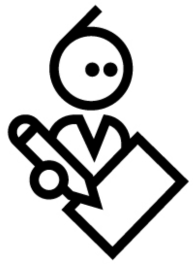
Nabto Platform Specifications

NABTO/001/TEN/029



Contents

1 Abstract

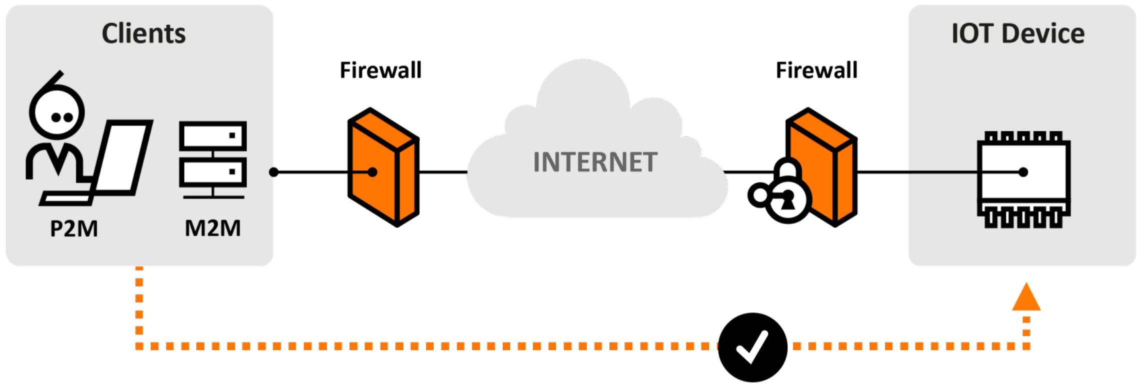
This document summarizes facts about the Nabto platform. Integration options, performance numbers and target platform requirements.

2 Bibliography

TEN017	NABTO/001/TEN/017: uNabto SDK - Compiling Source Code
TEN023	NABTO/001/TEN/023: uNabto SDK - Writing a uNabto device application
TEN024	NABTO/001/TEN/024: uNabto SDK - Writing a uNabto HTML client application
TEN025	NABTO/001/TEN/025: uNabto SDK - Writing a Nabto API client application
TEN036	NABTO/001/TEN/036: Security in Nabto Solutions

3 What is Nabto?

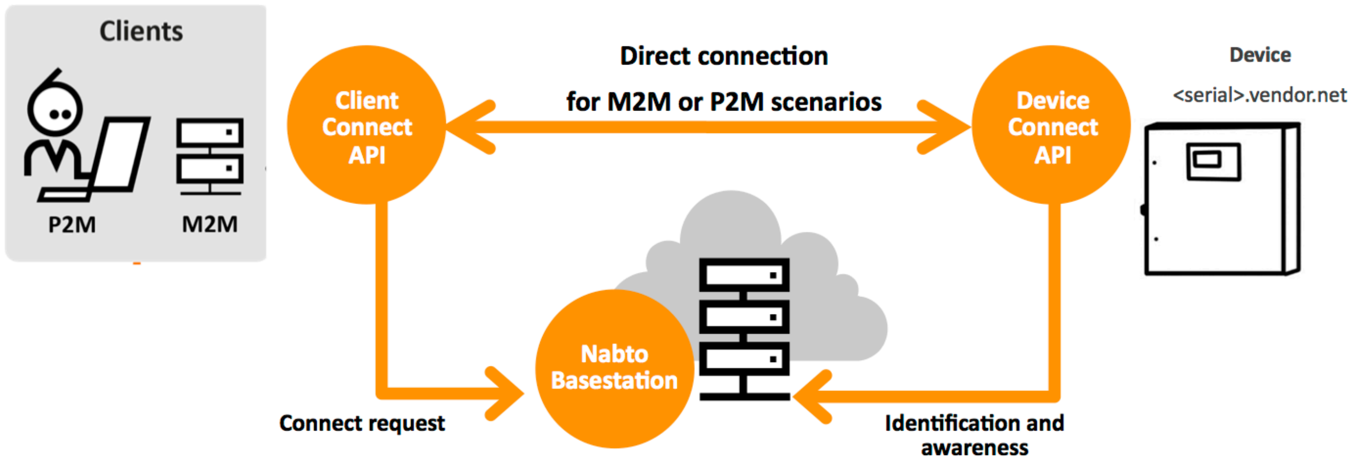
Nabto provides a full communication infrastructure to allow direct, encrypted communication between clients and even very resource limited devices – the Nabto communication platform. The platform supports direct peer-to-peer connectivity through NAT traversal. If either peer's firewall does not allow this, a transparent relay (TURN) is used to establish the connection.



