection is established after a minimum of 2295 milliseconds and takes a maximum of 20533 milliseconds.

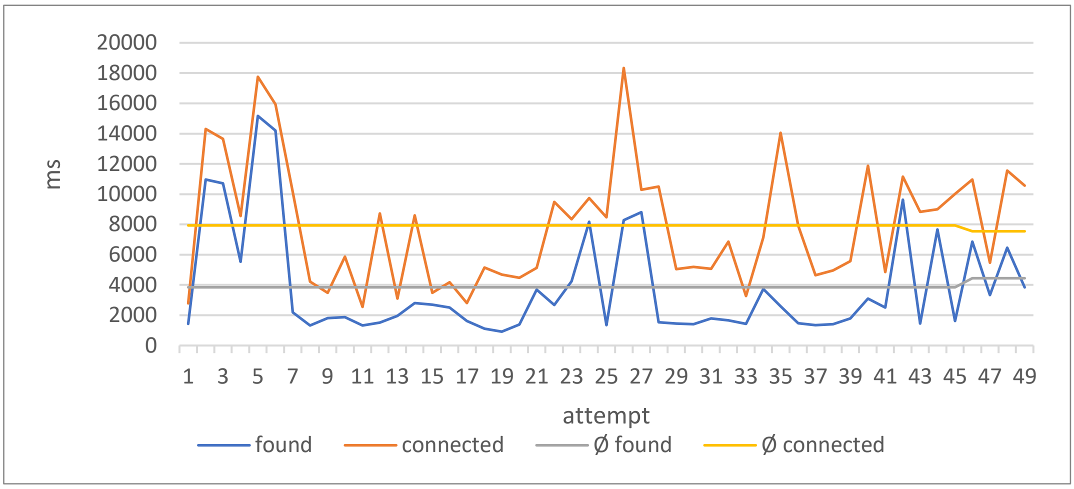


Illustration 12: Times required to establish a connection between S7 edge and S9

While values in **Illustration 11** fluctuate less, the connection establishment between S7 edge and S9 (as in **Illustration 12** can be seen) exposed to stronger fluctuations, but here again the S9 is found after an average of 3845 seconds and a connection is established after an average of 7934 milliseconds.

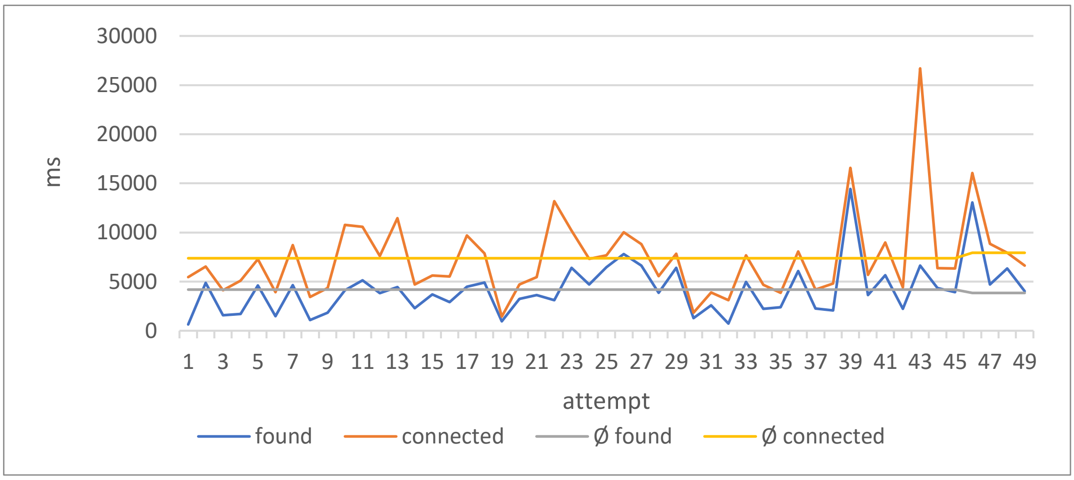


Illustration 13: Times required to establish a connection between G6 and S9

Between the establishment of a connection between G6 and S9 (as in **Illustration 13**) are similar, the average times according to the previous figures. While a node is found after an average of 4183 milliseconds, it takes an average of 7376 milliseconds to establish a connection.

