HÖHERE TECHNISCHE BUNDESLEHRANSTALT WIEN 10
ABTEILUNG FÜR ELEKTRONIK

DIPLOMARBEIT

Ethernetbasiertes Messsystem

Verfasser

Sebastian Lipp
Martin Pietryka

Betreuer

Prof. Dipl.-Ing. Herbert Kern

Jahrgang

5AHELI, 2011/12

Eingereicht

Wien, am 24. Mai 2012

Erklärung

Wir versichern,

dass wir die vorliegende Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und wir auch uns sonst keiner unerlaubten Hilfe bedient haben, dass wir dieses Diplomarbeitsthema bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt haben.

Wien, am 24. Mai 2012

Sebastian Lipp

Martin Pietryka

Zusammenfassung

Das Ziel dieser Diplomarbeit war die Entwicklung eines flexiblen Messsystems auf Ethernet-Basis, zur einfachen Einbindung in vorhandene Netzwerkstrukturen und zugleich die Bereitstellung einfacher Werkzeuge für die Erstellung eines solchen Systems. Deshalb stehen alle Entwicklungen bzw. die gesamte Diplomarbeit unter OpenSource. Da die Ethernet-Schnittstelle mit TCP/IP kein Echtzeitverhalten garantiert, wurde das System nur für unzeitkritische Aufgaben ausgelegt.

Hardware

Es wurden netzwerkfähige Messmodule entwickelt, die das von uns festgelegte Protokoll implementieren und Werkzeuge geschaffen für die Erstellung eigener Messmodule. Ein sogenannter UART-Umsetzer macht es sogar möglich eigene Messmodule um Netzwerkfähigkeit zu erweitern.

Software

Eine eigens entwickelte Software für die Verwaltung der Module stellt die Schnittstelle für andere Applikationen, wie Websiten und Smartphone-Apps bereit. Als Beispiel wurde eine einfache Website geschaffen, die alle im Netzwerk befindlichen Module und deren Messwerte anzeigt.

Abstract

The aim of this diploma project was the development of a flexible measuring system for easily integrating in existing Ethernet networks and to provide simple tools for creating such a system. Therefore all developments - every code line - are OpenSource. Due to the fact that the Ethernet interface and TCP/IP don't support real-time behaviour this system isn't constructed for time critical tasks.

Hardware

Network-compatible measuring modules were developed which implements our protocols and tools were built for creating own measuring modules. A so-called UART-Converter also makes it possible to extend network-compatibility to existing modules.

Software

A specially developed software for managing the modules provides the interface to other applications like websites and smartphone-apps. For instance a simple website were built to list all modules and measured values.

Vorwort

Sie wollen Umweltgrößen an mehreren Standorten (aus der Ferne) überwachen? An den Standorten ist lediglich ein gemeinsames Ethernet-Netzwerk verfügbar und für den Aufbau eines eigenen Netzes fehlt das Budget?

Im Rahmen dieser Diplomarbeit, wurde mit **netcon** ein quelloffenes, flexibles Messsystem auf Ethernet-Basis geschaffen.[1] Dabei wurde darauf Rücksicht genommen, erfahrene Endanwender, Unternehmen und Entwickler gleichermaßen zu bedienen. Je nach Anwendungsfall und Vorkenntnissen sollten Sie in der Lage sein ihr eigenes Messsystem aufzubauen.

Im ersten Kapitel folgt ein Überblick über die **allgemeine Konzeption** - Systemvoraussetzungen, Aufbau, Schnittstellen und grundsätzliche Funktionsweise. Ein zweites Kapitel gibt eine Einführung in **grundlegende Begriffe**, die für ein tieferes Verständnis der Entwicklungen erforderlich sind. Danach folgt das große Kapitel, **Hardware**, das den Aufbau und die Funktionsweise der Messmodule behandelt, sowie in die genaue Verwendung der Schnittstellen und Protokolle auf der Hardwareseite einführt. Das letzte Kapitel, **Software** beschreibt die Verwaltungsschicht und dessen Interfaces für die Anzeige der Moduldaten.

Die beliegende CD enthält die Diplomarbeit als PDF, sowie den gesamten Quellcode des netcon-Systems in den Verzeichnissen netcon Module und netcon Software.

[1] Maßnahmen für die spätere Implementierung eines Aktornetzwerks wurden getroffen. Diese wurde aber aus Zeitgründen nicht durchgeführt.

Danksagung

Zuallererst bedanken wir uns bei unserem Betreuer Prof. Dipl.-Ing. Herbert Kern, der diese Diplomarbeit überhaupt erst ermöglichte. Vielen Dank, dass Sie uns mit vielen tollen Ratschlägen zur Seite standen und irgendwie ein System in die Sache brachten.

Zudem möchten wir jedem danken, der uns Material für die Entwicklung unseres Messsystems zur Verfügung gestellt hat.

Und zuletzt vielen Dank an Matthias Subik, für viele tolle Tipps, die uns immer mal wieder aus dem Licht der Verzweiflung führten.

Vielen Dank!

Inhaltsverzeichnis

1 Überblick [Lipp]

Netcon ist zum einen ein Messsystem zur Einbindung in ein bestehendes Ethernet-Netzwerk, zum anderen aber auch das Ziel flexible Werkzeuge für die Erstellung eines solchen Systems bereitzustellen. Dabei stehen alle Entwicklungen unter OpenSource[2]

Da das System als Übertragungsmedium die Ethernet-Schnittstelle mit der TCP/IP-Protokollschicht verwendet, ist Echtzeitverhalten nicht garantiert. Dadurch ist es nur für unzeitkritische Aufgaben geeignet.

1.1 Zielgruppen

Erfahrene Endandwender, genauso Entwickler sollten mit netcon in der Lage sein, ein Messsystem zu realisieren. Es wurden hardware- und softwareseitig einfache Schnittstellen geschaffen um je nach Wunsch und vorherrschenden Kenntnissen eigene Anwendungen zu erstellen.

Der Anwender kann sich entscheiden, entweder entwickelt er auf Basis der Spezifikationen die netzwerkfähigen Module selbst, oder aber er verwendet die im Rahmen dieser Diplomarbeit gewählten Mikrocontroller-Systeme. Dazu stellt netcon die entwickelte Firmware zur Verfügung. Weiters besteht für netzwerktechnisch unerfahrene Entwickler die Möglichkeit, ihre Module netzwerkfähig zu machen.

[2]Das Projekt wurde über github organisiert. Näheres dazu im Kapitel Projektorganisation.

Auch auf der Softwareseite stehen mehrere Wege offen. Entwickler können eigene Applikationen über die Schnittstelle der Verwaltungsschicht aufsetzen, oder aber auch die entwickelte Weboberfläche zur Anzeige und Steuerung der Module verwenden.

1.2 Anwendungsbeispiele

Netcon sieht in seiner Spezifikation mehrere Typen von Messmodulen vor. Folgende Liste zeigt Anwendungen, die unter anderem mit diesem System verwirklicht werden können:

- Spannungsmessung
- Temperaturmessung
- Zeitmessung

1.3 Systemvoraussetzungen

Das netcon Messsystem wurde für dein Einsatz in einem Ethernet-Netzwerk konzipiert. Dieses muss zumindest über folgende Komponenten verfügen:

- Anschlussmöglichkeiten für die Module (Router/Switch/WLAN)
- DHCP-Server für die IP-Adressvergabe
- **Server** - Javafähige Betriebsumgebung für die Verwaltungsschnittstelle z.B. PC, Embedded System
- **Client** - Anzeigegerät z.B. Smartphone, Computer

Zusätzlich wird gegebenenfalls ein PHP-fähiger Webserver benötigt, um die bereits entwickelte Website verwenden zu können. Die genauen Anforderungen an die Softwareumgebung, sowie die Einrichtung einer Java Runtimte Environment (JRE) und eines Webservers sind im Kapitel Software nachzulesen.

Und nicht zu vergessen sind die wichtigsten Komponenten, die netzwerkfähigen Module. Diese können, wie bereits erwähnt, nach den netcon-Protokollen selbst entwickelt, oder aber auch nach Anleitung erstellt werden. Dazu mehr im nächsten Abschnitt.

1.4 Systemaufbau

Im folgenden sind die zwei grundlegenden netcon-Komponenten inkl. ihrer Schnittstellen beschrieben. Je nachdem wie netcon genutzt werden soll, wird auf weitere Kapitel verwiesen.

1.4.1 Module

Die **Module** sind Hardware, die über Ethernet und TCP/IP erreichbar und abfragbar sind.

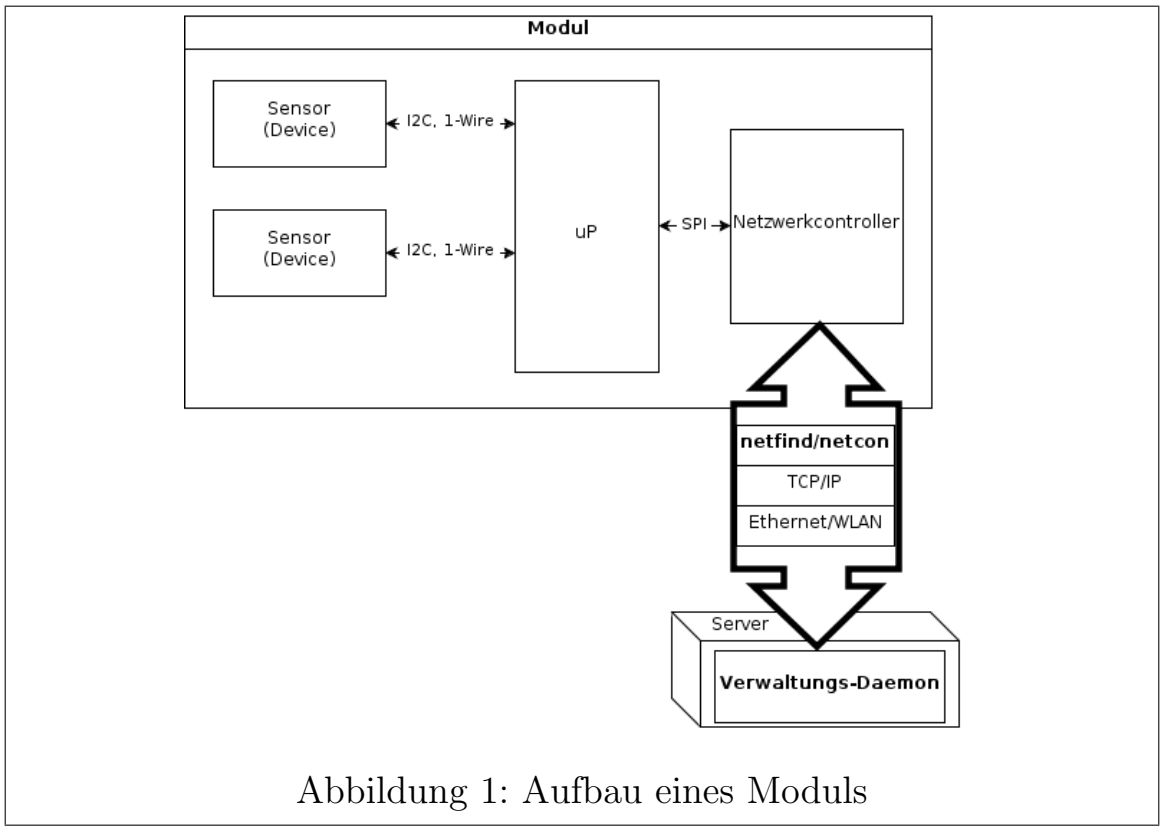


Abbildung 1: Aufbau eines Moduls

Sie vereinen alle benötigten Komponenten - Mess- und Netzwerkeinheit - auf einer Platine (siehe Abb. 1). Hier erfolgt die Übertragung zwischen den Sensoren und dem Mikroprozessor (uP) meist über Schnittstellen, wie I2C oder 1-Wire, während der Netzwerkcontroller per SPI mit dem uP kommuniziert. Über TCP und mit den beiden Protokollen *netfind* und *netcon* erfolgt die Abfrage und Steuerung durch den plattformunabhängigen Verwaltungs-Deamon *netcond*.

Messmodule umfassen beispielsweise Sensoren für Temperatur, Luftdruck und Zeit. Jeder dieser Sensoren wird von netcon als **Device** bezeichnet und kann mit seiner ID abgefragt werden.

Um den Modulen automatisch eine IP-Adresse zuweisen zu können und damit Konfigurationsarbeit zu ersparen, sollten diese auch das DHCP-Protokoll unterstützen.

Es bestehen grundsätzlich drei Möglichkeiten zur Erstellung von Modulen. Wenn Sie alle erforderlichen Kenntnisse besitzen, um die gesamte Entwicklung selbst zu übernehmen, informieren Sie sich im Kapitel Hardware über den Aufbau der netcon-Protokolle. Sind Sie in der Lage einfache Messmodule ohne Netzwerkfähigkeit zu erstellen, verbinden Sie doch ein zusätzliches Netzwerkmodul (siehe Abb. 2). Diese Möglichkeit erfordert lediglich die Implementierung der Seriellen Schnittstelle (UART). Genauso können die im Rahmen dieser Diplomarbeit konzpierten Module mit ihrer Firmware für die Erstellung eigener Module herangezogen werden. Egal welche Wahl Sie treffen, das Kapitel Hardware unterstützt Sie in allen drei Fällen.

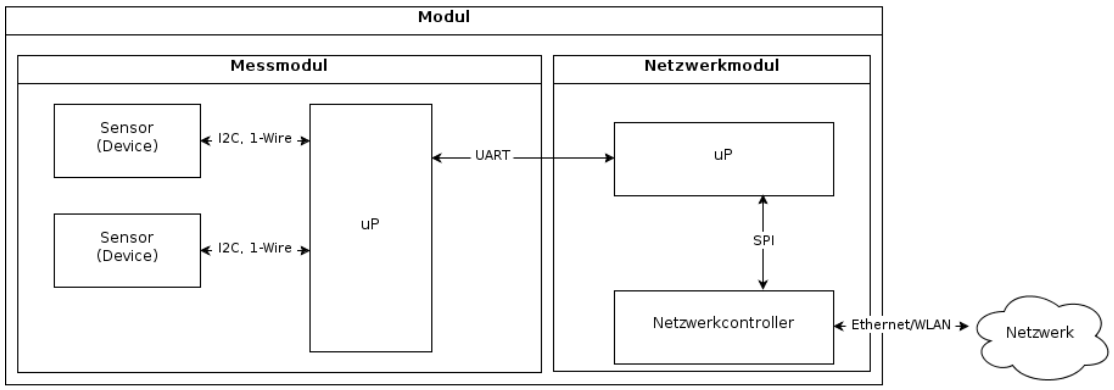


Abbildung 2: Netzwerkfähiges Messmodul mittels LAN-UART Umsetzer

1.4.2 Software

Die Verwaltungsschnittstelle **netcond** ist eine in Java geschriebene Hintergrundanwendung (Daemon), die sich um die Verwaltung der Module kümmert. Wie in Abb. 3 erkennbar können über das netcon Application Interface (netcon API) per TCP die Moduldaten abgefragt werden. Die Anwendung kann beispielsweise eine Website auf einem Webserver oder ein Smartphone-App sein.

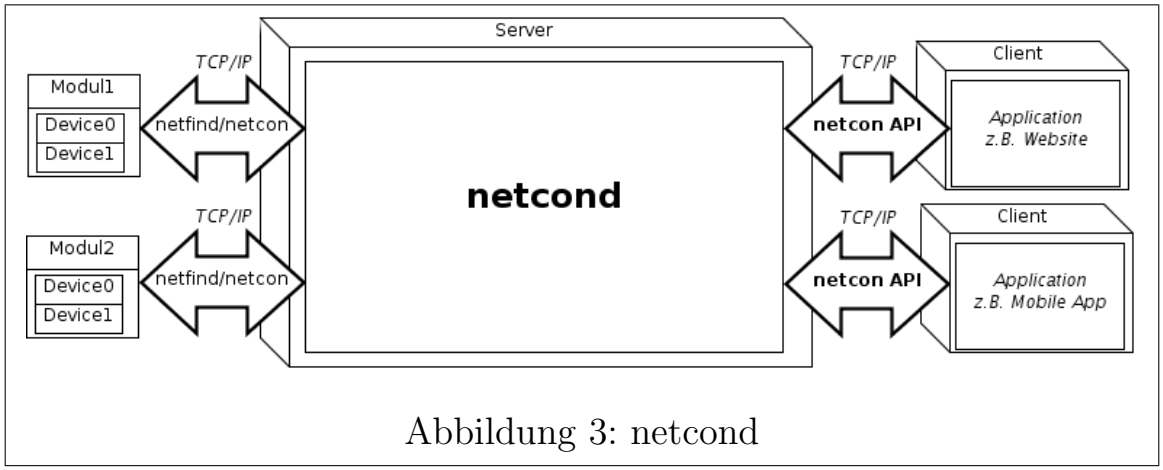


Abbildung 3: netcond

Soll die Anwendung zur Anzeige der Messdaten selbst entwickelt werden, führt das Kapitel Software in die Verwendung der Softwareschnittstelle ein. Sonst kann die bereits entwickelte Website **netcon web** (siehe Abb. 4) verwendet werden. Dazu ist zusätzlich zur JRE ein http-Webserver mit PHP-Unterstützung erforderlich. Deren Installation und Konfiguration ist im Kapitel Software erklärt.

