hochschule mannheim

# UMTS-Mobilfunkbrücke zur Messdatenübertragung mit OpenVPN

Gerrit Schröder

Schanzstraße 46; 67063 Ludwigshafen am Rhein

gs@linkus.de

Studienarbeit

Studiengang Automatisierungstechnik

Fakultät für Elektrotechnik

Hochschule Mannheim

15.01.2016

Betreuer

Prof. Dr.-Ing. C. Hübner, Hochschule Mannheim

**Schröder, Gerrit**:
UMTS-Mobilfunkbrücke zur Messdatenübertragung mit OpenVPN / Gerrit Schröder. –
Studienarbeit, Mannheim : Hochschule Mannheim, 2016. v Seiten.

**Schröder, Gerrit**:
Mobile UMTS-Bridge for transfering measuring data with OpenVPN / Gerrit Schröder. –
Studienarbeit, Mannheim : University of Applied Sciences Mannheim, 2016. v pages.

## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, 15.01.2016

Gerrit Schröder

# Abstract


***UMTS-Mobilfunkbrücke zur Messdatenübertragung mit OpenVPN***

In dieser Studienarbeit wird das Gerät *NB 1600* der Firma Netmodule zusammen mit einem Temperaturmessumformer in Betrieb genommen. Dazu wird eine VPN-Verbindung mit *openVPN* eingerichtet. Durch diesen Tunnel wird die serielle Datenverbindung vom *NB 1600* über eine TCP-Verbindung zu einem Linux-PC übertragen. Die Messdaten werden in einer Datenbank gespeichert und visualisiert. Das *NB 1600* ist über UMTS angebunden und überträgt Messdaten ohne kabelgebundene Netzwerke.

***Mobile UMTS-Bridge for transfering measuring data with OpenVPN***

In this project thesis the device *NB 1600* from the company Netmodule is put in operation in combination with an temperature transducer. Therefore an VPN-Connection is established with openVPN. Through this tunnel the serial data connection is forwarded from the *NB 1600* via a TCP-Connection in the Virtual Private Network (VPN). The measurig data is stored in a database und visualisated. The *NB 1600* is connected via UMTS and is able to transmit the measuring data without cable-networks.

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| ASCII | American Standard Code for Information Interchange |
| DB | database |
| DDNS | dynamisches DNS |
| DHCP | Dynamic Host Configuration Protocol |
| DNS | Domain Name System |
| IP | Internet Protocol |
| SIM | Subscriber identity module |
| SSH | Secure Shell |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| UMTS | Universal Mobile Telecommunications System |
| USB | Universal Serial Bus |
| VPN | Virtual Private Network |

# Kapitel 1

# Einleitung

## 1.1. Problemumfeld

Im australischen Outback wird fernab von Zivilisation Landwirtschaft betrieben, die zunehmend automatisiert wird. Die zu überbrückenden Entfernungen von Messeinrichtungen sind für kabelgebundene und Nahfunk-Systeme zu groß. Eine UMTS-Verbindung ist jedoch vorhanden.

## 1.2. Aufgaben und Ziele

Ziel dieser Arbeit ist es, die Temperaturmesswerte eines Temperaturmessumformers mit dem Industrie-Router *NB 1600* der Firma *Netmodule* über das UMTS-Netz zu übertragen und verfügbar zu machen. Außerdem ist zu ermitteln, wie viel Datenvolumen die Übertragung der Messwerte im Monat benötigt.

## 1.3. Struktur der Arbeit

Zunächst wird die benötigte Hardware aufgezählt und die Installation der vorausgesetzten Software erläutert. Die Bedienung des *NB 1600* wird anhand von Screenshots der Weboberfläche verdeutlicht. Die Scripte werden vorgeführt. Dabei werden immer erst Teilprozesse erklärt und in Betrieb genommen, bevor mit übergeordneten Strukturen fortgefahren wird. So ist es möglich, die aufgeführten Schritte nachzuvollziehen und nachzustellen.

# Kapitel 2

# Aufbau

## 2.1. Hardware

Die Hardware des Projektes besteht aus

- *NB 1600* der Firma Netmodule
- *ADAM-4520* (bidirektionaler serieller Umsetzer der Firma Avantech)
- *Sensor-Board* aus der Diplomarbeit von Muhammad Hanif Kamarulzaman
- Linux-PC
- Notebook
- Android-Mobiltelefon

Das *NB 1600* ist ein Embedded-Linux-System mit:

- einem Universal Mobile Telecommunications System (UMTS)-Modem
- zwei Ethernet-Schnittstellen
- zwei binären Ein- und zwei binären Ausgängen
- einer seriellen Schnittstelle nach RS232
- einem USB-Host-Anschluss

[NetModule AG, 2015b] Das Gerät kann auf einer Hutschiene montiert werden. Im Inneren wird eine SIM-Karte mit einem UMTS-Datentarif eingesteckt.

Das *ADAM-4520* ist ein bidirektionaler Umsetzer zwischen einer RS232[1]- und einer RS485[2]- oder einer RS422[3]-Schnittstelle. In diesem Projekt wird die RS485 Schnittstelle verwendet. Das Gerät ist auf 1200, 2400, 4800, 9600, 19200, 38400, 57600 und 115200 Baud konfigurierbar. Es ist auf 9600 Baud eingestellt. [Advantech, 2015]

Das *Sensor-Board* misst die Umgebungstemperatur und sendet diese zyklisch über die RS485-Schnittstelle. Die Übertragung erfolgt als ASCII-Text. Die Temperatur wird in der Einheit *Grad Celsius* mit 9 Nachkommastellen übertragen

Der Linux-PC hat einen Internetanschluss. Die Hardware-Ausstattung wird nicht näher erklärt. Die Einrichtung des openVPN-Servers wird im Kapitel 4.3 erklärt, die Funktionsweise der Scripte in den Kapiteln 5.3 und 6.

Das Notebook besitzt eine Ethernet-Schnittstelle, einen DHCP-Client, einen SSH-Client und einen Webbrowser.

Auf die genaue Ausstattung des Mobiltelefons wird hier nicht näher eingegangen.

## 2.2. Software

Der Linux-PC wird mit *Debian 7.7.0 (Wheezy)* und dem Kernel *3.2.0-4-686-pae* betrieben. Folgende Programme werden installiert:

- *openVPN*
- Webserver *Apache 2*
- Datenbankserver *mysql*
- Scriptsprache *perl*
- Scriptsprache *php*

*openVPN* in Version 2.2.1 wird konfiguriert (siehe Kap. 4.3). Das perl-Script (siehe Kap. B) wird in Betrieb genommen. Dieses empfängt später die Messdaten und speichert sie in der *mysql*-Datenbank (DB).

[1] Spannungspegelgesteuerte serielle Schnittstelle zur Punkt-zu-Punkt-Verbindung
[2] Differenzielle serielle Schnittstelle zur Mehrsender-Mehrempfänger-Verbindung [Manny Soltero, 2010]
[3] Differenzielle serielle Schnittstelle zur Einzelsender-Mehrempfänger-Verbindung [Manny Soltero, 2010]

*Apache 2* wird mit den Standardeinstellungen in Betrieb genommen. Das php-Script (siehe Anhang C) wird in den öffentlichen Ordner `/var/www` kopiert. Später liest dies die Messdaten aus der Datenbank und gibt sie als Grafik aus.

Auf dem *NB 1600* wird der openVPN-Client konfiguriert (siehe Kap. 4.3). Die serielle Schnittstelle des *NB 1600* wird in Betrieb genommen und über eine TCP-Verbindung durch den VPN-Tunnel weitergeleitet (Kap. 5.2). An die serielle Schnittstelle wird das *ADAM-4520* und an dieses das *Sensor-Board* angeschlossen. (siehe Kap. 5.1)

Auf dem Mobiltelefon wird der openVPN-Client *OpenVPN für Android 0.6.30* von Arne Schwabe installiert. Außerdem verfügt das Mobiltelefon über einen Webbrowser.

