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 22. 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 22. 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.

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

Danksagung

fehlt noch

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.

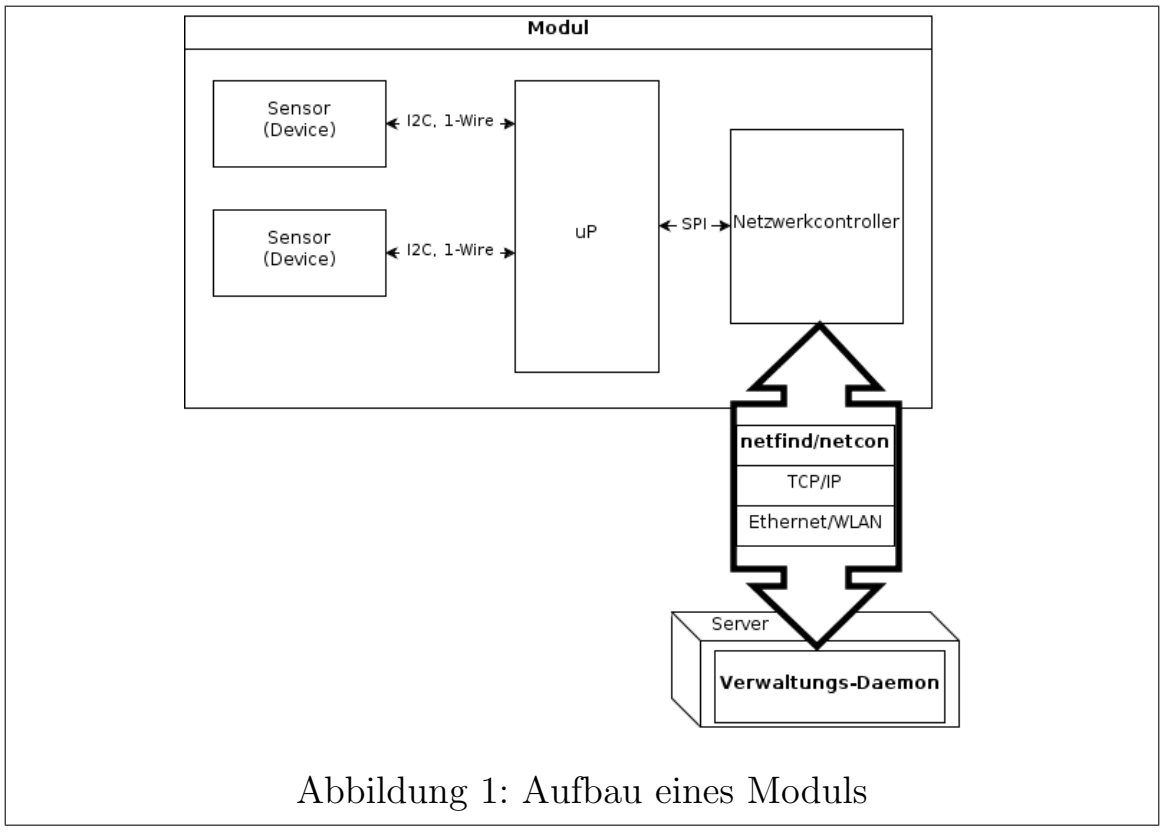


Abbildung 1: Aufbau eines Moduls

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.

