nabto

connect – simple and secure

Nabto Solution Design

Overview

BIBLIOGRAPHY

All below documents are available for download from https://www.nabto.com

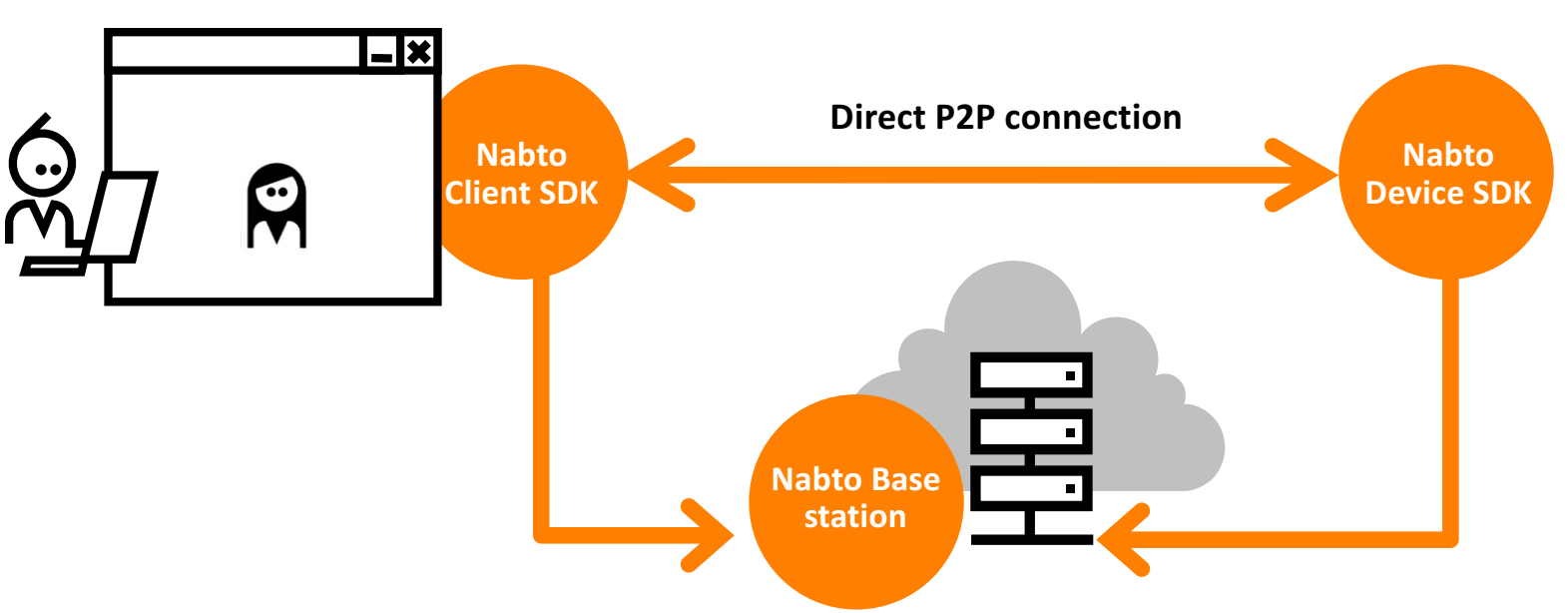
- [TEN023]: "TEN023 Writing a uNabto device application"
- [TEN025]: "TEN025 Writing a Nabto API client"
- [TEN029]: "TEN029 Nabto Platform Specifications"
- [TEN030]: "TEN030 Nabto Tunnels"
- [TEN036]: "TEN036 Security in Nabto Solutions"

PURPOSE

- This document provides a quick overview to get started building Nabto solutions
- For a general introduction to the Nabto platform and its capabilities, start on https://www.nabto.com. [TEN029] provides an overall introduction to the platforms and features.
- Please make sure to read and understand the concepts described in [TEN036] “Security in Nabto Solutions” regardless of the approach taken to Nabto application development

NABTO'S PEER-TO-PEER SOLUTION

nabto



- Nabto provides a simple, award-winning P2P solution
- A solid, secure and high-performing platform – Scandinavian quality software
- Platform in production since 2009
- 1,000.000+ devices deployed on 4 continents as of Q1 2016
- Extremely simple to integrate and operate
- Unsurpassed P2P ratio – symmetric NAT traversal yields 96% success in real life deployment

Fluent Technical and Business support in English, Mandarin, German, French and the Scandinavian languages

