Measuring the NFC Peer-to-Peer Data Rate

Geoffrey Ottoy, Sam Van Den Berge, Jean-Pierre Goemaere,
and Lieven De Strycker

Abstract. NFC is a relatively new short-range wireless technology. For bidirectional communication between two NFC devices, the NFC Forum specifies the Peer-to-Peer (P2P) standard. Several challenges remain for exchanging data between a mobile Android device and an embedded device. First of all, the current versions of Android implement the NFC P2P specification only partially. A second challenge is the implementation of the NFC P2P protocol stack on the embedded platform. For the developers to be able to make an educated choice on whether to use NFC (P2P) or not, knowledge of the data rate is essential as well. In this article we provide an overview of these challenges. We also create a representative setup with NFC P2P stacks according to the specification on both the mobile and the embedded device. With this setup we are able to measure the NFC P2P data rate and compare it to NFC connection handover to WiFi.

1 Introduction

The motivation of this work has been based on some emerging trends. That is, smartphones will play an increasingly prominent role in people's lives. They will be used to gain access to all kinds of services, both online and at designated terminals such as vending machines, electronic storage lockers or access control terminals. Near-Field Communication (NFC) is one technology that can be used when a smartphone needs to communicate with a such a terminal.

NFC is already applied in several practical cases. For example, in ticketing applications NFC is gradually replacing paper tickets. This is, for instance, the case in Dutch public transportation or in the London Metro, with respectively the

Geoffrey Ottoy · Sam Van Den Berge · Jean-Pierre Goemaere · Lieven De Strycker
DraMCo Research Group
www.dramco.be
e-mail: info@dramco.be
KU Leuven, Campus Gent (KAHO Sint-Lieven),
Faculteit Industriële Ingenieurswetenschappen, Gebroeders De Smetstraat 1, 9000 Gent
e-mail: geoffrey.ottoy@kuleuven.be

L. De Strycker (ed.), *ECUMICT 2014,*
Lecture Notes in Electrical Engineering 302,
DOI: 10.1007/978-3-319-05440-7_9,

OV-chipkaart[1] and Oyster Card.[2] A first advantage is that NFC cards can be reused over and over again, which saves on ink, paper and storage [1]. But tickets are also increasingly being stored on NFC-enabled smartphones, which are less prone to loss. In Paris' and London's public transportation, commuters can use their smartphone as a ticket [2, 3]. NFC ticketing systems are furthermore assumed to increase the throughput of the public transportation [1].

Predictions state that in 2016, 13% of US and Western Europe citizens will use their smartphones as a ticket [4]. Manufacturers such as Asus, HTC, Nokia and Samsung have all released NFC-enabled devices. It is expected that 46% of all the smartphones will support NFC by 2016. The only uncertainty is Apple, which has to date not launched any NFC phones.

The NFC Forum[3] regulates the design of all NFC specifications and norms. Started in 2004, it brings together manufacturers and service providers. Among its members are NXP, Nokia, VISA, Samsung, etc. The main goal of the NFC Forum is to ensure interoperability of devices and protocols and thus help in building the so-called *NFC ecosystem*.

NFC Modes of Communication. NFC can be seen as an extension to Radio Frequency Identification (RFID) operation at 13.56 MHz defined by the ISO 14443 standard [5, 6, 7, 8]. This implies that for NFC two types of devices are defined:

- *Passive devices.* These devices –often referred to as *tags*– don't carry a battery and typically have very limited processing power and memory. The functionality ranges of storage of a simple ID (e.g., a classic RFID tag), over storage of (secured) data (e.g., a Mifare card, smart posters), to devices with a cryptographic co-processor (e.g., Java cards). Passive devices draw their power from an RF field generated by an active device.
- *Active devices.* An active device –also called *NFC-enabled* device– has its own power source. This can be either a net supply or a battery. It typically has more processing power than a passive device. Examples of NFC-enabled devices are (some) smartphones and access points.