Diese Zusammenhänge sind in Abbildung 2.1 dargestellt. Drei Kommunikationswege sind mit *a*, *b* und *c* gekennzeichnet:

a) Vom Sensorboard wird die Temperatur über den *Adam-4520* an den *NB 1600* übertragen. Das Programm *ser2net* setzt die serielle Verbindung in eine TCP-Verbindung mit dem Linux-PC um. Das perl-Script empfängt die Daten und schreibt sie in die mysql-Datenbank.

b) Das php-Script auf dem Linux-PC liest die Messwerte aus der mysql-Datenbank und stellt diese als Webseite dar. Vom Webserver wird diese Seite an den Webbrowser des Notebooks übertragen und dargestellt.

c) Das Mobiltelefon konfiguriert den *NB 1600*, indem der Browser des Mobiltelefons mit dem Webserver des *NB 1600* kommuniziert.

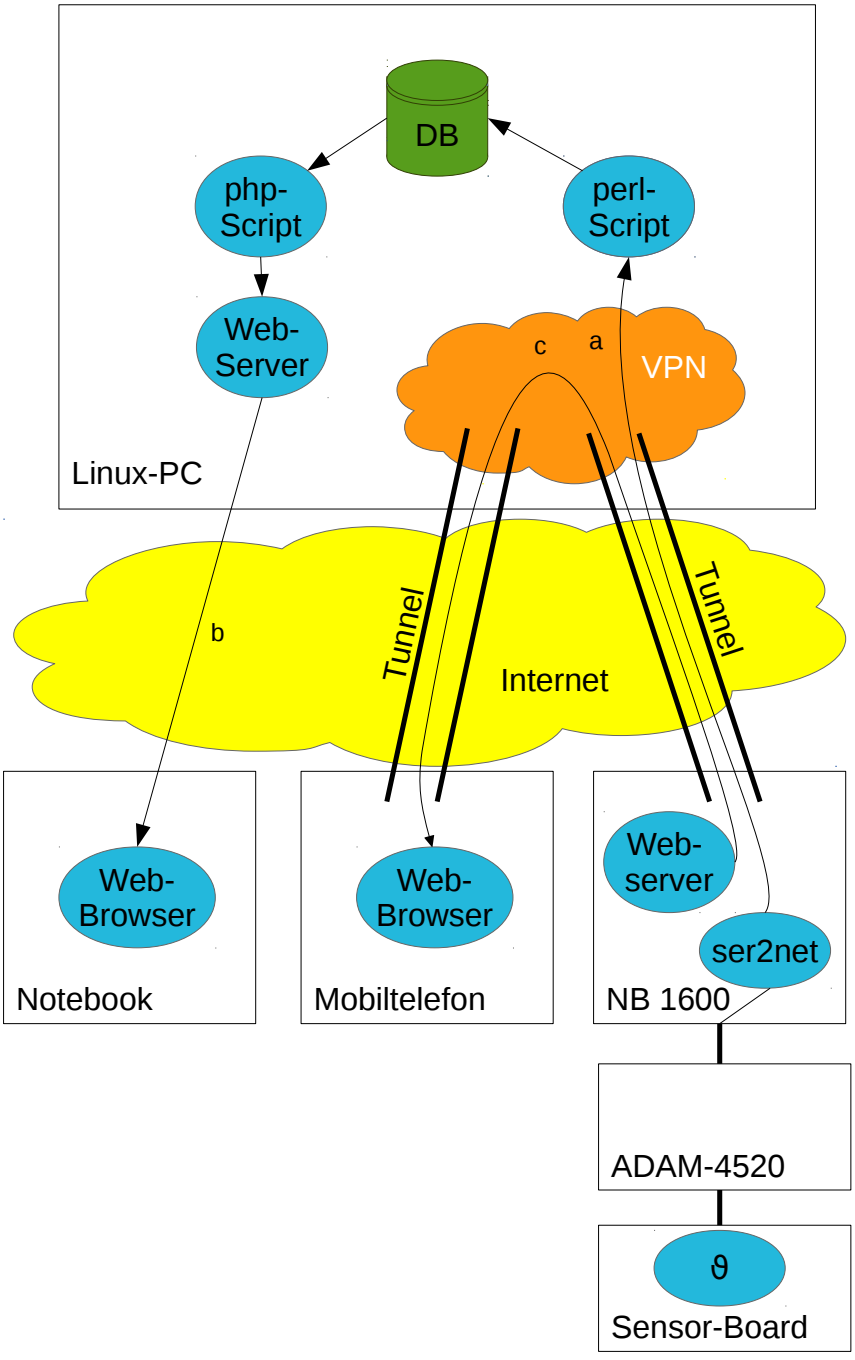


**Abbildung 2.1.:** Software-Schema mit drei Kommunikationswegen

# Kapitel 3

# Erstinbetriebnahme des *NB 1600*

Folgendes ist im Lieferumfang des *NB 1600* enthalten: (Siehe Abbildung 3.1)

- eine UMTS-Antenne
- ein 12V-Netzteil mit montiertem 2-poligem Platinensteckverbinder mit Schraubanschluss
- ein grüner 15-poliger Platinensteckverbinder mit Schraubanschluss für die Steckvorrichtung auf der Oberseite des *NB 1600*

Das 12V-Netzteil wird von dem 2-poligen Steckverbinder gelöst und gemäß den Angaben auf dem *NB 1600* mit dem 15-poligen Steckverbinder verbunden und in die Steckvorrichtung gesteckt. Auf der Seite befindet sich ein Deckel. Dieser wird geöffnet und eine SIM-Karte mit aktivem Datentarif eingelegt.

Zunächst wird der *NB 1600* konfiguriert. Dazu wird das Notebook über ein Patchkabel an die erste Ethernet-Schnittstelle angeschlossen. Diese Verbindung ist in Abbildung 2.1 nicht dargestellt. Auf dem Notebook ist ein DHCP-Client aktiv. Nun wird das *NB 1600* mit Spannung versorgt. Der DHCP-Client des Notebooks erkennt den DHCP-Server auf dem *NB 1600* und bekommt eine IP-Adresse zugewiesen. Mit einem Webbrowser wird nun die zum *NB 1600* gehörige IP-Adresse `192.168.1.1` aufgerufen. Es erscheint die Weboberfläche des *NB 1600*. Die Oberfläche verlangt die Definition eines Passworts für den Benutzer *admin*. Die Bestätigung dauert ca. 30 Sekunden.

Mit einem SSH[1]-Client auf dem Notebook wird der Zugang mit dem soeben definierten Passwort ebenfalls getestet. Um mit dem Notebook per SSH auf den *NB*

[1] Protokoll und Programm zur verschlüsselten Kommandozeilen- und Dateiübertragung

*1600* zuzugreifen lautet die einzugebende Verbindung `admin@192.168.1.1`. Es erscheint ein `NB1600 Command Line Interface`. Mit dem Kommando `shell` startet eine BusyBox[2]-Eingabeaufforderung. In der Weboberfläche wird nun unter dem Menüpunkt `INTERFACES->Ethernet->IP Settings->LAN 2` in dem Feld *IP adress* eine feste verfügbare IP-Adresse aus dem Netz des Linux-PCs eingetragen. *Mode* wird auf `WAN` gesetzt. Die zweite Ethernet-Schnittstelle wird nun mit dem Netzwerk des Linux-PCs verbunden. Die weitere Konfiguration des *NB 1600* kann nun vom Notebook, vom Linux-PC oder jedem anderem PC im Netzwerk des Linux-PCs erfolgen. Die Konfiguration der ersten Ethernet-Schnittstelle bleibt unverändert und ermöglicht ein späteres Zugreifen auf dieselbe Art, wenn kein anderer Zugriff auf den *NB 1600* mehr möglich ist.


**Abbildung 3.1.:** Lieferumfang des *NB1600*

[2]Stellt shell und Standard-Linux-Befehle zur Verfügung.

# Kapitel 4

# Einrichten des openVPN Servers mit Zerfikaterstellung

## 4.1. Zertifikate

Nun werden die Zertifikate für den openVPN-Server und die openVPN-Clients erzeugt. Eine Anleitung dazu findet sich im Anhang D. Die wichtigsten Befehle sind

- `source ./vars` zur Übernahme der Umgebungsvariablen
- `./clean-all` zum Aufräumen
- `./build-ca` zur Erzeugung des CA-Zertifikate
- `./build-key-server` zur Erzeugung des Server-Zertifikats
- `./build-key` zur Erzeugung eines Client-Zertifikats

Das Script `./build-key` fragt jeweils nach einem *Common Name*. Bei dem Zertifikat für das *NB 1600* wird hier `netmodule` und beim Zertifikat für das Mobiltelefon `mobile` eingegeben.