How to Benefit from Nabto

nabto

- Nabto provides low latency, direct interaction between two peers using the same P2P techniques known from e.g. VoIP applications and multiplayer games
- Nabto supports secure interaction between clients and very resource constrained devices
- Nabto supports 3 different communication patterns for different application types / scenarios (may be combined into a single application) – see next page

Supported Communication Patterns

nabto

Nabto P2P-RPC:

- Direct interaction for remote control and monitoring
- Device represented to client as regular JSON webservice
- Nabto framework encodes requests into compact binary representation and decodes response back into JSON

Nabto P2P-Streaming:

- Transmission of larger amounts of data through socket like abstractions
- Supports tunneling of existing protocols for zero-effort integration

Nabto Push:

- Device initiated communication towards basestation
- For push notifications or “classic” big data applications

Typical Use Cases

Examples of Nabto P2P-RPC use:

- End-user applications for controlling smart home devices
- Server-based (M2M) applications for server initiated, intelligent telemetry and data acquisition
- Industrial control

Examples of Nabto P2P-Streaming use:

- Small-scale video surveillance and monitoring (consumer installations and small businesses)
- Remote-enabling of legacy HTTP applications

Examples of Nabto Push use:

- Continuous reporting of statistics and sensor data
- Triggering of Nabto P2P-RPC or Nabto P2P-Streaming scenarios
- Push notifications to mobile devices

NABTO P2P-RPC QUICK START

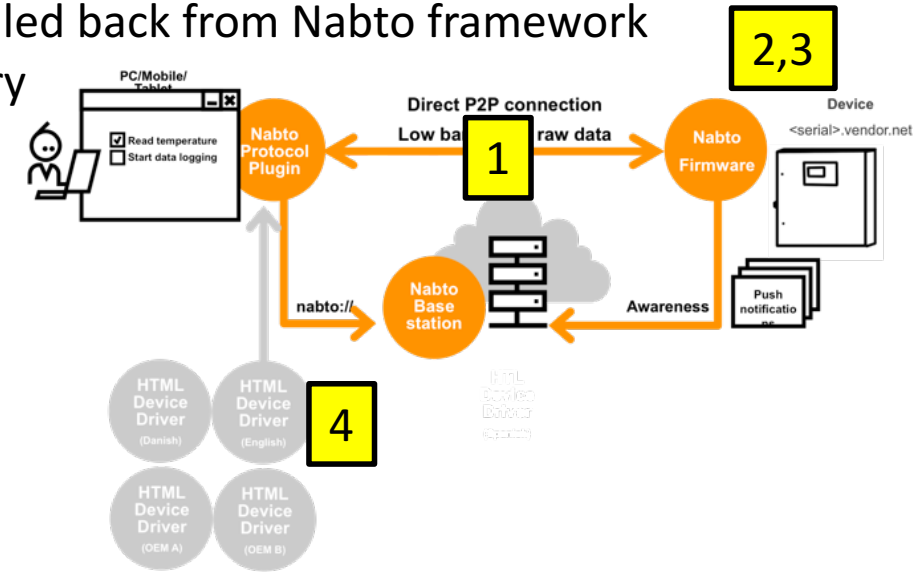
nabto

- It is suggested to start with the examples described on https://www.appmyproduct.com/docs.html
- This will enable you to get a fully functional Nabto P2P-RPC application running instantly
- The following pages describes a Nabto P2P-RPC application in a bit more detail

NABTO P2P-RPC IN 4 SIMPLE STEPS

Developing Nabto P2P-RPC applications is covered in detail in [TEN023] “Building a uNabto Device Application” (device) and [TEN025] (client).

1) **Device interface definition**
   - define available functions on device
   - a simple XML file
2) **Platform integration**
   - provide Nabto framework with network access
   - most likely not a single line of code necessary
3) **Back-end integration**
   - invoke your existing system when called back from Nabto framework
   - a few lines of code typically necessary
4) **Client development**
   - SDKs availalable for native iOS and Android apps, Cordova (+Ionic2) and Xamarin



DEVICE INTERFACE DEFINITION

nabto

An XML description of how the client can interact with the device

- map of function names specified in client to opcodes seen on device
- description of input and output parameters
- description of desired output format
- see [TEN023] sections 6.4 and 6.6 and [TEN025] section 6