One of the main differences between RFID and NFC is that with NFC, active devices are also able to communicate with each other. In this case both devices can alternately generate their own RF field when sending, or one of the devices can decide to act as a passive device. Because of the difference in communication between passive and active devices on the one hand and communication between two active devices on the other, the NFC forum has defined three different modes of communication:

- *Reader/writer mode.* This is the "classic" RFID communication. The active device acts as reader to read tags of the ISO/IEC 14443 A/B, Felica, etc.

[1] OV-chipkaart information website: `https://www.ov-chipkaart.nl/`

[2] What is Oyster? Transport for London – Information website: `http://www.tfl.gov.uk/tickets/14836.aspx`

[3] NFC Forum website: `http://www.nfc-forum.org`

- *Peer-to-peer mode (P2P).* In this mode, two NFC-enabled devices can exchange data with one another, e.g., digital business cards, an interesting URL, ... The data rate, however, is small. For this reason, a *connection handover* mechanism is used when large amounts of data have to be transferred. With this mechanism, all necessary parameters to set up a connection over, e.g., WiFi or Bluetooth, are transferred over NFC P2P. After the connection has been established, the NFC communication is terminated.[4]
- *Card emulation mode.* In this alternative to P2P, an NFC-enabled device will act as a traditional RFID tag. The main advantage is that it allows NFC smart phones to easily blend into the market, without the need to change the existing infrastructure (i.e., the existing card readers). However, in this mode the NFC chip of the smartphone communicates directly with a secure element (SE) on the phone (typically a SIM card), which eliminates intervention of the processor and the OS altogether.

The Need for Peer-to-Peer Communication. The number of applications that rely on NFC, and which implement some form of security (e.g., ticketing or mobile payments), is growing. Typically these security measures require bidirectional communication, for instance, advanced authentication protocols such as Idemix [10] or Direct Anonymous Attestation [11]. Currently, when bidirectional communication is required, the phone operates in card emulation mode. This is e.g., the case with *Google Wallet*,[5] or *Isis*.[6] However, this approach has several drawbacks.

Because the communication from the NFC chip is immediately routed to the SE, this element forms a bottleneck. First of all, payment apps are limited to the memory size of the SE. Second, the processing and access times to SEs are higher. A last drawback is that provisioning credentials to a SE is a complex process. In practice, this implies that every *e-wallet app* requires its own SE.

An alternative to using card emulation with SEs is the so-called *software card emulation*. Instead of passing the NFC communication to the SE, it is captured by an NFC service in the OS. This approach breaks the dependency on the SE i.e., credentials can be stored anywhere (e.g., in the phone's user memory, within a trusted execution environment (TEE), even in "the cloud"), but it also allows for several payment applications to use the NFC functionality.

Roland has written a comprehensive article on this subject [12]. His conclusion is that the main reason to go for software card emulation is that it does not require any changes to the existing payment infrastructure (i.e., the vendor terminals and network infrastructure). However, he also states that the most logical choice for payment and ticketing applications over NFC is to go for P2P because it was designed for easy communication between NFC devices.

[4] The connection handover mechanism is standardized in the ISO/IEC 18092 spec. [9]

[5] `http://www.google.com/wallet/`

[6] `https://www.paywithisis.com/`

With regard to software card emulation, BlackBerry is the only company that supports this approach. There have been patches submitted for Android,[7] but none have been merged with the main Android branch. The only way this can be achieved now on Android is by rooting the phone and installing a custom ROM like CyanogenMod;[8] something that would void your phone's warranty.

So why is card emulation used instead of NFC P2P? This is due to the current state of the technology. The NFC P2P specification, defined by the NFC forum, is the standard for bidirectional communication between two NFC devices. Unfortunately, the current version of Android implements this standard only partially.

Contribution. Our work focuses on NFC P2P communication between an Android mobile device and an (embedded) terminal. Android OS is one of the most-used operating systems for smartphones.[9] Moreover, developing apps for Android is easier compared to iOS or Windows 8 because no developer account is required. Also, when needed, root access to the phone can be enabled.

