Mittuniversitetet
MID SWEDEN UNIVERSITY

M.Sc Thesis within
Computer Engineering AV, 30 higher education credits

Implementations, simplifications and evaluations
around Nfc on Android

Johan Deckmar
Abstract

Near field communication (Nfc), a contact-range and short-lived message exchange technology, has in recent years become popular in relation to payment-cards, key-cards and ski-passes. With the release of, in particular, the Google Nexus S, which is capable of reading and writing Nfc tags as well as exchanging messages between devices by touch, the roles of consumers have changed from carriers of passive cards to that of active readers. This publicly available hardware technology, embedded into relatively cheap connected smartphones, creates a new field of possibilities in which a complete and complex Nfc-based system can be developed solely by means of software. In this thesis work, the research is in relation to the field of Nfc, ranging from the physical characteristics of the technology to the design of the Nfc API on the Android platform. Nfc-based apps, library and systems are designed, developed and evaluated in terms of performance. The Android apps which are implemented are WiFi and Bluetooth connectors as well as an Nfc-sensor value visualizer. Additionally, two full systems are developed which consists of an Android app, backend server, database and web or PC-client frontend. These two systems are: a web-based high-performance live statistical visualizer of Nfc sensor values and an Nfc-scan based product endorsement, Like-event, presenting the picture and name of a user endorsing a product on the in-store advertisement displays on top of the ads. The system for scanning, distributing and presenting statistics on sensor-values from a specially crafted NfcV-enabled sensor is evaluated in terms of performance along the chain of events starting at the Nfc tag introduction and being dispatched on the mobile phone and ending with the calculations and plot-generation carried out by the browser connected to the backend via a WebSocket. The evaluations show that in a use-case in which the user holds a tablet in one hand and an Nfc-enabled smartphone in the other, the user would experience a mean-delay of 515 milliseconds between putting the device against an NfcV-sensor and seeing the statistical plot, containing the last 1000 values, updated in the browser on the tablet. Additionally, the user would experience a mean-delay of 74 milliseconds between the values being presented on the mobile device itself until they are presented on the tablet in the case of constant streaming from an NfcV connected sensor. The conclusion is that this delay of 74 milliseconds is sufficiently low for the user to feel that the devices are tightly coupled, making it natural to pay attention to the tablet when scanning sensors. Possible use-cases for this include a technician walking between built-in passive NfcV sensors on the factory floor or a nurse walking between patients wearing bandages with built-in passive health-sensors. Finally, the work is discussed in terms of ethics and social impact as development in the tele-healthcare field continues, and a set of Nfc-based applications are suggested including Nfc-tags for interactive product customization, company car management and app deployment.

Keywords: Near field communication, Android, Nfc library, tele-healthcare, NodeJS, Redis, dnode, KnockoutJS, HTML5, WebSockets.
Acknowledgements

Thanks to...

Dr. Johan Sidén, Mid Sweden University for providing ideas, use-cases and self-made NfcV-enabled sensors used in large parts of this thesis.

Andreas Svanberg, Dewire Telecommunication Systems for bouncing ideas about apps and evaluations.

Sebastian Berggren, Netcorner Europe for providing the virtual server infrastructure used in this thesis and opportunity to test the Nfc Like-app.
## Contents

**Acknowledgements**  
2

**Terminology**  
6

### 1 Introduction  
9  
1.1 Background and problem motivation 9  
1.2 Overall aim 9  
1.3 Scope 9  
1.4 Concrete and verifiable goals 11  
1.5 Outline 11  
1.6 Contributions 11

### 2 Near field communication  
14  
2.1 NFC Forum Tag Types 15  
2.2 Near and far field  15  
2.3 ISO-14443 Proximity cards 16  
2.4 ISO-15693 Vicinity cards 16  
2.5 Differences between NFC and RFID 17  
2.6 NFC modes 17

### 3 NFC Data Exchange Format - NDEF  
18  
3.1 NDEF Message format 18  
3.2 NDEF Record format 18  
3.2.1 The type field 18  
3.2.2 The id field 19  
3.2.3 The payload field 19

### 4 NFC on Android  
20  
4.1 Basic NFC operation support on 2.3.3+ and 4.0+ 20  
4.2 The Android Activity lifecycle 20  
4.3 The Android NFC Dispatch System 21  
4.3.1 Two registration-points: Foreground-listening and Manifest 22

### 5 Methodology  
24  
5.1 Development process 24  
5.2 Hardware tools 24  
5.2.1 Android mobile 24  
5.2.2 Virtual server 24  
5.3 Software tools 24
5.3.1 Editor and Integrated Development Environment 25
5.3.2 Software deployment 25
5.3.3 Web authentication and navigation 25
5.3.4 Design- and image resource tools 25
5.4 Evaluation of system performance 25
   5.4.1 Point-description and methodology of measurements 26

6 Design 28
   6.1 Nfc library for Android 28
      6.1.1 Library use-cases and needs behind the design 28
      6.1.2 Helper-class and optional base-class 29
      6.1.3 Nfc to Bluetooth Handover 30
   6.2 Low latency multi-platform system 30
      6.2.1 Technology motivation 31
      6.2.2 Eliminating the delay-bottlenecks in a web-based system 31
   6.3 Wifi Connector and Writer 33
      6.3.1 Real-world scenarios 33
      6.3.2 UML design 33
   6.4 TimeSpot 33
      6.4.1 Functionality and use-cases 34

7 Results 35
   7.1 Android Nfc Library 35
      7.1.1 Class overview and usage 35
      7.1.2 Managing the Nfc life-cycle 35
      7.1.3 Nfc to Bluetooth handover 36
   7.2 Low latency multi-platform system 37
   7.3 Nfc to WiFi Writer and Connector 38
      7.3.1 Requirements and design 38
      7.3.2 Example NdefMessage 39
   7.4 TimeSpot 40
      7.4.1 User experience 40
      7.4.2 Keeping records in offline mode 40
   7.5 Nfc Like-tag to Digital Signage greeting 40
   7.6 Evaluation of sensor-system performance 42
      7.6.1 Time from tag introduction to app delivery 45
      7.6.2 Android NfcV tranceive-call time 45
      7.6.3 Time-of-flight from mobile to backend 47
      7.6.4 Time to store sensor-value in Redis database on backend 47
      7.6.5 Time-of-flight from backend to browser 48
      7.6.6 Calculating and plotting data-sets in browser 48

8 Conclusions 51
   8.1 Thesis goal and result 51
      8.1.1 Research 51
      8.1.2 The Nfc library 52
      8.1.3 Implemented Android apps 52
      8.1.4 Implemented server and frontend 53
   8.2 Evaluation summary 53
8.3 Societal impact ................................................. 54
  8.3.1 Ethics .................................................. 55
8.4 Related work .................................................. 55
  8.4.1 Android 4.1 Jelly Bean features ......................... 55
8.5 Future work .................................................. 55
  8.5.1 Nfc library features ..................................... 55
  8.5.2 Suggested applications ................................. 56

References ...................................................... 58
## Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad</td>
<td>Advertisement, a short film-, image- or text-message shown to potential customers to inform about certain products whose aim is to increase sales.</td>
</tr>
<tr>
<td>Android</td>
<td>Android is an operating system developed by Google, used on smartphones and tablets.</td>
</tr>
<tr>
<td>App</td>
<td>The term app originates from the word application and is used for smartphone, tablet and desktop applications distributed through a controlled marketplace. An app often has a small focused feature-set.</td>
</tr>
<tr>
<td>Backend</td>
<td>A server infrastructure consisting of one or several cooperating servers, accessible from clients via TCP/IP. The word backend includes the whole server infrastructure, not only one particular hardware or software server.</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>A radio communication technology used between a small group of active devices usually within 10 meters, but in some cases up to 100 meters (Class 1).</td>
</tr>
<tr>
<td>Frontend</td>
<td>A client application, connected to a backend, showing information and/or taking user input and actions.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface, a visual interactive interface shown to the user. The interface allows the user to make actions and updates when new information becomes available. Often includes navigation menus, buttons, text-fields and information.</td>
</tr>
<tr>
<td>Terminology</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>Handover</strong></td>
<td>In a handover one communication technology is used to exchange the necessary information to establish another connection using the same or a different technology. Handovers can occur between mobile cell-towers when the user travels while using the mobile - the mobile will automatically switch to the tower with the strongest signal. In this work a short-lived Nfc channel is used to exchange the necessary information in order to establish a Bluetooth connection without the need for user interaction - referred to as a Nfc to Bluetooth handover., 6</td>
</tr>
<tr>
<td><strong>Library</strong></td>
<td>In programming, a library is a module of code that manages a set of tasks. This module is then included in many different programming projects to save time and effort for the developers. A library is not an application of its own but is rather a generically written part that can, often, be used in many different ways depending on the requirements used in the project., 6</td>
</tr>
<tr>
<td><strong>Nfc</strong></td>
<td><strong>Near Field Communication</strong>, a technology enabling communication between two devices in close proximity, commonly up to 4cm or up 1.5m depending on antenna limitations. Using induction, a small battery-less micro-controller attached to an Nfc antenna, known as a passive tag, can be powered and is able to exchange a stored or generated message with an active Nfc reader. With an Nfc-enabled smartphone tags can be read from and written to. Additionally, two Nfc-enabled smartphones can exchange messages representing, for example, contact information. Nfc-connections are short-lived and frictionless - requiring no password or pincode for authentication before a message can be exchanged., 6</td>
</tr>
<tr>
<td><strong>RFID</strong></td>
<td><strong>Radio Frequency Identification</strong> is similar to Nfc in terms of some physical characteristics but is generally for read-distances up to several meters, making RFID suitable for inventory tracking., 6</td>
</tr>
</tbody>
</table>
RTT  
Round-trip time, the time it takes for a message to be send from A to B, then from B back to A. Measuring RTT does not require synchronization of clocks since only A needs to check its own time before sending and after receiving the round-trip message., 6

Socket / WebSocket  
A socket is an established connection between two applications, most likely running on different devices. A socket is used for reading and writing arbitrary data back and forth between the applications, following a pre-defined protocol so that the data can be interpreted correctly. Socket reading and writing is bi-directional and asynchronous. A WebSocket is an HTML5 feature and in which JavaScript is used in a modern browser to open a socket to a server that supports the WebSocket handshake protocol. It can then be used to stream data to and from the browser asynchronously and with low overhead., 6
Chapter 1

Introduction

Near Field Communication (Nfc) is a contactless technology enabling wireless communication within a range of a few centimeters. A short-lived Nfc channel is established by touching a reading device, such as a mobile phone, with another device or Nfc tag. During this connection a small amount of information, such as contact information, payment information or a passkey for handover to a longer distance technology, can be exchanged.

