NFC in Android

Martijn Coenen <maco@google.com>
Agenda

- State of NFC in mobile
- What can you do with NFC in Android?
- Android Beam
- NFC Tags
- Card emulation and HCE
- Q & A
State of NFC in mobile
NFC and Android

- First NFC-enabled Android device: Nexus S (2010)
- Running Android Gingerbread (2.3)
- Google Wallet launched in 2011
NFC availability in Android devices in 2013

- Supported by pretty much every Android device maker
- **100% of high-end** devices
- **Lots of mid-end** devices
- **Entering low-end devices** in 2014
- NFC will be ubiquitous technology in Android devices
What about Bluetooth Low Energy?

<table>
<thead>
<tr>
<th></th>
<th>BLE</th>
<th>NFC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Range</strong></td>
<td>50 meters</td>
<td>4 centimeters</td>
</tr>
<tr>
<td><strong>Proximity accuracy</strong></td>
<td>~1 meter</td>
<td>4 centimeters</td>
</tr>
<tr>
<td><strong>Power draw on device</strong></td>
<td>10 - 300mW</td>
<td>1 - 300mW</td>
</tr>
<tr>
<td><strong>Life-span of tags</strong></td>
<td>~1-3 years</td>
<td>passively powered</td>
</tr>
<tr>
<td><strong>Cost of tags</strong></td>
<td>20$</td>
<td>1$</td>
</tr>
</tbody>
</table>
BLE and NFC: different interaction models

- **BLE**: great for providing context and location
  - **Push** interaction model
  - Awesome if you push the right data and don’t spam
- **NFC**: great at showing intent
  - **Pull** interaction model, undisputed intent, low latency
- Intent could be added to BLE by external factors
  - Consistency is crucial (tapping, waving, twisting, in-app tap...)
  - Facial recognition
BLE and NFC: different connections

- **BLE**: great for persistent connections and broadcasts
  - Perfect for secure connection to wearables (once paired)
  - Public broadcasts (beacons)
- **NFC**: great for short-lived connections and bootstrapping
  - Quick and secure link because of proximity
  - Small quantities of data
  - Use of NFC tags for pairing devices, configuring Wi-Fi
Key takeaway for developers

- The number of NFC devices is still rapidly growing
- BLE is a great complimentary technology
- With device ubiquity, many opportunities for app developers to build magic experiences
What can you do with NFC on an Android phone?
A lot more than just payment

- P2P data exchange (Android Beam)
- Communicating with passive NFC tags/stickers
- Easily pair with Bluetooth audio devices
- Emulating (smart) NFC tags
Android Beam: magic sharing

- Share current context on screen
- Bring two NFC devices in range
- Touch the screen to confirm the send
Android Beam: how does it work?

- UI component of Android apps: **Activity**
- Foreground activity registers data to be sent over NFC
- On NFC tap and touch, the data is sent
- Receiving device launches Activity to handle the data
Android Beam data format: NDEF message

- **NFC Data Exchange Format**
- Standardized by the NFC Forum
- Can be read/written from/to NFC tags
- Can be exchanged over P2P link
NDEF Message

- NDEF Record 1
  - Type Name Format
  - Type
  - ID
  - Payload

- NDEF Record 2
  - Type Name Format
  - Type
  - ID
  - Payload

- Message has >= 1 records
- Each record is a single piece of data
Corresponding Android APIs

```java
NdefRecord rec1 = new NdefRecord(
    short tnf, // Use NdefRecord.TNF...
    byte[] type,
    byte[] id,
    byte[] payload);

NdefMessage msg = new NdefMessage(rec1, rec2);
```
Some Type Name Formats defined by NFC Forum

- **TNF set to NdefRecord.TNF_WELL_KNOWN**
  - Type field contains well-known Record Type Definition
  - Use NdefRecord.RTD_URI for Uri record
  - Use NdefRecord.RTD_TEXT for Text record

- **TNF set to NdefRecord.TNF_MIME_MEDIA**
  - Type field contains mime-type, eg “image/jpeg”
  - Use “application.vnd/...” for app-specific mimes
Creating an Uri record

```java
NdefRecord uri = new NdefRecord(
    NdefRecord.TNF_WELL_KNOWN,
    NdefRecord.RTD_URI,
    null,
    “http://www.google.com”.getBytes());

// Or, use the convenience method
NdefRecord rec = NdefRecord.createUri(
    “http://www.google.com”);
```
Creating a mime-type record

NdefRecord uri = new NdefRecord(
    NdefRecord.TNF_MIME_MEDIA,
    "application.vnd/my.pkg".getBytes(),
    null,
    payloadData);

// Or, use the convenience method
NdefRecord rec = NdefRecord.createMime(
    "application.vnd/my.pkg", payloadData);
Telling the NFC service which NDEF message to send

// In your Activity onCreate()
NfcAdapter adapter = NfcAdapter.getDefaultAdapter(this);

NdefRecord rec = NdefRecord.createUri("http://www.google.com");

NdefMessage msg = new NdefMessage(rec);

adapter.setNdefPushMessage(msg, this);

// Done!
Summarizing Android Beam data flow

Device A
- Activity X: Sending Application
- Data (NDEF)
- NFC Service
- Data (NDEF)

Device B
- Activity Y: Receiving Application
- Deliver Data in intent
- NFC Service
- Data (NDEF)
What happens on the receiving side?