## 4.2. dynamischer DNS und Portforwarding

Da in diesem Fall der Linux-PC eine wechselnde IP-Adresse im Internet zugewiesen bekommt und somit ohne Kenntnis dieser nicht aufrufbar ist, wird ein dynamischer DNS-Eintrag (DDNS) erstellt. Dadurch wird einem Hostname eine IP-Adresse zugeordnet, die nicht statisch ist, sondern sich täglich ändern kann, wie es

bei Internetprovidern der Fall ist. Es existierten mehrere DDNS-Anbieter. Hier wird *freedyn.de* verwendet. Voraussetzung für den Anbieter ist, dass der Linux-PC eine Software bereitstellt, die mit dem gewählten Anbieter kommunizieren kann um die aktuelle IP-Adresse zu übermitteln. Diese wird vom Anbieter in den dynamischen DNS-Eintrag geschrieben und somit einer Domain oder Sub-Domain zugewiesen. Die Konfiguration wird hier nicht weiter erläutert. Wenn sich zwischen dem Linux-PC und dem Internet ein Router befindet, muss der Router den gewählten Port im gewählten Protokoll für die openVPN-Verbindung zum Linux-PC weiterleiten. Dieser Menüpunkt kann *Portforwarding* heißen. Auch dies wird hier nicht weiter erläutert.

## 4.3. openVPN

Auf dem Linux-PC wird openVPN installiert und im Folgenden die Konfigurationsdatei erklärt: `/etc/openvln/server.conf` Folgende Auszählung beschreibt einen Auszug aus der Konfigurationsdatei mit Änderungen und Erklärungen.

- Der Server kann mit dem Gerät *tun* oder dem Gerät *tap* betrieben werden. *tun* realisiert einen VPN-Tunnel im OSI-Layer 3, *tap* im OSI-Layer 2. Hier wird *tun* benutzt.
- Die Anweisung `client-to-client` wird gesetzt damit auch VPN-Clients untereinander eine Verbindung aufbauen können. Die Verbindung *c* aus der Abbildung 2.1 wäre sonst nicht möglich.
- Es wird das verbindungslose UDP-Protokoll gewählt.
- Das Serverzertifikat wird mit den Befehlen `ca`, `cert` und `key` in den openVPN-Server eingebunden.

Die gesamte verwendete Konfigurationsdatei findet sich im Anhang A. Der openVPN-Server wird gestartet. Auf der verwendeten Debian-Distribution wird dazu der Befehl `service openvpn start` vom Benutzer *root* ausgeführt.

Auf der Weboberfläche des *NB 1600* unter `VPN->OpenVPN->Administration` wird das VPN mit *enabled* aktiviert. Unter *Tunnel Configuration* werden Parameter Eingestellt. Siehe Abbildung 4.1. Die Werte unter *Interface type* und *Protocol* gleichen denen des openVPN-Servers. Der *Operation mode* steht auf *client*. Unter *Server* wird der gewählte Name des dynamischen DNS-Eintrags eingetragen, bei *Port* der

in der Konfiguration des Servers definierte. Im Menüpunkt `SYSTEM->Keys&Certificates ->OpenVPN1` werden die zuvor erzeugten Dateien für das Client-Zertifikat des *NB 1600* hochgeladen.

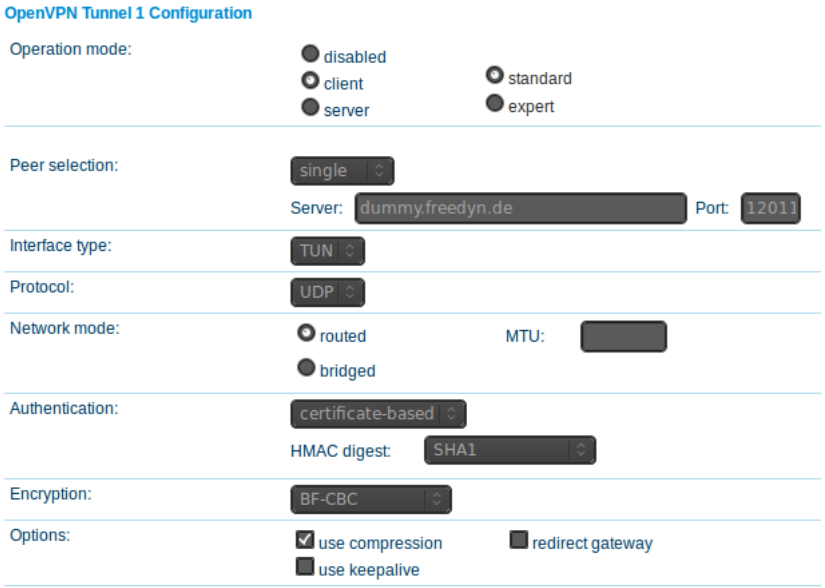


**Abbildung 4.1.:** openVPN Konfiguration auf der Weboberfläche des *NB 1600*

Die Zertifikate für das Mobiltelefon werden auf dem Linux-PC in das Format *pkcs 12* gewandelt. Dies hat den Vorteil, dass nur eine Datei auf das Mobiltelefon übertragen werden muss. Dazu wird der Befehl `openssl pkcs12 -export -inkey mobile.key -in mobile.crt -certfile ca.crt -out mobile.p12` im Ordner `/etc/openvpn/easy-rsa2/keys/` ausgeführt. Anschließend wird die Datei `mobile.p12` auf das Mobiltelefon übertragen und der openVPN-Client konfiguriert. Siehe Abbildung 4.2.

Die VPN-Verbindung über UMTS wird getestet. Dazu wird die Verbindung des zweiten Netzwerkanschlusses zum Netzwerk des Linux-PCs gelöst. Mit dem Befehl `cat /etc/openvpn/openvpn-status.log` auf dem Linux-PC wird die log-Datei des openVPN-Servers aufgerufen. Sie gibt Aufschluss über die aktuelle IP-Adresse des *NB 1600* im VPN. Es erscheint als `netmodule`. Anschließend wird mit dem Linux-PC die Weboberfläche des *NB 1600* mit dieser IP-Adresse im VPN aufgerufen. In diesem Fall lautet diese `10.8.0.6`. Damit funktioniert die VPN-Verbindung über UMTS.

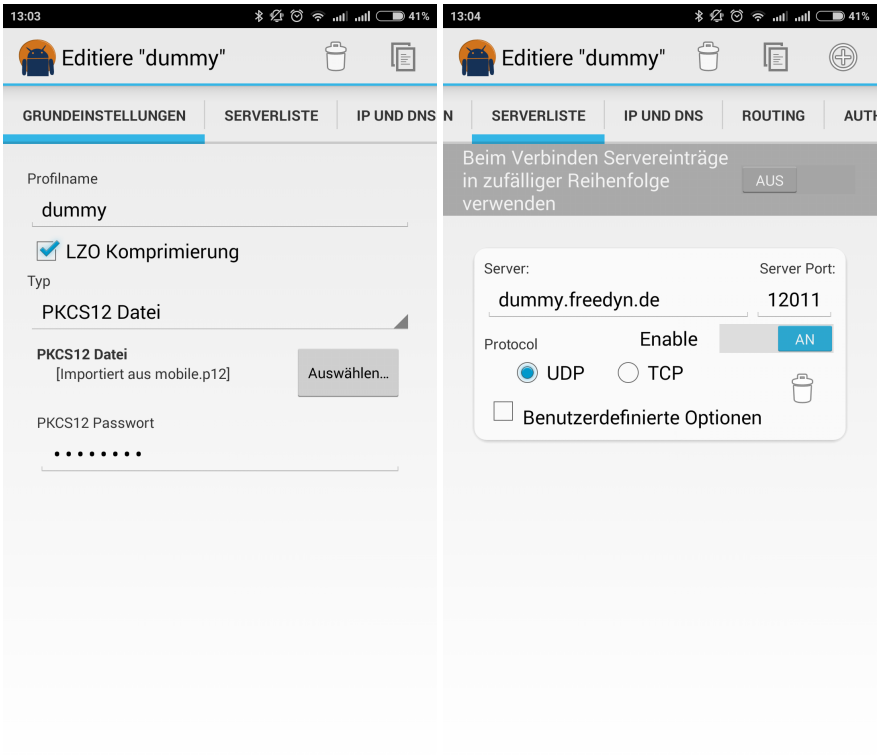


**Abbildung 4.2.:** openVPN-Konfiguration auf dem Android-Mobiltelefon

Mit dem Webbrowser wird ebenfalls die Weboberfläche des *NB 1600* mit der IP-Adresse `10.8.0.6` aufgerufen. Damit funktioniert auch die openVPN-Verbindung des Mobiltelefons.

