Provisioning IoT with Web NFC

Zoltan Kis (@zolkis), Intel
Background

✧ JavaScript APIs for IoTivity, Soletta
✧ W3C Web NFC editor
✧ Web access to hardware

Earlier:
✧ Network management (DSL)
✧ Mesh radio networks
✧ DSP (AI applied in robotics)
✧ Mobile, enterprise, cloud
IoT provisioning is complex

Objective: make it easier for IoT solution developers
Agenda

1. IoT deployment scenarios
2. The Physical Web
3. OIC/OCF provisioning
4. End to end provisioning
5. Web NFC details and examples
IoT deployment: sensors, connections, applications

- Smoke, fire, air pollution, CO
- Cameras, motion detectors
- Light sensors
- Temperature, CO2, humidity, barometer, air flow
- Biometric: HRM, EKG, ...

Applications:
- Heating
- Ventilation
- Energy management
- Security
- Emergency
- Medical services
Pivotal questions

✧ Who owns the data?

✧ Who can access the data?
The Physical Web

Google Physical Web: how it works

- Any smart device can have a web address
- Interaction on demand
- Discovery through broadcasting
- Eddystone: message format
  - 16 bit BLE service UUID
  - URL
- Data model: BLE →
  - https://github.com/google/physical-web
  - https://google.github.io/physical-web/
The Physical Web and NFC are complementary

<table>
<thead>
<tr>
<th></th>
<th>Physical Web</th>
<th>NFC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Carrier</strong></td>
<td>BLE, WiFi (mDNS/UPnP/SSDP)</td>
<td>Short range radio (13.56 MHz)</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>~10m/30ft</td>
<td>~10cm/4”</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Compressed URI (URI beacon)</td>
<td>URL, text, MIME (tag or peer)</td>
</tr>
<tr>
<td><strong>Initiated by</strong></td>
<td>Device (broadcast)</td>
<td>User (push and pull)</td>
</tr>
<tr>
<td><strong>Seen by</strong></td>
<td>All devices in range</td>
<td>One device (in practice)</td>
</tr>
</tbody>
</table>
Physical Web: discovery+CRUDN

**Create:** with Web NFC** or Web USB**

* https://github.com/sandeepmistry/bleno
* https://github.com/sandeepmistry/noble

** https://webbluetoothcg.github.io/web-bluetooth/
** https://w3c.github.io/web-nfc/
** https://wicg.github.io/webusb/
Topology: sensors to PC

Local, private setup

✧ Sensor data is private
✧ Storage: local device
Topology: sensors to PC or cloud

Hybrid setup

✧ Sensor data is shared
  ✧ at sensor level
  ✧ via cloud federation

✧ Storage:
  ✧ private cloud or local device
  ✧ enterprise or public cloud

✧ Separate solutions →
  → separate provisioning.

http://bit.ly/1X3x0In
Toplogy: sensors to gateway

- Hybrid setup with gateway
- Gateway can be a role (sensor to sensor topology)
- Separation of the solutions
- Separation of provisioning

http://bit.ly/1X3xOIn
Reality mix: sensors to gateway or cloud

✧ Multiple gateways possible
✧ Multiple topologies

http://bit.ly/1X3xOIn
How to provision all this

“It is unlikely that one provisioning solution will fit all…”

✧ Make simplifying assumptions where possible
  ✧ Application dependent

✧ Move provisioning complexity towards the cloud service
  Note: normal operation should not need the cloud

✧ Devices implement simple mechanisms
  and follow rules dictated by cloud
OIC/OCF concepts: platform, device, resource

- **Device**: 
  - di: "08854960-736F-46F7-BEC2-9E6CBD61BDC9"

- **Resource**: 
  - smallest addressable entity
  - data container
  - href: "/a/light1"
  - rt: "oic.r.light"
  - if: "oic.if.a"
  - status: "on"
  - dimmer: 50

- **Resource**: 
  - href: "/a/light2"
  - rt: "oic.r.light"
  - if: "oic.if.a"
  - status: "on"
  - dimmer: 40
  - color: "red"

- **Device**: the OIC/OCF stack
  - contains resources
  - Modeled as /oic/d resource

- **Platform**: the hardware
  - contains devices
  - Modeled as /oic/p resource
# OIC/OCF concepts: what needs provisioning

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connectivity</strong></td>
<td>CoAP, HTTP, XMPP → IP → WiFi or Bluetooth</td>
</tr>
<tr>
<td><strong>Identity, Addressing</strong></td>
<td>oic://&lt;deviceID&gt;/&lt;resourcePath&gt; → IP address</td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td>Multicast or unicast request on /oic/res</td>
</tr>
<tr>
<td><strong>Resource, CRUDN</strong></td>
<td>RESTful requests on resources →</td>
</tr>
<tr>
<td><strong>Access control</strong></td>
<td>Using /oic/sec/acl, /oic/sec/acm, ...</td>
</tr>
<tr>
<td><strong>Device management</strong></td>
<td>Using /oic/mnt</td>
</tr>
</tbody>
</table>
OIC/OCF concepts: operations