To that end, we will look at the current state-of-the-art, i.e., how is NFC P2P supported in Android OS, what is required to enable NFC P2P? Also on the side of the (embedded) terminal we add support for peer-to-peer communication by using two open-source libraries: `libnfc`[10] and `libnfc-llcp`.[11] Thus we are able to create a practical setup and use it to perform NFC P2P data rate measurements.

Outline. The remainder of this article is structured as follows. We start by giving some insight in the NFC P2P protocol stack (Sect. 2). Here we identify the key components and the current state-of-the art. In the next section (Sect. 3) we highlight the details of our practical setup. Section 4 then discusses the results we obtained. We state our conclusions in Sect. 5.

2 NFC Peer-to-Peer

2.1 Protocol Stack

For two active NFC devices to communicate with each other (e.g., a smartphone and a vending machine), the NFC Forum defines a *peer-to-peer* (P2P) communication mode. NFC P2P requires a stack of several protocol layers as shown in Fig. 1. The

[7] The patch by SimplyTapp adds software card emulation to the Android OS: `https://github.com/CyanogenMod/android_external_libnfc-nxp`

[8] `http://www.cyanogenmod.org/`

[9] According to IDC, Android and iOS Combine for 92.3% of all smartphone operating system shipments in the first quarter of 2013: `http://www.idc.com/getdoc.jsp?containerId=prUS24108913`. Android takes a market share of 75%. This share only includes the smartphone market and not, e.g., tablets.

[10] Project website: `http://code.google.com/p/libnfc/`

[11] Project website: `http://code.google.com/p/libllcp/`

SNEP layer and LLCP layer are defined by the NFC Forum itself. The lower layers are manufacturer and hardware dependent. In the following paragraphs, we will provide a short overview of all the layers.

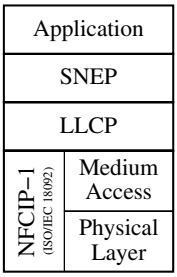


Fig. 1 NFC P2P communication stack

NFCIP-1. The ISO/IEC 18092 specification [9] describes the actual NFC communication i.e., how bits are physically transmitted from one device to another. In [13], the NFC forum has tried to make a more concrete description of the protocol.

The NFCIP-1 protocol works according to an initiator/target model. The initiator sends commands to the target, to which the latter answers. A target can never send data at its own initiative. An initiator is always an active device. Targets can be both active and passive devices. In active communication mode, a target will generate its own RF field when sending data to the initiator. A passive target relies on the RF field generated by the initiator. Through inductive coupling, the target can draw energy from the RF field. The target can be compared to the secondary winding and load of a transformer. By varying the load, the target can send data; the load variations can be detected by the initiator.

LLCP. The Logical Link Control Protocol (LLCP) provides to higher layers a logical connection between two endpoints [14]. This eliminates the inherent asymmetry of the lower NFCIP-1 layers with an initiator controlling the communication and a target device that only sends at a request of the initiator. LLCP creates a so-called *asynchronous balanced mode* (ABM) on top of this mechanism, through which each endpoint has the possibility to start a transmission.

Endpoints connect to the LLCP through *service access points* (SAPs). Typically, each protocol (endpoint) has its own SAP. For NFC, only SNEP has been assigned an SAP (`0x04`).[12]

Implementations on top of LLCP can either use connection-oriented or connectionless communication. Neither of these, however, have a guaranteed delivery time. This makes LLCP unusable for the streaming of audio or video. Other limitations are the fact that there is no possibility to send packets to different SAPs (multicast or

[12] The up-to-date list of assigned SAP addresses can be found at: `http://www.nfc-forum.org/specs/nfc_forum_assigned_numbers_register`

broadcast) and that there is no support for encryption or authentication. This should be implemented by higher layers.

SNEP. The Simple NDEF Exchange Protocol (SNEP) is a request/response protocol for exchanging NDEF[13] messages between a client and a server [15]. To exchange these messages SNEP uses the stable logic connection between two devices, provided by the underlying LLCP layer.