# Kapitel 5

# Messdaten sammeln

## 5.1. Serielle Verbindung

Das *ADAM-4520* besitzt an der RS232-Schnittstelle einen weiblichen zwei-Reihigen 9-poligen D-Sub-Anschluss. Das *NB 1600* sieht den Anschluss der seriellen Verbindung an dem 15-poligen Platinensteckverbinder mit Schraubanschluss vor. Dafür wird ein Adapter gelötet mit einem männlichen 9-poligen D-Sub-Anschluss und auf der einen und offenen Leitungsenden auf der anderen Seite. Abbildung 5.1 zeigt das Pinout. Pin 2,3 und 5 werden nach folgendem Schema verbunden verbunden:

**Tabelle 5.1.:** Belegung des RS232-Adapters

| D-Sub-Pin | NB 1600 |
|---|---|
| 2 | RS232 RxD |
| 3 | RS232 TxD |
| 5 | RS232 GND |

## 5.2. TCP-Verbindung

Wie bereits auf Seite 2 beschrieben, ist das *Sensor-Board* mit dem *ADAM-4520* verbunden und dieses auf 9600 Baud eingestellt. Nun wird die serielle Schnittstelle des *NB 1600* konfiguriert. Die Konfigurationsmaske befindet sich unter `INTERFACES ->Serial->Edit`. Im Reiter `Administration` wird unter *SERIAL1 is used by: device server* ausgewählt. Im Reiter *Port Settings* werden weitere Einstellungen vorgenommen. Die Baudrate wird wie beim *ADAM-4520* auf 9600 eingestellt. Außerdem werden 8 Databits, keine Parität und ein Stopbit verwendet. Dies ist abhängig

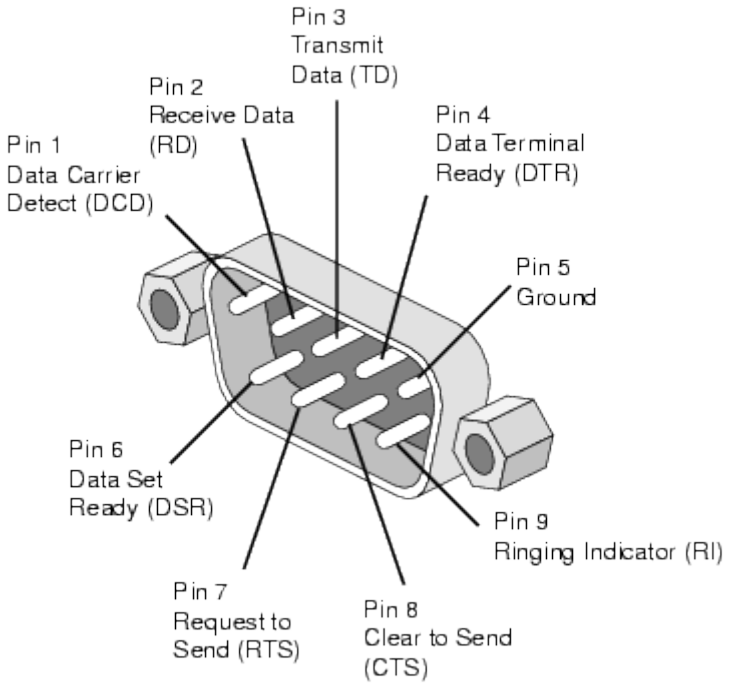


**Abbildung 5.1.:** Pinout des weiblichen D-Sub-9-Steckers bei der Verwendung von RS232 aus [ELSIST S.r.l., 2012]

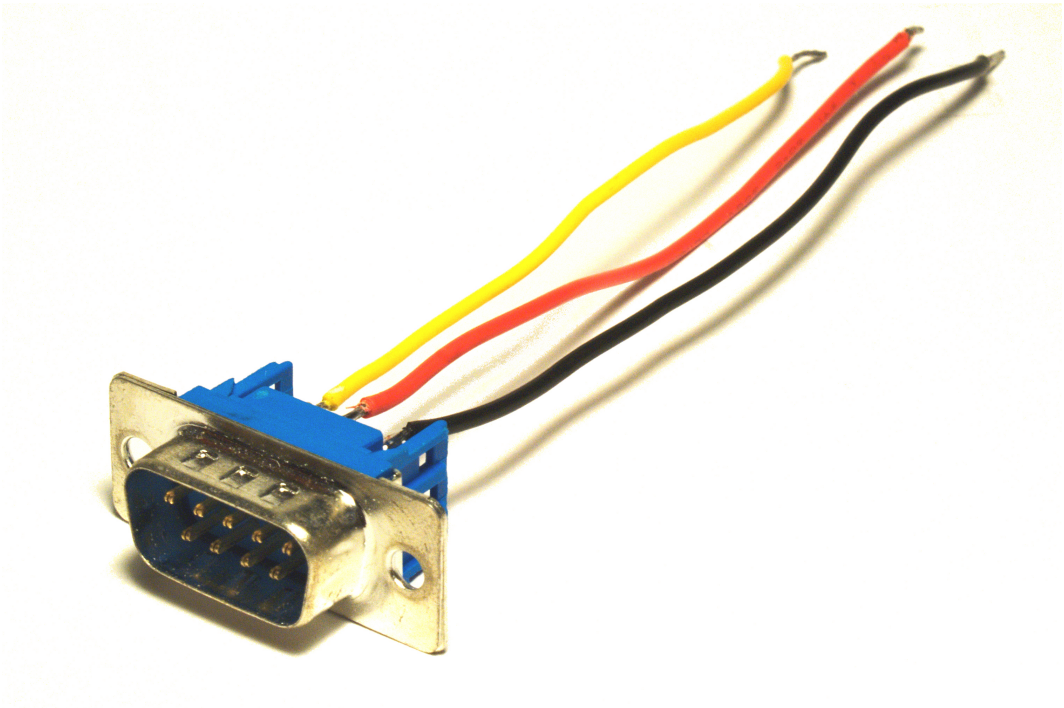

**Abbildung 5.2.:** Fertiger RS232-Adapter für den *NB 1600*

von den Einstellungen im *Sensor-Board*, die im Rahmen dieses Programms nicht verändert wurden.

Das Protokoll wird auf *Telnet* gesetzt und die Option *show banner* gesetzt. Mit dem Linux-PC wird eine Telnet-Verbindung zu der IP-Adresse des *NB 1600* im VPN und dem auf der Weboberfläche eingestellten Port aufgebaut. Der *NB 1600* schickt als erstes folgende Zeile: `ser2net port 12019 device /dev/serial0 [9600 N81] (NB1600 2.6.36)`. Daran lässt sich erkennen, dass es sich um das Programm *ser2net* handelt, welches auf dem *NB 1600* die serielle Schnittstelle über eine TCP-Verbindung zur Verfügung stellt. Die Verbindung wird mit Ctrl + C beendet.

Das Protokoll wird nun auf *TCP raw* gestellt. Dies ist nötig, da das perl-Script nur Kommazahlen und keinen anderen Steuerzeichen verarbeiten kann. Siehe Abbildung 5.3.