The technology is an enabler by design since it requires no setup from the user other than touching a device or tag with the phone. How the information is used after it has been exchanged depends on the receiving app. The information could, for example, be a store-name followed by a list of items and their prices for a mobile payment system, resulting in the payment app requesting the user’s pin-code to complete the payment transaction. The transaction could then be completed securely over an encrypted mobile Internet connection, leaving the Nfc as merely being a convenient enabler.

1.1 Background and problem motivation

Contactless smartcards and tags have become items of everyday use ranging from keycards and bus-cards to ski-passes and asset tagging. Tags are cheap, small, battery-less and easily managed in the form of stickers, plastic cards or key fobs.

Systems built on contactless technologies first came into commercial use in mid 1996 in Seoul, the capital city of South Korea, with their public transport payment card Upass[1]. In Europe, Nfc technology came into large-scale public use in mid 2003 with the London public transport payment card, Oyster. Oyster and Upass are based on a MIFARE, which adds an encryption technology on top of the ISO-14443 specification. ISO-14443 describes contactless smartcards and is the basis for many RFID and Nfc specifications such as NfcA[2], NfcB[2], MIFARE[3] and Calypso[4].

Nfc systems have, since they first came into commercial use, generally been set up such that the passive tags are in the hands of the customers and the active readers belong to the maintainers of the system. This changed in 2006 with the advent of the Nokia 6131 mobile phone, the first commercially available mobile phone capable of reading and writing Nfc tags[5][6][7].

When communicating between two devices using Nfc technology only one device is required to be active, which means that a reader can supply power for the microprocessor in the passive card, as well as allowing it to reflect RF energy back, in order to
communicate. Because of this, contactless smartcards do not require a battery, which simplifies and lowers the cost of production and usage.

With the release of Google Nexus S in December 2010, NFC technology has received a significant amount of publicity among developers, especially within the Android community. Google presented NFC as an enabler of frictionless sharing via smart posters and pairing between devices. With the word *frictionless* they are referring to the setup-less and password-less proximity-initiated information sharing or scanning, which requires nothing from the user other than holding the device towards a tag or another NFC-enabled device. For applications supporting contact sharing or connectivity handover to WiFi or Bluetooth, such as multiplayer games, a requirement is that at least one of the devices should have that contact or app in the foreground. An advantage of having NFC support on a smartphone platform, with an active app ecosystem, is that new features can be made globally available to customers by simple software app downloads. This enables disruptive innovation solely by, or almost solely by, software development since the potential customers already have all or part of the required hardware in their smartphone.

Research at Mid Sweden University aims to have RFID/NFC-enabled sensors printed into containers of everyday products. Both manufacturers and customers could then use RFID or NFC readers to view and track temperature, volume, moisture and potential transport shocks for products. These built-in sensors could also be used to create a customer-centric community around a shared passion for a type of product or the tracking of an item or production-batch. Another use of NFC-enabled sensors is in relation to health-care in which the patients themselves are able, in their homes, to scan an attached bandage with built-in sensors. By aggregating this data to a backend server, using the mobile Internet connection, the patients, their doctor and their families are able to monitor health trends and receive alerts if conditions change. Sidén et al. has predicted a growing interest in tele-healthcare and assisted living, especially among elderly, fueled by limited hospital resources and an increasing familiarity with smartphones among the elderly[8]. Even low-cost smartphones and tablets are packed with technology and stable connectivity allowing them to take a central place in bi-directional tele-healthcare and assisted living.

The completion of this thesis would provide insights for developers using NFC on Android. Additionally, a working high-performance system, containing similar important technology-parts as for a tele-healthcare system, is implemented and evaluated for performance at several points of interest, including the delay and bandwidth of scanning an NFC-temperature-sensor using a smartphone. The aim of the system is to assist in the development of new NFC sensor hardware and to provide a flexible proof-of-concept platform for demonstrating different NFC-sensor use-cases.

### 1.2 Overall aim

With the release of the NFC-enabled Samsung Nexus S in 2010[7], one in a line of Google Android-developer smartphones, the NFC technology has become available in a mobile device with an established developer community and app market ecosystem. An always-connected active NFC device, which is capable of reading/writing tags and peer-to-peer messaging, creates the conditions for innovative new systems to be developed without the necessity for significant costs or hardware manufacturing/maintenance.

The overall aim of this thesis is to explore the NFC technology and its possibilities by implementing a number of different NFC-based applications on the Android platform and, in so doing, identifying some key API use-cases and extracting these into a new...
open source Android Nfc library. Additionally, high-performance backend and frontend applications are created to distribute and visualize Nfc-based data and events.

The exploration of the Nfc technology includes the examination of some physical properties of a couple of relevant Nfc specifications in addition to exploring the programming interface created by Google for Android. This exploration has resulted in a number of theoretical chapters describing, for each respective field, characteristics, usage and how they relate to this thesis work.

The implementations in this project consists, in the first part, of a few isolated Android apps focusing on an Nfc-based key-less connectivity handover, such as a WiFi connector and device-to-device Bluetooth handover, as well as a sensor value visualiser for sensors which are built into packaging. The implementations consists, in the second part, of larger systems in which an Android app, a backend server and a PC client or web browser communicate in real-time to instantly visualize the following two types of Nfc scans, namely, a flow of sensor-values measured by a specially crafted NfcV sensor and the presentation of the customer endorsement of a product on the in-store Digital Signage monitors following the Nfc scan of a coupon tag. In addition, an Android application utilizing Nfc-tags for time-stamping has been developed for an existing system.

The evaluation in this project focuses on performance measurements in several points-of-interest. These points affect, to some extent, all apps and systems implemented in this project and range from Nfc polling delay and throughput, to streaming delay and throughput from the mobile via the backend to the browser or client software.

1.3 Scope

The theoretical chapters of this thesis aim to provide a knowledge base for the reader by explaining selected parts of the technologies that form the basis of this thesis. In the case of Near Field Communication, the physical workings and characteristics are explained to a target audience with entry-level knowledge in electronics. Layers between the physical and application, specifying details such as coding and anti-collision, are not dealt with since they are neither required to show how the technology works nor to explain the performance metrics.

The implementations in this thesis work are meant to be working proof-of-concept systems or apps and not as completed apps ready for distribution, with exception of the TimeSpot app. This means that some user interfaces are built to show only one or a few pre-defined use-cases, leaving general cases and adaptation for future work. Examples of this include the WiFi Connection designed to support only WPA-PSK networks and the Nfc Like app only showing one person promoting one product.

The evaluations in this thesis work provide detailed performance metrics about the systems implemented. A comparison between similar platforms or similar technologies are discussed in the system design, but no measurements are made using other platforms or technologies. Thus some performance claims made in the derivation of the system-design remain untested and unproven.

Security, alone, in Nfc is a broad and important topic studied by many security researchers. Because of the implementation, and the proof-of-concept-oriented nature of this thesis, security is omitted from both the theory and implementation and in viewed as part of future work in case a system is to be adapted for larger-scale use.
1.4 Concrete and verifiable goals

In this thesis the Nfc technology will be examined in terms of the physical characteristics, specification types, operational modes, message exchange data format and Android platform API design. This examination will result in three theoretical chapters answering the following questions:

- What is Near field communication? Which companies use it and for what?
- How does Nfc-communication work physically between devices and passive tags?
- What is the difference between proximity- and vicinity-based specifications?
- What is the difference between Nfc and RFID?
- Which operation-modes does Nfc support?
- What is the appearance of the Nfc data exchange format?
- How is the Nfc API designed on the Android platform?

The implementations in this thesis project will touch on the areas of: connectivity handover, user position- and time-logging, contactless sensor-polling, multi-platform data distribution and data-visualization. The following applications will be designed and implemented:

- An Android library handling the Nfc foreground life-cycle and reading/writing tags with Ndef messages
- An Android library handling an Nfc-to-Bluetooth connection handover
- Android application for storing credentials for a WPA-PSK WiFi network on a tag
- Android application for connecting to a WPA-PSK WiFi network upon scanning a prepared tag
- Android application for polling an NfeV sensor and streaming the values live to a backend
- NodeJS backend for receiving, storing and distributing streamed data between Android devices and web browsers.
- HTML5 web-application for visualizing sensor values being pushed via a WebSocket from the backend
- Android application for stamping in and out of work using Nfc tags
- Android application in relation to liking a product via a tag
- NodeJS backend to receive and distribute like-events
- A PC C# application for displaying like-events on in-store digital advertisement screens
CHAPTER 1. INTRODUCTION

The evaluation in this thesis consists of performance measurements in the important traffic points inside and between the applications and the hardware. The performance will be presented as a set of statistical summaries and, where appropriate, plots will be produced showing performance change in relation to varying input sizes. Performance will be evaluated for the following metrics:

- Time between the introduction of a tag to the mobile and the tag information delivery into app
- Time between the NfcV sensor POLL request and a value response
- Time-of-flight for a small packet from the mobile to the backend over an open socket (TCP)
- Time between the incoming value and the database write completion in the backend
- Time-of-flight for a small packet from the backend to the browser over an open WebSocket (TCP)
- Time between receiving data in the browser and calculating/generating plots

1.5 Outline

Chapters 2, 3 and 4 are theory chapters describing relevant background information on Near field communication, the NDEF message format and the Nfc API on Android respectively.

Chapter 5 details the methodology of the performance evaluations conducted on a system implemented in this thesis.

Chapter 6 details the design of all Android apps developed in this thesis, as well as the Nfc library, backend and web and PC-client frontends.

Chapter 7 presents the results of all implementations and evaluations.

In chapter 8 the results are summarized and discussed in terms of performance, ethics and society impact. The work performed in this thesis is compared to related work and additional Nfc-based features, applications and systems are suggested as future work.

1.6 Contributions

The Android apps, backend and frontend systems have all been designed and implemented by the author during the six month thesis period, with the exception of the app TimeSpot, which was implemented by the author before the thesis period as consultant work. The use-cases and backend for TimeSpot were provided by the company ordering the development of the app.

The requirements behind the idea of the Nfc Sensor Visualizer, in which the remaining contents of a wine-box are displayed upon scanning a fictive built-in Nfc weight sensor, originated from Dr. Sidén at Mid Sweden University. The NfcV sensor tags used in this thesis were provided by Dr. Sidén and the main use-cases of the Nfc Sensor Reader system, as a tool for the further development of Nfc-enabled sensors and as a flexible proof-of-concept system for tele-healthcare, were inspired by Dr. Sidén´s papers within the field of smartphone-based tele-healthcare and assisted living.
Chapter 2

Near field communication

Near field communication (Nfc) is a short-range wireless communication standard. It works on a distance limited by a fraction of one wavelength of the carrier frequency, and is additionally limited by the size of the antenna in comparison to the wavelength\[9\][10]. In practice, this results in Nfc communication being possible only within 4 cm of the Samsung Nexus S mobile phone. This means, the mobile phone must almost touch an Nfc tag in order to power it, connect and query its content.