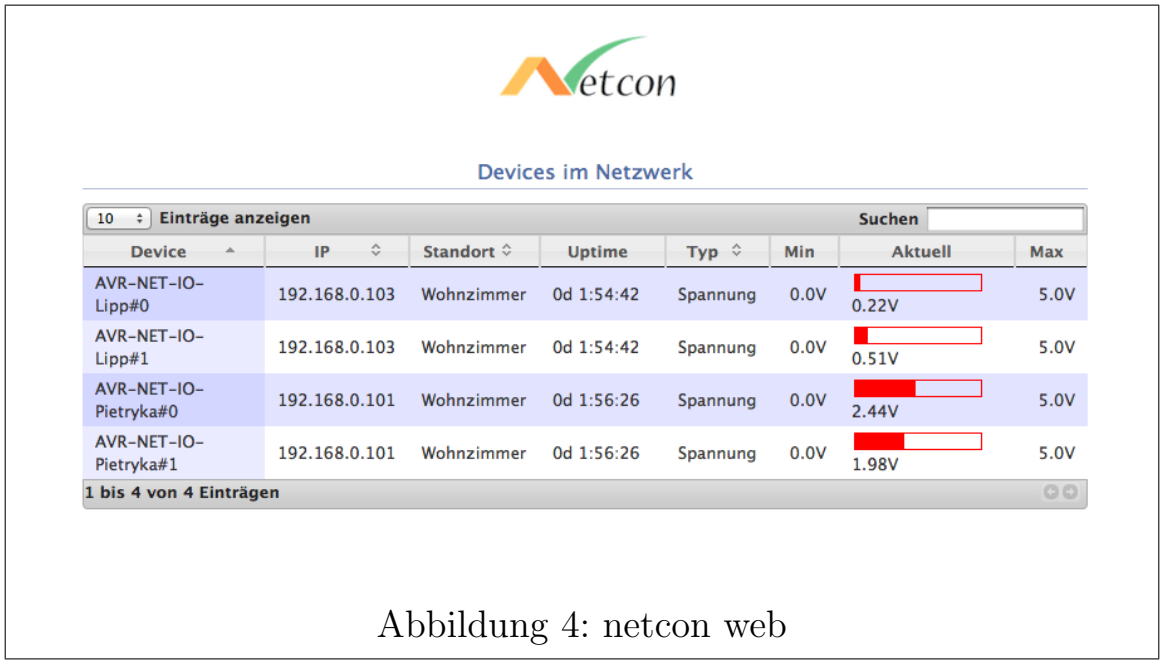


Abbildung 4: netcon web

2 Grundlagen [Pietryka]

2.1 Network Byte Order

In der Geschichte Prozessoren und Heimcomputer gab es verschiedene Prozessorarchitekturen, diese verwendeten auch eine unterschiedliche Anordnung der Zahlen (welche auch mehr als einem Byte bestehen) im Speicher. Bei der Kommunikation über das Netzwerk, sollten die Zahlen aber immer die gleiche Bedeutung haben, deshalb sollten auch immer alle Zahlen in der sogenannten Network Byte Order übertragen werden, diese wird auch als Big-Endian-Format bezeichnet, dabei werden die höherwertigen Bytes zuerst im Speicher gespeichert, also einer niedrigeren Adresse. Ein TCP/IP-Stack wie der uIP-Stack bietet verschiedene Funktionen um die der Host Byte Order, also der Byte Order des Gerätes, in die Network Byte Order umzurechnen, in beide Richtungen. Die Funktionen lauten:

```
Für 16 Bit Zahlen:
    Host to Network Byte Order:
        uint16_t htons(uint16_t i);
    Network to Host Byte Order:
        uint16_t ntohs(uint16_t i);

Für 32 Bit Zahlen:
    uint32_t ntohl(uint32_t i);
    uint32_t htonl(uint32_t i);
```

2.2 OSI-Schichtenmodell

Das OSI-Schichtenmodell ist ein von der ISO im Jahre 1983 standardisiertes Modell, welches als Designgrundlage für Kommunikationsprotokolle in Computernetzen dient. Dabei wird die Kommunikation beim OSI-Modell auf sieben Schichten bzw. Layern aufgeteil. Für jede dieser Schichten sind Anforderungen und Aufgaben definiert, welche von entsprechenden Protokollen realisiert werden müssen. Eine konkrete Umsetzung ist aber nicht vorgegeben, daher gibt es für eine Schicht mehrere in Frage kommenden Protokolle.

2.2.1 Schicht 1: Bitübertragungsschicht

Protokolle auf dieser Schicht kümmern sich darum, wie die unterschiedlichen Netzwerkgeräte untereinander verbunden sind, sowie welches Medium dafür verwendet wird. Hier geht es um die Übertragung einzelner Bits, es müssen je nach Übertragungsmedium verschiedene Codes oder Modulationsverfahren für die Übertragung von einzelnen Bits Festgelegt werden. Weiterhin sind Geräte wie Antennen, Verstärker, Stecker und Buchsen ebenfalls auf dieser Schicht definiert.

Protokolle und Normen: V.24, V.28, X.21, RS 232, RS 422, RS 423, RS 499

2.2.2 Schicht 2: Sicherungsschicht

Auf dieser Schicht definierte Protokolle, sollen sich darum kümmern, dass die Daten zuverlässig, im Sinnen von fehlerfrei, ankommen. Dazu wird der Bitstrom in Blöcke (auch Frames genannt) unterteilt, diese Frames werden mit

einer Prüfsumme versehen, welches es dem Empfänger erlaubt Fehler zu erkennen und ja nach Protokoll auch Fehler bis zu einem gewissen Grad zu korrigieren. Ein erneutes Anfordern von verworfenen Blöcken, oder das Erkennen von überhaupt nicht angekommenen Blöcken ist auf dieser Schicht nicht vorgesehen. Weiterhin findet wird im Protokoll eine sogenannte Zugriffskontrolle definiert, diese regelt, wann und wer auf das Medium zugreifen darf. Hardwareelemente auf dieser Schicht sind die Bridge und der Switch.

Bekannte Protokolle auf dieser Schicht sind das Ethernet-Protokoll, welches besser als das kabelgebundene lokale Netzwerk bekannt ist, oder das IEEE 802.11 Protokoll, welches das bekannte Wireless-LAN beschreibt.

2.2.3 Schicht 3: Vermittlungsschicht

Diese Schicht kümmert sich um das Weiterleiten von Paketen bei paketorientierten Diensten. Meist besteht zwischen Sender und Empfänger keine direkte Verbindung, daher muss das Paket mehrere Zwischenstationen durchlaufen bis es an seinem Ziel ankommt. Dieser Vorgang wird Routing genannt, die Hardware für diese Schicht ist der Router.

Protokolle und Normen: X.25, ISO 8208, ISO 8473 (CLNP), ISO 9542 (ESIS), IP, IPsec, ICMP

2.2.4 Schicht 4: Transportschicht

Die Aufgabe dieser Schicht ist einerseits die Stauvermeidung, andererseits Stellt diese Schicht mit ihren Protokollen für die höheren Schichten einen einheitlichen

Zugriff. Deswegen müssen die höheren Schichten auch nicht wissen welche Protokolle auf den unteren Schichten arbeiten. Bekannte Protokolle sind TCP, welches bereits die Datensicherheit durch Neuübertragungen vorgesehen hat, sowie UDP, welches, außer einer Prüfsumme, keine Sicherheitsmechanismen hat.

2.2.5 Schicht 5: Sitzungsschicht

Die Sitzungsschicht sorgt für die Prozesskommunikation zwischen zwei Systemen, Sie sorgt dafür, dass Zusammenbrüche einer Sitzung oder Ähnliche Störungen behoben werden. Dazu werden sogenannte Wiederaufsetzpunkte eingeführt, an denen die Sitzung nach einem Ausfall der Verbindung wieder fortgesetzt werden kann.

2.2.6 Schicht 6: Darstellungsschicht

Bei der Darstellungsschicht geht es darum, die systemabhängige Darstellung der Daten in eine unabhängige Form zu bringen. Diese erlaubt den korrekten Datenaustausch zwischen zwei unterschiedlichen Systemen. Ebenfalls zur Schicht 6 gehören Verschlüsselung und Datenkompression, sowie eventuell das Übersetzen zwischen verschiedenen Datenformaten.

2.2.7 Schicht 7: Anwendungsschicht

Die oberste Schicht, die sogenannte Anwendunsschicht verschafft Anwendungen den Zugriff zum Netz. Hierzu zählen Alle Anwendungen die einem Netzwerkkommunikation Benötigen wie E-Mail Client, Browser, etc.

Protokolle auf dieser Schicht sind unter Anderem HTTP, FTP, SSH, Telnet, etc.

2.3 Ethernet

Das Ethernet beschreibt unter anderem Protokolle sowie Hardware für ein kabelgebundenes Datennetz, es ist auch unter dem Namen Local Area Network (LAN) bekannt. Die Übertragungsraten sind mit 10Mb/s, 100Mb/s, 1Gb/s und 10Gb/s definiert. Das Ethernet umfasst die OSI-Schichten 1 und 2, die Übertragung läuft in Paketen, den sogenannten Ethernet-Frames ab. Diese Frames haben eine minimale Länge von 64 Bytes und können Maximal 1518 Bytes groß werden. Sind die Nutzdaten kleiner als 46 Bytes so werden der Rest mit Nullen aufgefüllt um die Minimale Größe von 64 Bytes zu erreichen.

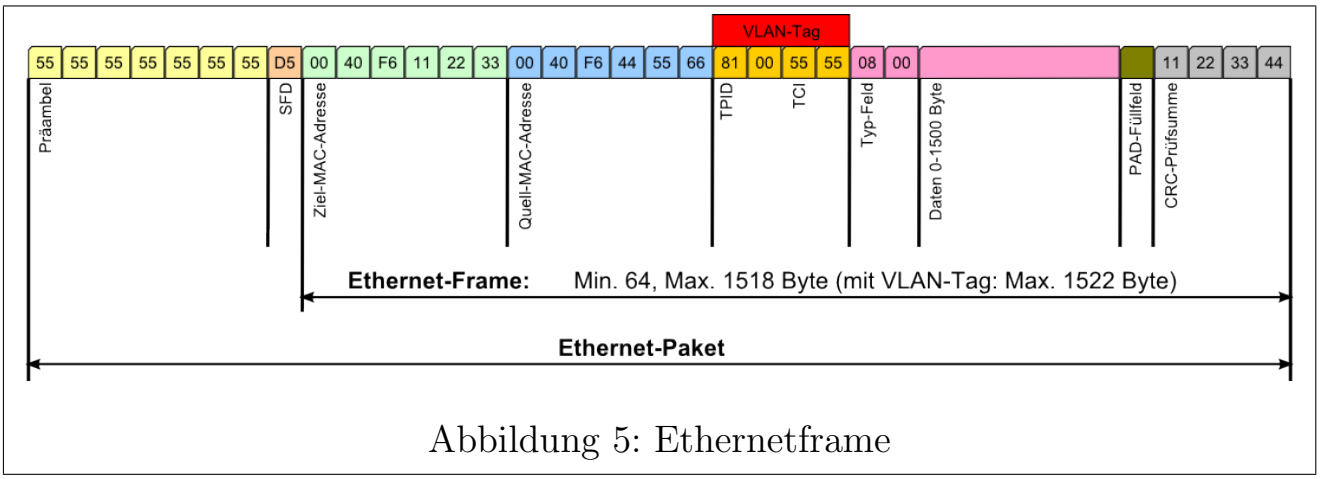


Abbildung 5: Ethernetframe

Das Präambel sowie der Start Frame Delimiter (SFD) dienen dazu, um auf Bitebene einerseits den Beginn eines Ethernetframes zu erkennen, andererseits um eventuell eine Taktsynchronisation durchzuführen, da das Präambel ein Alternierendes Bitmuster enthält. Danach kommt jeweils eine Ziel- sowie eine Quell-MAC-Adresse, diese Adressen sind jeweils 6-Byte lang. Das VLAN-Tag ist optional und dient dazu, innerhalb eines physikalischen Ethernet-Netzwerks noch

zwischen virtuellen (Virtual Local Area Network) Netzwerken unterscheiden zu können. Das Typ-Feld gibt an, welches Protokoll auf der nächsthöhrenen Schicht Verwendung findet, einige von denen sind:

```
0x8000:    IP Version 4
0x0806:    Address Resolution Protocol (ARP)
0x86DD:    IP Version 6
```

Danach kommen die Nutzdaten, diese müssen eine Länge von mindestens 46 Bytes haben, ist diese nicht der Fall, so werden die restlichen Bytes mit Nullen, dem Padding aufgefüllt, die maximale Länge der Nutzdaten beträgt 1500 Bytes. Die letzten 4 Bytes bilden eine sogenannte CRC-Prüfsumme, durch diese lässt sich berechnen ob alle Bytes im Ethernet-Frame korrekt übertragen wurden, ist dies nicht der Fall, so sollte dieser Frame verworfen werden.

2.3.1 MAC-Adresse

Die MAC-Adresse (Media-Access-Controll-Adresse) ist eine Adresse welche Netzwerkgeräte eindeutig identifiziert, Ethernet Pakete werden mithilfe dieser Adresse adressiert. Diese Adresse ist 48-Bit lang und ist auf der Welt eindeutig für jedes Netzwerkgerät. Die Darstellung erfolgt in Hexadezimaler Darstellung wie 00:80:41:ae:fd:7e oder 00-80-41-ae-fd-7e. Will man seine Netzwerkfähigen Geräte am Markt verkaufen, so benötigt man gültige MAC-Adressen, diese kann man in Blöcken kaufen. Kleinere Unternehmen können Blöcke mit 4096 MAC-Adressen kaufen, größere Unternehmen besitzen die Möglichkeit einen Block mit 16,8 Millionen Adressen zu kaufen.

Entwickelt man aber noch geringere Stückzahlen, so gibt es beispielsweise von der Firma Microchip EEPROMS, welche mit einer bereits vorprogrammierten

und natürlich gültigen MAC-Adresse geliefert werden. Eine weitere Möglichkeit gültige MAC-Adressen während der Entwicklung zu vergeben sind die sogenannten lokal administrierten Adressen, dabei wird im ersten Byte das zweite Bit auf 1 gesetzt. Ist dies der Fall, so handelt es sich um eine lokal administrierte Adresse, der Nachteil hierbei ist natürlich, dass diese Adresse nicht mehr eindeutig ist.

2.3.2 Broadcast

Bei einem sogenannten Broadcast wird im Ethernet-Header eine spezielle MAC-Adresse als Ziel angegeben, nämlich FF:FF:FF:FF:FF:FF, Pakete mit dieser Zieladresse werden an alle Geräte in einem LAN-Netzwerk verschickt, jedoch nicht in ein anderes Netzwerk geroutet.