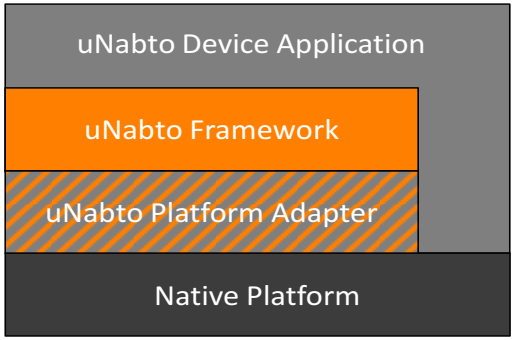
```
<query name="getTemperature" id="0x01">
        <request>
                <parameter name="sensorId" type="uint16"/>
                <parameter name="filter" type="uint8"
default="0"/>
        </request>
        <response format="json">
                <parameter name="temperature"
type="uint16"/>
        </response>
</query>
```

Platform Integration (all comms patterns)

nabto

Thin adapter between Nabto framework and platform:

- basic UDP/IP network access
- timer ticks
- random data for cryptography
- see [TEN023] section 12



- **Ready-to-use adapters available for 15+ hardware platforms**
- **Ready-to-use adapters for FreeRTOS, Linux, uLinux and Windows CE based systems**
- **New platforms can be added through a simple custom adapter**

ARDUINO

freeRTOS

freescale™

GainSpan®

RTX

MICROCHIP PIC® MCU

QUECTEL Wireless Module Expert

uC

Windows CE

BACKEND INTEGRATION

nabto

The Nabto framework invokes the vendor's backend system

- vendor glue code invokes backend system and passes back response
- see [TEN023] sections 6.4 and 6.5

```
application_event_result_t application_event(
    application_request_t* req, buffer_read_t* ibuf, buffer_write_t* obuf)
{
    switch (req->query_id) {
        case 0x01: {
            uint16_t sensor_id;
            uint8_t filter;
            uint16_t temperature;
            buffer_read_uint16(ibuf, &sensor_id);
            buffer_read_uint8(ibuf, &filter);
            temperature = readTemperature(sensor_id, filter);
            buffer_write_uint16(obuf, temperature);
            return AER_REQ_RESPONSE_READY;
        }
    }
    return AER_REQ_INV_QUERY_ID;
}
```

CLIENT DEVELOPMENT (API INTEGRATION)

nabto

Client SDKs available for many platforms

- Static libraries available for low level access to basic C-API for iOS, Android, Windows (win32), macOS and Linux
- iOS SDK Cocoatouch framework (via Cocoapods or www.nabto.com)
- Android SDK (via jCenter)
- Apache Cordova Plugin (via NPM)
- Ionic2 starter apps with high-level typescript wrapper
- Xamarin component
- Example applications for all platforms available at github.com/nabto

Non-app clients also supported

- for M2M communication and custom GUI applications
- allows e.g. server based monitoring and control applications
- allows server driven custom data acquisition, e.g. for integration with BI systems
- see [TEN025] for details

NABTO P2P-STREAMING IN 3 SIMPLE STEPS

Developing Nabto P2P-Streaming applications is covered in detail in [TEN023] section 7 and [TEN025] section 6.5.

TCP tunneling is a special streaming application that connects existing TCP clients to existing remote TCP servers through Nabto, described in [TEN025] section 6.4. A quick start guide for tunneling use is available in [TEN030] “Nabto Tunnels”, explaining the following:

1) Start device tunnel endpoint:

1. install unabto_tunnel executable on device (reverse proxy)
2. start executable: ./unabto_tunnel -d <id> -s -k <key>

2) Start client tunnel endpoint from client application:

1. nabtoTunnelOpenTcp(”<id>”, 554, 5554)

3) Connect existing client application:

1. rtspClient.play(”<userid>:<passwd>@rtsp://127.0.0.1:5554/feed-1”)

NABTO PUSH

nabto

- The Nabto Push implementation allows developers to issue push notifications from even the most resource constrained devices
- The notifications are issued through the standard uNabto framework, no need for integrating to e.g. an HTTPS service from the embedded system
- Current limitations:
  - The Nabto Push implementation currently only supports mobile push notifications
  - As push notification service (the provider between Nabto and the Apple and Android push services), we currently only support Google Firebase
- Near-term roadmap:
  - support for various push notification service providers (e.g., OneSignal, Urban Airship - or perhaps a Nabto hosted solution)
  - support for more use cases (e.g., push of data into a big data solution)

NABTO PUSH – FIREBASE PREPARATION

nabto

- The Nabto client must be integrated with the Google Firebase SDK:
  - Sign up for a free account on https://firebase.google.com/
  - Create a new Firebase app for both iOS and Android
  - Follow the wizard in the Firebase console to setup your app - including follow the guidelines in terms of preparing an iOS push certificate through the Apple dev center
  - Provide your Firebase server key to Nabto (for now, contact Nabto support - later it can be uploaded through the developer portal)