A common use-case is to have an Nfc buscard for payment, where, when being pressed against the active reader, part of the leather/cloth wallet and the user’s hand are in between the card and the reader. The distance and RF attenuation in this payment use-case is close to the maximum possible with the proximity-card specification and card-sized antenna diameter used, which is beneficial from a security and anti-collision standpoint. Another common use-case involve vicinity cards used to access ski-lifts. Vicinity cards can operate at a minimum field strength of 150 mA/m\[11\], as compared to the minimum strength of 1500 mA/m\[12\] specified for proximity cards. This translates, in practice, into a communication distance of up to 1.5 meters for vicinity cards if the reading antenna(s) are sufficiently large, thus enabling the reading of a card which has placed in the clothing of a person walking between two stationary antennas.

Nfc is designed to work between two nodes and in which at least one must be active. An active device creates an RF field around its Nfc-antenna by supplying the antenna with power in the specified frequency. In the case of all Nfc specifications mentioned in this thesis, the operating frequency is $13.56MHz \pm 7kHz$\[11\]. The RF power field from an active Nfc device is sufficient to power the microprocessor in a passive Nfc tag by means of magnetic induction and allows such a tag to reflect RF energy back to the reader in order for communication to take place.

The main Nfc standards are developed by the non-profit organization Nfc Forum, started 2004 by Philips, Sony and Nokia. The Nfc Forum ensures interoperability by creating, categorizing and freely providing specifications for communication and data formats (NDEF) which enables new readers and devices to build applications. The Nfc Forum currently has over 160 members (June 2012) including Google, Microsoft, VISA, PayPal and Assa Abloy\[13\].

Nfc enables information transfer without the need for searching, pairing or password input as is the case for Bluetooth and WiFi. This can be achieved since the close proximity necessary for discovery and communication greatly reduces the risk of unintended connections and makes a nearby man-in-the-middle difficult to remain unnoticed.

On the Android platform the Nfc chip is only powered when the screen-lock is off,
meaning that the user is actively using the mobile device. This removes the possibility of attacking a mobile device via the NFC while it rests in the pocket of an unsuspecting user[14].

2.1 NFC Forum Tag Types

The NFC Forum has defined four types (or categories) of NFC technologies. These types have been created to ensure interoperability between tag technologies and devices. The four NFC Forum standardized tag types are the following[15]:

<table>
<thead>
<tr>
<th>Forum Type</th>
<th>Based on</th>
<th>Read/Write</th>
<th>Read-only lockable</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>ISO/IEC 14443A</td>
<td>R/W</td>
<td>Yes</td>
<td>96B up to 2KiB</td>
</tr>
<tr>
<td>Type 2</td>
<td>ISO/IEC 14443A</td>
<td>R/W</td>
<td>Yes</td>
<td>48B to 2KiB</td>
</tr>
<tr>
<td>Type 3</td>
<td>JIS X 6319-4¹</td>
<td>Factory set²</td>
<td>Factory set²</td>
<td>Up to 1MiB</td>
</tr>
<tr>
<td>Type 4</td>
<td>ISO/IEC 14443</td>
<td>Factory set²</td>
<td>Factory set²</td>
<td>Up to 32KiB</td>
</tr>
</tbody>
</table>

NFC Forum Tag Types and some of their specified properties

[1] JIS X 6319-4 - Also known as FeliCa.

Tags in these categories are supported by Android under the names of NFC A, NFC B and NFC F. Tags can have additional special features, whose support may or may not be implemented by the Android NFC API, but which can be implemented manually via the transceiver-method if the protocols are known. Tag identification in the Android API consists of a list of supported technology classes and the ability of transforming generic Tags to a class-object representing one of the supported technologies for that tag. This means that any, possibly proprietary, tag building on top of a type in Table 2.1 can be used with Android mobiles since they follow the standardized specifications of communicating.

2.2 Near and far field

When an antenna radiates RF energy at a certain frequency, \( f \), the characteristics of the electromagnetic field around the antenna vary at different distances from the antenna.

Firstly, the distance from the antenna can be divided into a near-field and a far-field region. The border between the near-field and the transition to the far-field is about one wavelength

\[ \lambda = \frac{c}{f} \]

away from the antenna, if the antenna length is half or less of the wavelength. An antenna with a length smaller than half the wavelength is called an electromagnetically short antenna. In the case of NFC where

\[ f = 13.56[MHz] \]

then

\[ \lambda \approx 22.1[m] \]

Secondly, the near-field is divided into a reactive region closest to the antenna and a radiative region extending out towards one wavelength. The extent of the reactive
near-field depends on the properties of the antenna such as its size in comparison to the wavelength.

A characteristic in the near-field of an antenna is that it is inductive, meaning that it can induce a current (transfer electrical power) into correctly constructed antennas within this region. An additional characteristic, limited to the reactive near-field region, is that a load (resistance) on antenna B, present in antenna A’s reactive near-field, feeds back causing an increased load on antenna A. In other words, it is possible for antenna A to provide an electromagnetic field, drawing a certain power from its power-supply and only when antenna B, in the reactive near-field, applies (or increases) a load on its antenna will the load on antenna A’s power-supply increase.

Near-field communication is based on the inductive characteristic of the near-field region for supplying passive tags with the power required and on the feedback characteristic of the reactive near-field region for communicating back to the reading device with load modulation by varying the load it places on its antenna. This is the main characteristic shared by all NFC technologies and is also reflected in the name of the technology itself, namely, Near-field communication[10].

2.3 ISO-14443 Proximity cards

The standard ISO-14443 “Identification cards - Contact-less integrated circuit cards - Proximity cards” provides the base specifications for short distance one-to-one radio communication. This standard is a base in some, but not all, RFID and NFC standards such as NFC and MiFAR[20]. ISO-14443 specifies these, among other, properties[10]:

Carrier frequency 13.56MHz

Bit rate 424kbps

Modulation ASK

Field strength range 1.5 to 7.5 A/m

Anti-collision

Frame bit pattern (Start) (8 bits) (Odd parity) (End)

2.4 ISO-15693 Vicinity cards

The standard ISO-15693 “Contactless integrated circuit(s) cards - Vicinity cards” specify contactless cards similar to ISO-14443 but, operating at a distance up to 1.5 meters. The standard specifies, among others, these properties[10]:

Carrier frequency 13.56 MHz

Bit rate Long distance mode: [1.65 kbps to card and 6.62 kbps from card], and Fast mode: [26.48 kbps to and from card]

Modulation ASK

Field strength range 0.115 to 7.5 A/m

Anti-collision
Frame bit pattern (Start) (Byte 1) ... (Byte N) (End)

The higher maximum reading distance arrives from several differences. Firstly, ISO-15693 requires the contactless cards to operate at a field strength of as low as $0.115 \text{A/m}$, which is about $1/10$ compared to that of ISO-14443. Secondly, ISO-15693 has a long-distance mode, which is chosen by the reader, where the bit rate is $1/4$ to the card and $1/16$ from the card in comparison to the fast mode. Both these differences result in a lower bit-error rate, allowing the tag to be read from further away.

### 2.5 Differences between Nfc and RFID

One main difference between Nfc- and RFID-types of technologies is in their practical read-distances. RFID technologies use the same principles of power induction into passive tags and load modulation on the reader’s field to send data back to the active reader[10]. Many RFID specifications use frequencies other than the 13.56 MHz used by Nfc, which possesses other characteristics, such as a longer near-field and/or a higher allowed field strength resulting in longer read-distances[21].

### 2.6 Nfc modes

At least one of the two devices communicating over Nfc must be active, meaning that it is supplying an RF field. The power in this field is sufficient to power the chip in a close by passive tag, which allows for the second device (often a tag) to be without its own battery or power supply. This scenario is called read/write mode and takes place between an active and a passive device[22].

If two active devices are introduced, they are able to initiate a peer-to-peer communication in order to exchange data. An example use-case is the exchange of a contact between two mobile phones. In the case of larger transfers, or in the requirement for a persistent connection, the mobile phones can use the peer-to-peer Nfc mode to only exchange credentials in order to set up another connection with a longer range such as a Bluetooth or Ad-Hoc WiFi connection[22].

Before mobile Nfc was introduced, many systems were created which are targeted to work only with passive cards and not in peer-to-peer mode. To be compatible with these existing Nfc systems, a third mode called Card Emulation was specified. This allows the Nfc chip in mobile phones to emulate different card technologies and thus use some of the older systems. The mobile then appears to the reader as being a passive card[22]. It is important to note that the security of these systems is not bypassed since card emulation is performed by direct communication between a Secure Element in the mobile and the active reader. This means that a cryptographic relationship can be established between the mobile phone and, for example, the Google Wallet system. The cryptographic keys are stored and used only within the separated secure element chip and thus cannot leak out to a specially crafted malicious app on the mobile phone. Card emulation is, as yet, not supported on Android devices, and no documentation in relation to it is available from Google (August 2012).
Chapter 3

Nfc Data Exchange Format - NDEF

All Nfc specifications that fall into one of the four Nfc Forum Tag Types (see Table 2.1) support the storing of an NDEF Message, which consists of one or several NDEF Records. The NDEF Messages format is the one go-to format for all applications that must be interoperable with different non-proprietary readers, such as with smart posters, the sharing of URLs, plain text and device-to-device message exchanges such as contact sharing and wireless handover[23]. Additionally, NDEF is the only writable format supported by Android, other than writing into blocks on a MiFARE tag.

3.1 NDEF Message format

An NDEF Message is an ordered set of NDEF Records. Each NDEF Record has a type and a payload (see Figure 3.1).

3.2 NDEF Record format

An NDEF record consists of the following three parts: type, id and payload.

3.2.1 The type field

The first field in an NDEF Record is the type-field. It is used to identify the type of data being carried in the payload. The field is 3 bits long, containing a TNF (Type Name Format), which can assume one of the values seen in Table 3.1 [24]:

<table>
<thead>
<tr>
<th>Type Name Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>0x00</td>
</tr>
<tr>
<td>NFC Forum well-known type [NFC RTD]</td>
<td>0x01</td>
</tr>
<tr>
<td>Media-type as defined in RFC 2046</td>
<td>0x02</td>
</tr>
<tr>
<td>Absolute URI as defined in RFC 3986</td>
<td>0x03</td>
</tr>
<tr>
<td>NFC Forum external type [NFC RTD]</td>
<td>0x04</td>
</tr>
<tr>
<td>Unknown</td>
<td>0x05</td>
</tr>
<tr>
<td>Unchanged</td>
<td>0x06</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x07</td>
</tr>
</tbody>
</table>

Table 3.1: Nfc Forum NDEF TNF values
3.2.2 The id field

The id-field is an identifier in relative or absolute URI-format. This URI identifies the payload in more detail and is used by the reader to decide how to interpret the data in the payload and what actions to take.