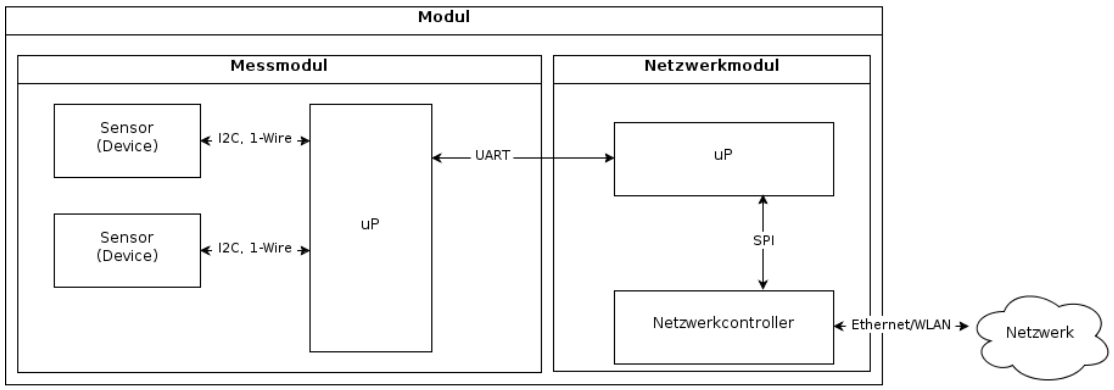


Abbildung 2: Netzwerkfähiges Messmodul mittels LAN-UART Umsetzer

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.

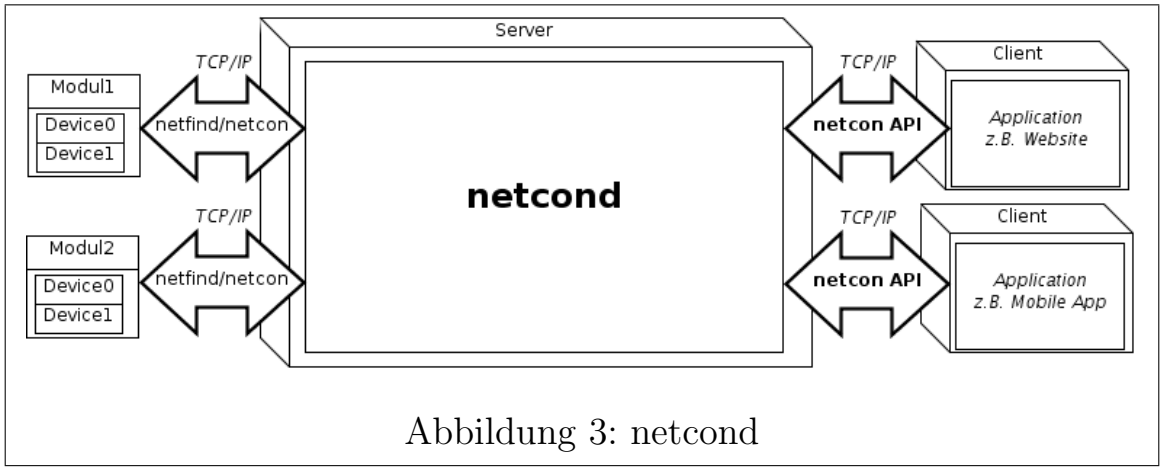


Abbildung 3: netcond

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.