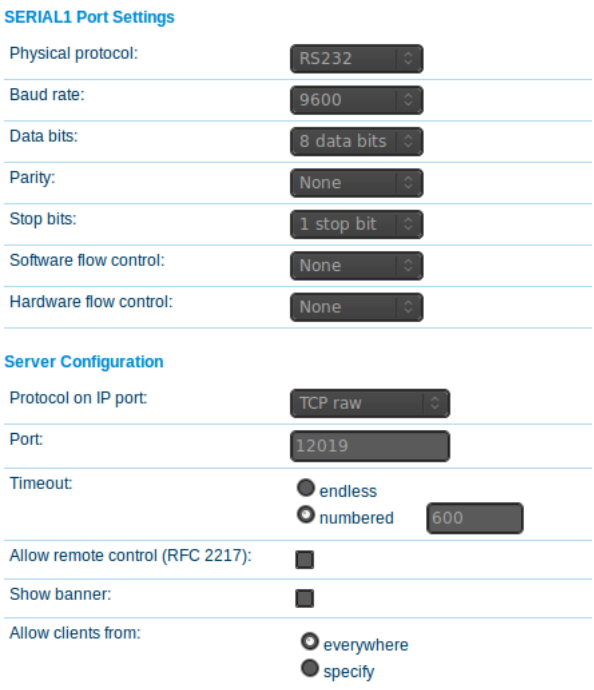


**Abbildung 5.3.:** Konfiguration der seriellen Schnittstelle auf der Weboberfläche des *NB 1600*

Die Telnet-Verbindung wird erneut aufgebaut. Der Linux-PC empfängt nur durch einen Zeilenvorschub voneinander getrennte Messdaten. Die Ausgabe ist in Abbildung 5.4 zu sehen. Die Verbindung wird mit Ctrl + C beendet.

```
gerrit@Yttrium:~$ telnet 10.8.0.6 12019
Trying 10.8.0.6...
Connected to 10.8.0.6.
Escape character is '^]'.
25.600000379
25.592967985
25.596094129
25.573436735
```

**Abbildung 5.4.:** Empfang der Temperatur-Messdaten über Telnet

## 5.3. Messdatenspeicherung

Im Rahmen dieser Studienarbeit wurde ein perl-Script entwickelt, um die Messwerte in der Datenbank zu speichern. Um das Script zu starten wird vorausgesetzt, dass *mysql* und *perl* installiert sowie eine mysql-Datenbank und eine Tabelle vorhanden sind. Die Spalten der Tabelle müssen lauten: id, value, time und zur Aufnahme einer ganzen Zahl, einer Kommazahl mit 9 Nachkommastellen und einer ganzen Zahl bestimmt sein. Abbildung 5.5 zeigt die Struktur der Tabelle angezeigt von dem Verwaltungstool *phpmyadmin*

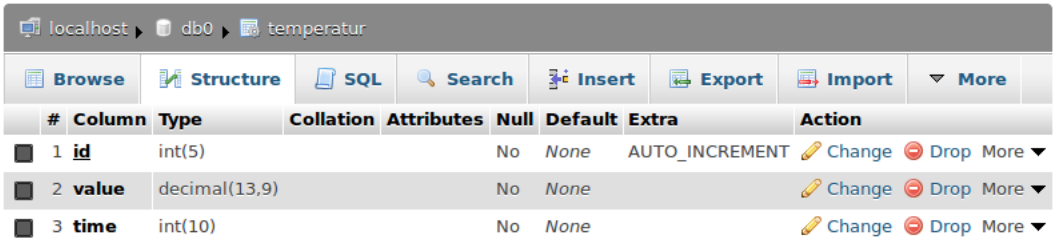


**Abbildung 5.5.:** Struktur der mysql-Tabelle

Im Folgenden wird auf die Funktionsweise des perl-Scripte eingegangen. Im Anhang B findet sich das gesamte Script mit Kommentaren. Das *DBI*-Modul baut eine Verbindung zum Datenbankserver auf. Mit dem Befehl `new IO::Socket::INET` wird eine TCP-Verbindung aufgebaut. Die Parameter sind `PeerHost`, `PeerPort` und `Proto`. Dies geschieht in einer Endlosschleife, damit die Verbindung nach einem Abbruch erneut aufgebaut wird. Diese *timeout*-Zeit ist auf 10 Sekunden eingestellt. Der Aufruf `$sel->can_read($timeout)` prüft, ob innerhalb von 10 Sekunden Daten empfangen werden könnten. Ist dies nicht der Fall, wird versucht eine neue Verbindung aufzubauen, sonst wird eine Zeile von der TCP-Verbindung gelesen und die aktuelle Uhrzeit bestimmt. Diese beiden Informationen werden mit dem SQL-Befehl `INSERT INTO` in die mysql-Tabelle geschrieben. Das Script gibt Informationen während der Laufzeit aus.

`perl tcp2sql.pl` startet das Script. Abbildung 5.6 zeigt den Aufruf. Die Endlosschleife wird mit Ctrl + C beendet.

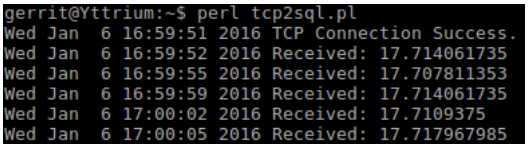


**Abbildung 5.6.:** Ausgabe des perl-Scripts

# Kapitel 6

# Ausgabe der Messdaten

Im Rahmen dieser Studienarbeit wurde ein php-Script entwickelt, um die Messwerte aus der Datenbank auszulesen. Um das Script zu starten muss *mysql*, *php* und ein Webserver installiert sein. In diesem Fall ist dies *Apache 2*.

Das php-Script liest Messwerte aus der Datenbank und gibt diese als X-Y-Diagramm aus. Die Funktion `unix2datum` wandelt die Unix-Sekunden in eine für den Menschen lesbare Uhrzeit um. Um das Diagramm auszugeben, wird die Bibliothek *jpgraph* verwendet. Mit dem Befehl `mysql_connection` wird eine Verbindung vom Datenbankserver aufgebaut. Die aktuelle Uhrzeit wird mit `time()` bestimmt. Das Script erlaubt die Auswahl verschiedener Zeiträume. Die Auswahl wird per `GET`-Parameter getroffen. Je nach Zeitraum wird mit dem SQL-Befehl `WHERE time >` und zum Beispiel dem PHP-Teil `$time-60` festgelegt, dass die Uhrzeit der gewählten Messwerte größer sein muss als die aktuelle Zeit abzüglich 60, was bedeutet, dass die Messwerte nicht älter als 60 Sekunden sein dürfen, um auf diese Abfrage zu passen.

Mit dem SQL-Befehl `SELECT value, time FROM` wird die SQL-Abfrage definiert. `mysql_fetch_object($sql)` liest jeden einzelnen Messwert aus der Datenbank. Mit den folgenden Befehlen an das Objekt `$graph` wird das Aussehen des Diagramms definiert. Mit `Stroke()` wird der Graph schlussendlich gezeichnet. Im Anhang C findet sich das gesamte Script. In Abbildung 6.1 ist die Ausgabe mit dem `GET`-Parameter *m* zu sehen, wodurch die Messwerte der vergangenen Minute ausgegeben werden.

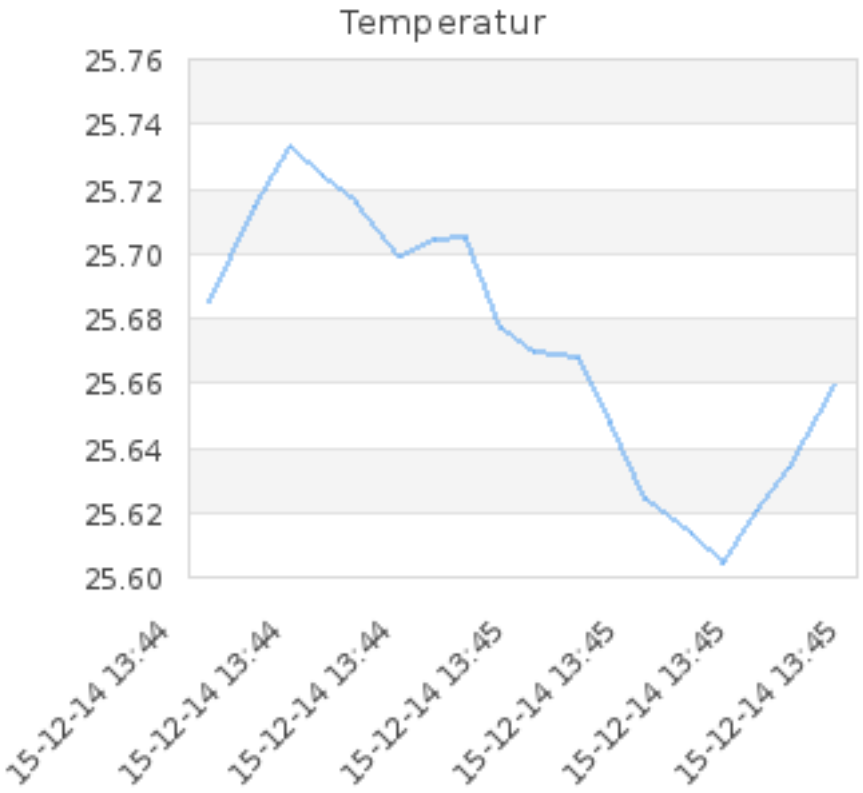


**Abbildung 6.1.:** Ausgabe des PHP-Scripts

# Kapitel 7

# Langzeitbetrieb

## 7.1. Datenvolumen

Während der Konfiguration und einem Zeitraum in dem nur Daten gesammelt werden und kein Zugriff auf die Weboberfläche des *NB 1600* statt findet, wird das Volumen des Datenverkehrs über das VPN am Gerät *tun0* des Linux-PCs gemessen. So wird abgeschätzt, welches monatliche Datenvolumen nötig ist, um das Gerät zu betreiben. Die Werte in Tabelle 7.1 werden mit dem Befehl `ifconfig tun0` aufgenommen. Die relevante Zeile von ifconfig lautet etwa: `RX bytes:69775072 (66.5 MiB) TX bytes:175618572 (167.4 MiB)`