2.4 IP

Das Internet-Protocol stelle eine hardwareunabhängige Schicht für die höheren Schichten zur Verfügung, dabei erfolgt die Adressierung durch logische Adressen, die IP-Adressen. Diese Adressen sind 4 Bytes lang und werden in der Form XXX.XXX.XXX.XXX angeschrieben wobei natürlich jedes Byte von 0-255 geht. Ein IP-Paket hat den folgenden Aufbau:

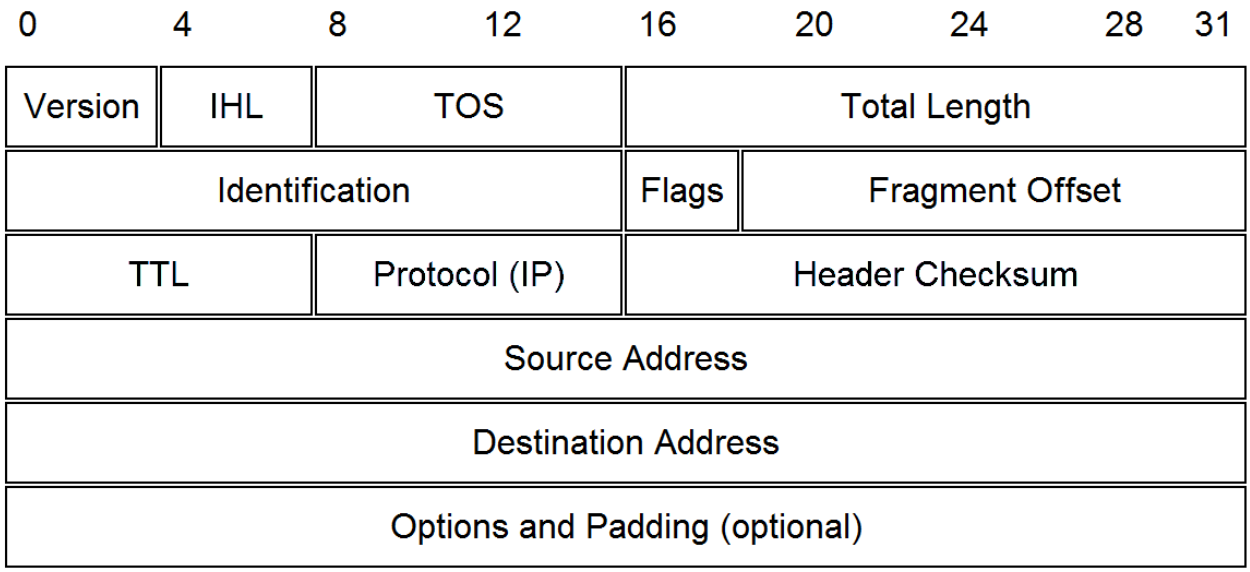


Abbildung 6: IP Paket

2.5 ARP

Will man nun ein IP-Paket an eine bekannte IP-Adresse schicken, so ergibt sich folgendes Problem. Welche Ziel-MAC-Adresse soll im Ethernet Header angegeben werden, man weiss zwar die Ziel-IP-Adresse aber eben nicht die MAC-Adresse, diese wird aber benötigt. Hier kommt das sogenannte Address Reso-

lution Protocol ins Spiel, dies sorgt dafür, dass das Netzwerkgerät die MAC-Adresse des Ziels herausfindet. Der Ablauf ist ungefähr folgender, ist die Ziel-MAC Adresse nicht bekannt, so wird ein ARP-Paket als Broadcast verschickt. In diesem steht quasi die Anfrage nach der MAC-Adresse zu einer passenden IP-Adresse. Fühlt sich nun ein Gerät im Netzwerk für diese IP-Adresse angesprochen, so antwortet es mit einem Paket, dieses mal nicht als Broadcast, im Paket steht nun als Quell-MAC-Adresse die vorher gesuchte MAC-Adresse zur angefragten IP-Adresse. Diese MAC-Adresse wird nun in einem Cache für eine gewisse Zeit geschrieben, damit man nicht bei jedem neuen IP-Paket diese ARP-Anforderung durchführen muss.

2.6 TCP

2.7 UDP

2.8 DHCP

DHCP steht für Dynamic Host Configuration Protocol, durch diese auf UDP basierende Protokoll wird es möglich Netzwerkgeräte in ein bestehendes Netzwerk einzubinden, ohne dass diese vorher manuell konfiguriert werden. Dabei sendet der Client beim Starten einen Broadcast, dieser wird von einem im Netzwerk befindlichen DHCP Server empfangen und verarbeitet. Über DHCP können unter anderem die IP-Adresse, die Netzwerkmaske, das Standardgateway, sowie der DNS-Server bezogen werden, neben diesen Optionen gibt es noch einige weitere, welche aber seltener verwendet werden, es kann zum Beispiel die Adresse eines Zeitservers angegeben werden. DHCP muss auf UDP aufbauen, da nur mit UDP Broadcast-Pakete möglich sind, dabei wartet der Server auf Port 67 auf Anfra-

gen, der Client erhält die Serverantworten auf Port 68. Zudem teilt der Server dem Client mit, wie lange ein sogenannter DHCP-Lease gültig ist, laut dem DHCP Protokoll muss nach der halben Zeit erneut eine Anfrage an den Server geschickt werden. Im Rahmen dieser Diplomarbeit wurde ein vollständiger DHCP Client implementiert, inklusive der Aktualisierung nach vorgegebener Zeit. Dazu musste aber die Framegröße des uIP Stacks auf 600 Byte erweitert werden.

3 Hardware [Pietryka]

3.1 Zufallsgenerator

Für das DHCP-Protokoll, sowie für das netfind-Protokoll wurde ein Zufallsgenerator benötigt, dabei bietet eine bereits beim AVR-Compiler mitgelieferte Bibliothek die Funktion eines Pseudozufallszahlengenerators, dieser generiert eine immer gleiche Zahlenfolge, diese Zahlenfolge hängt vom einem Startwert(dem Seed) ab. Damit die Module nun unterschiedliche Zufallszahlen generieren muss für den Generator ein wirklich Zufälliger Seed gesetzt werden. Um eine wirklich Zufällige Zahl für den Seed zu bekommen, wurde sich die Tatsache zu nutze gemacht, dass der RAM-Speicher beim Einschalten mit zufälligen Werten befüllt ist. Dabei wird der Speicher vom Ende bis zum Ende des Bereiches wo globale Variablen bereits initialisiert wurden durchlaufen, die einzelnen Speicherplätze werden miteinander XOR-Verknüpft. Diese Funktion sieht folgendermaßen aus:

```
uint16_t get_seed(void)
{
    uint16_t seed = 0;
    uint16_t *p = (uint16_t *) (RAMEND + 1);
    extern uint16_t __heap_start;

    while(p >= &__heap_start + 1)
        seed ^= *(--p);

    return seed;
}
```

Der zurückgelieferte Wert wird dem Zufallszahl-Generator der AVR-Bibliothek übergeben, somit hat man die Möglichkeit auf unterschiedlichen Modulen und nach dem Neustaren unterschiedliche Zufallszahlenfolgen zu erhalten.

3.2 Der Ethernet Controller

3.3 Auswahl des Ethernet Controllers

Damit ein Mikrocontroller über das Ethernet kommunizieren kann, wird eine entsprechende Hardware benötigt, der sogenannte Ethernet Controller. Ein Ethernet Controller übernimmt dabei die Aufgaben der OSI-Schichten 1(Physical) und 2(Data-Link). Der Controller benötigt zudem einen entsprechend großen Empfangspuffer, um mindestens einen vollwertigen Ethernet-Frame(1542 Byte) aufzunehmen zu können. Dabei standen für 8-Bit Mikrocontroller vorerst zwei verschiedene Bausteine zur Auswahl, einmal der CP2200 von SiLabs, und einmal der ENC28J60 von Microchip. Beide Controller haben, was die Netzwerkkommunikation angeht, so ziemlich die selben Features, der gravierende Unterschied liegt jedoch in der Ansteuerung dieser. Der CP2200 wurde von SiLabs, wie es scheint, nur für die Verwendung mit einem Mikrocontroller vom Typ 8051 entwickelt, die Ansteuerung erfolgt deshalb über einen parallelen Adress-/Datenbus wodurch man mindestens 16 Leitungen und Pins am Mikrocontroller benötigt. Beim ENC28J60 erfolgt die Kommunikation über den SPI-Bus, daher benötigt man nur vier Leitungen(MOSI, MISO, SCK, CS), dadurch hat auch der Netzwerkcontroller selber nur 28 Pins und ist auch im "bastlerfreundlichen"DIP-Gehäuse zu bekommen. Ein anderer Faktor für die Auswahl des ENC28J60 war das Vorhandensein einer günstigen Entwicklungsplatine, es gibt bei Pollin den AVR-NET-IO Bausatz, dieser kostet nur 20 € und enthält alle für die Netzwerkprogrammierung benötigten Komponenten(ATmega32, ENC28J60, RJ-45

Buchse).

3.4 ENC28J60 Beschaltung

Die Aussenbeschaltung benötigt neben einigen Standardbauelementen auch einige 1% Widerstände und einen 1:1 Übertrager, jedoch gib es RJ-45 Buchsen in denen bereits der Übertrager, sowie die LEDs bereits eingebaut sind.

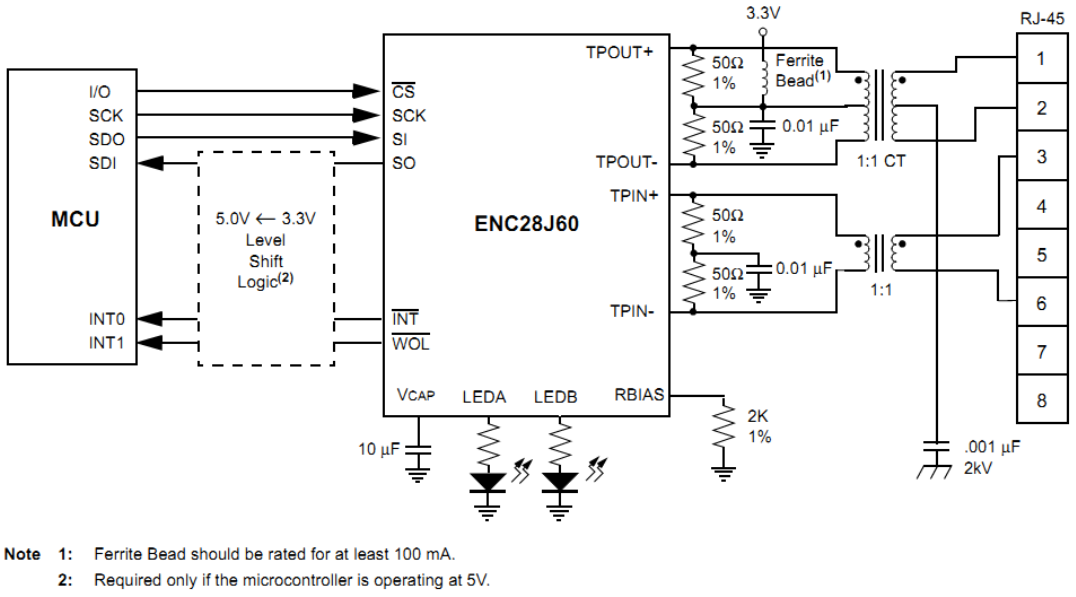


Abbildung 7: Aussenbeschaltung ENC28J60

3.5 ENC28J60 Treibersoftware

Die erste Aufgabe war es eine Bibliothek zu schreiben, welche es erlaubt, mithilfe des ENC28J60, Ethernet Pakete über das Netzwerk zu verschicken. Es

waren zwar im Internet verschiedene Programme für den ENC28J60 vorhanden, diese waren aber teilweise schlampig und unschön geschrieben. Am Ende Stand eine Bibliothek zur Verfügung, die auch für andere Projekte genutzt werden kann, diese wurde für den AVR geschrieben und besteht aus zwei Dateien „enc28j60.h“ und „enc28j60.c“. Zurzeit übernimmt die Bibliothek auch die Initialisierung des SPI-Busses, will man dies verhindern, so muss man die Zeilen 187-197 in der Datei „enc28j60.c“ entfernen, weiterhin wird es eventuell nötig sein die Pins für das Chip-Select anzupassen, dazu muss man die Funktionen get_cs() und release_cs() in den Zeilen 24-32 ebenfalls anpassen. Für die Benutzung der Bibliothek stehen dann drei Funktionen zur Verfügung. Die Bibliothek selber finde sich im Ordner „netcon Module/Pollin AVR-NET-IO/enc28j60/“ in den Dateien „enc28j60.c“ und „enc28j60.h“.

Der Treiber selbst kann als stabil angesehen werden, es sind in den mehreren Monaten der Arbeit mit diesem keine Fehler aufgefallen, auch wurde das Geräte mehrere Stunden über die Nacht laufen gelassen, mit Erfolg.

3.5.1 void enc28j60_init(const uint8_t *mac_addr)

Initialisiert die Hardware des ENC28J60 mit der angegebenen MAC-Adresse, diese muss als ein Zeiger auf ein Array mit 6 Byte übergeben werden. Zurzeit wird auch die SPI-Hardware eines ATmega32 ebenfalls initialisiert, ist dies nicht gewünscht, so muss der Code entsprechend abgeändert werden.

```
const uint8_t mac_addr[6] = {0x02, 0x00, 0x00, 0x00, 0x00, 0x01};
enc28j60_init(mac_addr);
```

3.5.2 void enc28j60_transmit(const uint8_t *data, uint16_t len)

Sendet ein Ethernet-Paket in das Netzwerk, welches über einen Zeiger auf data referenziert wurde, mit der Länge len.

```
uint8_t buf[512];

strcpy(buf, "Das ist ein ungueltiges Paket");
enc28j60_transmit(buf, strlen(buf));
```

3.5.3 uint16_t enc28j60_receive(uint8_t *data, uint16_t max_len)

Schaut ob ein Paket im Puffer des ENC28J60 angekommen ist, ist dies der Fall, so werden bis zu max_len Bytes in den durch data referenzierten Bereich kopiert. Als Rückgabewert liefert die Funktion die Anzahl der erfolgreich gelesenen Byte. Ist kein neues Paket beim Aufruf der Funktion vorhanden, so ist der Rückgabewert 0.

```
uint8_t buf[512], len;

while(1 > 0)
{
    len = enc28j60_receive(buf, 512);
    if(len > 1)
    {
        // Neues Paket
    }
}
```

3.5.4 Ethernet-DK Port

Weiterhin, wurde eine Portierung des ENC28J60C Treibers auf das Ethernet-DK von SiLabs durchgeführt. Dies lässt sich mit dem freien C-Compiler SDCC compilieren, die entsprechenden Projektdateien sind im Ordner „netcon Module/Ethernet-DK/enc28j60“. Damit dieses Projekt erfolgreich compiliert muss der SDCC Compiler entsprechen in der SiLabs IDE eingestellt konfiguriert werden, zudem muss sowohl dem Linker als auch dem Compiler der Parameter „–model-large“ übergeben werden.

3.6 CP2200

Auch wenn der CP2200 Ethernet-Controller nicht weiter verwendet wurde, so wurde ebenfalls ein Treiber geschrieben, sowie der uIP Stack auf diesen portiert, diese Projektdateien finden sich im Ordner „netcon Module/Ethernet-DK/cp2200“. Über die Stabilität kann keine Auskunft gegeben werden, jedoch hat die Netzwerkkommunikation über eine Stunde hinweg funktioniert. Auf eine weitere Dokumentation wird hier verzichtet, da der Treiber nur als Nebenprodukt verschiedener Experimente anzusehen ist.

3.7 Der uIP TCP/IP Stack

Die Kommunikation der Module soll im Netzwerk über TCP ablaufen, zum einen damit gewährleistet ist, dass die Daten auch ankommen, zum anderen damit die Module mit allen Betriebssystemen, Programmiersprachen und Netzwerkbibliotheken kompatibel sind, denn dies ist nur durch TCP oder UDP der

Fall, diese Protokolle sind in jedem modernen Betriebssystem ein fester Bestandteil. Eine Software die sich nun um das IP, TCP und eventuell auch um das UDP Protokoll kümmert, nennt man einen TCP/IP Stack.

Der von uns verwendetet Ethernet-Controller liefert uns jedoch nur Ethernet-Frames, daher benötigt man einen in Software geschriebenen TCP/IP Stack, einen solchen zu schreiben wäre an sich schon eine eigene Diplomarbeit, hinzu kommt noch die Tatsache, dass RAM-Speicher und Rechengeschwindigkeit des Prozessors begrenzt sind, das ganze soll ja auf einem ATmega32 mit 2K RAM laufen.

Ein hierfür entwickelter TCP/IP Stack ist der uIP Stack von Adam Dunkels, dieser wurde genau für die oben genannten Anforderungen entwickelt und ist komplett in C geschrieben, und zwar plattformunabhängig. Der Austausch der Ethernet-Frames zwischen dem ENC28J60C Treiber und dem Stack erfolgt über einen globalen Puffer, welcher in diesem Fall eine Größe von 600 Byte hat. Zudem ist es möglich die Routinen, welche zur Berechnung der Prüfsummen benutzt werden selber zu implementieren, um hier eventuell Geschwindigkeitsvorteile rauszuholen. Der Stack beherrscht die Kommunikation über UDP und TCP und erlaubt mehrere eingehende sowie ausgehende Verbindungen. Durch die Eigenheit des Stacks müssen Programme die den Stack benutzen als sogenannte Zustandsautomaten programmiert werden. Wenn ein Paket an einem für eine Anwendung bestimmten Port angekommen ist, so wird eine vorher definierte und vom Benutzer geschriebene Funktion aufgerufen. Diese Funktion verarbeitet nun das Paket und sendet eventuell eine Antwort. Es aber nicht unbedingt ein hereingekommenes Paket der Auslöser für einen Aufruf dieser Funktion sein, diese Funktion wird auch zum Beispiel alle 100ms automatisch Aufgerufen um es dem Benutzer zu ermöglichen eventuelle Verwaltungsaufgaben zu erledigen.