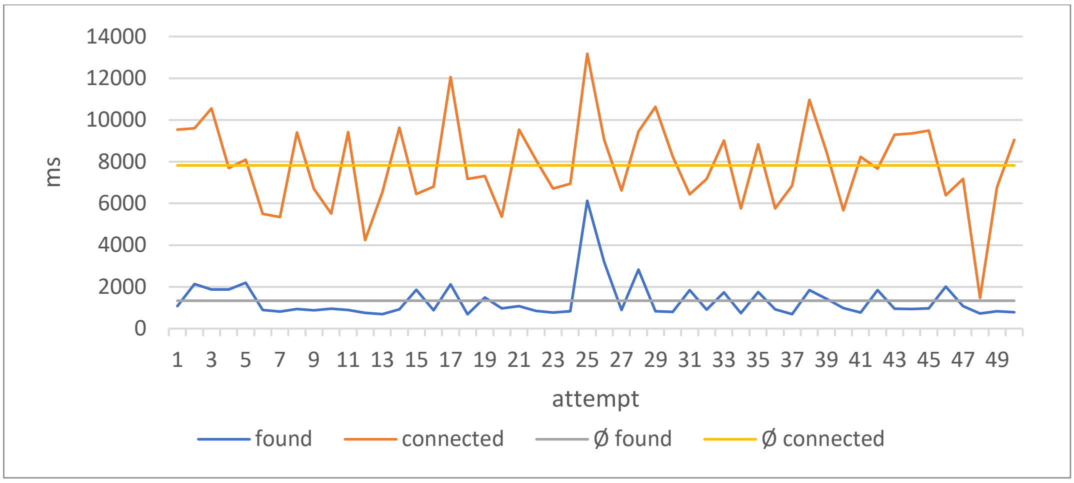


Illustration 14: Times required to establish a connection between LG and S9

Although the LG (as in **Illustration 14**) sometimes has the lowest version of the Bluetooth chip, it manages to find the S9 after an average of 1336 milliseconds and is therefore around three times faster than the previously tested combinations in terms of discovery times. When establishing a connection, however, the average connection time is again 7824 milliseconds and thus in the range of the previous device combinations.

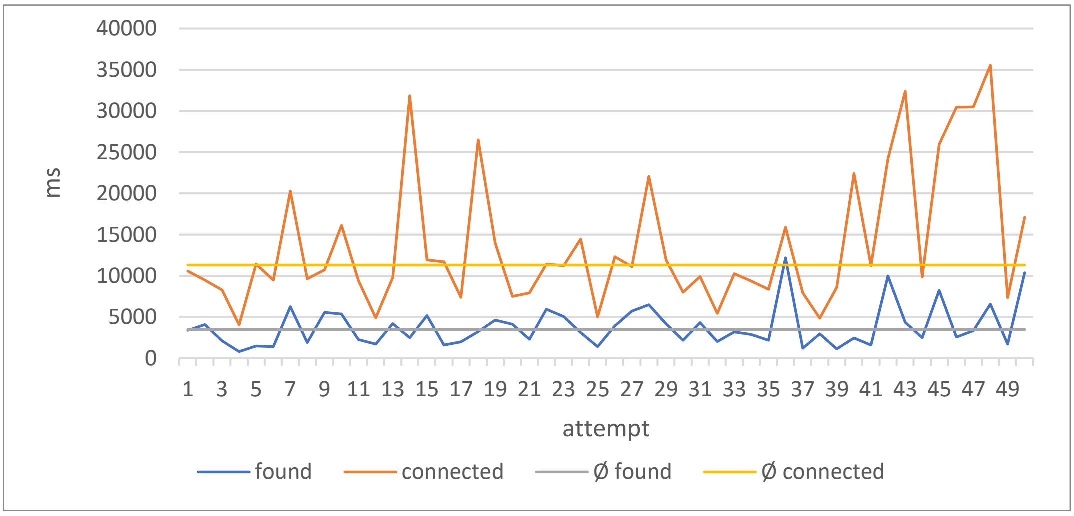


Illustration 15: Times required to establish a connection between G5 and G6

The combination between G5 and G6, which are both equipped with Bluetooth chips of version 4.2, takes about the same time as the previous combinations with 3848 milliseconds. However, it takes about 1.3 times longer to build the connection compared to the previous device combinations. Also shows **Illustration 15** high peak values in the region of 35 seconds.

As a result, it can be stated that a connection establishment takes an average of 7.5 to 11 seconds, although it cannot be ruled out that a connection establishment takes up to 35 seconds. In the best case, however, connections can also be established in 3 seconds.

Establishing a connection in the two-node network initially appears slow. However, the Nearby Connections API co-developer Will Harmon already gives an average connection latency of 2-7 seconds for unidirectional connection establishment (i.e. in the case of a single advertiser and a single discoverer)[8th] with the connection latency for devices that are in a worse state of up to one minute [24] can increase.