**Discovery:**

```
GET /oic/res?rt="/oic/light"
```

**Create:**

```
PUT oic://088...DC9/a/light/1?rt="/oic/light"
```

**Retrieve:**

```
GET coap://192.168.0.5:5683/a/light/1
```

**Update:**

```
POST oic://088...DC9/a/light/1?status="off"
```

**Delete:**

```
DELETE oic://088...DC9/a/light/1
```

**Notify:**

```
GET oic://088...DC9/a/light/1?obs=0
```

**Device**

```
di: "08854960-736F-46F7-BEC2-9E6CBD61BDC9"
```

**Resource**

```
href: "/a/light1"
rt: "oic.r.light"
if: "oic.if.a"
status: "on"
dimmer: 50
```

```
href: "/a/light2"
rt: "oic.r.light"
if: "oic.if.a"
status: "on"
dimmer: 40
color: "red"
```
Taxonomy of discovery

➔ During provisioning: discover non-provisioned devices
  ◆ By OIC/OCF methods
  ◆ Or by local access to HW, using NFC, USB, ...

➔ During operation: discover configured devices and resources
  ◆ OIC/OCF: Multi/unicast request on /oic/res
  ◆ Google Physical Web: Bluetooth LE broadcast + scanning
Taxonomy of IoT provisioning

OIC/OCF standardized
1. On-boarding (OBT)
2. Security provisioning (PT)
3. Configuration (OIC/OCF)

Application/service specific
- Configuring resources
- Provisioning cloud services
Provisioning flow with NFC using a PD

1. Open service web page
2. Tap NFC tags to PD
3. Send data to service
4. Service runs configuration
5. Tap PD to Gateway
6. Finish by OIC method.
Provisioning flow with NFC using a gateway

1. Tap NFC tags to Gateway
   \[\rightarrow\] transfer keys, parameters
2. Consult service, prepare bootstrap
3. Finish by OIC method.
How to use NFC in OIC/OCF

✧ Onboarding
✧ Provisioning
✧ Configuration
Step 1. OIC/OCF on-boarding

✧ Ownership Transfer Method
✧ Set up networking
  WiFi SSID, Bluetooth pairing etc
✧ Bootstrap next stage
  Provisioning Tool URI
  Credentials
OIC/OCF Ownership Transfer Method (OTM)

1. Discover devices needing OTM
2. OBT queries device ownership
3. Device returns /oic/sec/docxm resource including:
   - Ownership status, supported OTM, current deviceID
4. Establish DTLS session using a method:
   - "just works": anonymous Diffie-Hellman
     Clean room network needed (MitM) → NFC
   - "random pin": PSK-based DH with PIN
     (out of band from device to OBT) → NFC
   - "manufacturer certificate": signed Diffie-Hellman
     with manufacturer's certification
5. Deploy credential type → NFC
   - Symmetric: uses PRF to generate OwnerPSK
   - Asymmetric: owner's public key is deployed
   - Certificate
6. Establish device owner and device ID
   write /oic/sec/docxm and /oic/sec/pstat
On-boarding with NFC tag

1. Read NFC tag to get pre-shared key and network preference for step 4
2. Establish dedicated, secure communication channel
3. Configure device ownership: device ID, update security resources
4. Set up networking (e.g. WiFi SSID, Bluetooth pairing etc)
5. Bootstrap configuration stage (server URI, credentials)
On-boarding with NFC **adapter**

1. Tap OBT to device to read keys and network preference for step 5
2. Establish dedicated, secure communication channel
3. Establish device ownership: device ID, update security resources
4. Tap OBT to device to write device ID, configuration server URI, credentials
5. Device: set up networking (e.g. WiFi SSID, Bluetooth pairing etc)
6. Bootstrap configuration stage using the server URI and credentials.