Im Grunde sieht der Aufbau dieser Funktion so aus:

```
void generic_app_call(void)
{
    if(uip_newdata()) {
        // Neue Daten
    }

    if(uip_rexmit()) {
        // Beim Übertragen der TCP-Daten ist ein Fehler aufgetreten,
        // die Daten müssen neu gesendet werden
    }

    if(uip_poll()) {
        // Ereignis wird alle 100ms aufgerufen
    }
}
```

Neben den hier im Beispiel aufgeführten Ereignissen uip_newdata(), uip_rexmit() und uip_poll() gibt es noch weiter, auf diese wird aber jetzt nicht näher eingegangen, die weiteren Ereignisse sind in der uIP Dokumentation ganz gut beschrieben, welche sich in dem „uip-1.0.tar.gz“ Archiv findet.

Um selber Daten mit Hilfe des uIP-Stacks zu schicken gibt es die Funktion void „void uip_send(const void * data, int len)“, diese Funktion sendet die Daten über die aktuelle Verbindung, dabei wird ein Zeiger auf die Daten, sowie deren Länge übergeben. Diese Funktion lässt sich nur innerhalb einer der oben erwähnten Callback-Funktionen nutzen, daher will man kontinuierlich Daten senden geht dies nur höchstens alle 100ms.

3.8 DHCP Implementierung

Da aufgrund des uIP Stacks die DHCP-Implementierung als ein Zustandsautomat erfolgen musste, wurden folgende Zustände definiert:

DHCP_BOOT_WAIT: In diesem Zustand verbleibt das Gerät nach dem Einschalten eine zufällige Zeit lang(2-5 Sekunden).

DHCP_DISCOVER: Es wird ein Discover-Paket gesendet, dieses wird als Broadcast verschickt und dient dazu einen oder mehrere DHCP-Server im Netzwerk zu finden.

DHCP_OFFER: Es wird auf eine Angebot (Offer) von einem oder mehreren DHCP-Servern gewartet, dabei wird immer das erste Angebot gewählt. Das Offer-Paket vom DHCP-Server enthält bereits alle benötigten Daten wie IP-Adresse, Subnetzmaske, etc.

DHCP_REQUEST: Das Netzwerkmodul sendet nun eine Anfrage an den Server ob es die IP-Adresse für sich beanspruchen kann.

DHCP_ACK: Es wird nun auf eine Zustimmung (Acknowledge) des DHCP-Servers gewartet, kommt dieses an, so werden die Parameter wie IP-Adresse, Subnetzmaske, etc. auf das Netzwerkgerät übernommen und für die weitere IP-Kommunikation verwendet.

DHCP_WAIT_RENEW: Die DHCP-Implementierung verweilt in diesem Zustand bis die halbe Lease-Time abgelaufen ist, denn dann muss eine sogenannte Erneuerung (Renew) stattfinde, diese Erneuerung der Daten läuft aber genauso ab wie beim ersten Mal.

In einem Diagramm dargestellt sieht dieser Zustandsautomat folgendermaßen aus:

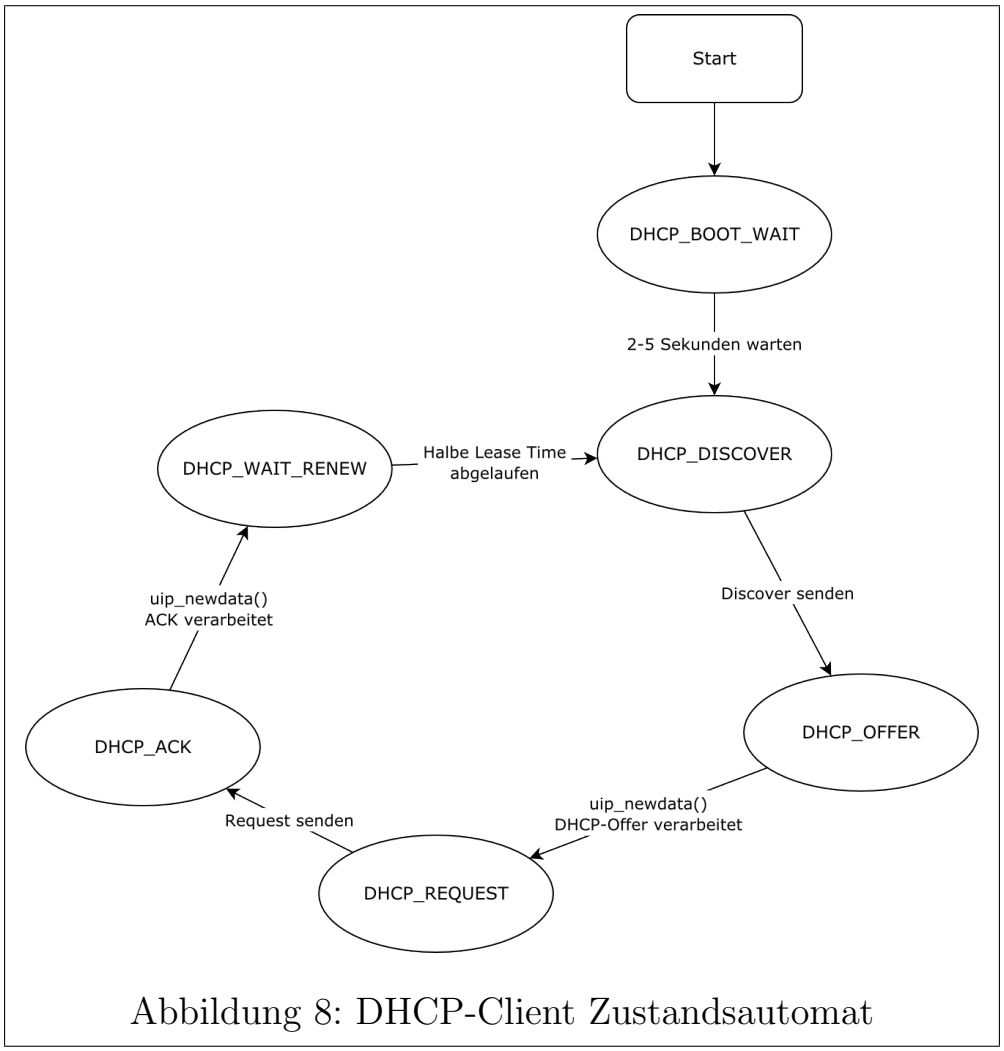


Abbildung 8: DHCP-Client Zustandsautomat

Da hier Broadcasts verwendet werden basiert DHCP auf dem UDP-Protokoll, dabei lauscht der Server auf dem Port 67 auf Anfragen, der Client erhält die Antworten der DHCP-Server am Port 68. Auf den konkreten Code wird hier nicht weiter näher eingegangen, eine vollständige DHCP-Implementierung findet sich im Ordner „netcon Module/Pollin AVR-NET-IO/netcon“ in den Dateien „dhcp.c“ und „dhcp.h“.

3.9 netcon Serial Protocol

Wie bereits erwähnt war es eines der Ziele, ein Netzwerkmodul zu entwickeln, welches es erlaubt andere Messgeräte „einfach“ in das Messsystem einzubinden. Dazu kommuniziert das Netzwerkmodul mit dem Messgerät über die serielle Schnittstelle nach einem bestimmten Protokoll, dem netcon Serial Protocol. Die Messgeräte verhalten sich dabei passiv und erhalten vom Netzwerkmodul verschiedene Anfragen, diese sollte innerhalb von 50ms entsprechend beantwortet werden. Jegliche Kommunikation läuft dabei über den standardmäßigen ASCII-Zeichensatz und mit einer Symbolrate von 9600 Baud.

Kommandos:

```
Anfrage: n
Antwort: <Name>\n
Fragt den Namen des Messgerätes ab.

Anfrage: o
Antwort: <Standort>\n
Fragt den Standort des Messgerätes ab.

Anfrage: a
Antwort: <Anzahl>\n
Fragt die Anzahl der Sensoren, die mit dem Messgerät gemessen
werden können ab, wobei Anzahl eine Zahl von 00-FF sein kann.

Anfrage: w##
Antwort: <Wert>\n
Fragt den aktuellsten Messwert von dem Sensor mit der Nummer
## ab.
```

```
Anfrage: f##
Antwort: <Format>\n
Fragt das Format für den Wert von Sensor ## ab.
Dieses gibt an wie der Wert an das Netzwerkmodul übergeben
wird, folgende Formate sind vorgesehen:
    h - Hexadezimal
    i - Dezimal
    f - Fließkommazahl

Anfrage: t##
Antwort: <Typ>\n
Fragt den Typ für den Sensor ## ab, dies soll es ermöglichen,
bei der Ausgabe nach Temperatursensoren, Luftdrucksensoren,
etc. zu filtern. Der Typ ist eine Zahl von 00-FF, jedoch
wurden noch keine Typen definiert, außer 00 für Spannung
definiert.

Anfrage: m##
Antwort: <Minimum>\n
Fragt das Minimum des Sensors ## ab.

Anfrage: x##
Antwort: <Maximum>\n
Fragt das Maximum des Sensors ## ab.
```

Beim Einschalten des Netzwerkmoduls werden zuerst der Name, der Ort und die Anzahl der Sensoren abgefragt, anschließend wird für jeden Sensor das Format, der Typ, der Min und der Max Wert abgefragt. Nach dieser Initialisierung werden alle 500ms die aktuellen Werte alles Sensoren nacheinander ab-

gefragt. Eine Implementierung dieses Protokolls befindet sich im Ordner „netcon Module/Pollin AVR-NET-IO/netcon_ser“ in den Dateien „serconn.c“ und „serconn.h“, diese Implementierung wurde noch nicht einem dauerhaften Test unterzogen, es könnten Fehler enthalten sein.

3.10 nefind Protocol

Da man die Module in ein bestehendes Netzwerk mit möglichst wenig bis gar keinen Konfigurationsaufwand einbinden soll, musste ein Protokoll entwickelt werden, welches es dem netcond-Daemon erlaubt angeschlossene Module im Netzwerk zu finden. Diese Protokoll basiert auf Broadcasts und baut deshalb auf UDP auf. Um die Module im Netzwerk zu finden, muss der Daemon eine Broadcaspaket mit dem Zielport 50000 senden, diese Anfrage muss folgendermaßen aussehen:

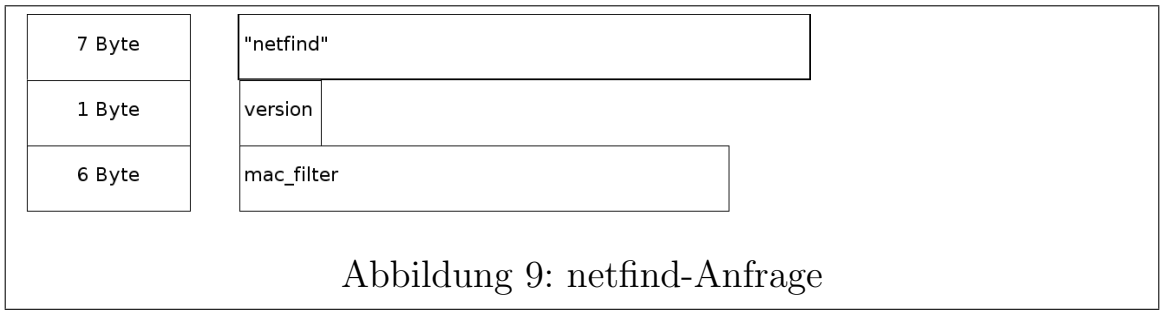


Abbildung 9: netfind-Anfrage

„netfind“ ist eine ASCII-Zeichenkette mit dem selben Inhalt, diese Zeichenkette dient zur Erkennung des Paketes, das Feld „version“ ist für eventuelle Unterscheidungen bei zukünftigen Protokollversionen vorgesehen, die einzige zurzeit gültige Version ist 0. Zuletzt erlaubt es das Feld „mac_filter“, nach Netzwerkmodulen mit bestimmten MAC-Adressen im Netzwerk zu suchen, dabei vergleicht das Netzwerkmodul seine eigene MAC-Adresse mit der im Paket angegebenen

Adresse, ist diese nicht gleich so sendet dieses Modul keine Antwort. Sollen alle Module gefunden werden, der Filter also nicht aktiv sein, so muss dieses Feld auf den Wert FF:FF:FF:FF:FF:FF gesetzt werden.

Nach dieser Anfrage antworten die verschiedenen Netzwerkmodule innerhalb von 1-3 Sekunden, dabei wird der exakte Zeitpunkt zufällig gewählt um die Broadcasts im Netzwerk etwas über die Zeit zu verteilen, die Antwort eines Moduls hat den folgenden Aufbau:

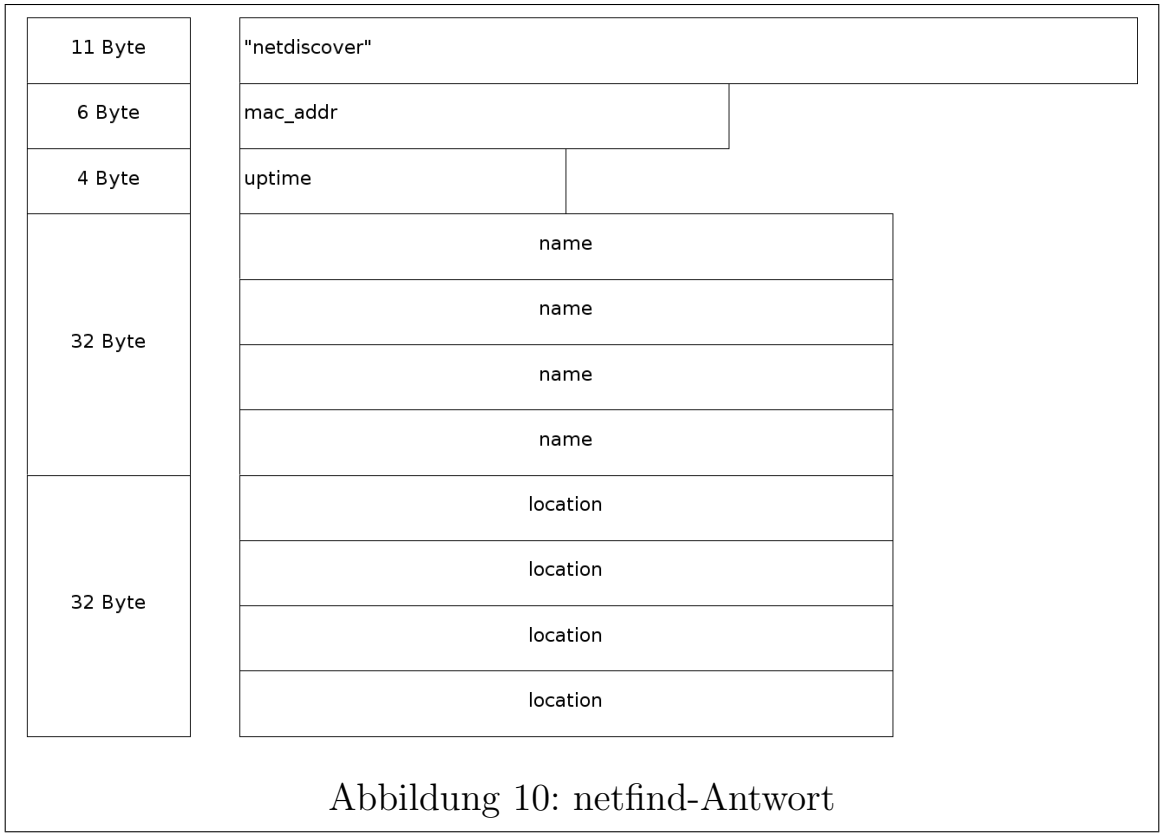


Abbildung 10: netfind-Antwort

„netfind“ ist ebenfalls wieder eine ASCII-Zeichenkette, welche signalisiert, dass es sich bei dem Paket um ein netfind-Antwortpaket handelt. Das Feld „mac_addr“ enthält die 6-Byte lange MAC-Adresse des entsprechenden Moduls. Als nächstes Feld folgt die ’‘uptime“, diese Feld enthält die Zeit seit dem Start des Moduls, diese Zeit ist in 10ms Schritten angegeben und ist ein Ganzzahlwert in der

Network Byte Order. Die beiden letzten Felder ''name“ und ''location“ geben jeweils den Namen und den Standort des Modules an, dabei handelt es sich um bis zu 32-Byte lange ASCII-Zeichenketten, welche aber vorzeitig mit einer Nullterminierung(Bytewert: 0) abgeschlossen werden können. Somit können beide Felder effektiv nur mit 31-Byte belegt werden, das letzte Byte muss die Nullterminierung sein.