However, since the advertiser (root) is not defined in the NeConAdapter from the start, the root identification phase, in which both advertising and discovering is activated, must be run through. This creates trashing[8th]. This phase is important because smartphones such as the LG G2 should not be selected as root if possible due to intermittent disconnections and poor performance for forwarding messages. Also, with the Samsung S5 Mini, which was not included in the above test cases, it could be determined that as root it cannot set up a network larger than two nodes and does not accept further nodes.

With the knowledge that there are already different latencies between two different smartphone combinations, the times for setting up a six node network (as in Illustration 16) are evaluated. This process of data collection is also automatic, in that this time the root nodes log the times from the initialization of the connection to the time all five nodes join the node. If the network of six nodes has been set up after 90 seconds, for example, the root node sends a "Finish" message to all network participants, which are then put to sleep. So that the network is not rebuilt by the nodes, they also switch to the STANDBY state. If you have waited so long that the current minute, a number that can be divided by three, all devices switch (once in this minute) to the FINDROOT state and try to set up the network again. The three minutes are intended to ensure that everyone wakes up at around the same time.

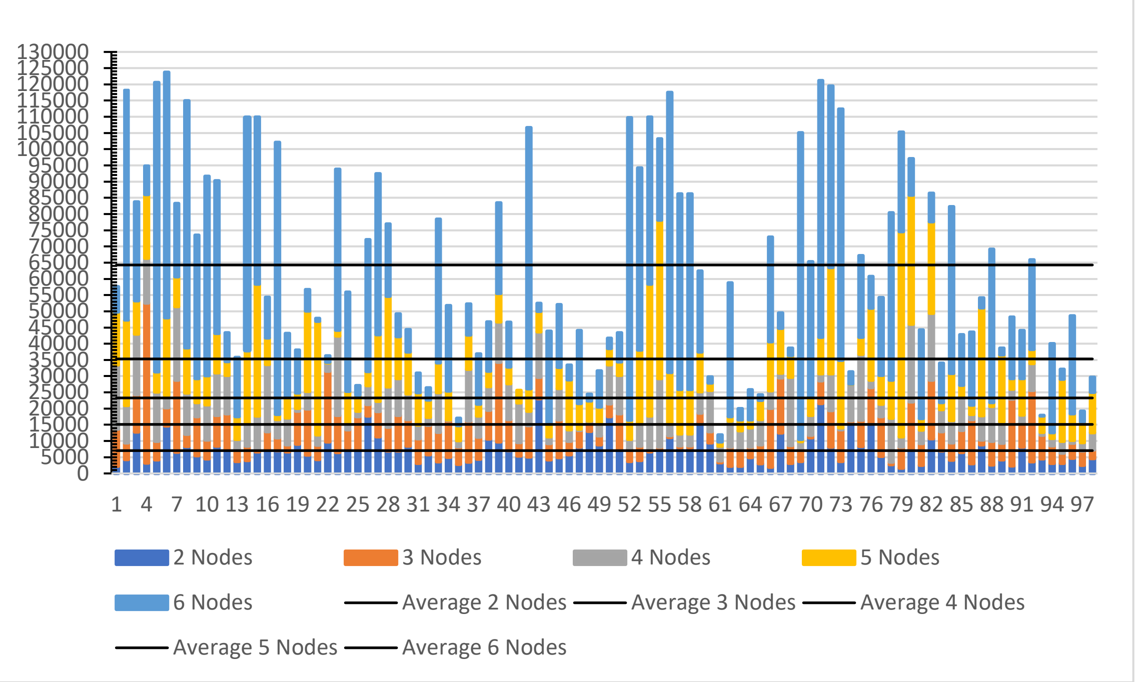


Illustration 16: Network structure in the six-node network

To set up the network, all available smartphones (as in **table 5**) used. Out Illustration 16 the connection between the first two nodes is established after an average of 7027 milliseconds. The third node joins the network in an average of 15120 milliseconds, while the fourth node connects in an average of 23278 milliseconds. After an average of 35,000 milliseconds, the fifth node is also connected. The last node that connects after an average of 64290 milliseconds is almost always the Samsung S5 Mini, which is included in this test case to demonstrate that a connection to six nodes is possible. This means that the last value does not necessarily apply as a guideline for the successful establishment of a connection between six nodes, as a network establishment could sometimes be created in less than 35-40 seconds. It should be noted that connection breaks can occur during the network establishment, which can in some cases greatly slow down the connection process. If a device connects to a wrong root, this often has the advantage that the other nodes in the network can establish a connection to the actual root without major mutual interference, while the other two first discover the root and connect to it.