**Tabelle 7.1.:** Datenvolumen im VPN

| Zeit | RX / MiB | TX / MiB |
|---|---|---|
| 14.12.2015 17:27 | 7,4 | 65,8 |
| 15.12.2015 17:33 | 11,9 | 99,3 |
| 17.12.2015 08:41 | 16,8 | 103,6 |
| 21.12.2015 11:54 | 29,2 | 114,3 |
| 21.12.2015 14:37 | 29,5 | 114,6 |
| 21.12.2015 23:40 | 30,7 | 115,5 |

Vom 17.12.2015 um 08:41 bis zum 21.12.2015 um 23:40 sind ca. 4,62 Tage vergangen. Es wurden 13,9 MiB Empfangen und 11,9 MiB Gesendet. In Summe sind das 25,8 MiB. Das sind pro Tag $\frac{25{,}8MiB}{4{,}62d} \approx 5,58\frac{MiB}{d}$.
Im Monat sind das $5,58\frac{MiB}{d} * 30d = 167MiB$
$167MiB = 167 * 1024 * 1024Byte = 175MB$
Ein Datentarif mit einem Volumen von 200 MB im Monat wird somit bei der gleichen Menge an Messwerten ohne Benutzung der Weboberfläche ausreichen.

## 7.2. Stabilität

Das perl-Script berücksichtigt den Fall, dass die Verbindung zum *NB 1600* unterbrochen wird und versucht danach eine neue aufzubauen. Die Verbindungsdaten werden mit Datum und Uhrzeit angezeigt. So ist erkennbar, wann die Verbindung unterbrochen wurde. Abbildung 7.1 zeigt das Szenario, in dem

1. um 16:56:10 das das Perl-Script gestartet wird,
2. um 16:56 das *NB 1600* mit Spannung versorgt wird,
3. um 16:57:50 die Spannungsversorgung des *NB 1600* getrennt wird,
4. um 16:58:00 die Spannungsversorgung des *NB 1600* wieder hergestellt wird.

```
gerrit@Yttrium:~$ perl tcp2sql.pl
Wed Jan  6 16:56:23 2016 ERROR in Socket Creation
Wed Jan  6 16:56:33 2016 ERROR in Socket Creation
Wed Jan  6 16:56:43 2016 ERROR in Socket Creation
Wed Jan  6 16:56:53 2016 ERROR in Socket Creation
Wed Jan  6 16:57:00 2016 TCP Connection Success.
Wed Jan  6 16:57:02 2016 Received: 17.646873472
Wed Jan  6 16:57:05 2016 Received: 17.650779722
Wed Jan  6 16:57:13 2016 Received: 17.653905866
Wed Jan  6 16:57:15 2016 Received: 17.646093366
Wed Jan  6 16:57:15 2016 Received: 17.657812116
Wed Jan  6 16:57:18 2016 Received: 17.657812116
Wed Jan  6 16:57:21 2016 Received: 17.667186735
Wed Jan  6 16:57:24 2016 Received: 17.664842603
Wed Jan  6 16:57:27 2016 Received: 17.663280485
Wed Jan  6 16:57:31 2016 Received: 17.657030103
Wed Jan  6 16:57:34 2016 Received: 17.657030103
Wed Jan  6 16:57:37 2016 Received: 17.663280485
Wed Jan  6 16:57:41 2016 Received: 17.653123853
Wed Jan  6 16:57:45 2016 Received: 17.660936353
Wed Jan  6 16:57:47 2016 Received: 17.661718366
 Socket read timed out
Wed Jan  6 16:57:57 2016 TCP Connection closed
Wed Jan  6 16:58:12 2016 ERROR in Socket Creation
Wed Jan  6 16:58:22 2016 ERROR in Socket Creation
Wed Jan  6 16:58:32 2016 ERROR in Socket Creation
Wed Jan  6 16:58:42 2016 ERROR in Socket Creation
Wed Jan  6 16:58:52 2016 ERROR in Socket Creation
Wed Jan  6 16:59:02 2016 ERROR in Socket Creation
Wed Jan  6 16:59:05 2016 TCP Connection Success.
Wed Jan  6 16:59:07 2016 Received: 17.688280103
Wed Jan  6 16:59:10 2016 Received: 17.69140625
Wed Jan  6 16:59:13 2016 Received: 17.701562879
Wed Jan  6 16:59:16 2016 Received: 17.698436735
Wed Jan  6 16:59:20 2016 Received: 17.70703125
Wed Jan  6 16:59:23 2016 Received: 17.696874616
Wed Jan  6 16:59:26 2016 Received: 17.707811353
Wed Jan  6 16:59:30 2016 Received: 17.716405866
Wed Jan  6 16:59:32 2016 Received: 17.697656629
```

**Abbildung 7.1.:** Stabilitätstest der Perl-Scripts

In diesem Projekt sind an den Linux-PC nicht direkt Tastatur und Maus angeschlossen, sondern es wird über SSH zugegriffen. Wenn das perl-Script mit dem Befehl `perl tcp2sql.pl` gestartet wird und danach die SSH-Verbindung beendet wird, läuft das perl-Script nicht weiter. Deshalb wird das perl-Script folgendermaßen mit dem textorientieren Programm *screen* gestartet:

- Die SSH-Verbindung wird aufgebaut.

- Das Programm wird mit `screen` gestartet.
- Das perl-Script wird mit `perl tcp2sql.pl` gestartet.
- Die *screen*-Sitzung wird mit Ctrl + A , D in den Hintergrund versetzt.
- Es erscheint eine Zeile beginnend mit `[detached from`.
- Die SSH-Verbindung wird nun mit dem Befehl `exit` beendet.

Das perl-Script läuft dennoch weiter und empfängt Messwerte. Um das Script nun zu beenden wird folgendermaßen vorgegangen:

- Die SSH-Verbindung wird aufgebaut.
- `screen -r` wird ausgeführt.
- Es erscheinen die Ausgaben des laufenden perl-Scriptes.
- Das Script wird mit Ctrl + C beendet.
- *screen* wird mit `exit` beendet.
- Die SSH-Verbindung wird mit dem Befehl `exit` beendet.

## 7.3. Leistungsaufnahme

In der technischen Spezifikation des *NB 1600* ist eine maximale Leistungsaufnahme von $5\ \mathrm{W}$ angegeben. [NetModule AG, 2015a]

Am Gleichspannungseingang des *NB 1600* wird ein Strom von $\approx 300\ \mathrm{mA}$ gemessen. Bei der Versorgungsspannung von $5\ \mathrm{V}$ entspricht dies $0,3\ \mathrm{A} \cdot 5\ \mathrm{V} = 1,5\ \mathrm{W}$

Am Gleichspannungseingang des *ADAM-4520* wird ein Strom von $\approx 150\ \mathrm{mA}$ gemessen. Bei der Versorgungsspannung von $5\ \mathrm{V}$ entspricht dies $0,3\ \mathrm{A} \cdot 5\ \mathrm{V} \approx 0,53\ \mathrm{W}$

In Summe werden also $1,5\ \mathrm{W} + 0,53\ \mathrm{W} \approx 2\ \mathrm{W}$ aufgenommen.