A SNEP message (Fig. 2) consist of four fields:

- The *version* field designates which version of the protocol is being used.
- The *request/response* field indicates the type of the SNEP message. Currently, four request codes are defined. We will only discuss the codes that are relevant for this article.

  `0x00` *CONTINUE*: The client indicates that the server can send another fragment of a fragmented SNEP message.[14] This can only follow a previously sent *GET*. The information field in a *CONTINUE* message is empty.
  `0x01` *GET*: This code is used to request an NDEF message (see later) from the server. The type of NDEF message is determined by the NDEF message that is being sent in the information field of the *GET* request.
  `0x02` *PUT*: Used, when the client wants to sent an NDEF message. The actual message is sent in the information field.

  The possible response codes are defined between `0x80` and `0xFF`. Again we only discuss the codes that are relevant for this article:

  `0x80` *CONTINUE*: Analogous to the request.
  `0x81` *SUCCESS*: Response to a *GET* or *PUT* request. In case of *GET*, the information field contains the requested NDEF message. Otherwise, the information field is empty.

  For a more detailed description and on overview of the other codes, we refer the reader to [15].
- The *length* field (4 bytes) indicates the length of the information field (number of bytes).
- The *information* field is used to send the actual NDEF messages.

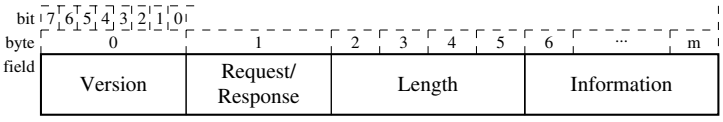


Fig. 2 SNEP message format

[13] NDEF stands for NFC Data Exchange Format. This is discussed later on in this article (p. 115).

[14] This is the case when the information that is being sent, is too large to fit in the information field of a single message.

NDEF Message Format. The NFC Data Exchange Format (NDEF) defines the format of data packets for NFC communication [16]. For P2P communication, when one application sends some information to another, it must encapsulate this information in an NDEF message. An NDEF message consists of one or multiple NDEF records. This can be either a normal record or a short record. Both have a comparable structure with the same fields. For detailed information on the NDEF record format and significance of the data fields, we refer the reader to [16]. Note that the main difference between a short and a normal NDEF record only lies with the allowed payload size.

2.2 Android NFC P2P Stack – API 16 and Higher

The mechanism that handles NFC P2P communication in Android is called *Android Beam* (introduced with API 14). The API provides support for the exchange of NDEF messages, with support for different types of data. There are methods to create the NDEF records and combine them into a message.

To send an NDEF message, the API provides two different approaches. The first one will register a static message to be sent when another NFC-enabled device establishes communication. The second method allows to dynamically create an NDEF message at run time. This is interesting when context-sensitive information needs to be sent.

To receive an NDEF message, an application needs to register the correct intent with the OS; in this case the `NDEF_DISCOVERED` intent. This is important because the *Tag Dispatch System* (Fig. 3) will try to immediately deliver all data received over NFC to the correct application. By using filters, the application can further specify if it is only interested in e.g., plain text NDEF messages. To prevent that a new instance of an application is started, every time an NDEF message is received, an active application can use the *Foreground Dispatch* to process the incoming intents.

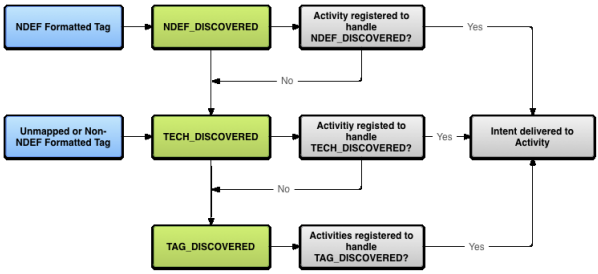


Fig. 3 Android tag dispatch system – Taken from `http://developer.android.com/guide/topics/connectivity/nfc/nfc.html`

If we look at the internal operation of the Android NFC service, one important flaw comes out, more specifically with respect to the NFC P2P specification. The procedure to send an NDEF message (whether it is a static message or composed at run time) is only called when the smart phone receives an *LLCP Link Activation* message. This message is part of the LLCP layer and is *only* sent when two devices are brought close enough to set up a connection. This means that, to be able to send multiple messages, the two devices have to be alternately brought together and moved away again [17]. It is obvious that this is highly impractical. It also implies that currently there is no bidirectional communication possible (with more than one pass), as is required in security protocols.

