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.

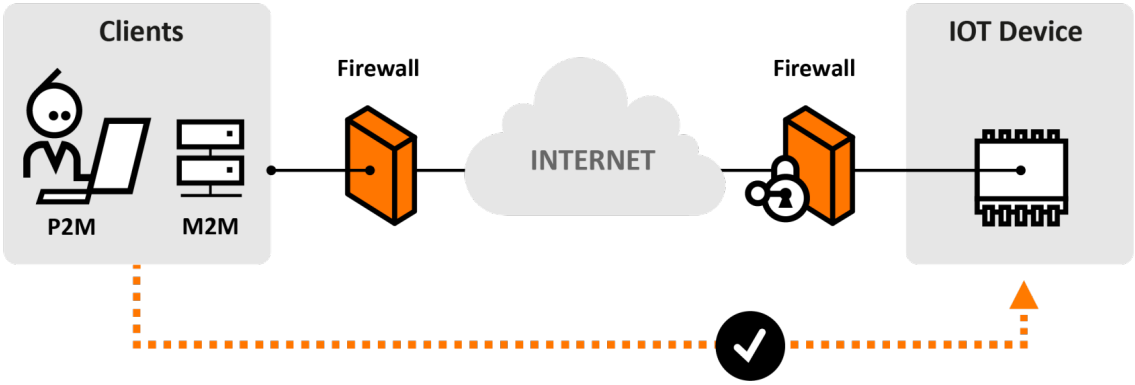
2 Bibliography

All documents are available from https://www.nabto.com/downloads.html:

TEN023	NABTO/001/TEN/023: uNabto SDK - Writing a uNabto device application
TEN025	NABTO/001/TEN/025: uNabto SDK - Writing a Nabto API client application
TEN036	NABTO/001/TEN/036: Security in Nabto Solutions
TEN040	NABTO/001/TEN/040: Nabto Solution Design Overview

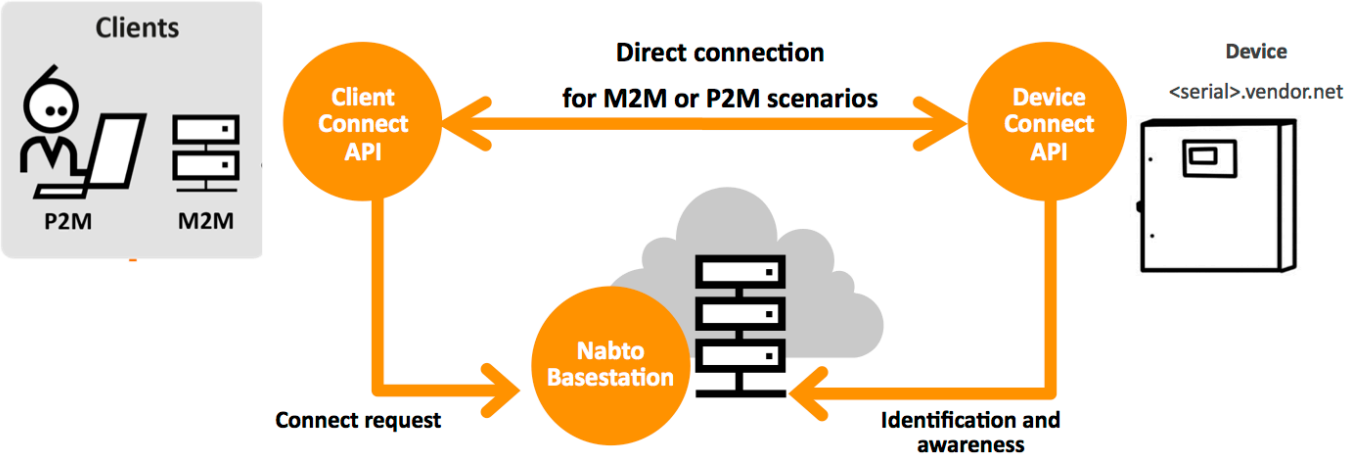
3 What is Nabto?

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: 20 kB flash, 2kB RAM needed (see section 6)
- Clients are enabled to seamlessly and securely "call" the device and request data, issue commands or start a data stream – directly, regardless of either peer's 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
- Secure, transparent tunneling of data between existing client and device applications

4 Nabto Platform Basics



The Nabto platform consists of 3 components:

- Nabto **client**: Libraries supplied by Nabto, used by the customer's 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.