Eine Implementierung dieses Protokolls findet sich in dem Ordner „netcon Module/Pollin AVR-NET-IO/netcon“ in den Dateien „netfind.c“ und „netfind.h“, diese ist vollständig getestet und funktionsfähig.

3.11 netcon Protocol

Das letzte für diese Diplomarbeit entwickelte Protokoll dient Dazu, es dem netcond-Daemon zu ermöglichen die Messwerte der Module einzuholen. Damit diese Messwerte auch sicher ankommen, basiert dieses Protokoll auf TCP. Dabei lauscht das Netzwerkmodul am TCP Port 50003 auf eingehende Verbindungen, kommt nun eine TCP-Verbindung zu Stande so kann der netcond-Daemon über das GET-Kommando verschiedene Daten des Moduls abfragen. Ein GET-Kommando sieht folgendermaßen aus:

```
GET <key>\r\n
```

Dabei handelt es sich um eine ASCII-Zeichenkette, wobei \r und \n, das Carriage-Return sowie das Line-Feed ADCII-Zeichen repräsentieren. Das Modul antwortet nun im Erfolgsfall mit:

```
OK\r\n
```

```
<value>\r\n
```

und im Fehlerfall mit:

```
ERROR\r\n
```

Für den <key>Wert gibt es nun eine Liste von verschiedenen Parametern, welche abgefragt werden können.

```
NAME        Fragt den Namen des Moduls ab

PLACE       Fragt den Standort des Moduls ab

UPTIME      Fragt die Laufzeit des Moduls in 10ms ab,
            dieses Mal als ASCII-Zeichenkette

DEVICECOUNT Fragt die Anzahl der Sensoren ab, welche
            für das Gerät vorhanden sind. Die
            Nummerierung dieser Erfolgt jedoch von 0.

TYPE ##     Fragt den Typ für den Sensor ## ab, dies
            soll es ermöglichen, bei der Ausgabe nach
            Temperatursensoren, Luftdrucksensoren, etc.
            zu filtern. Der Typ ist eine Zahl von
            00-FF, jedoch wurden noch keine Typen, außer
            00 für Spannung, definiert.

DTYPE ##    Fragt das Format für den Wert von Sensor ## ab.
            Dieses gibt an, in welchem Format die Werte
            geliefert werden. Mögliche Formate sind dafür:
```

```
                h - Hexadezimal
                i - Dezimal
                f - Fließkommazahl

MIN ##      Fragt den Minimalwert für den Sensor ## ab,
            dies wird benötigt um eine Balkendarstellung
            zu ermöglichen.

MAX ##      Fragt den Maximalwert für den Sensor ## ab.

MAX         Liefert die 6-Byte lange MAC-Adresse des Moduls
            im Format XX:XX:XX:XX:XX:XX.

VALUE ##    Liefert den letzten Messwert des Sensors ## in
            dem entsprechenden Format.
```

Eine beispielhafte Kommunikation könnte folgendermaßen aussehen, wobei D der netcond-Daemon ist und M das Modul:

```
D: GET NAME\r\n
M: OK\r\n
   Testmodul\r\n

D: GET DEVICECOUNT\r\n
M: OK\r\n
   4\r\n

D: GET TYPE 0\r\n
M: OK\r\n
   00\r\n
```

```
D: GET DTYPE 0\r\n
M: OK\r\n
   f\r\n

D: GET VALUE 0\r\n
M: OK\r\n
   4.75\r\n
```

Da sich im Betrieb außer dem Messwert diese Parameter nicht ändern werden, empfiehlt es sich, um die Belastung zu reduzieren, die Parameter für Name, Standort, Datentyp, etc. nur am Anfang bei der Initialisierung abzufragen. Lediglich den VALUE-Parameter muss man zwangsweise immer wieder für alle Sensoren abrufen. Eine vollständige Implementierung dieses Protokolls findet sich im Ordner „netcon Module/Pollin AVR-NET-IO/netcon“ in den Dateien „netcon.c“ und „netcon.h“, diese wurde während der Entwicklung ausführlich getestet und kann als stabil angesehen werden. Für die Zukunft ist eventuell noch ein SET Kommando vorgesehen um zum Beispiel ein Licht ein- oder auszuschalten.

3.12 Kompilieren der Quelldateien

Eine bereits kompilierte Version der Quelldateien findet sich in den Unterordnern von „netcon Module/Pollin AVR-NET-IO“ im Unterordner „bin“, dieses Hexfile kann dann beispielsweise mit avrdude auf den AVR mit dem folgenden Befehl auf der Kommandozeile gebrannt werden:

```
avrdude.exe -p atmega32 -P com3 -c stk500v2 -U flash:w:main.hex
```

Der COM-Port sowie eventuell der Programmer-Typ (hier STK500v2) müssen entsprechend angepasst werden.

Will man Veränderungen an den Quelltexten vornehmen, so muss man diese neu kompilieren, hierbei kommen sogenannte Makefiles zum Einsatz, dafür sind jedoch einige Vorbereitungen nötig, welche im Folgenden erläutert werden.

3.12.1 Installation der AVR-Toolchain

Die AVR-Toolchain kommt vom Atmel und liefert einen vollständigen C-Compiler, sowie die grundlegenden Bibliotheken zur AVR-Programmierung. Die Installationsdatei „avr-toolchain-installer-3.3.0.710-win32.win32.x86.exe“ befindet sich auf der CD, mit dieser Version der Toolchain wurden die Quelltexte erfolgreich kompiliert, andere Versionen wurden nicht getestet. Es empfiehlt sich die Toolchain in des Standardverzeichnis „C:/Program Files/Atmel/AVR Tools/AVR Toolchain/“ zu installieren.

3.12.2 Avrdude installieren

Avrdude ist ein Konsolenprogramm, welches es ermöglicht die erzeugten Hexfiles mithilfe eines Programmers auf Mikrocontroller zu programmieren. Damit dieser Schritt später komfortabler abläuft sollte die Dateien „avrdude.exe“ und „avrdude.conf“ aus dem „avrdude.zip“ Archiv in den bin-Ordner der Toolchain Installation kopiert werden. Hat man bei der Installation den Standardpfad behalten, so lautet der sollte die „avrdude.exe“ folgenden Pfad besitzen: „C:/Program Files/Atmel/AVR Tools/AVR Toolchain/bin/avrdude.exe“

3.12.3 Installation von Cygwin

Da die Buildvorgang hier auf sogenannten Makefiles basiert, wird ein Makefile-Interpreter, das Programm make, benötigt, dieses wird mit Cygwin mitgeliefert. Cygwin selber bietet eine Linuxähnliche Arbeitsumgebung auf einem Terminal unter Windows. Die Cygwin Installationsdatei findet sich entweder auf der CD, unter dem Namen „cygwin-setup.exe“, oder kann direkt von der Cygwin-Webseite http://www.cygwin.com/ heruntergeladen werden. Die Installation ist selbsterklärend, jedoch sollte man bei der Paketauswahl das Paket „make“ installieren, dies enthält das benötigte Programm. Nach der Installation kann man ein Cygwin-Terminal starten, es sollte jedoch hierbei Acht auf die Pfade gelegt werden, will man zu einem Windows-Pfad navigieren, so muss man den Laufwerksbuchstaben nicht als „X:“ angeben sondern als „/cygdrive/X/“, weiterhin werden Ordner in den Pfaden wie bei Linux üblich durch ein „/“ getrennt.

Also Beispiel sei hier folgender Windows Pfad gegeben:

```
C:\Program Files\Atmel\AVR Tools\AVR Toolchain\bin
```

dieser Pfad würde in Cygwin der folgenden Schreibweise entsprechen:

```
/cygdrive/C/Program Files/Atmel/AVR Tools/AVR Toolchain/bin
```

daher der Befehl um im Cygwin-Terminal in diesen Ordner zu wechselnd würde folgendermaßen lauten:

```
cd /cygdrive/C/Program Files/Atmel/AVR Tools/AVR Toolchain/bin
```

3.12.4 Kompilieren

Hat man bei der Installation der AVR-Toolchain einen anderen Pfad angegeben, so muss man diesen jetzt im entsprechenden Makefile des Projektes anpassen, der Pfad muss in der Zeile 41 entsprechend geändert werden, dabei muss dieser in der Cygwin-Schreibweise angegeben werden. Weiterhin kann man in der Zeile 277 und 280 dem Programmer und den COM-Port für avdude anpassen. Sind diese Anpassungen vorgenommen worden, so kann man nun mit dem Cygwin-Terminal in den Projektordner navigieren und mit:

```
make
```

den Kompiliervorgang starten. Dabei entsteht eine „main.hex“ Datei, welche nun zum Beispiel mit avrdude durch folgenden Aufruf in den Mikrocontroller programmiert werden kann:

```
make program
```

Am Schluss kann man noch die temporären und überflüssigen Dateien mit einem Aufruf von:

```
make clean
```

löschen, dadurch werden alle Dateien (inklusive Hexfile) gelöscht, welche beim Kompilieren erzeugt worden sind.

3.13 Inbetriebnahme

Nachdem das Programm in den AVR gebrannt worden ist, kann nun das Pollin-AVR-NET-IO in Betrieb genommen werden. Hat man das Projekt „netcon Module/Pollin AVR-NET-IO/netcon“ kompiliert und in den AVR Programmiert, so muss man nur mehr das Netzwerkkabel an das Netzwerk anschließen und das Kit mit einer 9V Versorgung in Betrieb nehmen. Das Kit fordert nun die IP-Adresse per DHCP an und ist dann bereit Kommandos vom netcond-Daemon zu erhalten. Es stehen dann die Werte der internen ADCs der Kanäle 0 und 1 zur Verfügung.

Hat man hingegen das Projekt „netcon Module/Pollin AVR-NET-IO/netcon_ser“ auf den AVR programmiert, so muss man an den Pins RXD (Pin 14) und TXD (Pin 15) des ATmega32 ein entsprechendes RS232 Gerät gehängt werden, welches sich an das netcon Serial Protocol hält.

4 Software [Lipp]

Das Kapitel beschreibt das Softwareinterface zwischen den Modulen und den Applikationen zur Anzeige der Moduldaten, um eigene Anwendungen, wie Webseiten oder Smartphone-Apps zu realisieren. Als Anwendungsbeispiel wurde bereits eine Website implementiert, die für den eigenen Gebrauch herangezogen werden kann.

4.1 Aufgaben

Die Software, genauer gesagt **netcond**, ist ein sogenannter Daemon (Hintergrundprozess), der im Allgemeinen die Module im Netzwerk verwaltet, indem er ihre Messwerte aufzeichnet, die über ein spezielles Interface abgefragt werden können. Auch modulseitig kommt ein eigens entwickeltes Protokoll zum Einsatz, das bereits im vorangegangenen Kapitel erklärt wurde. Wenn sich alle Module nach dieser Spezifikation verhalten, werden sie auch von netcond als solche erkannt.

4.2 Java

Die Anwendung wurde in Java geschrieben, um Plattformunabhängigkeit zu gewährleisten. Damit ist der Einsatz auf jeder Plattform möglich, die eine Java Runtime Environment (JRE) in Version 6 oder höher zur Verfügung stellt, egal ob PC, Server oder Embedded System. Die Installation dieser Laufzeitumgebung ist im Abschnitt Installation genau beschrieben.

4.3 Hardwareanforderungen

Netcond wurde für hardwareschwache Computer konzipiert. Demnach steht dem Einsatz auf eingebetteten Systemen nichts im Wege, Voraussetzung ist ausschließlich, wie bereits erwähnt, die Verfügbarkeit der JRE.

Da der Daemon im Hintergrund läuft, muss das Betriebssystem über keine grafische Oberfläche verfügen. Netcond kann auf einem Server/Embedded System ohne Monitor über zum Beispiel SSH ausgeführt und gewartet werden.

Empfohlende Hardware

- 1 GHz CPU
- 128 MB RAM (nur für netcond)
- Netzwerkkarte
- Textbasiertes Betriebssystem mit SSH-Zugang

Soll eine eigene Website oder die dieser Diplomarbeit für die Anzeige der Moduldaten genutzt werden, ist weiters ein PHP-fähiger Webserver, wie zum Beispiel Apache erforderlich. Wenn dieser auf der selben Hardware laufen soll, wird zusätzlich Arbeitsspeicher benötigt. Die Installation eines Webservers wird im Abschnitt Installation für die gängigen Betriebssysteme beschrieben.

4.4 Funktionsweise

Grunsätzlich ist die Funktionsweise des Java-Daemons netcond in einem Absatz beschrieben:

Zuerst sucht der Main-Tread nach dem Start alle paar Sekunden das Netzwerk nach verbunden Modulen ab und halt diese in einer Liste. Für jedes dieser Module wird ein neuer Programmfaden erzeugt, der ständig Messdaten abfragt und diese speichert. Zusätzlich startet der Daemon einen weiteren Subprozess, die Schnittstelle, über die eine Anwendung - in diesem Fall z.B. ein Webserver - die Daten abfragen kann. Verbindet sich ein Webbrowser zu diesem Webserver, weißt ein PHP-Script den Daemon an, die aktuellen Daten zu übermitteln. Diese werden dann auf der Website angezeigt. Dieses Prinzip ist nocheinmal in Abb. 11 grafisch verdeutlicht.

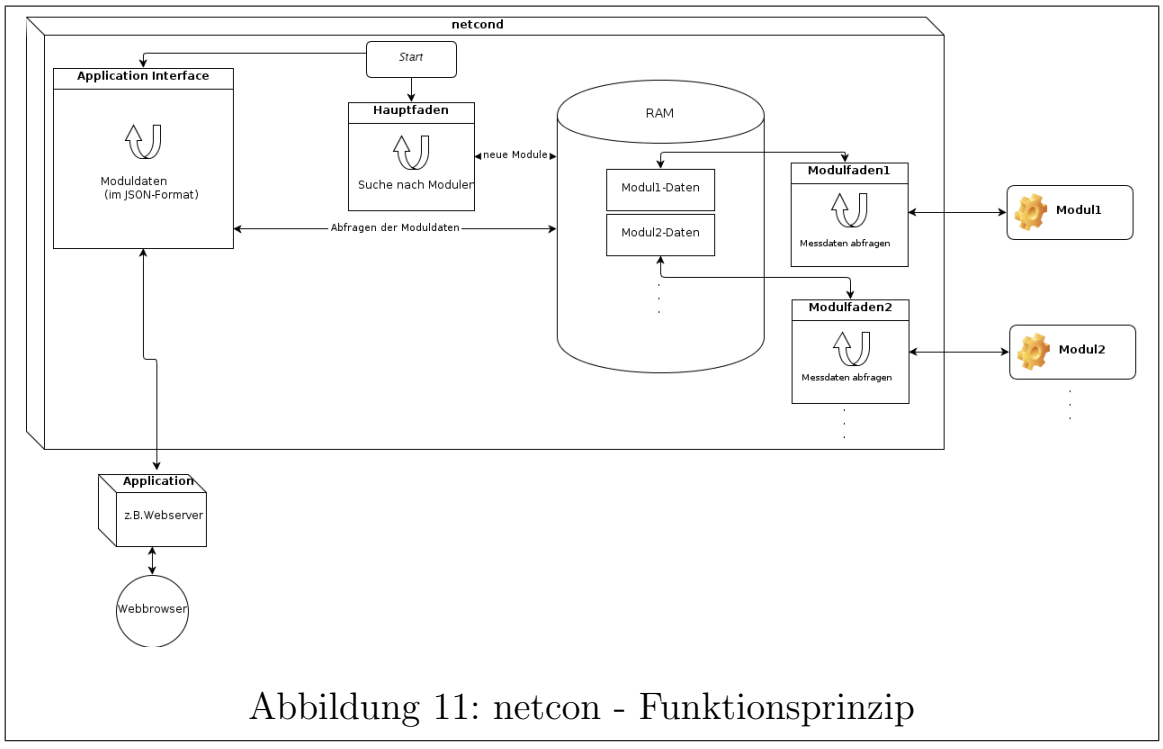


Abbildung 11: netcon - Funktionsprinzip

4.5 Aufbau

Nun folgt die Beschreibung des Quellcodes wobei Kentnisse in Java erforderlich sind. Erläutert werden dabei nicht die Programmzeilen im einzelnen, sondern für das Verständnis relevante Programmteile. Die Verwendung der netcon API Schnittstelle für die Programmierung eigener Anwendungen ist im nächsten Kapitel beschrieben:

Der Sourcecode besteht aus einigen Java-Klassen bzw. -Dateien, die in Paketen organisiert sind. Auf oberster Ebene, sind das **lib**, die Programmbibliothek und **program**, die Programmthreads. Diese sind je nach Funktion wiederum in mehrere Unterpakete unterteilt (siehe Abb. 12).