A possible explanation is that before the release of the NFC P2P specification (by the NFC Forum), Google had its own protocol for exchanging NDEF messages, the NDEF Push Protocol (NPP) [18]. In NPP it is specified that a client has to sever the LLCP connection after sending a NDEF message. It is likely that SNEP was implemented without making any changes to the underlying framework, which explains why this feature is still present.

We note again that this is the case for all Android versions to date.

3 Test Setup

Embedded Platform. With the help of an FPGA development board,[15] we implement a typical embedded platform. This consists of an embedded processor, in this case a 32-bit MicroBlaze™ [21], program memory, an Ethernet interface and an RS-232 interface. The MicroBlaze processor runs embedded Linux, which enables easy application development. Furthermore, it allows us to use open source libraries for NFC communication, as well as standard libraries for data manipulation, networking and input/output in general.

For the RF interface we use the NXP PN532C106 demo board [22]. The heart of this board is formed by the PN532 NFC chip. An antenna and matching network are also provided as well as a power supply circuit. The MicroBlaze processor controls the PN532 chip using the high-speed UART interface. The PN532 and successors such as the PN544 are among the most-used NFC chips in smartphones. The PN532 supports four different modes of operation:

- support for ISO/IEC 14443A (Mifare) and FeliCa as reader
- card emulation as ISO/IEC 14443A and FeliCa tag
- support for ISO/IEC 14443B only as reader
- NFCIP-1, required for NFC P2P; this is the mode that we use

Several open-source libraries exist that offer some functionality required for NFC P2P. One of the main requirements is that the libraries are able to run on an embedded platform. In particular we selected two open source libraries that work with

[15] We use the Xilinx ML605 development board [19], which houses a Virtex 6 FPGA [20].

Linux and that support the PN532. For the NFCIP-1 layer (mainly controlling the PN532) we use `libnfc`.[16] For the LLCP layer we opted for `libnfc-llcp`.[17] On top of that, we have written our own SNEP and NDEF implementation. Currently we only support short NDEF records. With this NFC P2P protocol stack, we are able to exchange messages with an Android smartphone.[18]

Android Phone. As pointed out before, bidirectional P2P communication with Android is currently not possible. To that end, we use a custom NFC P2P stack on the Android phone as presented in [17]. This solution uses the Java Native Interface (JNI) to create a link from Java applications to code written in e.g., C or C++. This code is typically time-critical code, or low-level driver code. Our replacement NFC service implements the NFC P2P stack as defined by the NFC Forum. This includes the NFCIP-1 and LLCP (both written in C) coupled through JNI to a Java API that is usable by applications that require NFC P2P.

Note the with this approach, the phone needs to be rooted because the access to the NFC chip is restricted to the root user or software that is signed by Google.

Another remark is that at the moment, the only solution that does not require root access or OS modification, is connection handover (standardized in ISO/IEC 18092 [9]). The drawback here is that this requires an extra wireless interface and that there is an increased connection setup time. The latter makes that this mechanism is only interesting when a certain amount of data will be exchanged.

Complete Setup. The embedded platform is connected via the on-board Ethernet connection to a WiFi router. With this setup we are able to evaluate both the data rate of NFC P2P, and the communication speed of NFC connection handover to WiFi (TCP/IP).

The complete setup is shown in Fig. 4. In the case of connection handover, the embedded platform transmits an NDEF message to the mobile device. This message contains the handover information that is required to set up the connection over WiFi, e.g., network ID, the platform's IP address, the TCP port to connect to.