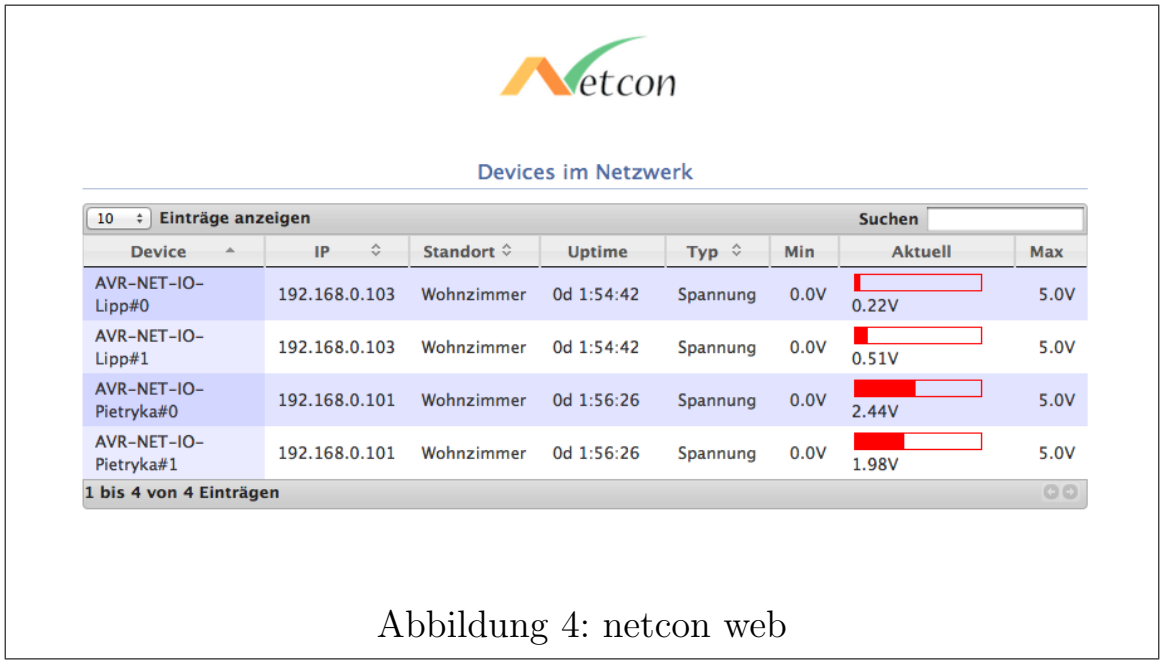


Abbildung 4: netcon web

2 Grundlagen [Pietryka]

2.1 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.

2.1.1 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

2.1.2 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.

2.1.3 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

2.1.4 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.

2.1.5 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.

2.1.6 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.

2.1.7 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.

2.2 Ethernet

2.2.1 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 adminis-

trierte Adresse, der Nachteil hierbei ist natürlich, dass diese Adresse nicht mehr eindeutig ist.

2.2.2 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.

2.3 IP

2.4 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 Resolution 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 ei-

ne gewisse Zeit geschrieben, damit man nicht bei jedem neuen IP-Paket diese ARP-Anforderung durchführen muss.

2.5 TCP

2.6 UDP

2.7 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 Anfragen, 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 Der Ethernet Controller

3.2 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).

3.3 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.

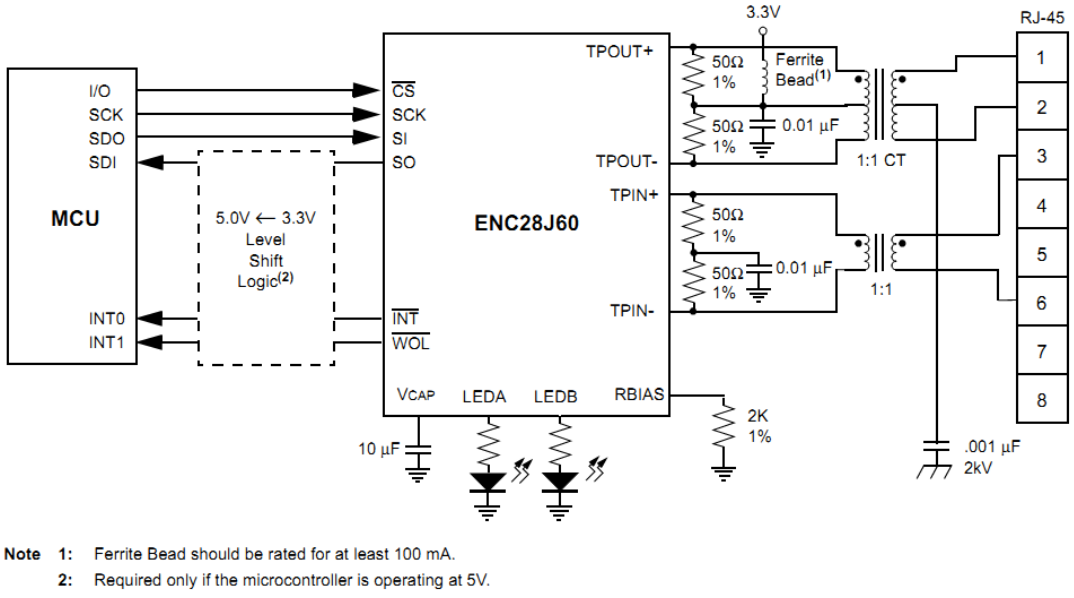


Abbildung 5: Aussenbeschaltung ENC28J60

3.4 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 wer-

den kann, diese wurde für den AVR geschrieben und besteht aus zwei Dateien ënc28j60.hünd ënc28j60.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 ënc28j60.cëntfernen, 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/".

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.