- Android intent system resolves to activity
- An intent is an abstract description of an operation to be performed
- An intent contains an action, eg “ACTION_VIEW”
- An intent contains data, eg the Uri “http://www.google.com”
- An intent may contain a type, eg “image/jpeg”
How do intents end up in Android apps?

- Activities can register **intent filters** to indicate they can handle certain intents.
- E.g., an Activity intent-filter with action “ACTION_VIEW” and type “image/jpeg” means: this Activity can view JPEG data.
- When an intent is “launched”, Android evaluates all matching Activities.
- If more than one, Activity chooser pops up.
How NDEF messages are delivered to your app

When the receiving device gets an NDEF message over NFC:

- Android creates an intent with action “NfcAdapter.ACTION_NDEF_DISCOVERED”
- Android maps the NDEF TNF/Type to intent data/type
- Android adds the NDEF message to the intent
- Tries to launch an Activity that handles the intent
Intent nfcIntent = new Intent(  
    NfcAdapter.ACTION_NDEF_DISCOVERED);

nfcIntent.setType(..); // or, setData()

nfcIntent.putExtra(  
    NfcAdapter.EXTRA_NDEF_MESSAGES,  
    ndefMessages); // Usually one

startActivity(nfcIntent);
Mapping of TNF/Type to intent type/data

<table>
<thead>
<tr>
<th>Type Name Format</th>
<th>Type</th>
<th>Payload</th>
<th>Intent type/data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNF_WELL_KNOWN</td>
<td>RTD_URI</td>
<td>“google.com”</td>
<td>setData(Uri.parse(“google.com”));</td>
</tr>
<tr>
<td>TNF_WELL_KNOWN</td>
<td>RTD_TEXT</td>
<td>“My Text”</td>
<td>setType(“text/plain”);</td>
</tr>
<tr>
<td>TNF_MIME_MEDIA</td>
<td>“application.vnd/my.pkg”</td>
<td>1101000010100111</td>
<td>setType(“application.vnd/my.pkg”);</td>
</tr>
</tbody>
</table>
What type do I choose for my record?

- Sometimes obvious, eg an Uri
- If you want to allow other/existing apps to handle data you send, use standard mime-types; eg “text/plain”
- If you want only your app to handle:
  - Either use an app-specific mime-type
  - Use an Android Application Record
Android Application Record

- Guaranteed delivery to a specified package
- Even if the package does not support NFC at all, it will still be launched
- Opens up the Play store if the package is not installed
Creating an NDEF message with AAR

```java
NdefRecord rec = NdefRecord.createUri("http://www.google.com");
NdefRecord aar = NdefRecord.createApplicationRecord("com.my.app");
// Creates a msg that delivers the Uri to // “com.my.app”
NdefMessage msg = new NdefMessage(rec, aar);
```
Creating an intent-filter on the receiving side

// In AndroidManifest.xml
<activity ...
   <intent-filter>
      <action android:name="android.nfc.action.NDEF_DISCOVERED" />
      <category android:name="android.intent.category.default" />
      <data android:mimeType="application/vnd.mine" />
   </intent-filter>
</activity>
In your receiving Activity

```java
protected void onCreate() {
    Intent launchIntent = getIntent();
    String action = launchIntent.getAction();
    if (action.equals(NfcAdapter.ACTION_NDEF_DISCOVERED)) {
        // Get the first NdefMessage
        NdefMessage msg = (NdefMessage) intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES)[0];
        // Get the payload of the first record
        byte[] payloadData = msg.getRecords()[0].getPayload();
        // Process payload (on different thread if needed)
        ...
    }
}
```
What about large data?

- Android Beam has APIs that handover to faster transports
- Allows NFC-initiated file transfer in 5 lines of code
- See NfcAdapter.setBeamPushUris()
More information on Beam

Android Beam is built using NFC Forum standards
- NDEF specification and RTDs
- LLCP 1.1 specification
- Simple NDEF Exchange Protocol specification
- Connection Handover specification for large data

See the NFC developer guide:
and the Google I/O 2012 NFC talk:
http://www.youtube.com/watch?v=HkzPc8ZvCco
Android Beam vs social/AirDrop/... sharing

- Beam requires no pre-existing relationship
- Beam requires no data/cloud connectivity
- Beam is less suited for 1-to-many transactions
Summarizing Android Beam

- Really easy to add to your app (~50 lines of code)
- Provides a simple and magic sharing experience
NFC tags
NFC tags

- 40 bytes - 16 KB memory
- Unique or random IDs
- Some allow crypto
- Full smartcards
What can you do with NFC tags?