As SNEP is a request/response protocol: a new message will only be sent when a *SUCCESS* response on a previous message has been received. We will use this mechanism to measure the time needed to send a certain number of bytes from the platform to the smartphone. At the moment when our embedded platform sends a SNEP *PUT* request, a time stamp is taken. The SNEP *PUT* message contains an NDEF record which in turn contains the actual payload. When the smartphone receives this message successfully, it will reply with a SNEP *SUCCESS* response. Upon receipt of this message at the embedded platform, a second time stamp is taken to measure the time needed to send a message with a certain payload size. Because the NFC stack on the embedded platform is influenced by the load on the OS, communication times will vary (several ms). To obtain an averaged result, we performed

[16] Project website: `http://code.google.com/p/libnfc/`

[17] Project website: `http://code.google.com/p/libllcp/`

[18] Using the standard "Android Beam" service as well as using our own NFC P2P service.

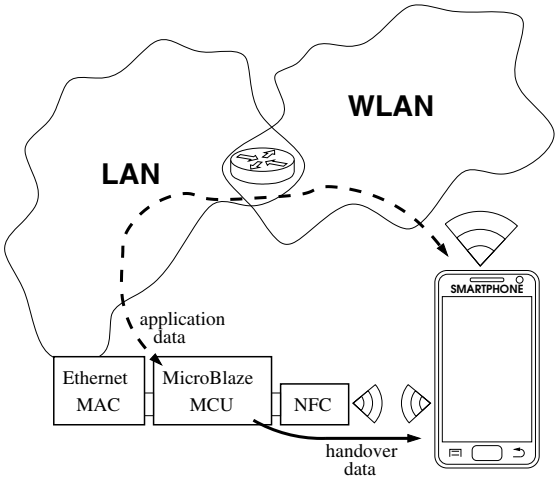


Fig. 4 Test setup

10 measurements for every payload size.[19] To eliminate the connection setup time in our data rate measurements we exchange data over a connection that has already been established.[20] Another simplification is that only short NDEF records are being used. The main reason is that the underlying hardware has a maximum payload length and thus normal NDEF records would have to be parsed over several LLCP messages anyway.

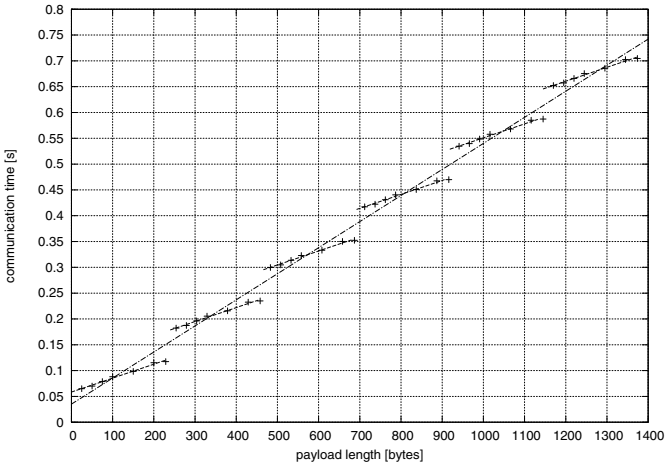
4 Results

NFC P2P Data Rate. We have set out the average time required to send an NFC P2P message (i.e., an NDEF record contained in a SNEP message) for different payload sizes in Fig. 5(a). By dividing the payload size by the time to send the data, we have obtained the effective data rate. This is set out in Fig. 5(b).