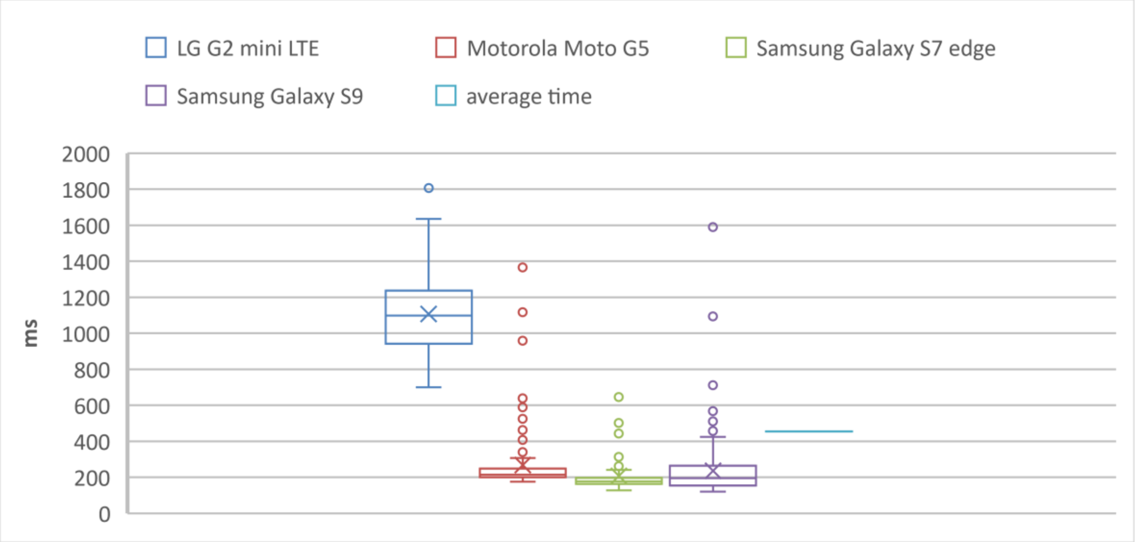


Illustration 17: Times for establishing a connection with MQTT

The connection establishment by MQTT, however, is noticeably faster, since the network establishment is hardly dependent on the network size. Instead, it depends on the broker's capacities. The data collection takes place here again by means of an automated test, with the difference time between the connection request and the CONNECT event of the onLogMessage method of the NearflyListener being recorded over 160 connection attempts. The connection is established in a WLAN network in which the smartphones have an average download speed of 5 to 90 Mbit / s[5]to have. The broker is about 4 kilometers away. **Illustration 17**shows that the LG G2 needs almost five times the time for the connection with an average of 1100 milliseconds, but that it is established much faster across all with an average of 454 milliseconds. The nodes hardly influence each other due to the server-based architecture, so that the network of two and three nodes is set up after a maximum of two seconds.

6.3 Times for data transfer

In order to compare the data transfers between the two connection modes, 100 bytes, which roughly corresponds to the average JSON string payload of the implemented scenarios, are transferred in over 100 attempts. Since an automatic test is to be used here too and the times differ in the millisecond range between the individual smartphones, the time that a message needs in the round trip is measured. The round-trip counterpart is a second smartphone that publishes the message immediately after it has been received. The time difference between the sending time and the receiving time of the sender divided by two is sent as the latency of a message.

[5] The Samsung meanwhile had a download speed of 90 Mbit / s. While it was 8 Mbit / s with the G6 and 31.5 Mbit / s with the G5. The LG managed 5 Mbit / s and the Samsung s7edge again 35 Mbit / s.

table 6: Comparison of the latencies in both nearfly connection modes for 100 bytes

Network participant	MQTT (in ms)	NeCon (in ms)	NeCon only Bluetooth (in ms)
S9 and S7edge	156	139	66
G9 and LG G2	152	151	-
G5 and G6	101	155	-

table 6, which contains the measured average values of several test attempts, the average latencies that arise when publishing byte messages of the Nearfly API are on average lower in the NeCon ConnectionMode for the S9-S7edge and S9-LG combination. The WiFi hotspot was activated almost continuously in NeCon mode. The first transmissions took place with Bluetooth, as the WiFi hotspot was not yet activated by the Nearby Connections API. In the G5-G6 combination, on the other hand, MQTT has the lower latencies despite the activated WiFi hotspot. It can be seen from the measurements that the sending of byte messages is approximately the same on average in both connection modes. Looking at the latencies of a single attempt as shown in **Illustration 18** is shown, however, it is noticeable that the transmission via the Nearby Connections API is subject to significantly more fluctuation (jitter) and has at times latencies in the range of 40 to 800 milliseconds.