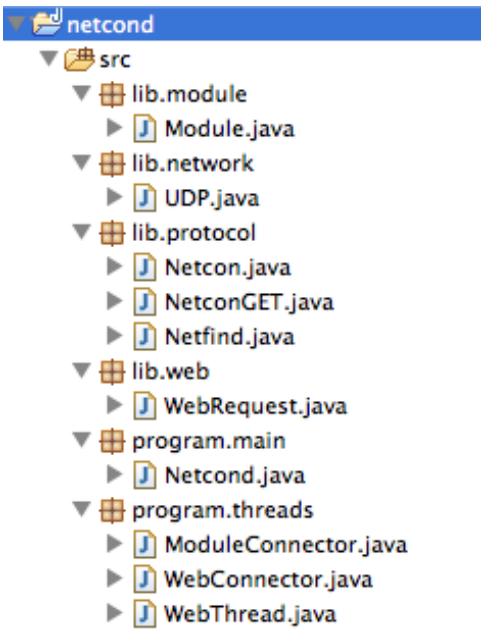
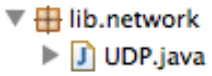


Abbildung 12: netcond - Übersicht Sourcecode

lib.network



Dieses Paket enthält die statische Klasse UDP.java, die rudimentäre UDP-Funktionen zum Senden und Empfangen von Daten- und Broadcastpaketen bereitstellt.

Die Funktion sendPacket dient dazu ein einfaches UDP-Paket zu verschicken. Dazu nimmt sie die zu sendende Nachricht *msg*, die Zieladresse *dstAdr* (vom Typ InetAddress), Zielport *dstPort* und den Quellport *srcPort* als Parameter. Die Angabe des Quellports ist erforderlich, damit der Empfänger auf anwendungsspezifische Pakete filtern kann.

```
public static void sendPacket
(String msg, InetAddress dstAdr, int dstPort, int srcPort)
```

Die Funktion receivePacket empfängt ein UDP-Paket. Dazu benötigt sie als Parameter den Zielport des zu empfangenden Pakets lstPort und den Timeout - also die Zeit bis der Empfangversuch abgebrochen wird - in Millisekunden und liefert das empfangene Paket in Form eines DatagramPacket zurück.

```
public static DatagramPacket
receivePacket(int lstPort, int timeout)
```

Für das Versenden von Broadcast-Paketen wird die überladene Funktion sendBroadcast verwendet, die als Parameter entweder einen String, oder ein Byte-Array bekommt, sowie den Ziel- und Quellport des Pakets.

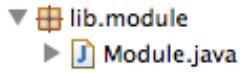
```
public static void
sendBroadcast(String msg, int dstPort, int srcPort)

public static void
sendBroadcast(byte msg[], int dstPort, int srcPort)
```

Die letzte Funktion der UDP-Klasse receiveBroadcast empfängt Broadcast-Pakete, wobei sie den Quellport des Broadcast-Pakets als Parameter bekommt und das empfangene Paket als DatagramPacket zurückliefert.

```
public static DatagramPacket receiveBroadcast(int lstPort)
```

lib.module



Das Paket lib.module enthält die Klasse Module.java, dessen Instanzen reale Module im Netzwerk abbilden. Diese bestehen aus folgenden Attributen:

```
private String hostname;
private String location;
private String ip;
private int port;
private String mac;

private String uptime;

private int devicecount;
private int type[];
private String value[];
private String minValue[];
private String maxValue[];
private String dtype[];

private ModuleConnector thread;
```

Ein Modul besitzt demnach einen Hostnamen, wie z.B. *Temperaturmodul*, einen Standort, IP, Port und MAC. Die Uptime zeigt an, wie lange das Modul bereits online ist. Wie bereits im vorherigen Kapitel erwähnt kann ein Modul aus mehreren sogenannten Devices bzw. Sensoren bestehen, deren Anzahl in der Instanzvariablen devicecount gepspeichert werden. Je nachdem wie groß diese Zahl ist (max. 9) enthält das type-Array die Messtypen und das value-Array die aktuellen Messwerte der einzelnen Devices. Die zwei weiteren Felder minValue[] und maxValue[] enthalten die minimalen und maximalen Messwerte der Devices während dtype[] anzeigt in welchem Datentyp die Messgrößen vorliegen.

Die letzte Variable thread ist an dieser Stelle wahrscheinlich noch nicht ganz verständlich, sei vollständigkeitshalber aber trotzdem erwähnt: Um die Messwerte der Devices abzufragen wird nach Erstellung der Instanz - also nach Auffinden eines Moduls - ein Thread gestartet, der über TCP mit dem Modul kommuniziert. Dazu benötigt jedes Modulobjekt seine eigene Threadvariable vom Typ ModuleConnector. Näheres dazu im Abschnitt program.threads.

Zusätzlich zu den Getter- und Setter-Methoden für jede Variable besteht die Klasse Module noch aus folgenden Methoden:

```
public void startThread()
```

Diese Methode ein Objekt vom Typ ModuleConnector an und startet damit den Modulthread um die Messwerte abzufragen. Sozusagen eine Setter-Methode für die Instanzvariable thread.

```
public JSONObject getJSON()
```

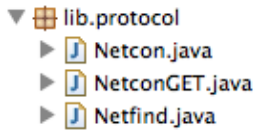
Die Methode getJSON() liefert die Moduldaten, also die Werte der Instanzvariablen verpackt in einem JSONObject zurück.

JSON ist ein ein kompaktes, einfach lesbares Datenformat, das speziell für den Datenaustausch zwischen Anwendungen erfunden wurde. Der Aufbau eines JSON-Objekts ist im Kapitel netcon API genau beschrieben, da es die Grundlage für die Schnittstelle zwischen netcond und anderen Anwendungen bildet. Hier sei noch erwähnt, dass Java in der Grundausstattung keine JSON-Klassen mitbringt und diese Funktionalität mit Bibliotheken von Drittanbietern nachgerüstet werden muss. In diesem Fall wurde JSON.simple, ein einfaches Toolkit für JSON eingebunden, um JSON-text zu encodieren und decodieren.

Jede (nichtstatische) Klasse besitzt einen Konstruktor, auch die Module-Klasse. Der Konstruktor wird bekanntlich beim Erstellen eines Objektes aufgerufen. Dies geschieht beim Auffinden eines Moduls im Hauptthread. Dazu mehr im Abschnitt program.main. Beim Aufruf werden die Werte aller Parameter an die erstellen Instanzvariablen übergeben, während devicespezifische Daten (devicecount, type[], value[], minValue[], maxValue[], dtype[]) und die uptime zunächst auf Standardwerte initialisiert (0,NULL) und erst später durch den Modulthread gesetzt werden.

```
public Module
(String hostname, String location, InetAddress ip,
int port, String mac)
```

lib.protocol



Im Kapitel Hardware wurden bereits die definierten Protokolle für das Auffinden und Abragen von Modulen im Netzwerk beschrieben. Dazu enthält das Paket lib.protocol die statischen Klassen Netfind.java und Netcon.java.

Die Klasse Netfind enthält zwei Funktionen, netfind() und netdiscover(). Netfind() erzeugt ein Byte-Array, das die nach Protokolle definierten Daten eines Netfind-Pakets zum Auffinden der Module enthält. Im Protokoll vorgesehen, aber nicht weiter genutzt, ist auch ein MAC-Filter, der im Programm auf den Standardwert FF:FF:FF:FF:FF:FF gesetzt wurde.

Der zweiten Funktion netdiscover() kann ein DataPacket übergeben werden, das zuallererst auf Richtigkeit des Protokolls geprüft wird. Danach werden die Moduldaten entpackt und eine neues Modul - also eine Instanz der Klasse Modul - erzeugt und zurückgeliefert.

```
public static byte[] netfind()
public static Module netdiscover(DatagramPacket packet)
```

Das netcon-Protokoll dient zur Kommunikation zwischen netcond und den Modulen, vorallem um sich veränderliche Daten, wie Messwerte aufzuzeichnen. Dazu besitzt die Klasse Netcon die Funktion netcon(), die den gewünschten Befehl vom enum-Typ NetconGET und die Devicenummer als Parameter erhält und ein Byte-Array mit den Daten zur Anfrage an ein Modul zurückliefert.

```
public static byte[] netcon(NetconGET c, String device)
```

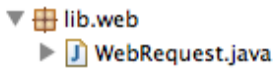
Die Auswahlmöglichkeiten für den enum-Parameter sind in der enum-Klasse NetconGET definiert. Einfacher gesagt lassen sich all diese Daten von einem Modul über das Netcon-Protokoll abrufen.

```
public enum NetconGET {

    uptime, name, devicecount, devicetype,
    value, min, max, dtype;

}
```

lib.web

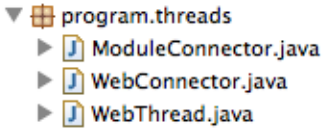


Im Paket lib.web enthalten ist die statische Klasse WebRequest.java, die Funktionen zur Verarbeitung der Anfragen von anderen Anwendungen bereitstellt. Dazu sind zwei Funktionen vorgesehen get() und set(), wobei letztere nur für die spätere Implementierung eines Steuernetzwerkes deklariert wurde und keinerlei Funktionen zur Verfügung stellt.

```
public static JSONObject get(String request)
```

Als Parameter bekommt die Funktion get() die Anfrage *request* einer Drittanwendung (z.B. einer Website) als String und liefert ein JSONObject zurück, das die Daten für die weitere Verarbeitung oder Anzeige enthält. Dazu legt die Funktion eine Modulliste im JSON-Format an, geht die Modulliste durch, ruft für jedes Modul die bereits beschriebene Instanzmethode getJSON() auf und fügt die Daten der JSON-Modulliste nacheinander hinzu. Ist die Modulliste leer gibt die Funktion null zurück.

program.threads



Das Paket program.threads enthält drei Klassen, die jeweils einen anderen Thread definieren.

Eine Thread der Klasse ModuleConnector wird wie bereits beschrieben, für jedes hinzugefügte Modul gestartet, um mit dem realen Modul zu kommunizieren und Messwerte abzufragen. Den Ablauf zeigt Abb. 13.

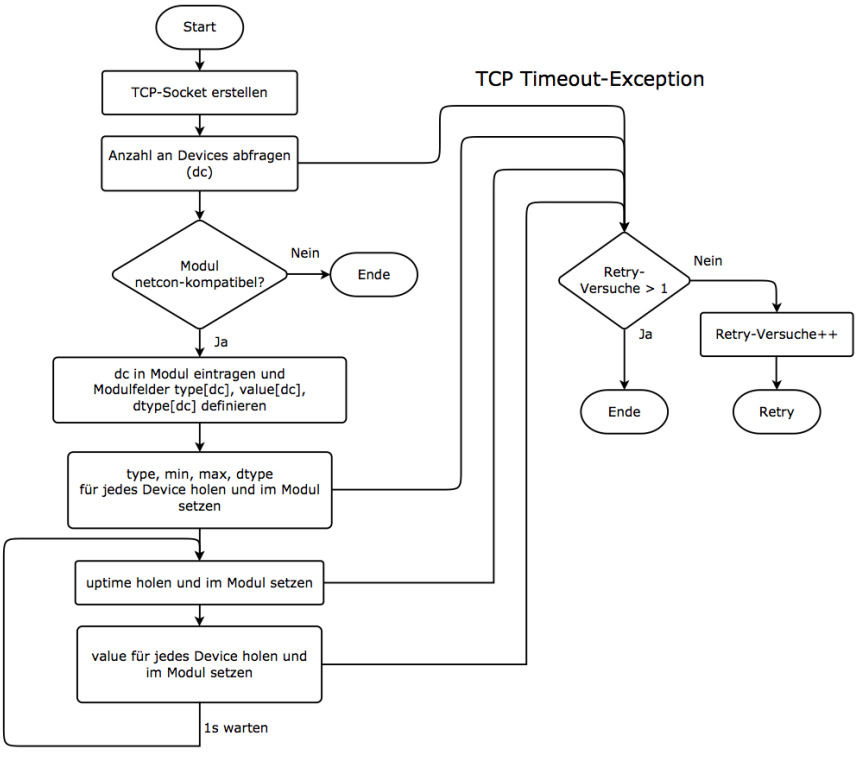


Abbildung 13: ModuleConnector

Nachdem also ein Modul im Netzwerk gefunden und ein Objekt erstellt wurde, wird sein ModuleConnector-Thread gestartet. Dieser erstellt einen TCP-Socket (Port 50003) mit den javainternen Bibliotheken und versucht anschließend per TCP die Anzahl der Devices abzufragen. Erhält er vom realen Modul keine netcon-konforme Antwort, wird er beendet. Im anderen Fall definiert er die Instanzvariable devicecount und erstellt die Felder der Instanz, da jetzt die Anzahl der Devices bekannt ist. In der nächsten TCP-Anfrage holt der Thread type, min, max und dtype für jedes Device und weist sie den Instanzvariablen zu. Danach beginnt eine Endlosschleife, in der die uptime und value (Messwert) geholt und gesetzt werden, wobei ein Delay von einer Sekunde eingebaut wurde. Alle TCP-Anfragen werfen bei längerer Wartezeit eine Timeout-Exception. Eine Variable für die Retry-Versuche wird dann um 1 dekrementiert und ein weiterer Versuch gestartet, vorausgesetzt die Variable war nicht 0. Die Retry-Versuche sind je TCP-Anfrage gültig, deshalb wird die Variable auch vor jeder Anfrage neu definiert. Wird der ModulThread im Programmblauf an einer Stelle beendet, wird auch das Modulobjekt gelöscht.

Der WebConnector-Thread bildet die Schnittstelle zu Drittanwendungen, wobei er für jede Anfrage einen Thread der Klasse WebThread startet und die Daten des Requests übergibt, um gleich darauf auf die nächste Anfrage zu warten. Die WebThreads bearbeiten die Anfragen mithilfe der Funktionen aus dem lib.web Paket, leiten die Antwort im JSON-Format an die Drittanwendung weiter und werden daraufhin beendet. Das Ablaufdiagramm in Abb. 14 macht diesen Vorgang verständlicher.

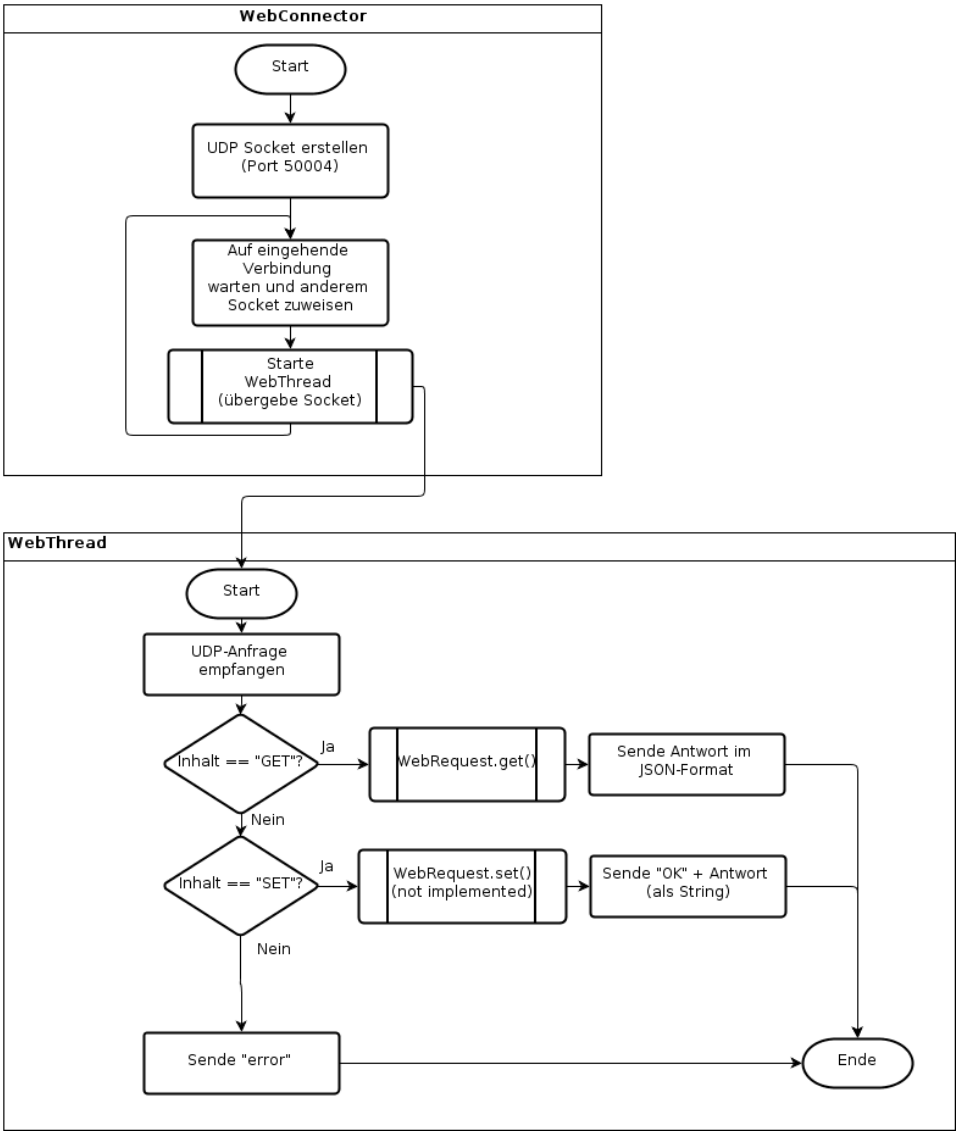
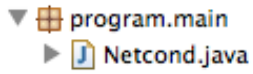