- Vendor integrates Nabto's highly optimized embedded software: 10 kB flash, 2kB RAM needed (see section "Device Requirements")[1]
- Each device is given a unique identity in DNS: e.g., <serial>.<vendordomain>.net
- Ability to seamlessly and securely "call" the device and request data, issue commands or start a data stream – regardless of its location
- Both online and offline environments: Skype™ style through cloud or Bonjour™ style local communication
- Only runtime data stored in cloud: Ensures high privacy and extreme scalability
- Interactive web experience: Nabto is fully integrated into browsers for ease of use, high adoption and simple development
- Secure, transparent tunneling of data between existing client and device applications

[1] The uNabto web protocol can be implemented in less than 1 kB flash if hand-coding the implementation and bypassing the abstractions provided by the SDK

4 Components in the Nabto Platform



The Nabto platform consists of 3 components:

- Nabto **client**: Binaries supplied by Nabto, used by the customer's HTML or native application
- Nabto **device**: The uNabto SDK - an open source framework supplied by Nabto, integrated with the customer's device application
- Nabto **basestation**: Services supplied by Nabto (Nabto- or self-hosted) that mediates connections between Nabto clients and devices. Also supplies the user interface to Nabto HTML clients.

The Nabto client initiates a direct, encrypted connection to the Nabto enabled device – the Nabto basestation mediates this direct connection: The device's unique name, e.g. <serial>.vendor.net, is mapped to the IP address of the Nabto basestation – this is where devices register when online and where clients look for available devices. After connection establishment, the basestation is out of the loop – no data is stored on the basestation, it only knows about currently available Nabto enabled devices.

The client can also discover the device if located on the same LAN and communicate directly without the basestation – useful for bootstrap scenarios or for offline use.

Integrating Nabto on the customer's device is the topic of [TEN023].

The customer's client application may use the Nabto client in different ways: The customer application can be an HTML application that uses the Nabto client to retrieve JSON data in a web application – in this scenario the Nabto client is typically a web browser plugin or mobile app, hosting the customer application. The latter is distributed from the basestation to the client and is denoted an HTML device driver bundle. Writing such an application is the topic of [TEN024].

The customer's client application can also be a native (non-HTML) application, linked with a Nabto client API library. The native client application can use the same request/response mechanism to invoke the device as HTML applications do. Additionally, the native client can establish streaming data connections with the device – this is a popular way of adding seamless, secure remote access capabilities to legacy client and device applications. Native client applications are the topic of [TEN025].

5 Supported Clients

5.1 HTML client applications

The plugin or app executes the customer HTML application. Both plugins and apps can be rebranded with the customer's theme (logos, texts, name in app stores where applicable).

Microsoft Windows (32/64-bit, Win XP SP3 and newer)	Internet Explorer plugin (all versions) Firefox extension (the 3 most recent releases)
Mac OS X	Firefox extension (the 3 most recent releases)
Linux (32/64-bit)	Firefox extension (the 3 most recent releases) Only the official Mozilla build is supported (not distribution specific versions)
Android 3.x and newer	Dedicated app running Nabto HTML applications
iOS 4.x and newer	Dedicated app running Nabto HTML applications
Hosted client	Any browser, no plugin needed - but requires Internet access and the hosted client component to be installed on the basestation

See [TEN024] for details on writing an HTML client application.

Note that there is currently no HTML client plugin support for Google's Chrome or Apple's Safari browsers. Until available, the Hosted client solution listed above is a usable workaround – with the limitation that it only works with an Internet connection and does not support local Bonjour-style discovery. It is still possible to build a custom browser component for all browsers using the Nabto Client API (described below) - to e.g. build a custom video streaming ActiveX or NPAPI based component.