# Kapitel 8

# Schlussfolgerungen

## 8.1. Zusammenfassung der Ergebnisse

In dieser Studienarbeit wurde die VPN-Verbindung eingerichtet und das *NB 1600* in Betrieb genommen. Die Messdaten werden nun übertragen, gespeichert und angezeigt.

## 8.2. Bewertung der Ergebnisse

Das *NB 1600* war während des gesamtes Projektes nie in einem undefinierten Zustand und ließ sich immer über das Netzwerk ansprechen. In Kombination mit dem Test des perl-Scriptes im Kapitel 7.2 ist der zuverlässige Betrieb sichergestellt.

Die Funktionsweise des Linux-PC ist hier mit einer *Debian*-Distribution erklärt. Die Verwendung einer anderen Distribution ist generell möglich und erfordert unter Umständen die Anpassung einiger Befehle.

## 8.3. Ausblick

Um die Anzahl der gespeicherten Messwerte zu reduzieren muss ein *swinging-door*-Algorithmus implementiert werden, der einen Messwert nur dann in die Datenbank schreibt, wenn dieser eine markante Änderung im Vergleich zu vorherigen Messwerten darstellt.

Für den Aufruf des Bildes, welches das php-script ausgibt, muss eine Weboberfläche geschrieben werden. Um die Messwerte in anderen als die bisher implementieren Zeitspannen auszugeben, muss das php-Script angepasst werden.

Um das perl-Script nicht mit *screen* zu starten, muss für die jeweilig verwendete Linux-Distribution das Script als Dienst eingerichtet werden.

# Literaturverzeichnis

[Advantech 2015] ADVANTECH: *ADAM-4510/S ADAM-4520 ADAM4521 Specifications*. Advantech, 2015. – URL http://downloadt.advantech.com/ProductFile/PIS/ADAM-4520/Product%20-%20Datasheet/ADAM-452020150714145057.pdf

[ELSIST S.r.l. 2012] ELSIST S.R.L.: *I/O Controller 2 Full COM port and I/O to Ethernet*. 2012. – URL http://www.elsist.it/WebSite/Html/English/Products/Hardware/Converface/EthSerial/EnIOController2.php

[Manny Soltero 2010] MANNY SOLTERO, Chris Cockril Kevin Zhang Clark Kinnaird Thomas K.: *RS-422 and RS-485 Standards Overview and System Configurations*. Texas Instruments, 2010. – URL http://www.ti.com/lit/an/slla070d/slla070d.pdf

[NetModule AG 2015a] NETMODULE AG, Switzerland: *NB1600 UMTS/3G Router Specification*. netmodule, 2015. – URL http://netmodule.com/products/industrial-routers/UMTS-router.html

[NetModule AG 2015b] NETMODULE AG, Switzerland: *NetModule Router NB1600 User Manual for Software Version 3.8*. 1.5. netmodule, 2015. – URL http://wiki.netmodule.com/_media/nrsw/latest/nb1600_manual.pdf

[OpenVPN Technologies, Inc. 2013] OPENVPN TECHNOLOGIES, INC.: *HOWTO*. 2013. – URL https://openvpn.net/howto.html

# Anhang A

# openVPN-Server einrichten

```
# Which TCP/UDP port should OpenVPN listen on?
port 12011

# TCP or UDP server?
;proto tcp
proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
# Use "dev tap0" if you are ethernet bridging
# and have precreated a tap0 virtual interface
# and bridged it with your ethernet interface.
# If you want to control access policies
# over the VPN, you must create firewall
# rules for the the TUN/TAP interface.
;dev tap
dev tun

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key). Each client
# and the server must have their own cert and
# key file. The server and all clients will
# use the same ca file.
ca /etc/openvpn/easy-rsa2/keys/ca.crt
cert /etc/openvpn/easy-rsa2/keys/dummy.freedyn.de.crt
key /etc/openvpn/easy-rsa2/keys/dummy.freedyn.de.key
dh /etc/openvpn/easy-rsa2/keys/dh1024.pem

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1.
server 10.8.0.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file. If OpenVPN goes down or
```

```
39 # is restarted, reconnecting clients can be assigned
40 # the same virtual IP address from the pool that was
41 # previously assigned.
42 ifconfig-pool-persist ipp.txt
43
44 # If enabled, this directive will configure
45 # all clients to redirect their default
46 # network gateway through the VPN, causing
47 # all IP traffic such as web browsing and
48 # and DNS lookups to go through the VPN
49 # (The OpenVPN server machine may need to NAT
50 # or bridge the TUN/TAP interface to the internet
51 # in order for this to work properly).
52 push "route-gateway"
53 push "route 192.168.178.0 255.255.255.0"
54
55 # Allow different clients to be able to "see"
56 # each other.
57 client-to-client
58
59 # The keepalive directive causes ping-like
60 # messages to be sent back and forth over
61 # the link so that each side knows when
62 # the other side has gone down.
63 # Ping every 10 seconds, assume that remote
64 # peer is down if no ping received during
65 # a 120 second time period.
66 keepalive 10 120
67
68 # Select a cryptographic cipher.
69 # This config item must be copied to
70 # the client config file as well.
71 cipher BF-CBC      # Blowfish (default)
72 ;cipher AES-128-CBC # AES
73 ;cipher DES-EDE3-CBC # Triple-DES
74
75 # Enable compression on the VPN link.
76 # If you enable it here, you must also
77 # enable it in the client config file.
78 comp-lzo
79
80 # The persist options will try to avoid
81 # accessing certain resources on restart
82 # that may no longer be accessible because
83 # of the privilege downgrade.
84 persist-key
85 persist-tun
86
87 # Output a short status file showing
88 # current connections, truncated
89 # and rewritten every minute.
90 status openvpn-status.log
91
```

```
92 # Set the appropriate level of log
93 # file verbosity.
94 #
95 # 0 is silent, except for fatal errors
96 # 4 is reasonable for general usage
97 # 5 and 6 can help to debug connection problems
98 # 9 is extremely verbose
99 verb 3
```

# Anhang B

# Messdaten sammeln

```
1  #!/usr/bin/perl
2  #tcp2sql.pl
3  #Gerrit Schröder 2016
4
5  use IO::Socket::INET;
6  use warnings;
7  use DBI;
8
9  # flush after every write
10 $| = 1;
11
12 my ($socket,$client_socket);
13 my ($db_user, $db_name, $db_pass) = ('perl', 'db0', 'perl');
14 my $tablename = 'temperatur';
15 my $timeout = 10;
16
17 #Database connection
18 my $dbh = DBI->connect("DBI:mysql:database=$db_name", $db_user, $db_pass);
19
20 while(1) {
21    # creating object interface of IO::Socket::INET modules which internally creates
22    # socket, binds and connects to the TCP server running on the specific port.
23    $socket = new IO::Socket::INET (
24    PeerHost => '10.8.0.6',
25    PeerPort => '12019',
26    Proto => 'tcp',
27    Timeout => $timeout
28    );
29
30    if( !$socket )
31    {
32       print localtime() . " ERROR in Socket Creation\n";
33       redo;
34    }
35
36    my $sel = IO::Select->new($socket);
37    print localtime() . " TCP Connection Success. \n";
38
```

```
39     while( $socket->connected() ){
40        unless ($sel->can_read($timeout)) {
41           print " Socket read timed out\n";
42           last;
43        } else {
44        $data = <$socket>; # read the socket data.
45        print localtime() . " Received: $data";
46        my $time = time();
47        #Define Query
48        $myquery = "INSERT INTO $tablename (value, time) VALUES ($data,$time)";
49
50        #Do mySQL
51        $dbh->do($myquery);
52        }
53     }
54
55     #Close Socket
56     $socket->close();
57     print localtime() . " TCP Connection closed\n";
58     sleep(5); #Sleep for 5 seconds
59  }
60
61  #Close Database
62  $dbh->disconnect();
```