By setting the strategy to P2P_CLUSTER (as in

table 6 can be seen) the average latency in NeCon mode can be measured without the aid of the WiFi hotspot, which is relatively lower with an average of 66 milliseconds for a 100-byte transmission between S9 and S7Edge.

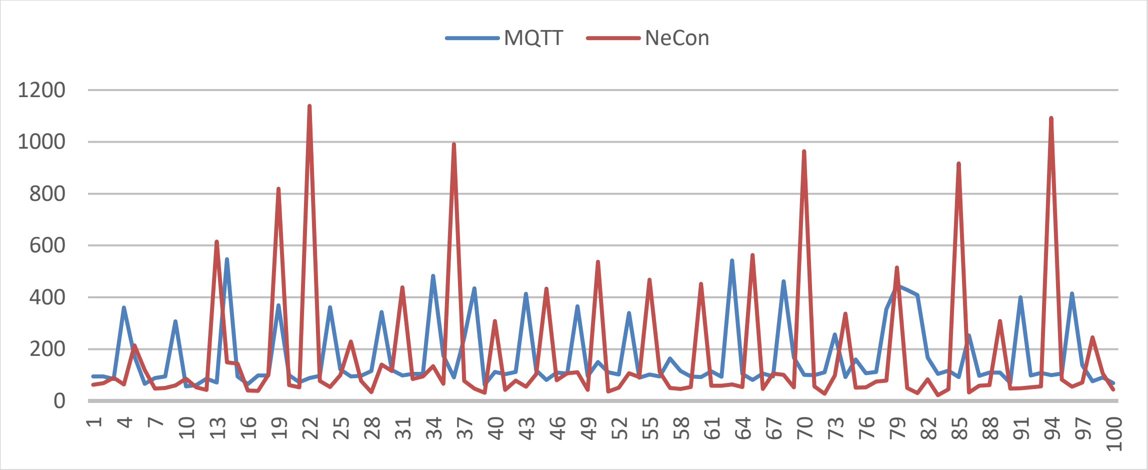


Illustration 18: Latency comparison of both connection modes with the S9-S7edge combination

Furthermore, the times for the transmission of multimedia data in both connection modes are to be recorded and compared. As in the previous test, the times here also correspond to the round-trip time of the message divided by two. In NeCon mode, the WiFi hotspot was continuously activated this time. In order to create a contrast between the deactivated and activated WiFi hotspot, the Nearby Connections API was also limited to the use of Bluetooth and BLE by temporarily using the P2P_CLUSTER strategy.

table 7: Comparison of the times of the pubFile method in both connection modes with the S9-S7edge combination

Data size	MQTT (in ms)	NeCon (in ms)	Bluetooth only (in ms)
67.19 Kbytes	286	103	1953
1.01 Mbytes	1614	556	13560
5.26 Mbytes	6121	1787	71308
55.57 Mbytes	43357	14077	-

table 7 shows the strength of the Nearby Connections API for larger data transfers. Compared to MQTT, the Nearby Connections API manages to transfer the same multimedia data almost three times faster. The situation is different before the WiFi hotspot is set up, where data transmission is slowed down on average by a factor of 27, with an increasing tendency for larger amounts of data. Unfortunately, the average time for transferring the 55.57 MByte APK in the cluster topology could not be measured because the connection

was disconnected again and again after a few MBytes had been transferred and the transfer was aborted within several attempts that lasted several minutes. This shows that it is advisable to wait until the WiFi hotspot is activated before transferring larger files. This can be recognized by

6.4 Sample apps with Nearfly

The exchange of text and multimedia data was successfully implemented and was tested with a messenger app, which, by using the Nearfly channels, offers users the opportunity to enter different chat rooms. The nearfly library is also corresponding to the in table 8 are able to buffer prioritizable messages in the event of a connection break by setting the retain flag. The connection is then automatically re-established, and the message is then sent if the connection is established.

table 8: Meeting the requirements

Functional requirements	status
Send text-based messages	✓
Send multimedia messages	✓
Separability of different messages within the same app	✓
Hold messages when disconnected	✓
Automatic connection establishment	✓
Near real-time (latency and throughput) for a smooth multiplayer game	O
Prioritization of the message types	✓
Visibility of messages only within the same app	✓
API of the underlying technologies is identical	✓