---

On-Boarding Tool (OBT):

```json
{ init: {
  deviceID: "088...DC9",
  certificate: "...",
  configServerURL: "https://..."
}
...}
```

NFC content pushed to device
Step 2. Provisioning

✧ Establish secure communication channel with PT
✧ Initialize security resources (credentials, ACL, AMS)
✧ Initialize Configuration Source (URI)

PT: Provisioning Tool
ACL: Access Control List
AMS: Access Management Service
CMS: Credential Management Service

Picture from OIC Security Specification 1.0
Security provisioning with NFC adapter

1. Create secure connection with Provisioning Tool as configured during ownership transfer (TLS using OwnerPSK)
2. Write /oic/sec/svc resource (BSS, AMS, CMS)
3. Write /oic/sec/cred resource (credentials)
4. Write /oic/sec/acl resource (access control lists)
5. Configure locally location, timezone, etc, or
6. Use configuration source and configure with OIC → see next

NFC content pushed to device

http://bit.ly/1oqcVLD
Step 3. Configuration

✧ Connect to OIC/OCF network
✧ Retrieve and update /oic/con
✧ Initialize location, time zone, security policies etc.
→ Can be done with NFC adapter.

1. UPDATE Request (configuration)
2. UPDATE Response (configuration)

1. On-boarding process successfully completed
3. Common configuration resource populated/updated

Pictures from OIC Security Specification 1.0
Device configuration mechanisms

1. Prepare configuration fragments with a Configuration Device (CD)
2. Tap CD to device (sensor)
3. Device config manager generates native configurations to services

Configuration fragments:
- CF1
- CF2
- CF3

Steps:
2. Deploy via NFC, USB, network etc
3. Parse route

Conf templates:
- Cf1
- Cf2
- Cf3
- Cf4
- Cf5

Conf trees:
- S1
- S1
- S1
- S1

Services: restart

http://bit.ly/1oqcVLD
Service and end-to-end provisioning

Service provisioning
- OIC services (CMS, AMS, ...)
- Data model: resource database
- Application logic

End-to-end provisioning
- A new resource to the service (owned)
- An existing (shared) resource to the service
Provision a new service with NFC

1. Open web page to cloud service portal
   Authenticate and open provisioning page
2. Tap PD to NFC tags on sensors
3. Provision to the cloud service
   → Update resource model
   → Observe resource (discovery optional)
4. Tap PD to GW to update configuration.
   ✧ Gateway is optional
   ✧ Service may be an app

 CircularProgress

Discover sensors and resources
Programmatically select and provision the ones needed

OR,

 CircularProgress

Tap sensor tags
Parametrized provisioning of the cloud service
Why and when Web NFC?

✧ Allows access to local NFC HW from a web page
✧ Allows the entire service code base on cloud side, without the need to deploy apps to device (if that’s a desired thing):
  - Service provisioning code
  - Application logic
✧ For browsers, but also Node.js
✧ Developed in a W3C Community Group

https://www.w3.org/community/web-nfc/
W3C Web NFC in a nutshell

-Control NFC adapters from a web page
- Secure context, origin → Web NFC ID
- URL, text, JSON, or media as NFC content
- Functionality:
  - Push/write NFC content
  - Watch/read NFC content
- Same technology in cloud, GW and device
- Implemented in browser (Chrome) and Node.js

Specification and code samples available at https://w3c.github.io/web-nfc/
Web NFC: write configuration tag

```javascript
var message = {
    url: "/myportal/iot/provisioning",
    data: [{
        recordType: "json",
        mediaType: "application/json",
        data: {
            networkPreference: "wifi",
            init: { "di": "08854960-736F-46F7-BEC2-9E6CBD61BDC9", "links": [
                {"href": "/a/light/1","rt": "oic.r.light","if": "oic.if.s"},
                {"href": "/binarySwitch","rt": "oic.r.switch.binary","if": "oic.if.a" }
            ] },
            rsaPublicKey: "-----BEGIN PUBLIC KEY----- ..."
        }
    }]
};

nfc.push(message).then(() => {
    console.log('Configuration written.');
}).catch((error) => {
    console.log('Failure, try again.');
});
```
nfc.watch(reader, { url: "*/myportal/iot/provisioning/*" });

function reader(message) {
    console.log("Source: " + message.url);
    // forward the configuration data to the service
    var xhr = new XMLHttpRequest();
    xhr.open("PUT", "https://myportal/iot/provisioning/devices/", false);
    xhr.send(message.data);
    // do other stuff
}
Web NFC implementations

✧ Chromium on Linux, based on neard
✧ Chromium on Android, based on Android NFC
✧ suitable for a provisioning device
✧ Node.js: node-webnfc
✧ Suitable for gateway (e.g. Ostro™OS)
1. Open the provisioning web page
2. Tap tablet on each sensor
3. Tablet sends provisioning data
4. Service provisioning
5. Tap tablet to the gateway
6. Sensors + gateway + service work

Ostro™ OS stack
neard + node-webnfc
REST API server

REST, DB
Application logic
Provisioning web page

Sensors: Intel Edison

Chrome + Web NFC

HTTPS

1,3
Thanks for listening!

Feedback to:
zoltan.kis@intel.com
zolkis @ github, linkedin, twitter, ...

http://bit.ly/1Ru4FFY