3.2.3 The payload field

The payload-field is the container for the item being transferred in this NDEFRecord, following a format implied by the type- and id-field. This could be a text with a certain encoding, contact-information for a person or map coordinates as part of a smart-poster NDEF message.
Chapter 4

Nfc on Android

The first Android version to support both reading and writing Nfc tags was Android 2.3.3[25]. Samsung Nexus S, released in the USA on December 16 2010, was the first mobile phone with Android to include the Nfc hardware[26]. Between Android versions 2.3.3 and 4.1 there has been no addition or change in the type of tag technologies supported.

4.1 Basic Nfc operation support on 2.3.3+ and 4.0+

From the Android version 2.3.3 reading, writing and interacting with the Nfc tag technologies in Table 4.1 have been supported.

4.2 The Android Activity lifecycle

The Android activity life-cycle explains how the Android platform handles apps that are running, not running or are frozen in the background. The different states are notified to the activity, and can be represented as a state-transition diagram as seen in Figure 4.1. To receive these state-transition notifications in an activity requires the overriding of a method, one for each state transition of interest, from the base class Activity. This is important in the case of the Nfc foreground dispatch since the activity using it must also ensure that it is disabled when the activity is taken out of foreground mode (onPause) and reactivate it when the activity returns to the front (onResume). Additionally, new intents are delivered into the activity by the onNewIntent notification, which must be handled and parsed in order to utilize the Nfc foreground dispatching.

<table>
<thead>
<tr>
<th>Tag technology</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFC-A</td>
<td>(ISO 14443-3A)</td>
</tr>
<tr>
<td>NFC-B</td>
<td>(ISO 14443-3B)</td>
</tr>
<tr>
<td>NFC-F</td>
<td>(JIS 6319-4)</td>
</tr>
<tr>
<td>NFC-V</td>
<td>(ISO 15693)</td>
</tr>
<tr>
<td>ISO-DEP</td>
<td>(ISO 14443-4)</td>
</tr>
<tr>
<td>MIFARE Classic</td>
<td>(ISO 14443-3A)</td>
</tr>
<tr>
<td>MIFARE Ultralight</td>
<td>(ISO 14443-3A)</td>
</tr>
</tbody>
</table>

Table 4.1: Supported Nfc Technologies from Android 2.3.3 and onwards
4.3 The Android Nfc Dispatch System

The Android platform contains a system which is managing all the actions with regards to Nfc and apps, called the Nfc Dispatch System. The dispatcher mainly determines which app should receive the message when an Nfc tag has been scanned. There are three types
of messages that can result from a scan, namely:

- NDEF_DISCOVERED
- TECH_DISCOVERED
- TAG_DISCOVERED

These three are prioritized in the presented order and which app they will go to depends on the following:

- Content on Nfc tag (NDEF or not)
- Any currently focused app requesting foreground dispatch
- Nfc-filters in Manifest of installed apps

### 4.3.1 Two registration-points: Foreground-listening and Manifest

When an app registers interest in receiving any Nfc events, it must always state what, in particular, it is interested in receiving. This registration can vary from a specific registration, such as a certain URI pattern in an NDEF Record, to a general wildcard of any NDEF Message and/or any Nfc Tag technology. To ensure the app receives what it requires, whilst also not interfering unnecessarily with other Nfc apps on the device, it is important for developers to know how the Nfc dispatch system on Android prioritises and chooses which apps (one or several) can receive the Tag or NdefMessage scanned. This is in order to, if possible, avoid any conflict with other apps. A conflict results in the application-picker being presented to the user, in which it is possible to choose to dispatch the Nfc scan to an app once or by default. The choices taken by the Nfc dispatcher with regards to the Tag content and app filters are shown in a flow-chart in Figure 4.2.

![Figure 4.2: The priority of NDEF and Tag dispatching by the Android system](image)

An app has two means of registering interest in receiving Nfc tags, one involves the use of foreground listening, which can be activated or deactivated at any time the application
CHAPTER 4. NFC ON ANDROID

is active in the foreground on the device (i.e. seen by the user) and the other involves specifying certain filters in the Manifest. The foreground dispatch is always prioritized over other apps. For both foreground dispatching and Manifest filtering, the following filters are available:

**Filtering for NDEF_DISCOVERED** allows for the specifying of a certain mime-type or scheme, host and path for NDEF messages containing mime-content or an absolute URI, respectively. Filtering for a mime-type can be useful when sharing contacts, while filtering for a URI (or AAR) is useful for catching application-specific NDEF messages.

**Filtering for TECH_DISCOVERED** allows for the specifying of one or several tag technologies, such as NfcA or NdefFormatable. These filters will only be iterated when the tag did not contain an NDEF message or no app request matched it. This is useful when making an app targeting, for example, MiFARE tags in general.

**Filtering for TAG_DISCOVERED** is a means of catching tags that did not match any TECH_DISCOVERED request. It is not very useful on its own since any higher priority event would override it. A possible use-case is for an app presenting the type and content of any Nfc tag. Having only the TAG_DISCOVERED filter in its Manifest would allow it to catch unknown tags, while not risking any undesired conflict with other apps.
Chapter 5

Methodology

In this chapter the development-process, tools and evaluation methodology used in this thesis work are described.

5.1 Development process

The development of the apps, library, backend and frontend was conducted using an agile iterative approach. This methodology was chosen as several software projects were dependent on each other. For example, the requirements in the Nfc library came from the needs of the different apps created and thus it evolved throughout the thesis work. The steps in the development process are summarized as follows:

Analysis $\rightarrow$ Design $\leftrightarrow$ Implementation $\leftrightarrow$ Testing

5.2 Hardware tools

5.2.1 Android mobile

The mobile phone used throughout this thesis work was the Google Nexus S, developed by Samsung. The operating system powering this was Android 4.1.1 Jelly Bean.

5.2.2 Virtual server

The backend server created in this thesis work was a virtual machine running on the Amazon EC2 cloud service in a data-center in Ireland, Europe. The performance and characteristics of this virtual machine were as follows:

**Operating system** GNU/Linux Ubuntu 11.10 oneiric

**CPU** Intel® Xeon® CPU E5430 x64 2.66 GHz

**RAM** 512 MB
CHAPTER 5. METHODOLOGY

5.3 Software tools

5.3.1 Editor and Integrated Development Environment
The programming Integrated Development Environment (IDE) used was Eclipse (Helios Service Release 2) with the Android Development Toolkit (ADT) version 20. This was used for all Java Android app development.

The IDE Visual Studio 11 was used for the development of the C# PC-client.

The text-editor Sublime Text 2 was used for the development of the backend NodeJS-based JavaScript software.

5.3.2 Software deployment
In order to continually deploy the backend software to the remote virtual server in Ireland, the application WinSCP was used. WinSCP monitors the software development directory on the local computer and uploads it as soon as any changes are made.

5.3.3 Web authentication and navigation
In order to avoid the development of custom authentication and navigation on the web frontend the open-source web-project, Drupal was used. Drupal was essentially used as a frame around the web frontend created in this thesis work and, additionally, enables continual frontend changes and development through its in-browser content editing tools.

5.3.4 Design- and image resource tools
The application used for creating the design- and class-diagrams was yEd Graph Editor. To generate app launch icons the Android Asset Studio\footnote{Android Asset Studio - http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html} was used. Additionally, some diagrams in the report were borrowed and used for educational purposes from the Android Developer Platform.

5.4 Evaluation of system performance
The aim of the system performance evaluation in this project is to measure the main throughput- and transmission-bottlenecks. Summarizing and generalizing the bottlenecks throughout the applications in this project yields the following points of interest:

- Time between the introduction of a tag to the mobile and the Tag delivery into the app
- Time between the NfcV sensor POLL request and the value response
- Time-of-flight for a small packet from the mobile to the backend over an open socket (TCP)
- Time between the incoming value and the database write completion in the backend
- Time-of-flight for a small packet from the backend to the browser over an open WebSocket (TCP)
• Time between receiving data in the browser and calculating/generating plots

The points of interest stated above are all present in the NfcV sensor scanning system (including backend and browser app). Therefore, it was decided to perform all measurements on that system and use appropriate combinations of those results to estimate the performances of the other apps.

The NfcV sensor scanning system was outlined (see Figure 5.1) and the points of interest for performance measurements, as listed above, were marked. See subsection 5.4.1 for details about each point and the corresponding performance measurement method.

![Figure 5.1: System overview marked with points-of-interest for performance measurement, detailed in subsection 5.4.1](image)

5.4.1 Point-description and methodology of measurements

**T1 - Tag field-entry to dispatch delay** is the mean delay between physically introducing an Nfc tag into the mobile’s Nfc field and the successful dispatch of the Tag to a running Android Nfc app with foreground dispatch activated. Since the Android Nfc dispatch system does not provide data in relation to the start of Nfc communication this had to be provided to the app by a different method. Two possible means of approximately recognizing the introduction of the tag could be: sense the physical bump of the tag brought to the back of the mobile using the accelerometer, or, use a screen finger touch event synchronized in a best-effort manner with the introduction of the tag to the back of the mobile. Due to its simplicity the latter method was chosen to measure the tag field-entry to dispatch delay.

**T2 - NfcV sensor POLL time** is the time between sending the (pre-configured 10 byte array) POLL request, using the transceive-method against a connected NfcV sensor tag and the successful reception of the sensor value in the app. This is measured by storing and comparing time-stamps before and after the transceive-call. This was repeated 4000 times, then plotted as a histogram and presented together with mean- and standard deviation calculations.
**T3 - Mobile-to-backend network transmission time** is the time it takes between writing a sensor value into the open TCP socket on the mobile and the corresponding socket read of the value in the backend. This is measured using a reflector in the backend that responds into the socket when the value is received much like a ping-pong round-trip-time (RTT) measurement. The one-way time is then half of the RTT.

**T4 - Database storage time** is the time it takes between executing the request for the database to append one sensor value to the current sensor’s list of values and the successful completion callback. This is performed by storing the time of the request as well as the time of the execution of the success-callback.