By comparing both ConnectionModes in the Bouncing Ball App, the app can be tested for its near real-time capability. It shows how the Bounce Ball App already exhausts the Nearfly API. In the MQTT network, the latency continues to rise even in two nodes, which means that after a while gaming is no longer possible. In fact, such near real-time capability is possible by reducing the Quality of Service (QoS) to zero. However, this is currently not offered by the Nearfly API, as no counterpart to QoS has been implemented in the NeCon adapter. In the NeCon network, the latencies are significantly worse without an activated WiFi hotspot and a game is not possible. If, on the other hand, the WiFi hotspot is activated, noticeable latencies occur at times with the same number of nodes, However, if there are three or more network participants, the NeCon network is also overloaded and cannot keep up with the data transfer[6]. This means that the requirement for near real-time capability (10-30 FPS) is only met to a limited extent for a NeCon network with two nodes and an activated WiFi hotspot.

In order to better assess the limit for the relationship between speed and scalability of byte messages, however, a system test can be carried out using the Share Touchpoint Canvas. The results of this system test are in **table 9** and show how many points of contact could be sent at the same time without major delays on the part of the recipient (about less than a second) on average. The points of contact were sent by a non-root node and the experiment was repeated about every third second. It turns out that the transmission in NeCon ConnectionMode, as already known from the Bouncing Ball App, is faster in the two-node network than in MQTT. If a third node is added, the behavior roughly corresponds to that of MQTT. If a fourth node enters the network, an average of two points of contact can be sent relatively quickly. At the end of the day, when sending more than two simultaneous points of contact, there are occasionally very high latencies in five and six-node networks.

table 9: Transferable touchpoints with latencies of less than 1 second

node	2	3	4	5	6
NeCon	7	4	2	1-2	1
MQTT	4	4	4	4tj	4

Furthermore, the requirement of a dynamic change between the ConnectionModes could be realized. The connection to the existing network is only established when there is one to the network of the other technology. With the introduction of the room string, it was also possible to implement a visibility restriction for apps that use the same Nearfly API but should not see each other.

[6] It should be noted that if the transmission is too frequent, as in this case 90 messages per second, the smartphone itself can become a bottleneck.

The Score Board Notepad App is also well served as an application with little data volume through both ConnectionModes of the Nearfly API. In the NeCon ConnectionMode, especially with older devices, it can happen that the connection is terminated during a game session. Specifically, this occurs less often with the LG G2. This has also happened with the S7 and G5. This problem especially exists with the S5 Mini, which was therefore hardly used. In order to counteract the establishment of a connection, only messages with absolute values and a set retain flag are sent. The probability that a connection problem will arise should not be disregarded in other applications that use the Nearfly API in NeCon ConnectionMode.

6.5 Use of the Nearfly API

After the library has been successfully integrated and the mandatory authorizations entered in the manifest, the activity can be started. Before connecting to the NearflyClient, the mandatory dangerous authorizations must be granted, e.g. by calling the askForPermissions method[7] queried and by the application user (as in Illustration 19) be granted.