5.2 Native client applications

The Nabto Client API is available as a basic C library with access to all functionality on the platform. Additionally, an object oriented .NET library is provided, wrapping the lower level API in the typical abstractions used on the .NET platform – e.g., it can replace traditional NetworkStream objects in applications upgrading from a proprietary client/server implementation to using Nabto.

The Nabto streaming data capabilities (e.g. video streaming) are only available through the Nabto Client API (not available for HTML clients).

Microsoft Windows (32/64-bit)	C library, .NET 4.0 abstraction
Mac OS X	C library, .NET 4.0 abstraction (requires Mono)
Linux (32/64-bit)	C library, .NET 4.0 abstraction (requires Mono)
Android 3.x and newer	C library with JNI wrapper
iOS 4.x and newer	C library

6 Supported Devices

The Nabto SDK for embedded devices (the uNabto[2] SDK) is available to device vendors as open source.

Basically, a uNabto device application consists of the following components:

- The uNabto **framework**: Abstracts away all the complexity of e.g. security and NAT traversal. Provided entirely by Nabto, can be configured by the vendor.
- The uNabto **platform adapter**: Glue between the uNabto framework and the device platform in question. Enables the uNabto framework to e.g. send/receive UDP packets. Several adapters provided by Nabto as part of the open source SDK (see below), vendor may implement adapters for non-supported platforms.
- Glue between the uNabto framework and the vendor's backend application (e.g., invoke backend upon client request). Implemented by vendor.

Nabto provides a set of platform adapters, ready for use as is or as basis for new vendor specific adapters:

RAKWireless	RAK415 and LX520
Microsoft	WIN32 (x86 and x64), Windows CE
Linux	Any Linux and uClinux variants, just need a cross gcc toolchain
FreeRTOS	Full integration through FreeRTOS+
MicroChip	PIC18 and PIC32
Freescale	ColdFire
Renesas	RL78 and RX600
Atmel	AVR gcc
Quectel	M10
RTX	RTX4100 and RTX4140
Gainspan (on chip)	GS1100

[2] Pronounced *micro-Nabto*

Gainspan (at cmds)	GS1100 and GS1500
Arduino	
Mbed	NXP LPC1768 (Cortex M3)

See [TEN023] for details on the components constituting a uNabto device application.

The resource requirements for a uNabto device application very much depend on the target architecture and desired features. The platform is module based so features can be omitted from compilation as desired to save memory / flash. See [TEN017] for details on uNabto source code configuration.

7 Security

The Nabto platform uses X509/PKI for client authentication and for initiating a secure communation channel from client to device. Devices uses shared secret based authentication (HMAC-SHA256/AES-128) and for establishing a secure communication channel back to the initiating client. The basestation plays a mediating role, passing identity of the authenticated client to the device and exchanging a session key between client and device for data confidentiality.

Full PKI based security (vs shared secret based device security) is intended as a later platform feature (will be prioritized upon customer request).

Access control is enforced at three levels:

1. Coarse grained access control on the basestation: Is the connecting client allowed to connect to devices in the requested domain?
2. Connection level access control on the device: The device receives the encrypted identity of the connecting client from the basestation and may compare this against an Access Control List maintained on the device.
3. Function level access control on the device: For each function invoked by the client on the device, the encrypted identity of the client is supplied by the client. The device may compare this identity against an authorization matrix maintained on the device.

Access control is supported through basic mechanisms on the uNabto platform (access to identity and connection information) as well as through application level modules provided as part of the SDK (to maintain access control and privilege lists). See [TEN023] for details on the basic mechanisms and the supplied modules.

Security in Nabto solutions is described in detail in [TEN036] – a very important read for integrators of the Nabto technology.

8 Network

8.1 Peer-to-peer support

The Nabto platform ensures direct, peer-to-peer connection will be established in all network configurations that theoretically allow this. If either peer’s firewall does not support UDP hole punching, a transparent relay (TURN) is used to establish the connection. It is completely transparent to the client application, although native client applications can query the actual connection type (to e.g. disconnect long running relays if streaming HD video).