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 client and device communicates directly with each other, 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].

Nabto client applications are developed using the Nabto Client SDK described in [TEN025]. The Nabto Client SDK is the lowest level way of developing a Nabto application - several wrappers exist on top of this lowest level SDK to provide a more abstract experience, for instance for developing Cordova/Ionic hybrid apps or just simplify native Android and iOS app development.

4.1 Nabto Communication Patterns

The Nabto platform supports 3 communication patterns that will be referenced throughout this document:

RPC: The Nabto P2P-RPC communication mechanism allows a client to securely invoke a remote function on a Nabto device. The device implements an interface definition shared between client and device, the client works with normal JSON documents, exchanged in a compact representation with the device.

Streaming: Nabto P2P-Streaming can be used for retrieving larger amounts of data from a device or sending e.g. a firmware update. With sufficient resources available on the device, Nabto P2P-Streaming can be used for high performance streaming suitable for video scenarios.

Push: Nabto Push is used for communication initiated by the device, for instance to implement mobile push notifications or to support big data scenarios where data is collected centrally for further analysis. Nabto Push can also trigger an M2M scenario using RPC or Streaming - e.g. when a certain condition is triggered, the device sends a Nabto Push message and a server function invokes an RPC function or streams data.

5 Supported Clients

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.

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 and high level Java abstraction, jCenter support for simple installation
iOS 4.x and newer	C library with high level Objective-C wrapper and example integration from Swift, Cocoapods support for simple installation
Cordova	iOS and Android cordova plugin available for hybrid apps
Ionic	Typescript based wrapper using the Nabto Cordova plugin, production ready starter apps to use as starting point

6 Supported Devices

The Nabto SDK for embedded devices (the uNabto[1] 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
Gainspan (at cmds)	GS1100 and GS1500
Arduino	

[1] Pronounced *micro-Nabto*

Mbed	NXP LPC1768 (Cortex M3)

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

6.1 Module requirements

List of necessary conditions for Nabto P2P to be integrated on an IoT module:

1. Module can run custom applications using an SDK or similar
2. Module has resources to run a C-application and UDP stack
3. Module has a UDP/IP stack and can send/receive UDP packets
4. Module can run without a host computer connected
5. Manufacturer can provide toolchain and build C code

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.

7 Security

The Nabto platform uses X509/PKI for client authentication in remote access scenarios 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.

Local (offline) communication between two devices on the same LAN can either be in cleartext or encrypted using a pre-shared secret. For instance, this can be boot-strapped through a QR code scanned by the client, representing a secret installed at the factory on the device. On the established channel, a client specific secret can then be exchanged.

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? A webhook can be configured through the Nabto Cloud console where customers where a custom service is queried for each connect request if a connection should be allowed to proceed. See section 9.2.1 below.
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**	**relay**	**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 support IPv6 through relay. The next generation Nabto Edge platform released in 2020 have full IPv6 support, ie also for P2P connections.

8.3 uNabto Device Network Environment

uNabto devices need *outbound* Internet access to two UDP ports on the Nabto basestation servers. Nabto can provide the specific IP addresses for the servers in your solution if necessary. Per default the ports are configured as follows:

- Basestation's Controller service: UDP ports 5566 (and port 5565 for Nabto Cloud based solutions)
- 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 servers (default port numbers):

- Basestation's STUN service: UDP ports 3478 and 3479
- Basestation's Controller service: UDP ports 5566 (and port 5565 for Nabto Cloud based solutions)
- 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 3478-9**	**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

9 Basestation

The Nabto basestation is as discussed in the introduction a central service in Nabto solutions: Devices register with the basestation to make themselves available for connections from client applications. It mediates connection requests from clients to setup P2P connections and supports fallback to a relay if not possible. The relay gateway is also part of the basestation.

The basestation can either be used as a pure SaaS solution where the vendor does not have to worry about anything in terms of deployment, scaling, geographical distribution etc. We denote this the Nabto Cloud. Or the basestation software can be purchased and hosted on premises.

9.1 Nabto Cloud

With the Nabto Cloud SaaS basestation deployment, Nabto provides clusters of basestations in datacenters all around the world to maximize availability and performance: The vendor does not have to worry about where users will be located, the individual device will automatically register with a basestation in the closest datacenter. Clients will automatically be routed to the correct basestation. If a device is moved, it automatically registers with the new closest datacenter.