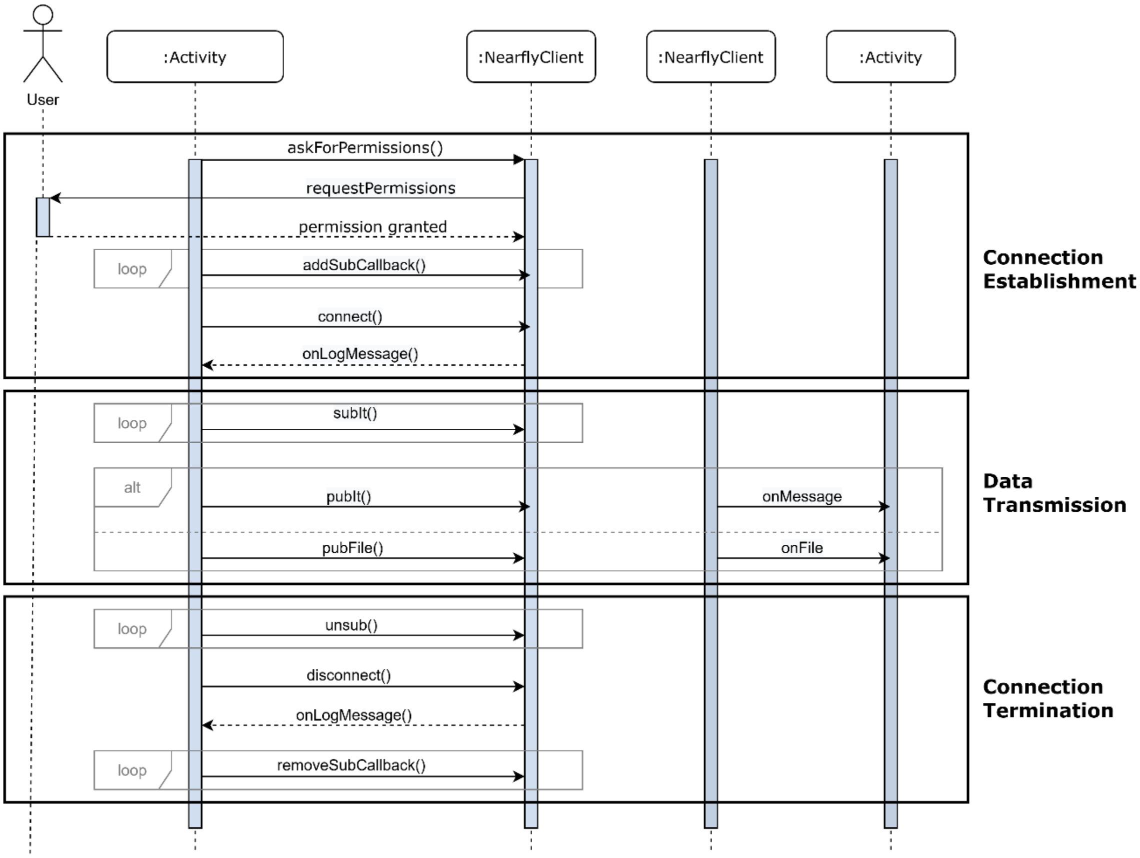


Illustration 19: Procedure for using the Nearfly API

[7] When using the Nearfly API via the NearflyService, it should be noted that the NearflyService method calls are only possible after the service has been successfully bound.

If the app has all the necessary authorizations, one or more NearflyListeners can be registered and the connection to the underlying technology can be established by specifying the ConnectionMode. If the connection was successfully established, the onLogMessage event is called up by the NearflyListener with the CONNECT keyword. This is particularly useful when using the Nearby Connections API, as the connection can take several seconds, and no data exchange is possible beforehand. It should be noted that the NeConAdapter defines a connection as an established network with at least two nodes. If more nodes are expected, this must be implemented by the API user.

You can now subscribe to any channels using the subIt method and then send any messages using pubIt and pubFile. The receipt of these messages triggers the onMessage or onFile event for the other network participants.

If the connection is to be terminated, the counter-operations corresponding to the connection establishment (as in Illustration 19) can be used in reverse order. According to this, the subscribed channels have to be disconnected by unsub and a disconnect has to be called. By logging off the registered NearflyListener, the NearflyClient is finally reset to its initial state.

7 Résumé

The implemented wrapper library allows Nearby Connections and MQTT to be addressed via a uniform API. This gives the API user the opportunity to create an application that allows communication between several smartphones both in local internet-free environments and in environments with internet access. The Nearfly API was created for this purpose, which extends the underlying technologies by using adapter components and enables alternative use without boilerplate code.

It has been shown that the network setup with Nearby Connections is currently quite slow and therefore only has a limited application. Choosing a different methodology, such as manually deciding on advertisers (root) and discoverer, speeds up network construction, but the current Nearby Connections API does not allow significant improvements in times. However, if the connection establishment is ignored, Nearby Connections shows good data rates when the hotspot is activated. MQTT, on the other hand, manages to establish connections quickly and achieve a good data rate with tolerable latencies.

If there is the possibility of internet access, MQTT should be preferred as ConnectionMode when creating networks with more than three network participants. Networks with two to three nodes are better served with the NeCon ConnectionMode and can exchange larger amounts of data at a much faster rate when the WiFi hotspot is activated. If a network consists of only two nodes, close real-time communication of up to 25 data packets per second is even possible with an activated hotspot for games. If the Internet is not accessible, the NeCon ConnectionMode can be used for applications with low data traffic, such as local survey apps or turn-based games for up to seven participants. However, the prerequisite also applies here.

Bibliography

[10] Banks, A, Gupta, R (October 29, 2014): MQTT Version 3.1.1 - OASIS Standard.

[16] E. Fallis, P. Spachos (2018): Power Consumption and Throughput of Wireless Communication Technologies for Smartphones. In:, 2018 Global Information Infrastructure and Networking Symposium (GIIS).