The table below defines the possible combinations: If a field contains "ok", a peer-to-peer connection can be established. For instance, it is possible to establish a peer-to-peer connection between a peer behind a port restricted NAT and another peer behind an address restricted NAT. It is not practically (even though theorectically) possible to establish a connection between a peer behind a symmetric NAT to a device behind another symmetric NAT.

	full cone	**address restricted**	**port restricted**	**symmetric**	**open**
full cone	**ok**	**ok**	**ok**	**ok**	**ok**
address restricted	**ok**	**ok**	**ok**	**ok**	**ok**
port restricted	**ok**	**ok**	**ok**	**ok**	**ok**
symmetric	**ok**	**ok**	**ok**	**fail**	**ok**
open	**ok**	**ok**	**ok**	**ok**	**ok**

Firewalls on the Internet are not evenly distributed between the different types above. Nabto has experienced that the population of firewalls is very different between consumer (normally inexpensive and hence simpler) and industrial/corporate, but also from country to country and the type of end-user client (cellular vs. fixedline based like ADSL/cable).

This chart presented is a based on real life data from an application that is both consumer and industrial based. The geographical location of the collected data is mainly US.

The overall distribution of firewall types in the series:

Observations	Percentage
	ratio of total
full cone	15,36
address restricted	10,47
port restricted	51,29
symmetric	21,34
open	1,54

Which amounts to the following (P2P) success matrix:

Probability (success – P2P)						
	full cone	address restricted	port restricted	symmetric	open	**TOTAL**
full cone	2,4	1,6	7,9	3,3	0,2	**15,4**
address restricted	1,6	1,1	5,4	2,2	0,2	**10,5**
port restricted	7,9	5,4	26,3	10,9	0,8	**51,3**
symmetric	3,3	2,2	10,9	0,0	0,3	**16,8**
open	0,2	0,2	0,8	0,3	0,0	**1,5**
TOTAL	**15,4**	**10,5**	**51,3**	**16,8**	**1,5**	**95,4**

And the following failure (relay) matrix:

Probability (failure – relay)						
	full cone	address restricted	port restricted	symmetric	open	**TOTAL**
full cone	0,0	0,0	0,0	0,0	0,0	**0,0**
address restricted	0,0	0,0	0,0	0,0	0,0	**0,0**
port restricted	0,0	0,0	0,0	0,0	0,0	**0,0**
symmetric	0,0	0,0	0,0	4,6	0,0	**4,6**
open	0,0	0,0	0,0	0,0	0,0	**0,0**
TOTAL	15,4	**0,0**	**0,0**	**4,6**	**0,0**	**4,6**

In a pure consumer setup the P2P connection success rate will be higher.

8.2 IPv6 support

The Nabto platform currently does not natively support IPv6; all network interaction is currently IPv4 based. Support for IPv6 is scheduled for Q3 2015.

8.3 uNabto Device Network Environment

uNabto devices need *outbound* Internet access to two UDP ports on the Nabto basestation (the host to which a given device name resolves to – e.g., demo.nabto.net resolves to the demo basestation at 195.249.159.159). Per default these are configured as follows:

- Basestation's Controller service: UDP port 5566
- Basestation's uDirectory service (GSP): UDP port 5562
- Basestation’s TCP gateway (if device needs TCP relay): TCP port 5568

To be able to establish a peer-to-peer connection, the device must be able to send packets to any UDP host and port through its firewall.

8.4 Client Network Environment

Nabto clients need *outbound* Internet access to a subset of the following ports on the Nabto basestation (default port numbers):

- Basestation’s STUN service: UDP port 3478
- Basestation’s Controller service: UDP port 5566
- Basestation’s TCP gateway: TCP port 5568
- Basestation’s HTTPS service: TCP port 443
- Basestation’s HTTP service: TCP port 80

Ideally outbound access through the firewall is needed for all these services, but the Nabto peers also work in more restrictive configurations if only partial functionality is needed, as seen from the table below.

Additionally, to be able to establish a peer-to-peer connection, the client must be able to send packets to any UDP host and port through its firewall.