**T5 - Backend-to-browser network transmission time** is the time it takes between writing a sensor value into the open WebSocket connection to a browser and the reception of that value in the browser. This is measured using a reflector in the browser that responds into the WebSocket when the value is received, thus enabling the measurement of the RTT. The one-way time is then half the RTT.

**T6 - Browser calculation- and plot drawing-times** are the collection of times it takes to calculate and plot the set of different statistic visualisations of sensor values in the browser. Each visualization consists of some or all of the following steps: copy, sort, calculate, group into structure and generate plots. The work-load depends on the sample size, that is; how many of the latest values are included in the calculation/plot. The following sample sizes are used to plot the work-time: 1 000, 5 000, 10 000, 15 000, 20 000, 25 000, 30 000.
Chapter 6

Design

This chapter presents the designs of the applications implemented in this project.

6.1 Nfc library for Android

This section concerns the planning steps taken and the choices made before implementing the Android Nfc Library which was a central part of the work in this thesis.

6.1.1 Library use-cases and needs behind the design

A set of library use-cases, i.e. how developers would like to use the library, were defined based on recurring requirements in the Android Nfc apps developed in this thesis work:

**Handling the parsing and foreground life-cycle of Nfc scanning** were the first requirements to offload to the library. All apps using Nfc require a certain boilerplate code to distinguish between different and extracting information from Nfc events (such as NDEF, TECH or TAG intent), and all foreground Nfc apps are required to correctly act on certain states of their Activity life-cycle so as to notify the Nfc dispatch system (in particular onPause and onResume).

**Arbitrary flag in Tag/NDEF callback** were required in the app TimeSpot to note what user-action caused the need for a Tag scanning. The user had a two button choice to either stamp-in or stamp-out from a project, and both options would trigger the need for scanning an Nfc tag in order to provide confirmation. When the callback with the tag arrived the app was required to know whether this was a stamp-in or stamp-out. This information could have been saved in a flag in the app, but this use-case appeared likely to frequently occur exactly at the point when the app wants to start listening for tags. Therefore, the request of listening to tags also includes the passing of an arbitrary flag-object (or null), which will be sent back at a later stage together with the Tag or NDEFMessage.

**Separated callbacks for NDEF and Tag events** allowed for the library to offload some intent parsing and thus make coding easier, in particular for beginner developers. This separation also allows the callbacks (in interface NfcTagListener) to provide the app only with the objects of interest for that specific event and to disregard the rest.
Writing an arbitrary NdefMessage on a tag allows for the multi-purpose writing of an NdefMessage to a newly found writable tag, without any assumptions of a protocol or NdefRecord layout from the library. The only lower level form of tag writing allowed by the Android Nfc system would be Mifare-specific block writing or specially crafted trancive messages for special Nfc tags, such as the NfcV sensor tags.

Writing an apk URI followed by String-String HashMap allows for the writing of a tag which contains, firstly, a unique URI pointing to where the application apk-file can be downloaded from, such as a http(s) or Play Store link. This causes, by default, those Android devices without the installation of that app to ask the user whether they wish to download and install it. In case the app has already been installed it can specify that unique URI in an NDEF_DISCOVERED filter in its manifest to (in most cases) guarantee that this app will be launched. The app then proceeds to parse the settings previously saved on the tag in the String-String HashMap format.

6.1.2 Helper-class and optional base-class

When making a library that augments functionality in an Activity it is beneficial to make the library a base class that inherits from Activity. This would allow an app to simply inherit the class, for example NfcHelperActivity, which would then directly manage life-cycle events such as onPause/onResume as required. The class being created could then choose to override certain methods that act in a similar manner to Nfc event-callbacks from the library. The one crucial problem associated with this is that it forces the new class to inherit from NfcHelperActivity, thus removing the possibility to inherit some other subclass of Activity. For example, if another Android library was constructed in an identical manner then the developer would have to choose which library to inherit from when creating their Activity subclass(es) since Java allows inheriting to take place from only one class. In brief, designing the library as a base class provides benefit for the developer using it in many cases, but is not scalable because of the inherit restriction placed on the new class.

Having the library as a helper class (see Figure 6.1) i.e. an independent class which is used via an object, forces the developer using the class to notify the library of the Activity life-cycle events in order to be able to use it. This boiler-plate notification code can be avoided from Android 4.0 (API Level 14) and onwards using the Activity Life-cycle callback interfaces[27], but since this library is targeted towards Android 2.3.3 (API Level 10) these Activity life-cycle hooks are unavailable. In brief, designing the library as a helper-class forces the developers using it to include boiler-plate code in their Activity for the library to be able to manage the Nfc foreground dispatch life-cycle. This design is restriction free concerning the inheritance and, thus in a way is more scalable, but introduces non-aesthetic boiler-plate which, if not pasted in will cause the application to crash when the user navigates away from it.

The solution to the design dilemma between the base-class or helper-class is to provide both the NfcHelper and NfcHelperActivity. This allows developers, in the vast majority of cases, to simply inherit from the NfcHelperActivity and take advantage of the callbacks for interacting with Nfc tags or other devices (see Figure 6.2). In the case in which the developer is required to inherit some other subclass of Activity, it would be safe to assume they are actually quite experienced and would see the correct usage of the helper class in the examples.
6.1.3 Nfc to Bluetooth Handover

Bluetooth handover is a feature of the Nfc library. It is designed to be transparent and simple, yet flexible to use. It is transparent since the class using it will not receive the NDEF messages used to establish the connection. It is simple yet flexible since the developer can choose to merely send strings to the other device through the library object or take control of the Bluetooth socket themselves to implement their own byte-stream protocol.

6.2 Low latency multi-platform system

This section describes the design of a low latency multi-platform system for distributing and visualising sensor-values. The system includes:

- an Android application which polls values from a sensor using Nfc
- a backend server receiving, storing and managing sensors and values
- a web front-end for managing known sensors and displaying value-statistics

Regardless of the technologies used, a generic web-based system containing the parts above would fit into the overview in Figure 6.3.
6.2.1 Technology motivation
The purpose of this particular system is to connect an Nfc sensor, an Android smartphone and a browser so that they are able to communicate with each other in real-time. The main use-case, in which this system is planned to excel, is when the user is holding a tablet in one hand for value visualization and an Nfc-enabled Android device in the other. This provides high mobility, allowing the user to walk around to different sensors and obtain instant feedback without the need to go back to a stationary computer. This use-case demands a high responsiveness in the browser when scanning, in order that the user is able to feel that the tablet is tightly connected, thus quickly visualizing the actions made with the reader.

6.2.2 Eliminating the delay-bottlenecks in a web-based system
The driving point is low latency between sensor scanning and browser visualization, and it will become apparent that even an enhanced AJAX-based web interface will have too high a latency, due to arbitrary thread sleeps in the browser and unnecessary database access in the PHP backend, in order to allow an as-instant-as-possible visual to sensor scanning.

As a starting point for this system design, the common pure LAMP\(^1\) approach will be considered and analyzed for bottlenecks, using the technology agnostic web-system overview in Figure 6.3.

---

\(^1\)LAMP - Linux, Apache, MySQL, PHP

---

**Figure 6.3: A technology agnostic overview of a web-based system for sensor management**

**Mobile to backend**

With a RESTful PHP system the mobile would collect a set of values and send them periodically to the backend via an HTTP POST with some chosen time interval in between. This bottleneck would cause a delay between the reading and presentation that could not be fixed by improving any other part of the system. Even lowering the chosen...
time interval for the periodic POST messages would, at one point, make the overheads from TCP handshake and HTTP POST headers be at the limit of delay minimization. The solution for this is to discard the RESTful approach and instead establish a live connection from the mobile to the backend into which the sensor values are streamed live without any arbitrary delay.

**Backend to browser**

With a RESTful PHP system, the browser could, at best, use the AJAX technique to periodically poll the backend for new sensor values, which would be served in XML or, preferably, JSON format. This approach would require a chosen time to wait between each AJAX request so as not to place too much load on the backend, since each request causes overhead in the form of HTTP header traffic and database lookups in the backend. It would, again, be desirable to discard the arbitrary polling and instead establish a live connection to the backend. An asynchronous bidirectional live connection between the browser and the backend would make it possible for the backend to push new sensor values to the browser as soon as they arrive from the mobile phone. The live connection takes away the HTTP header overhead, while switching from poll-based to push-based value distribution removes the need for an arbitrary sleep delay in the browser. This can be achieved using HTML5 WebSockets.

**A single-context backend with TCP- and WebSockets**

The design-choices so far point at a backend capable of supporting TCP and WebSockets and having all these connections available in a single context. The single context means, for example, that the browser connections are directly available at the context of a sensor-value which is just being received from a connected mobile phone.

The choice was made to use NodeJS as the backend and Redis as the database. The main reasons for using NodeJS instead of, for example, Java are the following:

- Simplicity in implementation
- NodeJS is a new interesting technology worthy of exploration
- Availability of many easy-to-use real-time libraries via a package manager
- Consistency of JavaScript in both server and frontend, especially for managing arbitrary JSON structures

**Web frontend for statistics**

For the web frontend it was decided to use an HTML5-based approach in the design by utilizing dnode for the WebSocket communication to the backend and KnockoutJS for scripting the interactive web UI and keeping the JavaScript models up to date in the browser. These are directly reflected to and from the UI on user action or from updated information from the backend. Using a setup of synchronized models any changes would automatically be distributed among all the browsers which are currently connected, such as when changing the name of a sensor or the range of a statistical plot.

---

2 AJAX - Asynchronous JavaScript and XML
6.3 Wifi Connector and Writer

The idea behind the WiFi Connector is to allow a device to connect to a password protected WiFi network by tapping an Nfc tag with a specially crafted NdefMessage stored on it or by achieving the same via a generated peer-to-peer NdefMessage push.

6.3.1 Real-world scenarios

The three primary use-cases for the Wifi Connector are the following.

**In the entrance of a small restaurant** there is a tag and poster saying *Tap for free WiFi access*. Customers can easily use the in-house WiFi without additional set-up.

**In the lobby of a hotel** there is a specially built Nfc device, connected to the hotel’s WiFi system, that generates one or several new unique sets of credentials for paying customers in order that they can use the WiFi throughout the hotel but only for the time-span that they have paid for.

**The hotel customers** can share their WiFi connection with a pre-defined number of friends by tapping their phones together while the app is active. This works since the phone received and stored multiple credentials from the device in the lobby, namely, one set of credentials for the phone itself and multiple others for sharing with a limited number of friends. (This use-case remains for future work).

6.3.2 UML design

The WiFi Connector usage-flow has the following appearance: Scan a tag; Assert tag is valid for this purpose; Handle/parse the NdefMessage on the tag; Connect to WiFi network. These steps are separated into methods as seen in the UML diagram in Figure 6.4.