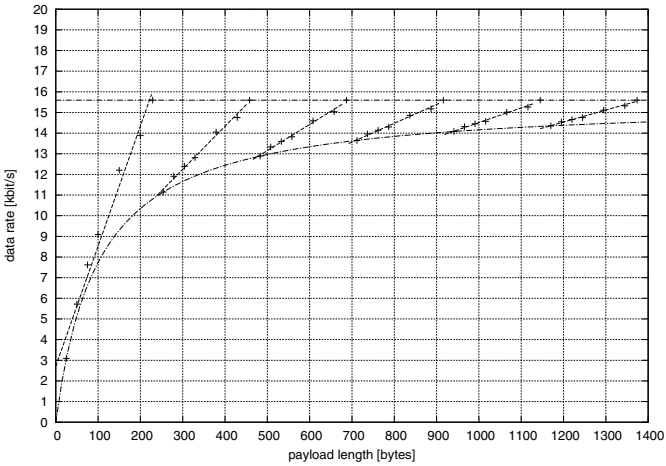
A first remarkable observation is that the time increases linearly in intervals of 229 bytes. Whenever the payload size exceeds a multiple of 229, an extra overhead is introduced. The reason for this is that 229 bytes is the maximum payload size that can be sent with one packet by the underlying layers. This means that when exceeding a payload size of 229 bytes (or a multiple) an extra message needs to be sent, which introduces overhead in the form of headers and the processing of intermediate SNEP *CONTINUE* responses.

[19] We limited ourselves to 10 measurements per payload, because the measurement itself is a tedious procedure that involves resetting the Android application and the embedded NFC stack, as well as "touching" the platform with the smartphone.

[20] Typically connection setup times for NFC are under 100 ms.



(a) Time required to send a number of bytes over an NFC P2P link (in one direction).



(b) Effective data rate for NFC P2P in function of the payload length.

Fig. 5 Speed measurements for NFC P2P communication

This obviously has its influence on the data rate as well. Only for payloads above 1024 bytes, the data rate more or less stabilizes at about 15 kbit/s, with a maximum of 15.6 kbit/s. This is only 3.7% of the physical data rate of 424 kbit/s.

NFC Connection Handover to WiFi. As an alternative to NFC P2P communication, we investigate connection handover to WiFi (TCP/IP). The timing measurements have been performed with the smartphone already connected to the WLAN. This eliminates variations due to the wireless router's response times. The time to

handover the connection has been measured as time difference between sending the handover data and accepting the TCP connection. Over this connection we have send a message of 683 bytes.[21] We have measured this transmission time as well. The connection handover with TCP/IP communication has been repeated 40 times.

We have generated a histogram for the handover timing, the time required for TCP/IP data transmission, and the total communication timing (handover and data transmission). This has been set out in Fig. 6. It is clear that the communication handover is responsible for a large portion of the total communication time. Moreover, it is impossible to do the handover under 0.4 seconds. This is more than the 350 ms for the total communication when NFC P2P is used. The TCP/IP data transmission on the other hand clearly outperforms the NFC P2P communication. This shows that communication handover only pays off when the amount of data to be sent is large enough. More detailed measurements, however, are required to determine the exact tipping point. Specifically for attribute-based credentials it should be investigated if it pays off to use handover, when more attributes are being used.

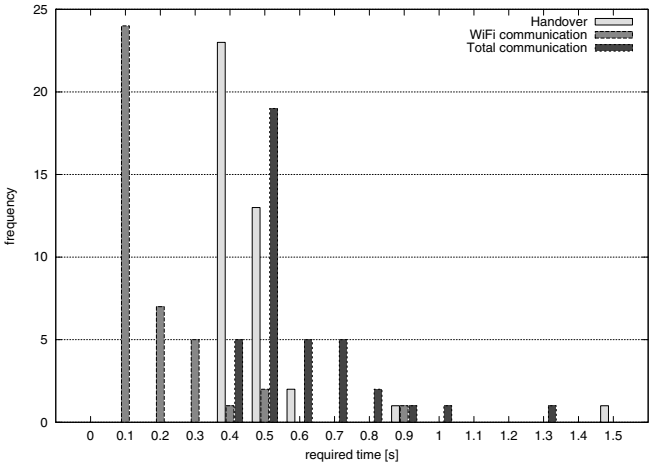


Fig. 6 Communication handover timing

5 Conclusion

As a first practical result, we have been able to determine the effective data rate for NFC P2P communication i.e., 15.6 kbit/s. These are likely the first figures published regarding this subject. However, more measurements are required to determine the

[21] As a practical example we took this value because it is the size of an Identity Mixer [10] credential proof (1536-bit modulus).

effect of the connection set up and the effect of transmitting application data in the other direction as well. Furthermore, when an new Android release implements NFC P2P (according to the specification), it would be interesting to repeat the measurements using this Android version and compare the results with our measurements.