CLIENT PORTS OPEN FOR OUTBOUND ACCESS	**STUN UDP 3478**	**Controller UDP 5566**	**Gateway TCP 5568**	**HTTPS TCP 443**	**HTTP TCP 80**	**Full open UDP**
P2P connections	Yes	Yes	No	No	No	Yes
TCP relay fallback connections	No	Yes	Yes	No	No	No
HTTP relay fallback connections (client only)	No	No	No	No	Yes	No
Initial installation of HTML device driver	No	No	No	Yes	Yes	No

9 Basestation

9.1 Capacity

The Nabto basestation capacity is defined by the chosen Nabto license and available ressources on the host machine (memory, CPU, network bandwidth). Each online device requires about 10 kB of memory on the basestation. Our reference hosting platform is an Amazon EC2 small instance (single core) tested capable of handling 10.000 devices. An Amazon EC2 c3.xlarge instance (4 cores) is tested capable of handling 100.000 devices.

Each device registered with the basestation sends an alive message every 10 seconds per default: 25 bytes sent to and received from the basestation. This amounts to 12 MB per device per month. For an EU Amazon EC2 instance, this amounts to about 70 USD/month in idle traffic charge for 100.000 devices (April 2014 prices). Cost for actual usage comes on top of this (e.g., connect requests, relayed traffic, HTML device driver updates).

9.2 Operation / Deployment

The Nabto basestation is a set of services running on a public IP address, listening on a few UDP and TCP ports. For production use, the basestation is currently supported on Linux type systems. The basestation software as such also runs on Windows and Mac OS X (and with slight adaptations likely also on any other Unix variants). But no management / monitoring service integration is available there – this can be added upon customer request.

The Nabto basestation can be hosted in the customer's own server environment, at Nabto's hosting facilities or in the cloud – it runs well with a variety of VPS providers, including Amazon EC2.

Load can be distributed amongst multiple basestations through DNS – by changing the DNS mapping, the basestation instance with which a device registers can dynamically change.

The basestation does not maintain any persistent state, hence simple failover is possible through a hot standby: Once the spare instance comes online, all devices re-register with the new instance and the exact state as before the failover is re-established.

9.3 Hosting of the Basestation

Nabto offers optional hosting of customers' basestations. Hosting typically takes place in Nabto's data center or at Amazon AWS. Standard hosting comes with no SLA and with no automatic failover mechanisms employed.

A high availability hosting option with SLA is also available:

- Guaranteed 99.95% uptime on a monthly basis.

- Compensation of 1% discount on the monthly hosting fee per minute downtime that exceeds the threshold, capped at 25%/month.

High availability hosting SLA fine print:

- "Downtime" means that the basestation's state prevents users from being able to access devices associated with the basestation through the device's name.
- High availability hosting requires that the customer accepts Nabto runs the basestation at Amazon AWS.
- Controlled service windows announced well in advance do not affect SLA.
- Force majeure: SLA does not cover if Amazon hosting services becomes unavailable simultaneously in multiple availability zones. DoS attacks towards the basestation is not covered by the SLA.

High availability hosting is approximately 25% more expensive than standard hosting. Please see http://nabto.com for more info on pricing or contact sales@nabto.com.

10 Streaming data performance and limitations

When using either Nabto TCP tunnels or raw Nabto streams as detailed in [TEN025], the possible throughput is defined by the following parameters:

- the Nabto stream MTU size - as of writing, it defaults to 1311 bytes (**mtu**)
- the roundtrip latency between peers - either directly or through basestation for relayed connections (**rtt**)
- the Nabto stream window size (**win_size**)

The theoretical throughput in an ideal scenario without packet loss, duplication or reordering:

throughput = mtu_size * win_size / rtt

For instance, if roundtrip time between peers is 200 ms, the default window size of 100 yields a throughput of about 5 Mbps. In a relay scenario, the rtt is often roughly the double, meaning that the expected throughput drops to 2.5 Mbps.

The current implementation of the Nabto stream has the following limitations:

- It is assymmetrical in the sense that performance is optimized for throughput from device to client (i.e., best performance in a typical video streaming scenarios and only limited performance when e.g. pushing a firmware update).
- The window size is static (configured at compile time), a worst case estimate must be made during development - no feedback is possible from actual network conditions.
- If the window size is too large on platforms with limited CPU power, there is a risk that the tunnel will consume too much CPU.

Careful analysis and tests must be performed to balance the window size with throughput requirements, CPU and memory consumption on the target platform.