3.4.1 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);
```

3.4.2 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));
```

3.4.3 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
    }
}
```

3.4.4 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.

3.5 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.

3.6 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.

3.7 DHCP Implementierung

3.8 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.

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.

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 abgefragt. Eine Implementierung dieses Protokolls befindet sich im Ordner "netcon Module/Pollin AVR-NET-IO/netcon_ser", diese wurde bis jetzt nicht einem dauerhaften Test unterzogen, es könnten Fehler enthalten sein.

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. 6 grafisch verdeutlicht.

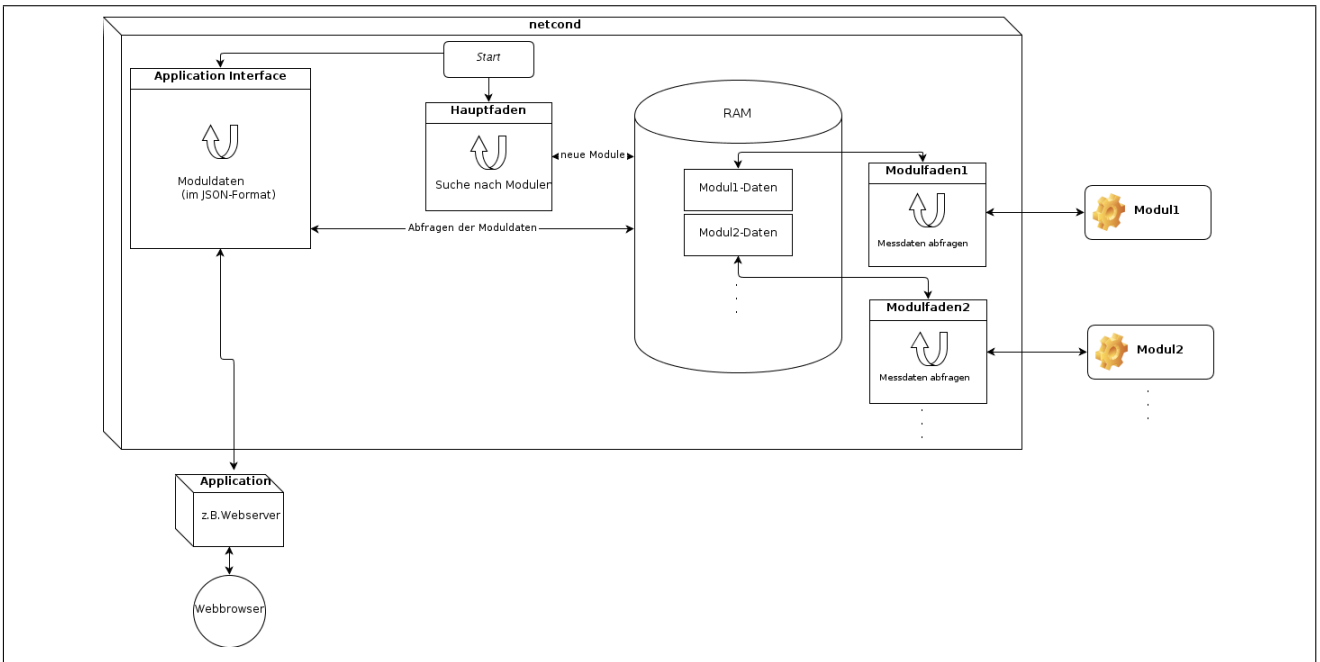


Abbildung 6: netcon - Funktionsprinzip

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. 7).

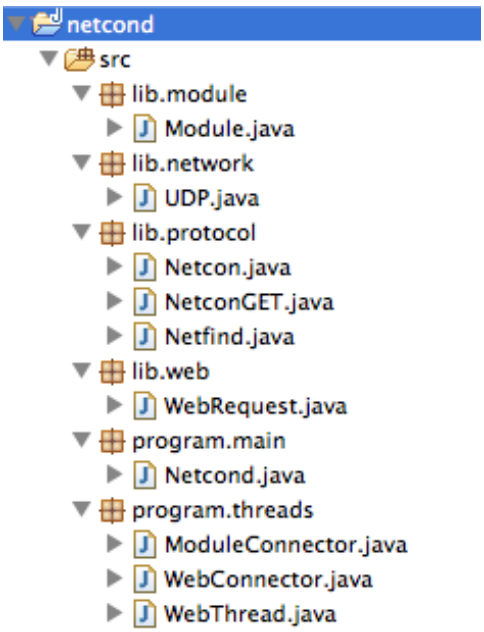
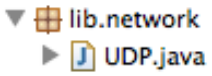


Abbildung 7: netcond - Übersicht Sourcecode

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.

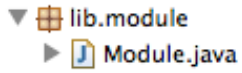
```
public static void
sendBroadcast(String msg, int dstPort, int srcPort)