Capacity is automatically adjusted to maintain connection mediation and relay performance, regardless of workload. So the vendor does not have to worry about load balancing.

The Nabto Cloud is compatible with all Nabto protocol versions so both existing and new projects can benefit from it. The infrastructure currently includes data centers in EU, US, Southeast Asia and mainland China - and more will be added as demand requires.

This is the recommended approach to simplify the overall solution while minimizing cost and maximizing the user experience in terms of availability and performance. However, some customers have requirements for isolated services (no co-hosting with other customers), so the basestation can also be managed as standalone instances, see below.

9.2 Solution management using the Nabto Cloud Console

Nabto customers manage Nabto Cloud based solutions through https://console.cloud.nabto.com. Through the console, the customers can create licenses, analyze usage and activity and manage integrations.

Console Account Logout

Home
Domains

Devices for product1.nabto-enterprise-customer.net

Create New Devices | Show All Device Keys | Batch number: 1 | Show Batch

Device ID	Region	Last Update	Created	Batch ID
7qfnhwbg.product1.nabto-enterprise-customer.net	eu-west-1	2017-10-20T04:29:23.383Z	2017-10-20T04:27:48.434Z	1
e4wrus4w.product1.nabto-enterprise-customer.net			2017-10-20T04:27:48.467Z	1
fhq3pjzj.product1.nabto-enterprise-customer.net			2017-10-20T04:27:48.402Z	1

Figure 1: Device overview

Account Logout

Device Info for 7qfnhwbg.product1.nabto-enterprise-customer.net

Enable Tracelog | Show Device Key

No device info available, device offline

Device Log

Timestamp	Event type	
2017-10-20T04:29:00.292Z	DEVICE_CONNECTION_OPEN	{ "eventType": "DEVICE_CONNECTION_OPEN", "basestation": { "region": "eu-west-1" }, "device": { "id": "7qfnhwbg.product1.nabto-enterprise-customer.net", "ip": "109.153.142.54", "port": 52862, "versionString": "4.1.1" }, "connection": { "type": "P2P", "duration": 480, "receivedBytes": 0, "sentBytes": 0, "receivedPackets": 1, "sentPackets": 0 }, "domain": "product1.nabto-enterprise-customer.net", "timestamp": "2017-10-20T04:29:00.292Z" }
2017-10-20T04:29:00.207Z	CLIENT_CONNECTION_OPEN	