The WiFi Writer usage-flow has the following appearance: User chooses WiFi network type; User inputs SSID and password; User presses Write-button; An NdefMessage is created and queued for writing; When an Nfc tag is connected the NdefMessage is written. This app requires separate GUI-parts for each supported WiFi technology. These are designed as separate classes, inheriting the Fragment-class present in Android 3.0 (API Level 11) and onwards. The NdefMessage queuing and writing are designed as separate methods, as seen in the UML diagram in Figure 6.4.

6.4 TimeSpot

TimeSpot is an Android app which provides stamping in and out of projects, sometimes requiring confirmation via Nfc tags. TimeSpot works together with a RESTful backend system, which manages users and their projects as well as receiving all stamp-actions taken by the user. The development of TimeSpot, contracted by *TimeTerminal* via *Dewire Telecommunication Systems*, concerns only the Android app and not any other parts of the system.
Figure 6.4: UML diagram portraying key parts of the WiFi Connector and WiFi Writer Android apps

6.4.1 Functionality and use-cases

The app was designed to have the following functionality for each user:

- A main state of being stamped in or out
- A list of projects, each with a state for being stamped in or out
- Depending on subscription; forced Nfc scan before any state-change by the user
- Receiving projects and settings from and sending all stamps to the backend

These basic functionalities allow for, for example, the following use-cases:

**One or several consultants working on different projects on the move** can stamp in and out of projects and thus keep track of their hours. Stamping in and out of the office could require scanning an Nfc tag at the office entrance. At the point of sending out invoices and paying salaries the finance department has the aggregated data of the hours which have been spent.

**A security guard walks a security-round** during night-time to check stores, warehouses and other facilities in an area. All companies paying for this service have received an Nfc sticker to place in their facility. Each such Nfc tag thus represents a geographical position, and scanning it proves a presence at a certain time. The guard carries an Nfc-enabled Android smartphone and uses the app to create stamping-events when walking through each facility. These stamps are automatically sent to the backend when an Internet connection becomes available, allowing the security company to summarize their service and track the progress of the guards.
Chapter 7

Results

This chapter presents an overview of the implementations made in this project. The languages and platforms for the implementations are Java in Android, C in Windows, JavaScript in NodeJS and HTML/JavaScript in a browser.

7.1 Android Nfc Library

The library was created to assist in the use-cases detailed in design-section 6.1.1, namely the following:

- Handling the Tag/NDEF parsing and foreground life-cycle of Nfc scanning
- Arbitrary flag in Tag/NDEF callback
- Separated callbacks for NDEF and Tag events
- Writing an arbitrary NdefMessage on a tag
- Writing an apk URI followed by String-String HashMap

7.1.1 Class overview and usage

The library consists of one class which manages the Nfc life-cycle, when necessary, and one interface with Tag and NdefMessage discovery callbacks to the Activity using the library (see Figure 7.1).

7.1.2 Managing the Nfc life-cycle

The helper-class part of the library is dependant on the correct usage of four hook methods from the Activity: onCreateHook, onNewIntentHook, onPauseHook and onResumeHook, to correctly manage the Nfc life-cycle and receive Nfc Intents. Therefore, for simplicity in most Nfc app development cases, an additional base class called NfcHelperActivity, inheriting Activity, has been created which, when inherited, manages the Nfc life-cycle.
7.1.3 Nfc to Bluetooth handover

The idea behind the Nfc to Bluetooth handover is to simplify the addition of multiplayer, or similar, for the developers to an Android game or other app. The handover is a feature built into the library and relies on the Android peer-to-peer NDEF message push. To remain transparent to the developer using the library, this requires parsing of the NDEF messages before they are passed on to the app. In the case in which the library recognizes an incoming NDEF message as a special Nfc-to-Bluetooth handover request, it will act upon it instead of passing it on to the app.

On a device running the game, the developer would run a method in the code to prepare for the Bluetooth handover. A Bluetooth socket is then created and an NdefMessage created which contains the device's Bluetooth MAC address and the UUID for the Bluetooth server. When the other device receives this NdefMessage, the library will transparently connect to the Bluetooth server and notify the activity that they can start exchanging messages, without relaying the NdefMessage containing the Bluetooth handover request.
Example NdefMessage

An example NdefMessage triggering a Bluetooth handover.

<table>
<thead>
<tr>
<th>NdefRecord type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNF_WELL_KNOWN - RTD_URL</td>
<td><a href="http://www.deckmar.net/apk/multiplayer_game.apk">http://www.deckmar.net/apk/multiplayer_game.apk</a></td>
</tr>
<tr>
<td>TNF_WELL_KNOWN - RTD_TEXT</td>
<td>NfcToBluetoothHandoverRequest</td>
</tr>
<tr>
<td>TNF_WELL_KNOWN - RTD_TEXT</td>
<td>F8:DB:7F:04:63:1D</td>
</tr>
<tr>
<td>TNF_WELL_KNOWN - RTD_TEXT</td>
<td>d4dcdc1b-c537-4fbb-9a11-44943df6d10</td>
</tr>
</tbody>
</table>

NdefMessage triggering a Bluetooth handover between two apps using the Nfc library

7.2 Low-latency multi-platform system

The multi-platform system, with Android clients and web frontend, was implemented using NodeJS. To speed up the development the following libraries were used:

Express A web-server library for NodeJS, supporting live compilation of CoffeeScript\(^1\) to JavaScript and LESS\(^2\) to CSS

dnode A bi-directional JavaScript Remote Method Invocation (RMI) library utilizing WebSockets between the NodeJS application and connected browsers. This is used as the basis for synchronizing JavaScript models and pushing sensor values to browsers.

KnockoutJS A JavaScript library enabling bi-directional binding to models and event-handling in the browser by means of convenient in-line HTML syntax. Using this library, all UI components are automatically created according to the layout scripting and adding and removing, for example, sensors in the UI requires only pushing and popping the models from an array.

Flotr2 A JavaScript library for generating in-browser HTML5-based statistical plots, such as the histogram and candle-plot.

Sockets in NodeJS were used to receive sensor values from mobile devices. A simple line-based protocol with the sensor id followed by streaming values was created. The single context nature of NodeJS then made it simple to distribute all sensor values to browsers via RMI as soon as they arrived. When the function was triggered in the browser, parametrized with the sensor-id, list-name and a value, it was appended into the list of values for that sensor, whereas, KnockoutJS automatically updated the UI to show, in this case, the number of values scanned from that sensor. Generating the plots demanded rather more preparation, in terms of filtering, sorting and ordering the sensor-values into groups, and was conducted in code triggered by the sensor value distribution event, instead of only through KnockoutJS as is the case for the remainder of the web UI.

The web UI was wrapped in Drupal for the sake of navigation and authentication, but not implemented as a Drupal module. The Drupal platform enables the development of new forms of presentations directly via the web UI by editing the source, which is similar to editing the contents of a blog-post. This brings flexibility since it does not require the setup of a development environment to make additions or changes to the frontend.

\(^1\)CoffeeScript - [http://coffeescript.org/](http://coffeescript.org/)

\(^2\)Less - [http://lesscss.org/](http://lesscss.org/)
7.3 Nfc to WiFi Writer and Connector

The WiFi Writer was implemented as a tabbed screen, with one tab for each WiFi type. The text-fields are in the tab and there is a button in the ActionBar for writing these credentials to a tag (see Figure 7.2). For the sake of demonstration only, the WPA-PSK type of WiFi was implemented. When pressing the write-button, the user sees a dialog requesting the introduction of the Nfc tag to write this to.

![Figure 7.2: Screen of Nfc WiFi Writer when creating a tag with credentials](image)

The WiFi Connector was implemented as an Nfc-based app with only a dialog, on top of a transparent background, to show its state (see Figure 7.3). This makes it very easy for the user, since no additional actions need or can be taken.

The separation of writing and connection into two different apps was performed to simplify the UI for the users. If a user scans a prepared tag without having the connector app then his/her mobile will ask if he/she wants to download and install it. The user must then scan the tag again, which would result in the app connecting to the WiFi using the stored credentials.

7.3.1 Requirements and design

The WiFi Connector is likely not to be pre-installed in the mobile, so when tapping an Nfc tag the mobile should, if needed, ask the user to install the app. Additionally, the WiFi Connector should, if installed, directly connect to the WiFi upon an Nfc scan without having first to be started by the user. These two requirements are met by writing an Apk
CHAPTER 7. RESULTS

Figure 7.3: Screen of NFC WiFi Connect after scanning a tag with credentials

URI as the first NdefRecord in the NdefMessage on the NFC tag, and then filtering to that URI in the app’s Manifest.

To make the WiFi Connector app as simple to use as possible, the functionality to write tags was separated out as its own app and was called the WiFi Writer. When launching the WiFi Writer, the users are able to choose which type of WiFi network they want to create a tag for, then enter the SSID and password and press the write-button. A dialog requests the user to hold the phone to the writable tag, onto which an appropriate NdefMessage will be written.

In order not to inconvenience the user the app was designed to simply pop up a dialog, upon scanning an appropriate NFC Tag, indicating that it is connecting to the network. The user need not make any confirmation. After four seconds the dialog vanishes.

7.3.2 Example NdefMessage

An example NdefMessage for connecting to a WPA-PSK network with SSID “example” and password “secret” has the following appearance.

<table>
<thead>
<tr>
<th>NdefRecord type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNF_WELL_KNOWN - RTD_URL</td>
<td><a href="http://www.deckmar.net/apk/wifi_connector.apk">http://www.deckmar.net/apk/wifi_connector.apk</a></td>
</tr>
<tr>
<td>TNF_WELL_KNOWN - RTD_TEXT</td>
<td>type=WPA-PSK</td>
</tr>
<tr>
<td>TNF_WELL_KNOWN - RTD_TEXT</td>
<td>ssid=example</td>
</tr>
<tr>
<td>TNF_WELL_KNOWN - RTD_TEXT</td>
<td>password=secret</td>
</tr>
</tbody>
</table>
CHAPTER 7. RESULTS

Table stamps

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER PRIMARY KEY AUTO-INCREMENT</td>
</tr>
<tr>
<td>time_added</td>
<td>LONG</td>
</tr>
<tr>
<td>user_id</td>
<td>TEXT</td>
</tr>
<tr>
<td>tag_id</td>
<td>TEXT</td>
</tr>
<tr>
<td>confirmed</td>
<td>INTEGER</td>
</tr>
<tr>
<td>time_confirmed</td>
<td>LONG</td>
</tr>
<tr>
<td>direction</td>
<td>TEXT</td>
</tr>
<tr>
<td>project_id</td>
<td>TEXT</td>
</tr>
</tbody>
</table>

Table 7.1: Database definition of table: stamps

NdefMessage triggering the WiFi Connector app to launch and connect to a WiFi network

7.4 TimeSpot

TimeSpot is an app used for stamping into and out of work and/or work-projects. The app was built for a company with an existing backend system where user-specific settings are retrieved and stamps are sent.