A second conclusion is that NFC P2P is a valid candidate as communication standard when only small amounts of data need to be transmitted (e.g., up to 1024 bytes). Newer versions of the standard (at 848 kbit/s and higher) will result in even shorter communication times. For protocols that require more data transmission, communication handover could be used. However, more detailed measurements are required to determine the exact conditions for which it is beneficial to use either of the approaches. As these are preliminary results, also more research is needed to determine the influence of the protocol overhead, the stack implementation and the OS overhead on the data rate.

We must note that we have only measured the time of setting up a TCP connection and not the time required to connect to a WiFi network. This should deserve some further attention. Other future work could be targeted to comparing connection handover to other technologies such as Bluetooth or GPRS.

One could say that for the further evolution of NFC applications that require P2P communication, the ball is now in Google's court. On the other hand, we assume that eventually they will support the P2P standard. This is a reasonable assumption because 1) an NFC P2P standard is available and 2) applications, other than payment apps, could benefit from this as well.

References

1. NFC Forum, NFC in public transport – white paper (2011), http://www.nfc-forum.org/resources/white_papers/NFC_in_Public_Transport.pdf
2. Clark, S.: Paris commuters to get travel passes that work with NFC phones (2012), http://www.nfcworld.com/2012/01/30/312832/paris-commuters-to-get-travel-passes-that-work-with-nfc-phones/
3. McLean, H.: Transport for London to accept NFC payments from 2012 (2011), http://www.nfcworld.com/2011/07/12/38537/transport-for-london-to-accept-nfc-payments-from-2012/
4. The Point of Sale News, Where Is NFC Going? New Reports Forecast Growth (2012), http://pointofsale.com/20120319953/Mobile-POS-News/where-is-nfc-going-new-reports-forecast-growth.html (date consulted October 10, 2013)
5. ISO/IEC 14443-1:2008 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 1: Physical characteristics
6. ISO/IEC 14443-2:2010 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 2: Radio frequency power and signal interface
7. ISO/IEC 14443-3:2011 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 3: Initialization and anticollision
8. ISO/IEC 14443-4:2008 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 4: Transmission protocol

9. ISO/IEC 18092:2013 Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)
10. IBM Research – Zurich. Specification of the Identity Mixer Cryptographic Library – Version 2.3.2 (2010)
11. Brickell, E.F., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: Proceedings of the Eleventh ACM Conference on Computer and Communications Security, CCS 2004, pp. 132–145. ACM (2004)
12. Roland, M.: Software Card Emulation in NFC-Enabled Mobile Phones: Great Advantage or Security Nightmare. In: Fourth International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use (IWSSI/SPMU 2012), p. 6 (2012)
13. NFC Forum. NFC Digital Protocol Technical Specification (2010)
14. NFC Forum. Logical Link Control Protocol Technical Specification (2011)
15. NFC Forum. Simple NDEF Exchange Protocol Technical Specification (2011)
16. NFC Forum. NFC Data Exchange Format (NDEF) Technical Specification (2006)
17. Berge, S.V.D.: Onderzoek en realisatie van P2P communicatie tussen een Android smartphone en een embedded NFC terminal. Master's thesis, KAHO Sint-Lieven – Dept. I.I., Electronics-ICT, Jean-Pierre Goemaere, promotor (2013)
18. Google. Android NDEF push protocol specification (2011), http://source.android.com/compatibility/ndef-push-protocol.pdf
19. Xilinx®. ML605 Hardware User Guide UG534 (v1.8) (2012), http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf
20. Xilinx®. Virtex-6 Family Overview DS150 (v2.4) – Product Specification (2012), http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf
21. Xilinx®. MicroBlaze Processor Reference Guide – Embedded Development Kit EDK 14.1 UG081 (v14.1) (2012), http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf
22. NXP. AN10609_3 PN532 C106 application note Rev. 1.2 (2010)