Abbildung 14: WebConnector

Der WebConnector-Thread wird beim Start des Daemons automatisch gestartet. Dabei erstellt dieser einen UDP-Socket (Port 50004), um auf eine eingehende Verbindung zu hören, die dann einem weiteren Socket zugewiesen wird. Für die weitere Bearbeitung der Anfrage wird nun ein WebThread gestartet, wobei ihm der Socket übergeben wird. Während der WebConnector jetzt wieder auf eine eigehende Verbindung wartet, führt der WebThread folgende Schritte durch:

Er empfängt die eigentliche UDP-Anfrage der Anwendung. Dabei versteht er alle Requests, die mit GET oder SET beginnen, sonst sendet er ein "error" zurück. Für die verstandenen Anfragen werden die bereits erwähnten Funktionen get() und set() der WebRequest-Klasse aufgerufen, die entsprechende Antworten zurückliefern. Diese werden per UDP an die Anwendung gesendet.

program.main




```
// Modulliste
public static List<Module> moduleList =
new ArrayList<Module>();

public static void main(String[] args) {
 ...
/*1*/   UDP.sendBroadcast(Netfind.netfind(), 50000, 50001);
 ...
/*2*/   recv = UDP.receivePacket(50001, 2000);
 ...
/*3*/   module = Netfind.netdiscover(recv);
 ...
}
```

Die Klasse Netcond.java bildet den Main-Thread des Daemons, der nach dem Start zuerst ausgeführt wird. Dabei wird, wie in Abb. 15 erkennbar, zunächst der WebConnector-Thread gestartet. Daraufhin folgt eine Endlosschleife, die zuerst einen UDP-Broadcast - mithilfe der bereits beschriebenen UDP-Funktionen und netfind() - von Port 50000 an Port 50001 aussendet (/*1*/) und danach in einer weiteren Endlosschleife 2 Sekunden lang auf die Antworten der Module wartet. Dazu wird wieder eine UDP-Funktion aus dem Paket lib.network benutzt (/*2*/).

Diese Antworten werden der Funtion netdiscover() aus dem Paket lib.protocol übergeben, die entweder ein erstelltes Objekt der Modulklasse, oder null zurückliefert, wenn die Antwort nicht dem Protokoll entsprach (/*3*/). Wurde ein Modulobjekt zurückgegeben, wird die Modulliste - eine ArrayList die die Modulobjekte hält und in der Netcond-Klasse deklariert wurde - daraufhin dursucht, ob sie das gefunde Module bereits enthält. Ist dies nicht der Fall wird der ModuleThread des Moduls gestartet und das Modul zur Modulliste hinzugefügt.

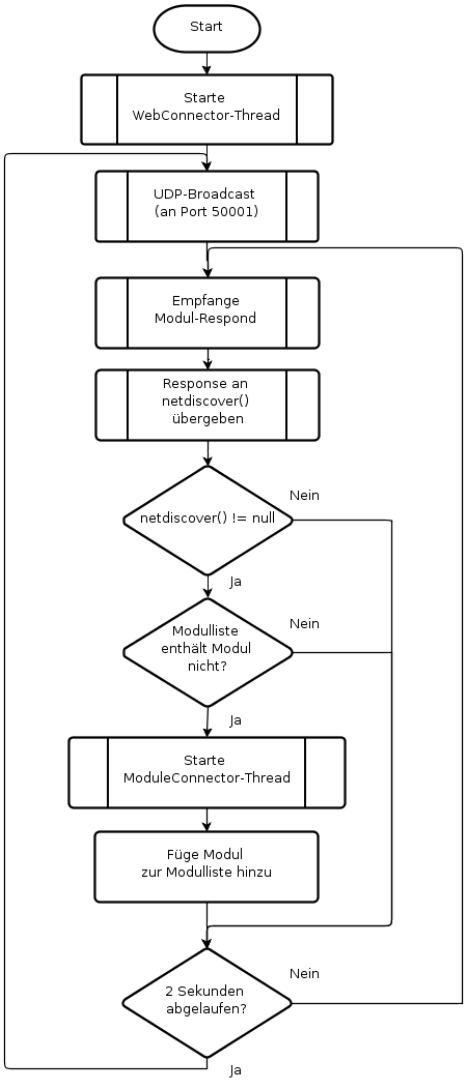


Abbildung 15: WebConnector

Vergleich der Module

Damit die Instanzmethode contains() der ArrayList auch feststellen kann, ob ein gefundenes Modul mit einem in der Liste enthaltenen Modul übereinstimmt, mussten die Object-Methoden equals() und hashCode() in der Klasse Module überschrieben werden. In diesem Fall stimmen zwei Module überein, wenn ihr Hostname gleich ist.

```
@Override
public boolean equals(Object o) {

Module mod = (Module) o;

if (getHostname().matches(mod.getHostname()))
return true;

return false;
}

@Override
public int hashCode() {

return port * super.hashCode();
}
```

Warten im Thread

An mehreren Stellen im Programmcode kommt es vor, dass in einem Thread gewartet werden muss. In einer Multithreading-Anwendung würde bei alleiniger Verwendung der gewohnten while-Schleife die Rechenzeit im Thread draufgehen. Stattdessen werden die Threads mit der Funktion Thread.sleep(time) schlafen gelegt und die Rechenzeit kann für die anderen Threads bzw. dem Betriebssystem genutzt werden. Um beispielsweise eine Sekunde zu warten dient folgendes Konstrukt:

```
long startTime = System.currentTimeMillis();

while ((System.currentTimeMillis() - startTime) < 1000) {
    try {
            Thread.sleep(time);
        }
}
```

Synchronisation

Greift ein Thread auf Daten zu, die auch von anderen Threads genutzt werden, muss an der Stelle im Code eine Schreibsperre für diese gesetzt werden, bis der eigene Schreibvorgang abgeschlossen ist. Auch in den Threads von netcond erfolgt ein Mehrfachschreibzugriff und zwar auf die Modulliste. Der Main-Thread fügt Module hinzu, während die einzelnen ModulThreads Daten in die Instanzvariablen der Module schreiben und auch Module löschen, am Ende ihrer Lebenszeit. Deshalb wurde jede Stelle im Programmcode, an der in die Modulliste geschrieben wird, durch ein synchronized()-Block eingehüllt.

```
synchronized ( Netcond.moduleList ) {
    Netcond.moduleList.remove(module);
}
```

4.6 netcon API

Das netcon Application Programming Interface (API) ermöglicht, wie oftmals erwähnt, die Entwicklung eigener Anwendungen zur Anzeige der Moduldaten. Diese Anwendung kann beispielsweise eine Website (inkl. Webserver), oder ein Smartphone App sein. Egal wie diese auch aussehen mag, vorausgesetzt ist, dass sie UDP-Pakete versenden und empfangen kann.

In diesem Abschnitt wird nun der Grundstein für die Erstellung eigener Anwendungen gelegt und das dazu notwendige API erläutert, wobei als Programmbeispiel auch die im Rahmen dieser Diplomarbeit programmierte Website herangezogen werden. Näheres dazu im Abschnitt Website.

Request

Wurde der Daemon netcond gestartet - näheres dazu im Abschnitt Installation - wird sofort ein Thread ausgeführt, der auf Anfragen von Anwendungen wartet. Diese Anfrage muss ein UDP-Paket, gerichtet an die IP-Adresse des netcon-Servers und Port 50004 sein. Schickt man nun dieses Paket, noch mit undefinierten Inhalt an den Server, erfolgt die Konsolenausgabe **E: Incomprehensible web request** inkl. der IP von der die Anfrage kam. Damit kann schon einmal getestet werden, ob das Paket beim Server ankam.

Respond

Die UDP-Antwortpakete, die der netcon Daemon zurücksendet, sind bereits an den Port gerichtet, von der die Anfrage kam. In der Anwendung muss also nach dem Aussenden der Anfrage, auf ein UDP-Paket gewartet werden, das die Moduldaten enthält.

Moduldaten empfangen

Da zurzeit nur ein Messystem implementiert wurde, existiert auch nur ein Befehl, mit dem man eine Liste aller Module inkl. ihrer Daten erhält und zwar im JSON-Format. Dazu muss der Inhalt des UDP-Pakets folgendermaßen aussehen:

```
GET\nlist\n
```

JSON

Das Antwortpaket besteht ausschließlich aus einem leserlichen JSON-Objekt, aus dem entweder mit eigens programmierten Funktionen oder mithilfe diverser Bibliotheken die gewünschten Daten entpackt werden können. Da hier nicht auf alle verfügbaren Bibliotheken für alle Programmiersprachen eingegangen werden kann, folgt die Beschreibung des Aufbaus eines JSON-Objekts. Damit sollte es möglich sein eigene Funktionen zu programmieren. Aber auch bei der Verwendung von bereits verfügbaren Bibliotheken empfielt es sich, den Aufbau zu verstehen, um die Funktionen richtig einzusetzen.

JSON wurde entwickelt, um den Datenaustausch zwischen Anwendungen möglichst einfach zu gestalten und das, in einer für den Menschen gut lesbaren Form.

Ein JSON-Objekt liegt demnach in Reintext vor, aber in genau definierter Anordnung der zu vermittelnden Daten, wobei standardmäßig die UTF-8 Kodierung verwendet wird. Es beginnt und endet mit einer geschwungenen Klammer und kann verschiedene Elemente (=Werte) enthalten, wobei jedes durch "Titel": (=Schlüssel) eingeleitet wird und mit Beistrich vom nächsten Element getrennt ist. Weiters sind Leerraum-Zeichen beliebig verwendbar.

```
{
  // Zeichenkette
    "Firmenname":"Muster Industries",
  // Boolescher Wert
    "Aktiengesellschaft":false,
  // Nullwert
    "Index":null,
  // Zahl
    "Vermögen":-150.00,
  // Array
    "Inhaber": ["Mustermann","Musterfrau"],
  // Objekt
    "Unternehmen":{
        "Name":"Muster Industries",
        "Mitarbeiterzahl": 100,
        "Mitarbeiter":["Adam","Eva"]
   }
}
```

Wie man am Beispiel der vorherigen Seite erkennen kann, ist es sogar möglich, JSON-Objekte in sich zu verschachteln. Dieser wichtige Aspekt ist bei der Anordnung der Moduldaten zu beachten.

Auch wichtig zu beachten ist, dass bei Verwendung von Funktionen fertiger JSON-Bibliotheken die Daten nicht in der gleichen Reihenfolge ankommen, wie sie hinzugefügt wurden. Es ist also beim Empfänger bzw. der Anwendung beim Auslesen der Daten nach dem Schlüssel zu suchen und von keiner bestimmten Reihenfolge auszugehen. Es sollte demnach immer angenommen werden, dass die Module und ihre Devices in irgendeiner Reihenfolge im JSON-Objekt vorliegen. Außerdem kann die Verwendung von Leerzeichen variieren.

Wird nun ein UDP-Antwortpaket vom Server an die Anwendung ausgesendet, enthält es ein JSON-Objekt, das laut folgender Seite formatiert ist, wenn beispielsweise ein Module mit zwei Devices im Netzwerk gefunden wurde. Konnte der netcon Daemon keine Module im Netzwerk finden, sie das JSON-Objekt folgendermaßen aus:

```
{"modulliste":null}
```

Wurde die Anfrage nicht verstanden, wird mit “error” geantwortet.

```
{
  "modulliste":[
    {
      "name":"AVR-NET-IO-Lipp",
      "location":"Wohnzimmer",
      "ip":"192.168.2.3",
      "port":50000,
      "uptime":"14403",
      "devicelist":[
       {"id":0,
        "type":1,
        "dtype":"f",
        "min":"0.0",
        "max":"5.0",
        "value":"0.11",
        },
        {"id":1,
        "type":1,
        "dtype":"f",
        "min":"0.0",
        "max":"5.0",
        "value":"2.0",
        }
       ]
   } //, weitere Module ...
  ]
}
```

Das JSON-Objekt besteht aus einem Array, das die einzelnen Module trägt. Jedes dieser Module ist wiederum ein JSON-Objekt. Die Modulobjekte bestehen aus den Moduldaten und einem weiteren Array, das die Devices in Form von JSON-Objekten enthält. Ein Device-Objekt besteht aus seiner ID, dem Typ der Messeinheit (z.B. Volt) und des Messwerts (z.B. f für Fließkommazahl), dem minimalen und maximalen Messwert und dem Messwert. Der Typ für die Messeinheit ist in diesem Fall 1, was bedeutet, dass es sich um eine Spannung (V) handelt (von uns so festgelegt).

Die Bedeutung der einzelnen Typen kann beliebig gewählt werden und ist im Kapitel Hardware im Zusammenhang mit dem netcon-Protokoll, zur Kommunikation zwischen den Modulen und dem Daemon, beschrieben. Dort zu finden sind auch nähere Informationen über dtype, den Typ des Messwerts. Diese Daten sollten reichen, um die Anzeige der Moduldaten und Messwerte (fast) unabhängig von der Messgröße zu ermöglichen. Nochmals der Hinweis, das JSON-Objekt ist in den meisten Fällen nicht so geordnet, wie die letzte Seite zeigt. Auch hier wurde im Nachhinein die Reihenfolge im Zuge der Lesbarkeit angepasst.

Bei Problemen mit der Entwicklung eigener Anwendungen und Module ist es empfehlenswert den nächsten Abschnitt Debugging zu lesen, um diverse Fehlerquellen auszuschließen.

4.7 Debugging

4.7.1 Ports

Damit ein reibungsloses Zusammenspiel zwischen den Modulen und dem Daemon bzw. den Anwendungen un dem Daemon stattfinden kann, muss auf die richtige Verwendung der Ports geachtet werden. In einer Liste sind diese nocheinmal zusammengefasst. Diese sollten überprüft werden, bevor mit dem Debugging des netcon-Systems begonnen wird. Abb. 16 zeigt je nach Kommunikation, an welchen Ports die Module liegen müssen bzw. von welchen Ports empfangen werden muss. Analog dazu Abb. 17 für selbst programmierte Anwendungen.

Art der Kommunikation	Modul an Port	Netcond an Port
Netcond sendet UDP-Broadcast	50001	50000
Modul sendet UDP-Antwort	50001	x
TCP Kommunikation	50002	50003

Abbildung 16: Ports für die Kommunikation mit Modulen

Art der Kommunikation	Anwendung an Port	Netcond an Port
Anwendung sendet UDP-Anfrage	x	50004
Netcond sendet UDP-Antwort	50004	x

Abbildung 17: Ports für die Kommunikation mit Anwendungen

4.7.2 Programmausgaben

Netcond ist, wie bereits erwähnt, eine konsolenbasierte Anwendung, die in bestimmten Fällen Textausgaben macht, um Fehler im netcon-System einfach zu entdecken:

```
I: WebConnector started

E: Incomprehensible web request (Request)

E: UDP-response isn't conform to netcon protocol (IP)

I: ModulThread started # (Hostname, IP)

E: Timeout # (Hostname, IP)

E: ModulThread stopped # (Hostname, IP)
```

I: und **E:** geben an, ob es sich um eine Information, oder einen Fehler handelt.

Anfragen von Anwendungen

Wird netcond gestartet erscheint sofort **I: WebConnector started** , nachdem der WebConnector ausgeführt wurde. Diese Information gibt Auskunft darüber, ob das netcon API bereit ist, auf Anfragen von Anwendungen zu hören.

Schickt eine Anwendung eine nicht definierte Anfrage erscheint **E: Incomprehensible web request** inklusive dem fehlerhaften Request.

UDP-Broadcast

Antwortet ein Modul auf einen UPD-Broadcast mit einer nicht nach dem netcon-Protokoll definierten Antwort, wird **E: UDP-response isn't conform to netcon protocol** inklusive der IP des Moduls ausgegeben.