7.4.1 User experience

The resulting TimeSpot Android app consists of a home-screen showing the currently logged in user and the user’s main- and project-state. Additionally, buttons for changing the main state to stamped-in or -out are reachable with the thumb along the bottom (see Figure 7.4).

Navigating to Projects lists the current user’s projects and allows for the stamping in or out of any of them (see Figure 7.5).

Depending on the settings which have been made for the user in the backend by the administrators, the user might need to scan an Nfc tag to complete any stamp-in or -out. This causes the app to display a dialog requesting the user to hold the mobile against a tag (see Figure 7.6). When a tag is connected the stamp is completed and the dialog closes.

7.4.2 Keeping records in offline mode

The application’s state, which includes user-information, settings and projects, is required to be stored even if the app has quit. Additionally, in case the mobile lacks an Internet connection when a stamp-event occurs, the app must have it stored so it can attempt to send it in again at a later stage. This was solved using a local SQLite database. The database relations were created as seen in Table 7.1, Table 7.2, Table 7.3 and Table 7.4:

7.5 Nfc Like-tag to Digital Signage greeting

The Nfc Like app was implemented as a simple Nfc app, which notifies the backend when a certain Nfc tag has been scanned. The backend then has a list of connected C# clients
Figure 7.4: Home screen of TimeSpot, showing logged in user, main- and project-state. Buttons to stamp in or out.

(frontends) which it notifies in order to show the product endorsement message (as seen in Figure 7.7). For demonstration purposes the app and system only support a hard-coded name and product, and do not have a connection to Facebook, which is referenced by the popup-dialog.
Figure 7.5: Current user’s projects in TimeSpot. Buttons to stamp in or out of selected project.

### 7.6 Evaluation of sensor-system performance

The evaluation of the implementations in this thesis consists of performance measurements at a collection of points in the sensor-scanning system. The points-of-interests were chosen as those that impact upon the performance, in some combination, for all applications.
CHAPTER 7. RESULTS

Figure 7.6: TimeSpot app requesting user to scan an Nfc tag to complete a stamp-event.

implemented in this thesis. The following metrics were measured:

- Time between the introduction of a tag to the mobile and Tag delivery into app
- Time between the NfcV sensor POLL request and the value response
Table 7.2: Database definition of table: projects

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>TEXT PRIMARY KEY</td>
</tr>
<tr>
<td>caption</td>
<td>TEXT</td>
</tr>
</tbody>
</table>

Table 7.3: Database definition of table: conf

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>TEXT PRIMARY KEY</td>
</tr>
<tr>
<td>value</td>
<td>TEXT</td>
</tr>
</tbody>
</table>

Table 7.4: Database definition of table: log

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER PRIMARY KEY AUTO-INCREMENT</td>
</tr>
<tr>
<td>time_added</td>
<td>LONG</td>
</tr>
<tr>
<td>type</td>
<td>TEXT</td>
</tr>
<tr>
<td>message</td>
<td>TEXT</td>
</tr>
</tbody>
</table>

Figure 7.7: Nfc Like-scan resulting in a product endorsement message on in-store advertisement screens.

- Time-of-flight for a small packet from the mobile to the backend over an open socket (TCP)
- Time between the incoming value and the database write completion in the backend
- Time-of-flight for a small packet from the backend to the browser over an open WebSocket (TCP)
• Time between receiving data in the browser and calculating/generating plots

7.6.1 Time from tag introduction to app delivery

The time it takes from the introduction of the Nfc tag to the mobile device until the data has been delivered into the foreground-listening app was measured using a specially crafted application. The main hurdle in the measurement was to approximate when the tag was introduced, since this information cannot be accessed from within the app via the Android Nfc API. Instead, a manual touch-event from the GUI was used to identify the time to start the measurement (see Figure 7.8), and the Nfc data callback event to identify the time to stop. The accuracy of this approach depends largely on the touch-timing accuracy of the person performing the test. This method was deemed as providing a reasonable accuracy for the purpose of discussing the Nfc introduction-to-data-delivery delay.

The resulting time, using 60 measurements, has a mean value and standard deviation in milliseconds as follows:

\[ N(374, 148.4) \]

For a histogram showing the time-distribution, see Figure 7.9.

7.6.2 Android NfcV tranceive-call time

The time it takes to complete a full tranceive from and to the Android device was measured by comparing timestamps before and after and was repeated 4000 times. In this case, a full tranceive consists of sending a byte-array of length 10 and then receiving a byte-array
CHAPTER 7. RESULTS

Figure 7.9: Histogram showing NFC tag-introduction-to-data-delivery time distribution in milliseconds.

of length 3 from the sensor. This time includes both the sending and receiving as well as all the time it takes the sensor tag to prepare its sensor and obtain a measurement.

The resulting time, using 4000 measurement points, has a mean value and standard deviation in milliseconds as follows:

\[ N(67.4767, 2.6208) \]

For a histogram showing the time-distribution, see Figure 7.10.

Figure 7.10: Histogram showing tranceive-call time distribution in milliseconds.
7.6.3 Time-of-flight from mobile to backend

Time-of-flight from the mobile to the backend server is the time it takes a sensor value to be sent through an established TCP socket. This is measured using a reflector in the backend, which allows the round-trip time (RTT) to be measured in the mobile device, and at a later stage the RTT is divided in two. The start-time is just before the Socket write-call in the mobile app and the stop-time is after the read, in the mobile app, of a short special reflector message confirming the sensor value has been received.

The resulting time, using 4000 measurement-points, has a mean value and standard deviation in milliseconds as follows:

\[ N(25.8546, 6.6170) \]

For a histogram showing the time-distribution, see Figure 7.11.

![Histogram showing time to send a value one-way from a mobile to the backend via an open TCP socket.](image)

Figure 7.11: Histogram showing time to send a value one-way from a mobile to the backend via an open TCP socket.

7.6.4 Time to store sensor-value in Redis database on backend

The backend has a local Redis database with a list of sensors, and each sensor has a list of time-value pairs. The implementation is designed such that the values streaming one-by-one to the backend are stored in the database before they are distributed to any currently connected browsers. Because of this, the delay in storing the values forms part of the chain of delays from the mobile to the browser. The delay is measured at nano-second precision using the NodeJS built in `process.hrtime`-method to compare current high resolution time starting before the append command is sent to the database and ending in the success callback.

The resulting time, using 5800 measurement-points, has a mean-value and standard deviation in nanoseconds as follows:

\[ N(162232, 56072) \]
Represented as rounded milliseconds these become:

\[ N(1.62, 0.56) \]

For a histogram showing the time-distribution, see Figure 7.12.

\[ \text{Figure 7.12: Histogram showing time to append one value into the Redis database.} \]

### 7.6.5 Time-of-flight from backend to browser

Time-of-flight from the backend to the browser is the time it takes a sensor value to be sent through an established WebSocket. This is measured using a reflector in the backend, which allows the round-trip time (RTT) to be measured using a JavaScript in the browser, and to be later divided into two. The measurement utilizes a remote-method-invocation library operating over a WebSocket, supplying a callback which is triggered from the backend reflector method. The start-time is just before the RMI reflector call in the browser and the stop-time is in the callback, which is triggered in the browser per request of the reflector. This measurement does not cause any overhead other than identifiers appended by the JavaScript RMI framework `dnode`.

The resulting time, using 4000 measurement-points, has a mean-value and standard deviation in milliseconds as follows:

\[ N(21.1629, 0.8808) \]

For a histogram showing the time-distribution, see Figure 7.13.

### 7.6.6 Calculating and plotting data-sets in browser

Every time a sensor-value arrives into the browser, or at most every 200 milliseconds depending on settings, the three different representations are re-created in the browser: calculating mean and deviation, histogram plot and candle-bar plot. Re-creating these
CHAPTER 7. RESULTS

Figure 7.13: Histogram showing time to send a value one-way from backend to browser via open WebSocket.

representations uses processing power, demanding all or some of the following: copying, sorting, calculating, grouping into structures, generating plots and updating the DOM. Depending on the size of the sample being represented, which can be limited per representation in the settings, this involves different amounts of time.

For this evaluation, sample sizes between 1 000 and 30 000 were used. The first of the three representations consists of calculating both the mean- and standard deviation values. Since these calculations took a very short time in comparison to the two in-browser-plots, even for 30 000 samples, it was ignored in relation to this evaluation. The second and third representation are visual plots, more specifically a histogram plot and a candle-plot. The time in milliseconds for creating these in-browser-plots using sample sizes of 5000 to 30000 in steps of 5000 were measured several times and drawn with bars representing the mean plotting-time for each sample size.

The resulting times for the in-browser bar plot and the in-browser candle plot are seen in Figure 7.14 and Figure 7.15 respectively.
CHAPTER 7. RESULTS

Figure 7.14: Time in milliseconds for creating in-browser histogram plots using $N$ samples.

Figure 7.15: Time in milliseconds for creating in-browser candle plot using $N$ samples.
Chapter 8

Conclusions

To conclude; the work planned in this thesis, including research, implementation and evaluation, was successfully completed. This has yielded a knowledge-base, a set of diverse Android apps, a system including a high-performance backend and frontend, an Android Nfc library simplifying the use of Nfc for reading, creating and writing tags as well as facilitating transparent mobile-to-mobile Bluetooth handover and a set of performance metrics detailing many points-of-interest present in any Android Nfc system.

8.1 Thesis goal and result

The overall aim of this thesis was to (1) research and build a knowledge-base around the workings and usage of Nfc, in particular on the Android platform, to (2) implement a diverse set of Android Nfc apps, for some part of a system including both the backend and frontend, to (3) isolate shared and useful use-cases from the apps into an Android Nfc library and to (4) evaluate performance at several points-of-interest in an Android Nfc system, including its backend and frontend.

8.1.1 Research

The research presented in this thesis consists of the following:

- Near field communication
  - Physical layer
  - Nfc Forum tag types and modes
  - ISO-14443 Contactless cards
  - ISO-15693 Vicinity cards
  - Comparison with RFID

- Nfc Data Exchange Format

- Nfc on Android
  - Nfc functionality in 2.3.3+ and 4.0+
  - Activity life-cycle and Nfc foreground management
  - The Nfc dispatch system priority-rules for NDEF and/or Nfc technologies
CHAPTER 8. CONCLUSIONS

This research serves as a knowledge-base and is used for motivating implementation, evaluation methods and evaluation results.