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()
```

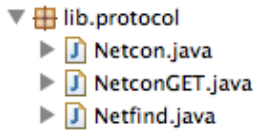
Die Methode getJSON() liefert die Moduldaten, also die Werte der Instanzvariablen verpackt in einem JSONObject zurück.

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)
```

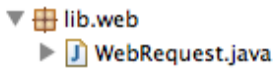
Die Auswahlmöglichkeiten für den enum-Parameter sind in der enum-Klasse NetconGET definiert. Einfacher gesagt lassen sich all diese Daten von einem Modul über das Netcon-Protokoll abrufen.

```
public enum NetconGET {

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

}
```

lib.web

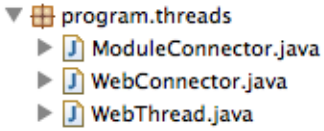


Im Paket lib.web enthalten ist die statische Klasse WebRequest.java, die Funktionen zur Verarbeitung der Anfragen von anderen Anwendungen bereitstellt. Dazu sind zwei Funktionen vorgesehen get() und set(), wobei letztere nur für die spätere Implementierung eines Steuernetzwerkes deklariert wurde und 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. 8.

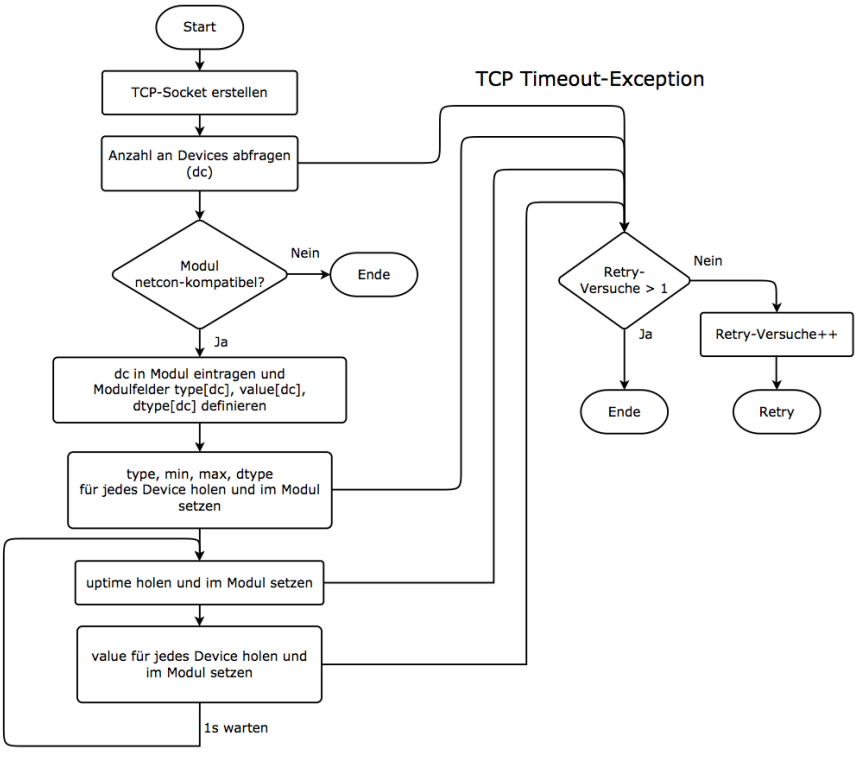


Abbildung 8: ModuleConnector

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. 9 macht diesen Vorgang verständlicher.