[18] Shah, Rahul C. and Nachman, Lama and Wan, Chieh-yih (2008): On the Performance of Bluetooth and IEEE 802.15.4 Radios in a Body Area Network.

Online sources

[1] Alexander Kunst (2017): Statista-Survey Telecommunication 2017. https://de.statista.com/statistik/daten/studie/722248/umfrage/umfrage-for-use-von-smartphone-funktions-nach-frequency-in- Germany/. Retrieved on June 4th, 2020.

[2] Zhang, L (2011): Building Facebook Messenger. https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920. Retrieved on April 13, 2020.

[3] Perez, S (July 31, 2017): Google opens its Nearby Connections tech to Android developers to enable smarter offline apps. https://techcrunch.com/2017/07/31/google-opens-its-nearby-connections-tech-to-android-developers-to-enable-smarter-offline-apps/. Retrieved on May 24, 2020.

[4] Boshell, B (2017): Average App File Size: Data for Android and iOS Mobile Apps. https://sweetpricing.com/blog/2017/02/average-app-file-size/. Retrieved on May 25, 2020.

[5] Harmon, W (2018): Google Nearby Connections 2.0 capabilities. https://stackoverflow.com/questions/51976470/google-nearby-connections-2-0-capabilities. Retrieved on April 13, 2020.

[6] Nearby Connections Team (2018): Nearby Connections API Guide. https://developers.google.com/nearby/connections. Retrieved on 04/04/2020.

[7] Harmon, W (2019): How performant is Nearby Connections? https://stackoverflow.com/questions/54434616/how-performant-is-nearby-connections/54470958#54470958. Retrieved on April 13, 2020.

[8th] Harmon, W (2018): How can I speed up Nearby Connections API discovery? https://stackoverflow.com/questions/52825617/how-can-i-speed-up-nearby-connections-api-discovery/52882054#52882054. Retrieved on April 13, 2020.

[9] Google I / O (May 19, 2017): How to Enable Contextual App Experiences (Google I / O '17). https://youtu.be/1a0wII96cpE?t=1819. Retrieved on June 4th, 2020.

[11] HiveMQ Team (2019): Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3. https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/. Retrieved on April 14, 2020.

[12] Roger Light (02/27/2020): mosquitto.conf man page. https://mosquitto.org/man/mosquitto-8.html. Retrieved on 04/04/2020.

[13] Ihlenfeld, J (April 22, 2009): Bluetooth 3.0 HS with WLAN acceleration - Golem.de. https://www.golem.de/0904/66633.html. Retrieved on May 17, 2020.

[14] Schnabel, P (05/17/2020 22:47:49): Bluetooth Low Energy (4.0 / 4.1 / 4.2). https://www.elektronik-kompendium.de/sites/kom/1805171.htm. Retrieved on May 18, 2020.

[15] Schnabel, P (05/17/2020 22:48:00): Bluetooth 5. https://www.elektronik-kompendium.de/sites/kom/2107121.htm. Retrieved on May 18, 2020.

[17] 4G.co.uk (oJ): How fast is 4G? https://www.4g.co.uk/how-fast-is-4g/. Retrieved on June 4th, 2020.

[19] Google (2019): App permissions. https://developer.android.com/guide/topics/permissions/overview. Retrieved on May 2nd, 2020.

[20] Harmon, W (2018): Be able to send Messages / Bytes Simultaneous to multiple devices using Nearby Connections. https://stackoverflow.com/questions/52773197/be-able-to-send-messages-bytes-simultaneous-to-multiple-devices-using-nearby-con/52785805#52785805. Retrieved on June 5, 2020.

[21] MacKenzie, D (2020): nice (1) - Linux man page. https://www.man7.org/linux/man-pages/man1/nice.1.html. Retrieved on June 2nd, 2020.

[22] Harmon, W (2019): NearbyConnection: payload.asFile.asJavaFile is null when retrying to saveFile after storage permissions initially not granted. https://stackoverflow.com/questions/55328027/nearbyconnection-payload-asfile-asjavafile-is-null-when-retrying-to-savefil/55345391#55345391. Retrieved on June 5, 2020.

[23] Siahaan, APU (2017): Base64 Character Encoding and Decoding Modeling. https: //10.31227/osf.io/ndzqp. Retrieved on June 4th, 2020.

[24] Harmon, W (2020): Is there a lower bound for the connection time of Nearby Connections API? https://stackoverflow.com/questions/61614967/is-there-a-lower-bound-for-the-connection-time-of-nearby-connections-api/61639915#61639915. Retrieved on June 5, 2020.