8.1.2 The Nfc library

The library is central to all apps implemented in this thesis. It includes the following features:

- Helper-class (scalable) or base-class (easy) usage
- Nfc life-cycle management
- Parsing of tags and NDEF messages from intent
- Separate callbacks for tag or NDEF discovery, both including arbitrary flags from listen-request
- Assist-methods for creating URI or Text NdefRecords
- Methods for creating common NdefMessages with a certain pattern
- Builder-class for creating or writing an NdefMessage in one line.
- Transparent Bluetooth handover and messaging

The library resulted in an easier use of Nfc in the apps implemented, and, in particular, removed around 30+ lines of boilerplate code for extracting Tag or NDEF objects and managing the Nfc life-cycle in foreground dispatch mode. The main aim for the library was to simplify Nfc usage in non-advanced application. This goal was achieved, with the assumption that developers using it are familiar with including libraries in their project.

8.1.3 Implemented Android apps

The apps implemented were the following:

- Nfc WiFi Connector
- Nfc WiFi Writer
- Nfc Like (+backend, +frontend)
- Nfc Sensor Reader (+backend, +frontend)
- Nfc Sensor Visual
- Nfc Measurements
- Nfc Bluetooth handover test
8.1.4 Implemented server and frontend

The backend server consists of one NodeJS application and one Redis database running on a virtual machine in an Amazon EC2 data-center in Ireland. The backend has several open ports awaiting connections from: the Nfc Sensor Reader, the browser WebSocket RMI library dnode, and an HTTP server serving CSS and JavaScript files.

The single-context design and simplicity of JavaScript made NodeJS an excellent candidate in order to implement the versatile backend. The resulting server has very high performance and low hardware utilization, mainly because of the single-context nature of NodeJS removing a large amount of overhead in areas such as the threads per connection, internal communication, client polls and HTTP headers. NodeJS is single-threaded, single-context and has great support for sockets, which means that no internal database access is required, and that hundreds of thousands of long-lived simultaneous connections can be managed, and HTTP headers can be avoided when building on top of sockets.

8.2 Evaluation summary

The performance evaluation focused on only one system, namely the Nfc Sensor Reader system including NodeJS+Redis backend and web frontend. The motivation behind this was based on the fact that this system contains all points-of-interest seen in this thesis. These points-of-interest, and their corresponding methods and performance metric(s), were:

- Time between the introduction of a tag to the mobile and Tag delivery into app
  
  **Method** Touch screen at the same time as introducing the tag. Measure time-difference.
  
  **Mean-value and standard deviation** 374 & 148.4 [ms]
  
  **Repeated** 40 times

- Time between the NfcV sensor POLL request and the value response
  
  **Method** Measure the difference in time before and after the tranceive-call.
  
  **Mean-value and standard deviation** 67.48 & 2.62 [ms]
  
  **Total traffic** 10 bytes out, 3 bytes in
  
  **Repeated** 4000 times

- Time-of-flight for a small packet from a mobile to the backend over an open socket (TCP)
  
  **Method** Measure the RTT from a mobile to the backend using a reflector. Divide the RTT by two.
  
  **Mean-value and standard deviation** 25.85 & 6.62 [ms]
  
  **Repeated** 4000 times

- Time between the incoming value and the database write completion in the backend
  
  **Method** Measure the difference in time before a database append-call and in callback.
Mean-value and standard deviation 1.62 & 0.56 [ms]
Repeated 5800 times

• Time-of-flight for a small packet from the backend to the browser over an open WebSocket (TCP)
Method Measure the RTT from the browser to the backend using a reflector. Divide RTT by two.
Mean-value and standard deviation 21.16 & 0.88 [ms]
Repeated 4000 times

• Time between receiving data in the browser and calculating/generating plots
Method Measure difference in time before and after plotting.
Mean-value for plotting histogram (N=1000) 19 [ms]
Mean-value for plotting histogram (N=30000) 73 [ms]
Mean-value for creating candle-plot (N=1000) 6 [ms]
Mean-value for creating candle-plot (N=30000) 30 [ms]

Using these results it is possible to conclude that the time between placing the phone against an Nfc sensor until it can be seen in both plots in the browser by the user is (assuming 1000 values are plotted in the browser) 515 ms.

When streaming sensor-values from the mobile (when it is in constant connection with the sensor tag) it is possible to subtract the time for the tag discovery and the transmit time. This means that the time for each value, after being read from the sensor to being displayed in both plots in the browser is (assuming 1000 values are plotted in the browser) 74 ms.

Although the total times calculated above can differ several tens of milliseconds it is safe to say that a high-performance system with such low delay and variance is difficult to create using a RESTful approach.

8.3 Societal impact

The NfcV Sensor Reader system was designed and developed bearing three aspects in mind: (1) as a tool for Nfc sensor development, (2) as a platform supporting proof-of-concept tele-healthcare as well as (3) assisted living scenarios, as described by Dr. Sidén et al in *Home care with NFC sensors and a smart phone* [8]. The technological and financial development within the fields of sensors, smartphones and health-care over the last few years combine to make tele-healthcare both cheap and desirable in many parts of the world.

As development in this field progresses, it is important to bear in mind the transformative impacts that technology can have on society and social behaviour. Just as telephones enabled keeping in touch with friends throughout the world it also isolated people from those nearby, making it more unusual to go visit a friend who lives nearby without first calling. Tele-healthcare could yield similar unforeseen social changes, such as the possibility that the health of elderly becomes something which is displayed rather than involving a visit from relatives because of automated notifications based on health-statistics. It is, in other words, important to develop humane systems and not think of tele-healthcare as merely status-monitoring of a machine.
8.3.1 Ethics

In many cases an Nfc tag can be tied to a physical position at a certain time. This includes all sorts of payments, facility check-ins and smart posters. When scans take place it is reasonable to assume that information about the tag and the person scanning the tag is stored on a server. Thus, if one person does this often, either through work or in private, it is possible to track movements and make accurate assumptions about that person without their consent. Information from an often used a payment system could, within days, be used to estimate where the person lives and works. The products they buy could be correlated to groups of similar interest, family situation, health, and social situation, which, among other metrics, are valuable for market researchers. It is easy to imagine any heavily used Nfc-based system gathering a significant amount of information about its users, whether planned or not, without their knowledge, which could, at a later stage, prove very tempting to share with anything from market researchers to political organizations. As a developer of such systems it is important to ensure that users know what information is stored about them and/or remove personally identifiable data as soon as it is no longer needed.

8.4 Related work

At least one library similar to the one designed and implemented in this thesis exists, called EasyNFC\(^1\). EasyNFC, whose first versions were published March 2011, currently supports most of the features for this thesis library and more. Supported features include help to create NdefRecords and facilitating Bluetooth handover. EasyNFC is designed only as a helper-class requiring the developer to notify the library upon Activity life-cycle events in order for it to function. In contrast, the library created in this thesis can also be used as a base-class, which takes care of those life-cycle notifications.

8.4.1 Android 4.1 Jelly Bean features

In June 2012 Google introduced Android 4.1 Jelly Bean. This version supports a Nfc-to-Bluetooth handover standard called Simple Secure Pairing\(^2\)\(^3\). This is used to pair hardware, such as the music streamer Google Nexus Q, with an Android device for remote control.

8.5 Future work

Suggested future work consists of two parts, namely, features that can be appended to the library and separate Nfc-based smartphone apps.

8.5.1 Nfc library features

Partially looking at EasyNFC for inspiration, the following features are suggested as future work:

- Transparent fallback to Bluetooth where there is a lack of Nfc hardware
- GUI elements, such as dialog requesting the user to scan, to simplify implementations

\(^1\)EasyNFC - https://github.com/Mobisocial/EasyNFC

\(^2\)Simple Secure Pairing

\(^3\)Android 4.1 Jelly Bean features
• Ad-hoc WiFi handover
• Support for NDEF and Technology filtering in foreground dispatch mode

8.5.2 Suggested applications

The following Nfc-based applications are suggested as technology examples and future work:

Interactive purchases

Imagine a use-case taking place in IKEA in front of a sofa collection, for which an Nfc tag would invite the user to scan in order to interactively explore color and material choices available for this product. The user sees the sofa collection in their mobile after scanning the smart poster and is then able to select different colors and materials, which updates the model seen in the mobile display and/or a large display in front of the sofa collection.

Flexible tele-healthcare system

The Nfc Sensor Reader system implemented in this thesis supports rapid prototyping of ideas directly in the frontend, assuming that the user is logged in with an administrative account, without the requirement to set up a developer environment or being tied to certain computer tools. In relation to future work, the suggestion is to create several such proof-of-concept modules to show the possibilities to those people who are interested in developing or investing in such technology.

In-store recipe scanning

An app was designed for this thesis in the first stages of the project, but was later discarded. This concept app would be used to scan food recipes stored in a tag in a store. It would then display prices and locations for the products required. A brand-loyalty discount was suggested, as seen in Figure 8.1.

Company car management

Company cars are often shared among the employees. Swedish law requires travel and the reason for it to be logged for tax reasons. In addition to this, refilling gas sometimes means that the employee must spend their own money and file a request to have this reimbursed on the next paycheck. These two repetitive and time-consuming hurdles could be made more efficient by developing an Nfc-based app allowing employees to scan a tag showing the distance the car has traveled and the amount of gas in the tank, possibly in combination with a photographic analysis of the dashboard panel. Reports could then be automatically generated and knowledge about the car’s whereabouts could be available at all times.

Application deployment

In ongoing smartphone application development, new versions are often sent out to the client awaiting feedback. One means of doing this is by emailing an apk-file, in the case of Android development, or a link to one. The business client can then open this on the mobile device to install the new or updated application. Scaling this to as few as five
devices requires a more automated approach in order to be practical for the client. An Nfc-based application is suggested as future work such that the company developing the app can send Nfc tags to the business client. Scanning these tags causes the mobile to download the new version from a URL by default. Using this approach means that the smartphones are not required to be prepared or in any way connected to an email server to ensure an efficient deployment of frequently occurring events. The client can, at any time, scan the tag with all the devices, which will trigger them to download the apk-file and to ask for confirmation to install it.

Figure 8.1: Mockup of a food-recipe scanning app with brand-loyalty discounts.
References


REFERENCES

http://www.nfc-forum.org/specs/spec_list/.


http://www.nfc-forum.org/resources/faqs#operating.


http://developer.android.com/about/versions/android-2.3.3.html.

[26] Google, “Features Galaxy Nexus S.” Available: 


http://www.android.com/whatsnew/.

2011.