# Anhang C

# Messdaten auswerten

```php
<?php
/*
 *  pic.php
 *
 *  Reads measuring data of the the
 *  last minute,
 *  the last hour,
 *  the last day or
 *  the whole table from mysql-Database.
 *  Creats a X-Y graph and displays as picture.
 *  Gerrit Schröder 2016
 */

//Converts unix timestamp in a human readable format
function unix2datum ( $unix )
{
   return date("y-m-d H:i",$unix);
}

//jpgraph functions
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_line.php');

//Data for SQL connection
define ('SQL__HOST', 'localhost');
define ('SQL__DB', 'db0');
define ('SQL__USER', 'perl');
define ('SQL__PWD', 'perl');
define ('SQL_TEMP', 'temperatur');

//SQL connection
$sql_connection = mysql_connect(SQL__HOST, SQL__USER, SQL__PWD)
or die("Verbindung zum Datenbankserver fehlgeschlagen");
mysql_select_db(SQL__DB, $sql_connection) or die("Konnte die Datenbank nicht
    waehlen.");

//Current time in seconds
$time = time();
```

```
38
39 //Choosing the area bei GET-Parameter
40 if(isset($_GET['m']))
41  $extra = "WHERE time > " . ($time-60);
42 else if(isset($_GET['d']))
43  $extra = "WHERE time > " . ($time-24*60*60);
44 else if(isset($_GET['h']))
45  $extra = "WHERE time > " . ($time-60*60);
46 else
47  $extra = '';
48
49 //SQL-Request
50 $sql = mysql_query( 'SELECT value, time FROM '.SQL_TEMP. ' ' . $extra ) or die(
       mysql_error());
51
52 //Fill the Variables
53 $xdata = array();
54 $ydata = array();
55
56 while ( $messwert = mysql_fetch_object($sql) )
57 {
58   $xdata[] = $messwert->time;
59   $ydata[] = (float) $messwert->value;
60 }
61
62 //Size of Graph
63 $width=350;
64 $height=300;
65
66 // Create the Graph
67 $graph = new Graph($width,$height);
68 $graph->SetScale('intlin');
69
70 // Setup margin and titles
71 $graph->SetMargin(80,20,20,80);
72 $graph->title->Set('Temperatur');
73 $graph->xaxis->SetLabelAngle(45); // 45 degrees angle
74 $graph->xaxis->SetLabelFormatCallback('unix2datum');
75
76 $lineplot=new LinePlot($ydata, $xdata); // Create the linear plot
77 $graph->Add($lineplot); // Add the plot to the graph
78 $graph->Stroke(); // Display the graph
79 ?>
```

## Anhang D

# Zertifikaterstellung für openVPN

[OpenVPN Technologies, Inc., 2013]

## Setting up your own Certificate Authority (CA) and generating certificates and keys for an OpenVPN server and multiple clients

### Overview

The first step in building an OpenVPN 2.x configuration is to establish a PKI (public key infrastructure). The PKI consists of:

- a separate certificate (also known as a public key) and private key for the server and each client, and
- a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates.

OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established.

Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server).

This security model has a number of desirable features from the VPN perspective:

- The server only needs its own certificate/key -- it doesn't need to know the individual certificates of every client which might possibly connect to it.
- The server will only accept clients whose certificates were signed by the master CA certificate (which we will generate below). And because the server can perform this signature verification without needing access to the CA private key itself, it is possible for the CA key (the most sensitive key in the entire PKI) to reside on a completely different machine, even one without a network connection.
- If a private key is compromised, it can be disabled by adding its certificate to a CRL (certificate revocation list). The CRL allows compromised certificates to be selectively rejected without requiring that the entire PKI be rebuilt.
- The server can enforce client-specific access rights based on embedded certificate fields, such as the Common Name.

Note that the server and client clocks need to be roughly in sync or certificates might not work properly.

### Generate the master Certificate Authority (CA) certificate & key

In this section we will generate a master CA certificate/key, a server certificate/key, and certificates/keys for 3 separate clients.

For PKI management, we will use *easy-rsa*, a set of scripts which is bundled with OpenVPN 2.2.x and earlier. If you're using OpenVPN 2.3.x, you need to download easy-rsa separately from here.

If you are using Linux, BSD, or a unix-like OS, open a shell and cd to the **easy-rsa** subdirectory. If you installed OpenVPN from an RPM or DEB file, the easy-rsa directory can usually be found in **/usr/share/doc/packages/openvpn** or **/usr/share/doc/openvpn** (it's best to copy this directory to another location such as **/etc/openvpn**, before any edits, so that future OpenVPN package upgrades won't overwrite your modifications). If you installed from a .tar.gz file, the easy-rsa directory will be in the top level directory of the expanded source tree.

If you are using Windows, open up a Command Prompt window and cd to **\Program Files\OpenVPN\easy-rsa**. Run the following batch file to copy configuration files into place (this will overwrite any preexisting vars.bat and openssl.cnf files):

```
init-config
```

Now edit the **vars** file (called **vars.bat** on Windows) and set the KEY_COUNTRY, KEY_PROVINCE, KEY_CITY, KEY_ORG, and KEY_EMAIL parameters. Don't leave any of these parameters blank.

Next, initialize the PKI. On Linux/BSD/Unix:

```
. ./vars
./clean-all
./build-ca
```

On Windows:

```
vars
clean-all
build-ca
```

The final command (**build-ca**) will build the certificate authority (CA) certificate and key by invoking the interactive **openssl** command:

```
ai:easy-rsa # ./build-ca
Generating a 1024 bit RSA private key
............++++++
...........++++++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [KG]:
State or Province Name (full name) [NA]:
Locality Name (eg, city) [BISHKEK]:
Organization Name (eg, company) [OpenVPN-TEST]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:OpenVPN-CA
Email Address [me@myhost.mydomain]:
```

Note that in the above sequence, most queried parameters were defaulted to the values set in the **vars** or **vars.bat** files. The only parameter which must be explicitly entered is the **Common Name**. In the example above, I used "OpenVPN-CA".

### Generate certificate & key for server

Next, we will generate a certificate and private key for the server. On Linux/BSD/Unix:

```
./build-key-server server
```

On Windows:

```
build-key-server server
```

As in the previous step, most parameters can be defaulted. When the **Common Name** is queried, enter "server". Two other queries require positive responses, "Sign the certificate? [y/n]" and "1 out of 1 certificate requests certified, commit? [y/n]".

### Generate certificates & keys for 3 clients

Generating client certificates is very similar to the previous step. On Linux/BSD/Unix:

```
./build-key client1
./build-key client2
./build-key client3
```

On Windows:

```
build-key client1
build-key client2
build-key client3
```

If you would like to password-protect your client keys, substitute the **build-key-pass** script.

Remember that for each client, make sure to type the appropriate **Common Name** when prompted, i.e. "client1", "client2", or "client3". Always use a unique common name for each client.

### Generate Diffie Hellman parameters

Diffie Hellman parameters must be generated for the OpenVPN server. On Linux/BSD/Unix:

```
./build-dh
```

On Windows:

```
build-dh
```

Output:

```
ai:easy-rsa # ./build-dh
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
.................+.......................................
...................+.............+.................+.........
.....................................
```

### Key Files

Now we will find our newly-generated keys and certificates in the **keys** subdirectory. Here is an explanation of the relevant files:

| Filename | Needed By | Purpose | Secret |
|---|---|---|---|
| ca.crt | server + all clients | Root CA certificate | NO |
| ca.key | key signing machine only | Root CA key | YES |
| dh{n}.pem | server only | Diffie Hellman parameters | NO |
| server.crt | server only | Server Certificate | NO |
| server.key | server only | Server Key | YES |
| client1.crt | client1 only | Client1 Certificate | NO |
| client1.key | client1 only | Client1 Key | YES |
| client2.crt | client2 only | Client2 Certificate | NO |
| client2.key | client2 only | Client2 Key | YES |
| client3.crt | client3 only | Client3 Certificate | NO |
| client3.key | client3 only | Client3 Key | YES |

The final step in the key generation process is to copy all files to the machines which need them, taking care to copy secret files over a secure channel.

Now wait, you may say. Shouldn't it be possible to set up the PKI without a pre-existing secure channel?

The answer is ostensibly yes. In the example above, for the sake of brevity, we generated all private keys in the same place. With a bit more effort, we could have done this differently. For example, instead of generating the client certificate and keys on the server, we could have had the client generate its own private key locally, and then submit a Certificate Signing Request (CSR) to the key-signing machine. In turn, the key-signing machine could have processed the CSR and returned a signed certificate to the client. This could have been done without ever requiring that a secret **.key** file leave the hard drive of the machine on which it was generated.

# Tabellenverzeichnis

# Abbildungsverzeichnis