NABTO PUSH – SUBSCRIBING AND NOTIFYING

nabto

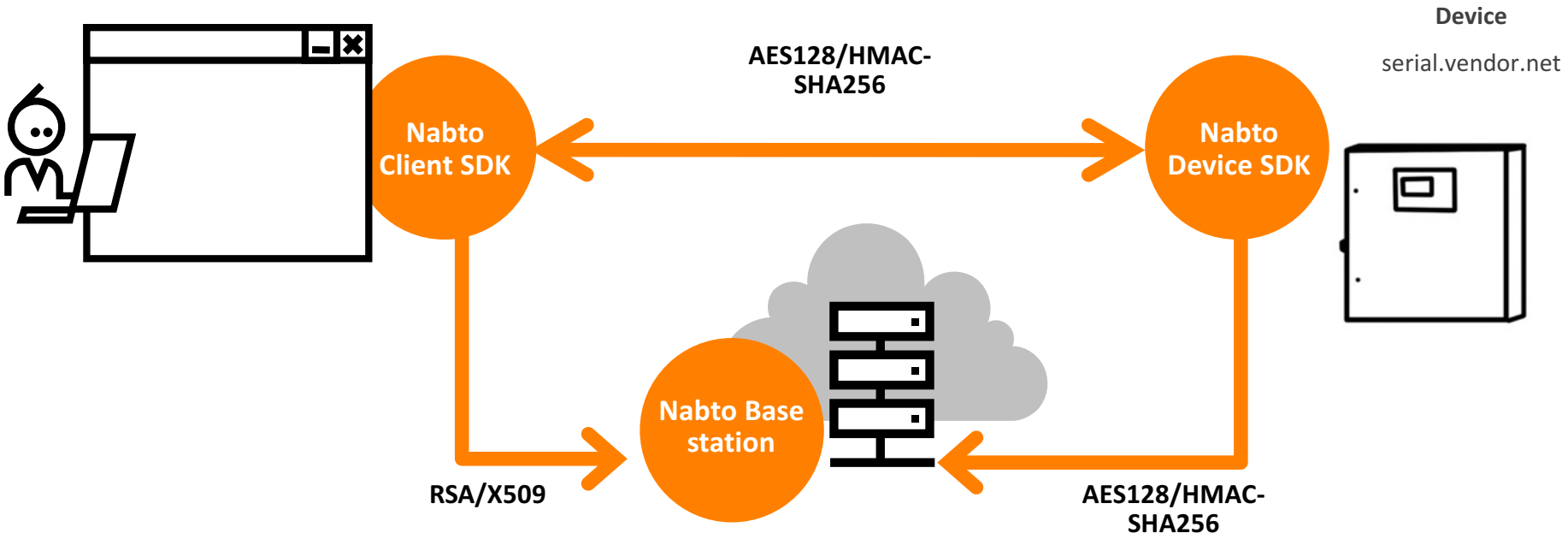
- To subscribe an iOS or Android device to notifications from a uNabto device, a *Firebase Cloud Messaging token* for the app instance is needed
- The token must be passed on to the uNabto device and used there when issuing notifications
  - Nabto provides example apps for iOS and Android showing how to obtain and forward this token
  - Nabto provides a uNabto demo app showing how to use the tokens and the uNabto push module to issue a push notification
- See https://github.com/nabto/nabto-push-demo

nabto

connect – simple and secure

Security

Security in The Nabto Platform



- The platform has been designed from the ground up with security as a focal point
- RSA/X509 authentication of clients and shared secret based auth of devices
- End-to-end encryption using AES128 and HMAC-SHA256
- Session key exchanged through basestation

Please read and understand [TEN036] about Nabto security before deploying any Nabto solution to production!

KEY CONCEPTS

nabto

- Device authentication:
  - a device is a uNabto SDK application, e.g. a camera or thermostat
  - authentication ensures a device with a specific id is what it claims to be
- Client authentication:
  - a client is a Nabto Client SDK application, e.g. a mobile app or desktop application
  - authentication ensures the identity of a specific client is what it claims to be
- Authorization:
  - the device decides which Nabto clients are authorized to connect and which actions the client is authorized to perform
- Secure session establishment:
  - if a client is authorized to connect to a device, a secure connection between client and device is established
  - the Nabto framework ensures confidentiality and integrity (i.e., an eaves-dropper cannot intercept the communication)