Figure 2: Event data for an individual device

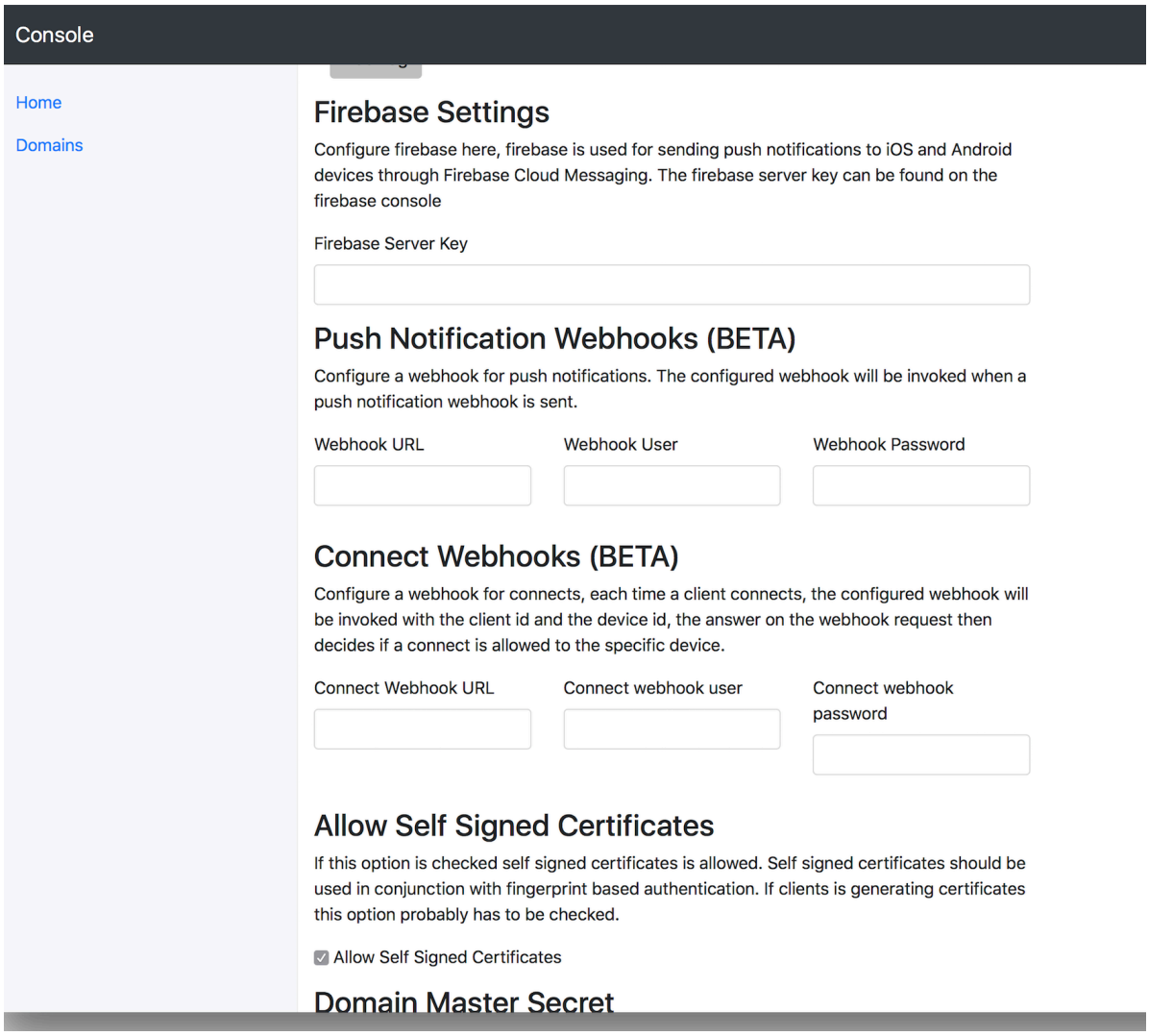


Figure 3: Manage integrations for solution

The Nabto Cloud management console is seen in the screenshots above.

As of writing, 4 different integration options are provided as seen in the following sections.

9.2.1 Central authorization

A webhook can be configured that allows a 3rd party webservice to decide if a connection is allowed to be established to a specific device for the given user: Whenever a client requests a connection to be established to a device, this webhook is invoked. Nabto passes along the identity of the current user (always the RSA public key fingerprint and also the user identity if using signed certificates), the device id, IP addresses of both peers and any optional authorization information provided by the user (ie, an application specific token). See section 8 in [TEN036] for further information about authentication and authorization in Nabto.

9.2.2 Mobile push notifications through Firebase

This integration allows uNabto devices to send push notifications to iOS and Android mobile devices (and more) through Google Firebase. This requires the vendor (not the end-user) to have signed up for a Google Firebase account. See [TEN040] for more information about using mobile push notification including links to examples for managing tokens and sending a push notification from the the uNabto device.

9.2.3 General push integration

This integration allows the uNabto device to send data on its own initiative to an arbitrary webservice, for instance for an analytics solution or a notification mechanism that does not require the device management complexity as mobile push notifications do.

This can be used with other Nabto communication patterns - for instance, if the device provides a Nabto RPC interface, a central Nabto client can trigger specific operations on the device in response to e.g. an alert issued by the device using a push message. Or larger amounts of data can be retrieved from the device if it supports Nabto Streaming.

9.2.4 Nabto Cloud API integration

External applications can use the Nabto Cloud functionality through the Nabto Cloud API. This allows development of custom software and services that can e.g. access device status, usage statistics and create licenses. See https://nabto.github.io/nabto-cloud-api-doc for further information.

9.3 Private basestation instances / self hosting

Customers who prefer not to use the Nabto Cloud Services may either host individual basestation instances themselves or purchase individual instance hosting at Nabto, this way of hosting (as opposed to using the Nabto Cloud) is described below. It is in the roadmap to provide a Nabto Private Cloud with the general abilities as described above for the Nabto Cloud but running on premises at the individual customers.

9.3.1 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 and relayed traffic).

9.3.2 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.

9.3.3 Hosting of the Basestation

Nabto offers optional individual hosting of customers' basestations in addition to the Nabto Cloud deployment described above. Individual hosting typically takes place in at Amazon AWS. Standard individual 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.