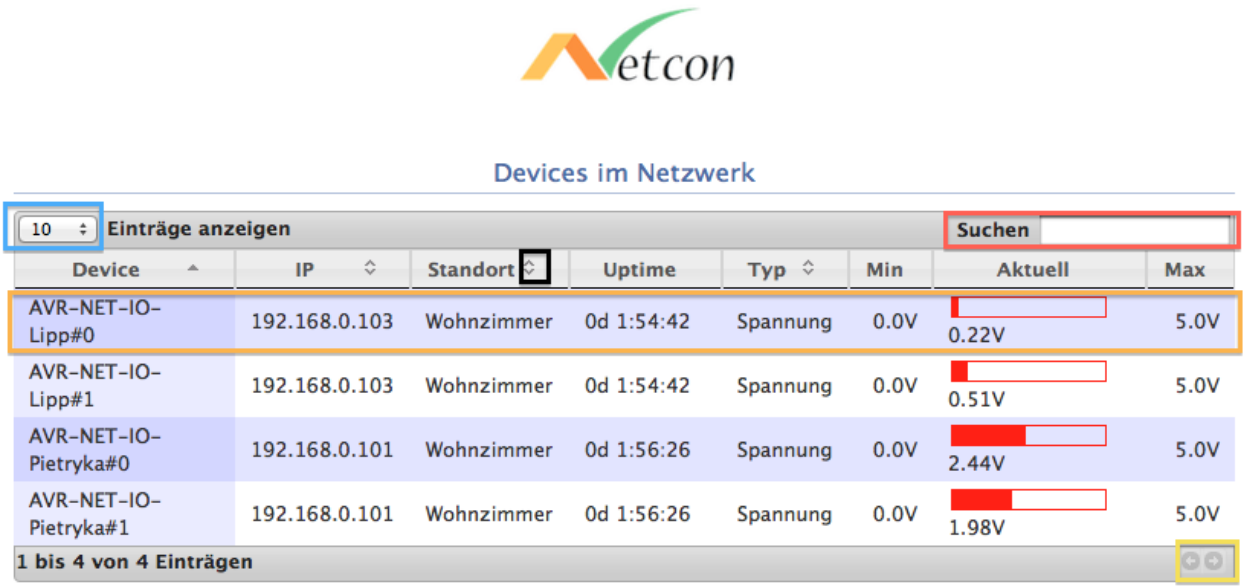
TCP-Kommunikation

Wurde ein Modul zur Modulliste hinzugefügt und der ModulThread gestartet erscheint **I: ModulThread started #** mit Hostnamen und IP des Moduls, wobei # für die Prozess-ID steht.

Antwortet das Modul beispielsweise nicht oder falsch auf die Messwertabfrage, gibt netcond **E: Timeout # (Hostname, IP)** aus. Nach zwei misslungenen Retries erscheint dann **E: ModulThread stopped # (Hostname, IP)** und der ModulThread wird beendet, sowie das Modul aus der Modulliste gelöscht.

4.8 Website

Informationen zur Einrichtung des Webservers für die Verwendung der Website im Kapitel Installation. Wird der Webserver nicht zusammen mit dem netcon Daemon auf dem selben Computer betrieben, muss eine Variable im PHP-Script angepasst werden. Näheres dazu auf den folgenden Seiten.



[ORANGE] Die Anzeige der Moduldaten erfolgt auf der netcon Website in einer Tabelle, wobei für jedes Device eines Moduls ein eigener Eintrag angezeigt wird. Dabei wird an den Hostnamen des Moduls einfach die Devicenummer angehängt.

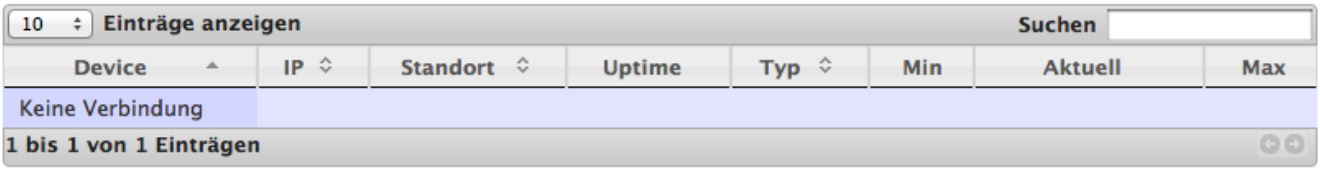
[SCHWARZ] Standardmäßig werden die Einträge nach dem Hostnamen sortiert. Dies kann durch Drücken der entprechenden Buttons für IP, Standort und Typ verändert werden.

[BLAU] Mit einer Dropdown-Liste oben links kann die Anzahl der Einträge pro Seite eingestellt werden.

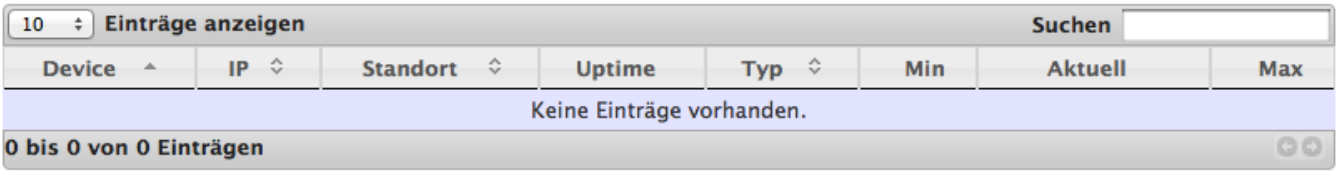
[GELB] Die Seiten können mit den Pfeiltasten unten rechts gewechselt werden.

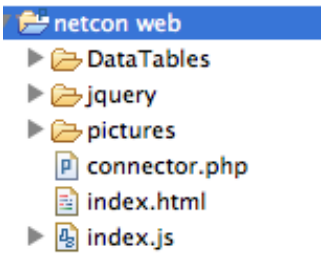
[ROT] Über die Suche oben rechts kann nach allen Ausdrücken gefiltert werden z.B. Spannung oder Wohnzimmer.

Kann keine Verbindung zum netcon Daemon hergestellt werden, weil dieser nicht gestartet wurde, oder die host-Variable im PHP-Script falsch gesetzt ist, sieht die Website folgendermaßen aus:



Wurde die Verbindung hergestellt, aber keine Module im Netzwerk gefunden, macht die Website diese Ausgabe:





Die Grundlage der Website bildet das PHP-Script connector.php, das bei Aufruf den Webserver dazu veranlasst ein UDP-Anfrage an den netcon Deamon zu richten und die Moduldaten im JSON-Format entgegennimmt und ausgibt.

Dazu wird mithilfe der PHP-Funktionen ein UDP-Socket (1) erstellt mit dem Zielport 50004. Trat kein Fehler auf folgt nun die Anfrage GET list (2), dessen Antwort daraufhin (3) eingelesen und ausgegeben (4) wird. Dieses Script liefert lediglich eine Ausgabe der Moduldaten im JSON-Format bzw. “modulliste”:null wenn kein Modul gefunden wurde, oder “error” wenn keine Verbindung zum Daemon hergestellt werden konnte. Zur Verarbeitung und benutzerfreundlichen Anzeige sind noch weitere Schritte notwendig.

Wichtig für die eigene Verwendung der Website ist die Anpassung der $host-Variablen, wenn Webserver und netcon Daemon nicht gemeinsam auf einer Maschine ausgeführt werden. Ist dies der Fall, muss hier “localhos” durch die IP-Adresse des Webservers (unter Anführungszeichen) ersetzt werden.

```
    <?php
    $out = "";
    $host="localhost";
    $port=50004;
    $timeout=30;

(1) $sk=fsockopen($host,$port,$errnum,$errstr,$timeout);

    if(!is_resource($sk)) {
        exit("error");
    }

    else {
(2)     fputs($sk, "GET\n", 4);
        fputs($sk, "list\n", 5);

(3)     $zeichen = fgetc($sk);

        while($zeichen != "\n") {
            $out .= $zeichen;
            $zeichen = fgetc($sk);
        }

(4)      echo $out;
    }

    fclose($sk);
    ?>
```

Die eigentliche Website index.html enthält Befehle für die Einbindung von jQuery (Bibliothek mit erweiterten JavaScript-Funktionen), DataTables (Tabellen-Plugin für jQuery) und einer JavaScript-Datei, die die eigentliche Aufbereitung der Daten übernimmt. Im Body befindet sich lediglich das Logo, der Titel und eine leere Tabelle inkl. Spaltennamen, die von der DataTables-Library benötigt wird.

Wurde die Website - also index.html - fertig geladen, folgt die Ausführung des JS-Scripts index.js, das sofort die Funktion $(document).ready() ausführt, der als Paramater die Definition einer Funktion mit dem auszuführenden Programmcode übergeben wird.

```
$(document).ready(function(){
    updateTable(oTable);
    window.setInterval(updateTable, 1000);
});

function initTable() {
    // ...
}

function updateTable() {
    var oTable = initTable();
    // ...
}
```

In diesem Fall ist das updateTable(), eine im selben Script definierte Funktion. Diese ruft die Funktion initTable() auf, die die Initalisierung einer leeren DataTable übernimmt und dabei ihre Formatierung festlegt (Sortierreihenfolge, Aussehen,...). Dazu nimmt sie die HTML-Tabelle als Grundgerüst, baut darauf die DataTable auf, die noch immer nur aus den Spaltennamen besteht und gibt sie an die Variable oTable zurück.

Danach wird das connector.php Script aufgerufen (1), die Ausgabe in einem JSON-Objekt gespeichert (3) und verarbeitet und für jedes Device ein Eintrag zur DataTable hinzugefügt (4) vorausgesetzt das JSON-Objekt enthält Module. Die Ausgabe des connector.php Scripts wird natürlich schon am Anfang darauf geprüft, ob sie "error" enthält (2).

```
(1)  $.get('connector.php', function(result){

(2)      if(result == "error") {
           //...
           return;
         }

(3)       var json = $.parseJSON(result);

         // Verarbeitung ...

(4)       oTable.fnAddData ( [

           // ...

         ]);
    }
```

Bisher würde die Website einmal die Daten vom Deaemon abfragen und in der DataTable ausgeben. Eine Methode, die den Namen Ajax trägt macht es aber möglich die Moduldaten abzufragen, ohne die Website neu laden zu müssen. Dazu dient die zweite Zeile. Diese ruft die Funktion updateTable() jede Sekunde auf:

```
window.setInterval(updateTable, 1000);
```

Zurzeit versteht die Website nur den Typ 1 (Spannung), da keine anderen Typen definiert wurden. Die Typen können in eigenen Anwendungen beliebig vergeben werden. Neue Typen können den folgenden zwei Funktionen mitgeteilt werden:

```
function getTypeunit(typeNum) {
    switch (typeNum) {
        case 1:
            return 'V';
            break;
    }
}

function getLongType(typeNum) {
    switch (typeNum) {
        case 1:
            return 'Spannung';
            break;
    }
}
```

4.9 Installation

Der Java Daemon netcond liegt als .jar Datei vor[3]. Auf einem Betriebssystem mit grafischer Oberfläche könnte er einfach per Doppelklick gestartet werden, vorausgesetzt eine Java Runtime Environment in Version 6 (oder höher) ist installiert. Da es sich aber um eine konsolenbasierte Anwendung handelt, würden dann keinerlei Ausgaben sichtbar sein und der Daemon unsichtbar im Hintergrund laufen. Deshalb empfiehlt es sich, auch unter grafischen Betriebssystemen die netcond.jar Datei über die Konsole auszuführen:

```
java -jar netcond.jar
```

Auf Systemen ohne Monitor müsste die .jar Datei beispielsweise per SSH-Zugang übertragen und ausgeführt werden. Die Einrichtung des SSH-Zugangs für Systeme ohne Monitor wird in dieser Diplomarbeit nicht weiter behandelt, wobei der SSH-Zugang unter Linux-basierten Systemen in vielen Fällen bereits eingerichtet ist. Danach sollte mit dem Befehl ssh und der IP des Zielsystems zugegriffen werden können.

Für die Verwendung der Beispielwebsite wird ein PHP-fähiger Webserver, wie zum Beispiel Apache, oder lighttpd (inkl. PHP-Modul) benötigt, der nicht gemeinsam mit dem netcon Deamon auf einem Computer laufen muss. Wurde der Webserver gestartet, müssen nur noch alle Dateien aus dem Verzeichnis netcon web[4] ind das root-Verzeichnis des Servers kopiert werden. Tippt man nun http://localhost in die Adresszeile seines Browsers, erscheint die Website.

[3] /netcon Software/netcond/netcond.jar
[4] /netcon Software/netcon web

4.9.1 JRE

Die Java Runtime Environment (JRE) ist ein Java-Interpreter, der benötigt wird um den in Bytecode vorliegenden Daemon netcond auszuführen. Sie muss in Version 6 oder höher vorliegen.

Windows

Für die Installation der JRE unter Windows (XP,Vista,7,8) muss lediglich die neuste Version von http://www.java.com/de/download/ geladen und die .exe Datei installiert werden. Der java-Befehl ist danach in der Eingabeaufforderung verfügbar.

Mac OS X

Alle User der neuesten Version Mac OS X Version besitzen bereits eine JRE, die womöglich noch über die Softwareaktualisierung aktualisiert werden muss.[5] Danach ist lediglich das Terminal zu öffnen und die .jar Datei auszuführen.

[5] Apple wird in naher Zukunft Java nicht mehr selbst entwickeln und aktualisieren. Die Weiterentwicklung wurde wieder an Oracle abgegeben und in der nöchsten Mac OS X Version wird die JRE womöglich von der Oracle-Website bezogen werden müssen.

Linux

Da jede Linux-Distribution einen anderen Paketmanager verwendet um Anwendungen zu installieren/deinstallieren, hier nur eine allgemeine Erklärung. Der jeweilige Paketmanager muss gestartet und nach einem Paket mit den Inhalten "java" und "oracle" oder "sun" (ehemaliger Hersteller) gesucht werden. Nachdem das Paket installiert wurde, ist der java-Befehl in der Konsole verfügbar.

Manche Distributionen, vorallem gekaufte, besitzen in vielen Fällen bereits eine JRE. Es empfiehlt sich einfach in der Konsole zu prüfen, ob der Befehl "java" verstsanden wird. In freie Distributionen sind oftmals proprietäre Pakete, wie java, aus lizenzrechtlichen Gründen nicht standardmäßig freigeschaltet. Um solche Pakete finden und installieren zu können, muss dies in den Einstellungen des Paketmanagers gesetzt sein.

4.9.2 Webserver

Hier wird am Beispiel von XAMPP die Installation eines PHP-fähigen Webservers erklärt. XAMPP ist ein kostenloses Programmpaket, das alle nötigen Komponenten (Apache, PHP,...) eines Webservers mitbringt und für Windows, Mac OS und Linux verfügbar ist.

Windows

Unter Windows ist die Installation von XAMPP genauso einfach, wie die der JRE. Von der XAMPP-Website wird die .exe-Datei geladen und per Doppelklick installiert. Danach werden die Dateien der Website nach *C:\xampp\htdocs* kopiert und XAMPP über das Programmsymbol XAMPP Control Panel Application ausgeführt. Nun kann über den Start/Stop-Button der Webserver gestartet werden. Im Browser sollte durch Eingabe von *http://localhost* die Website verfügbar sein.

Mac OS X

Mac-User laden die .dmg-Datei von der XAMPP-Website und mounten diese per Doppelklick. Danach ist bloß der XAMPP-Ordner nach */Applications/* zu kopieren. Nachdem die Website in den htdocs-Ordner im Applications-Verzeichnis kopiert wurde, kann der Webserver über das Programm XAMPP Control über den Start/Stopp-Button gestartet und die Website im Browser mit *http://localhost* aufgerufen werden.

Linux

Unter Linux ist für die Installation und Inbetriebnahme von XAMPP keine grafische Oberfläche notwendig. Somit steht dem Betrieb auf Embedded Systems nichts im Wege.

Das tar.gz-Archiv muss von der XAMPP-Website heruntergeladen werden. Für die Installation werden zunächst mit dem Kommando *su* root-Rechte erlangt und das Archiv mittels *tar xvfz xampp-linux-1.7.7.tar.gz -C /opt* entpackt. XAMPP wurde damit ins Verzeichnis */opt* installiert. In den Ordner */opt/lampp/htdocs* ist nun die Website zu kopieren und der Webserver mit dem Kommando */opt/lampp/lampp start* auszuführen. War der Start erfolgreich, sollte "XAMPP gestartet" ausgegeben werden. Verfügt das Linux-System über eine grafische Oberfläche, kann jetzt im Webbrowser mit *http://localhost* die Website aufgerufen werden. Sonst verbindet man sich über einen grafischen Client (z.B. Smartphone) mit dem Webserver.