DEVICE AUTHENTICATION

nabto

- Device is authenticated towards basestation (and vice versa) using device id and preshared key (PSK)
- Id and key must be available on device before Nabto can run, normally installed at factory
  - the vendor can implement runtime provisioning by providing the id and key from an app to device using e.g. bluetooth
  - Nabto offers some optional services to use for runtime provisioning
- See section 9 of [TEN036] for details

CLIENT AUTHENTICATION

nabto

- Clients can be authenticated using Nabto CA signed certificates (section 8.1 of [TEN036])
- ... or using self-signed certificates (“Paired Public Key Authentication” (PPKA), section 8.2 of [TEN036])
- ... or using application level authentication (e.g. HTTP/RTSP basic auth, section 8.3 of [TEN036])
- Examples of the different approaches shown a bit later in this document

AUTHORIZATION / ACCESS CONTROL

nabto

- The device decides which Nabto clients are allowed to connect
- With application level authentication (section 8.3 of [TEN036]):
  - no access control enforced at the Nabto level, all Nabto clients are authorized to access the device application – *the application* then performs authentication and authorization
  - useful if e.g. an existing RTSP or HTTP basic auth mechanism exists on the device
- With CA based authentication or PPKA (sections 8.1 and 8.2, respectively):
  - the device applications looks up the client identity in an Access Control List (ACL) located on the device
  - if the ACL does not allow the client access to the device and/or requested function, it is rejected
  - the uNabto SDK includes ACL modules to simplify ACL maintenance from the device application
- Maintaining the ACL (adding/removing user identities):
  - either a human user can add/remove user identities from the ACL, i.e. by invoking functionality on the device to edit the ACL from a Nabto Client SDK based mobile app or desktop app
  - or a central service can synchronize a central user/device mapping to the device ACL from a Nabto Client SDK based server app (3rd party solutions exist that do exactly this)

SECURE SESSION ESTABLISHMENT

nabto

- Secure session establishment:
  - a secure session is established by exchanging a unique session key through the basestation
  - the exchange happens through the two secure channels established using the device PSK and a client / basestation RSA SSL handshake, respectively
  - see section 7 and appendix A in [TEN036] for details

EXAMPLE 1: LOCAL PAIRING, NO CENTRAL SERVICE

nabto

- This PPKA scenario allows a user to setup a secure connection to an installed device with no involvement of a central service:
  - When the user starts the vendor's app for the first time, an RSA keypair is created
  - The user *pairs* the app with the device In a *trusted setting*
    - i.e. on a local network
    - perhaps combined with a WPS-like function on the device to provide a temporal restriction ("allow pairing for the next 2 minutes")
  - The pairing step involves transferring the public key from the client app to the device
    - specifically, a SHA-256 hash of the public key is transferred, denoted *the public key fingerprint*
    - the device adds the user's public key fingerprint to its Access Control List (ACL)
  - Later, when the client connects from a remote location, the basestation provides the public key fingerprint to the device
    - if the public key fingerprint is in the ACL, the user is allowed to connect
    - prior to this step, the basestation has verified that the user possesses the matching private key through a standard RSA SSL challenge/response
- The benefit of this scenario is the simplicity and strong security, the drawback is there is no central knowledge of user/device ownership

EXAMPLE 2: PAIRING THROUGH CENTRAL SERVICE

nabto

- This PPKA scenario involves a central service that maintains user/device mappings to centrally control the device ACL:
  - The user logs into his client application and the vendor's app authenticates the user towards the vendor's central user management services
  - If not done before, an RSA keypair is created using the Nabto Client SDK and the public key is transferred from the client to the vendor's central *user management services*
    - specifically, a SHA-256 hash of the public key is transferred, denoted *the public key fingerprint*
- The vendor's central user management services has the following responsibilities:
  - it maintains a user/device ownership relation, ie *this user* has access to *these devices*
  - when a relation is updated or a new client fingerprint is received from a user, the service updates the individual device's ACL by adding or removing a user's public key fingerprint
  - the service invokes the Nabto device to update the ACL using the Nabto Client SDK
  - all devices must then be delivered from the factory with the public key fingerprint of the central services in the ACL
- Later, when the client connects from a remote location, the basestation provides the public key fingerprint to the device
  - if the public key fingerprint is in the ACL, the user is allowed to connect
  - prior to this step, the basestation has verified that the user possesses the matching private key through a standard RSA SSL challenge/response
- This solution gives a better user experience at the cost of increased complexity
  - a 3rd party solution exists to manage user/device mappings and synchronize to the ACL, it integrates with various authentication backends