- **Attach actions** to real physical objects!
- **Launched automatically** when you tap it
- Many places to stick tags:
  - Consumer products
  - Digital signage (smart posters)
  - Smart devices (for pairing)
How are NFC tags handled?

- The beauty of NDEF: this format is used on NFC tags, too
- Reading an NDEF NFC tag ~= receiving the same NDEF data over Android Beam
- AARs can be used on tags as well!
Reading/writing NDEF to tags

- All Android devices support NFC Forum standard tags
  - Type1, Type2, Type3, Type4
- APIs for reading/writing NDEF to tags:

Android also supports low-level tag ops

- Sending/receiving raw data over various interfaces/protocols
- ISO-DEP (ISO/IEC 14443-4)
- Nfc-A or Nfc-B (ISO/IEC 14443-3 type A or type B)
- Nfc-F (FeliCa / JIS-X 6319-4)
- ISO15693
- NFC Barcode (Kovio -> Thinfilm)
Platform tag use cases

- NFC Forum defined “Connection Handover” specification and “Bluetooth SSP”
- Allows using NFC to setup alternate carriers (eg WiFi, Bluetooth)
- Android has implemented Bluetooth audio pairing since JellyBean (Android 4.1)
BebaPay: mobile payments in Kenya
Card emulation and HCE
Card emulation on Android: how did it start?

- Emulate contactless cards
- Contactless payment card contains a secure element
- How would you move this to mobile?
Traditional card emulation in Android

- Put Secure Element in Android phone
- Connect it to NFC Controller
- Let it handle all transactions
- Done...right?
Well...

- A mobile device is not a plastic card
  - Plastic SE is owned by the issuer
- Who owns the mobile phone software?
  - OEM, carrier, OS manufacturer
  - App developers
- Who owns the secure element in a phone?
  - embedded secure elements: platform owner / OEM
  - UICC/SIM: carriers
- Different provisioning models
  - Plastic pre-provisioned
  - Mobile runtime provisioned
Issues around SE: need for protection

- Payment applets contain sensitive data
- Limited space in SE creates scarce resource
- Easiest solution: restrict access
- No public APIs => no 3rd party app developers
What about the app developer?

- You don't want to deal with OEMs, MNOs or platform owners
- You just want to use platform APIs
- What about use cases where a SE is not needed?
Goals for HCE

- Open up NFC card emulation for any app developer without cost but with scale
- Allow card emulation usecases that certainly don't require SE
- Revitalize the NFC ecosystem
HCE Architecture

- Apps in Android can emulate cards
- Android already provides native app sandboxing
- Supports many apps at the same time
- No infrastructure changes
First question: what about security?

- Not using the SE doesn’t mean “not secure”
- Android 4.4 has many security features
  - App sandboxing
  - SELinux
  - ASLR
Thinking about security

- But also, it’s a **smart** phone, not a plastic card
- You can use **location, connectivity, cloud**
- You can use **short-lived tokens** or hardware-backed **keystore**
Second question: Application selection

- **Many cards** can live inside **many apps** on a HCE device
- Which one to **invoke**?
- Android uses ISO7816-4 application IDs (**AIDs**)
- HCE apps register AIDs they handle
- Reader **selects** AID it wants to talk to
HCE protocol flow

Reader (Master) -> Android OS
Select AID “F0123456”
OK + additional data
Command

Android OS -> HCE app X AID “F0123456”
Lookup, resolve to app X
Select AID “F0123456”
OK + additional data

HCE app X AID “F0123456” -> Reader (Master)
Response
What protocol?

- EMVCo has deployed existing specifications for payment
  - These are implementable with HCE
- For closed-loop payment or non-payment apps:
  - Follow “select AID” part of protocol to get app invoked
  - All subsequent data exchange between reader and your app
Application invocation

- HCE apps specify a **service** component in their manifest
- The HCE service can run in the **background**
  - Supports no UI, notification, full UI
- **Service will be launched automatically** on tap
  - (No need for your app to be already running)
- **Significant advantage vs QR**
  - “Tap”, vs
  - Unlock, launch app, get QR code screen to show, get QR code scanned
HCE service AID registration

- OS keeps a lookup table of AID->HCE service(s)
- OS invokes correct HCE service when the reader selects AID
- Conflict resolution
What about the reader?

- Android device can also act as reader
  - eg Nexus 10 device
- See NfcAdapter.enableReaderMode()
- Can build Kiosk/loyalty app with Android devices
Getting started with HCE

Read the developer guide:
Android Beam vs HCE for your app

- HCE can be used with existing reader infrastructure
- HCE signifies a clear master/slave relationship
  - Your phone is the boarding pass, payment card, movie ticket.
- HCE is background-enabled
- Beam is a more equal relationship (peers)
- Beam provides instant sharing of the current context