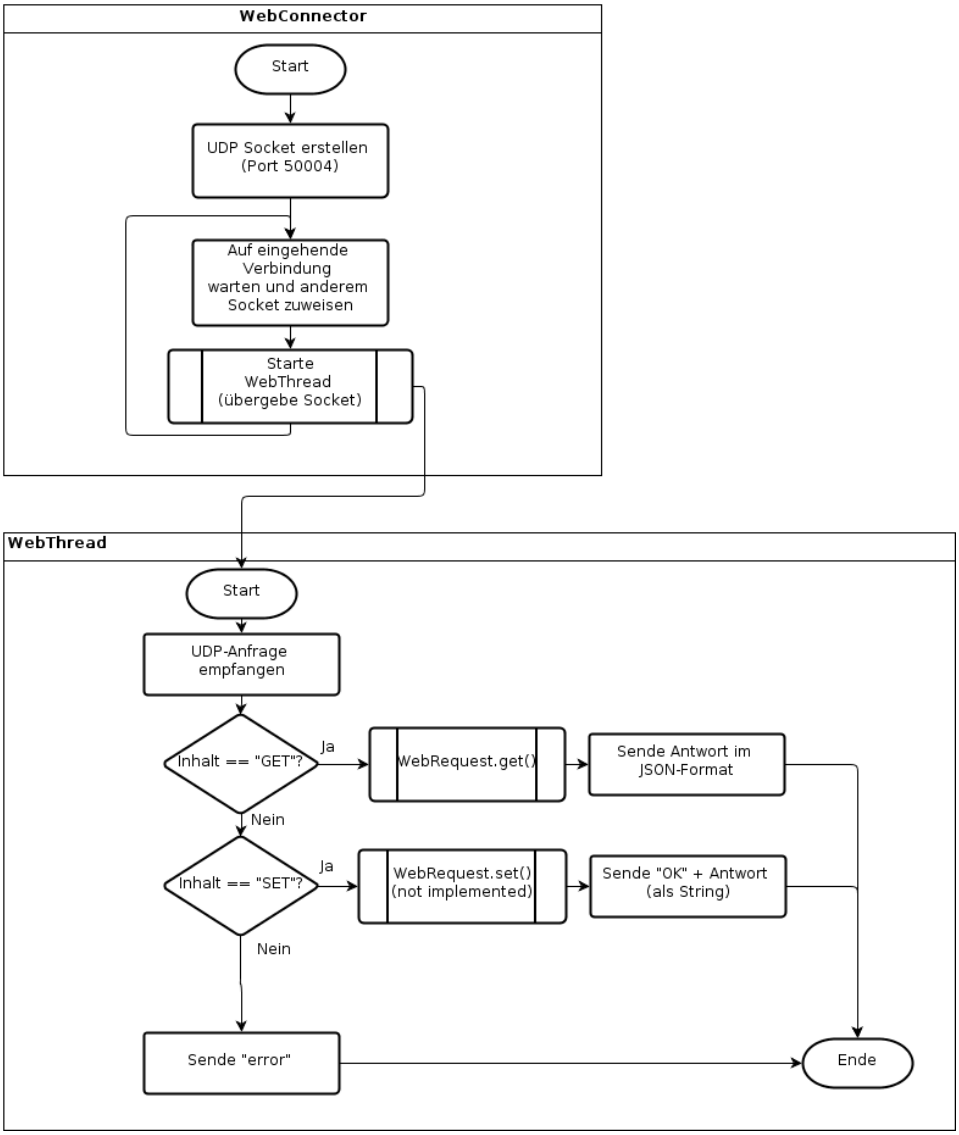
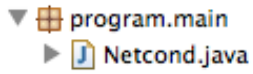


Abbildung 9: WebConnector

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) {
// ...
}
```

Die Klasse Netcond.java bildet den Main-Thread des Daemons, der nach dem Start zuerst ausgeführt wird. Dabei wird, wie in Abb. 10 erkennbar, zunächst der WebConnector-Thread gestartet. Daraufhin folgt eine Endlosschleife, die zuerst einen UDP-Broadcast - mithilfe der bereits beschriebenen Funktionen - von Port 50000 an Port 50001 aussendet 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. 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. 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.

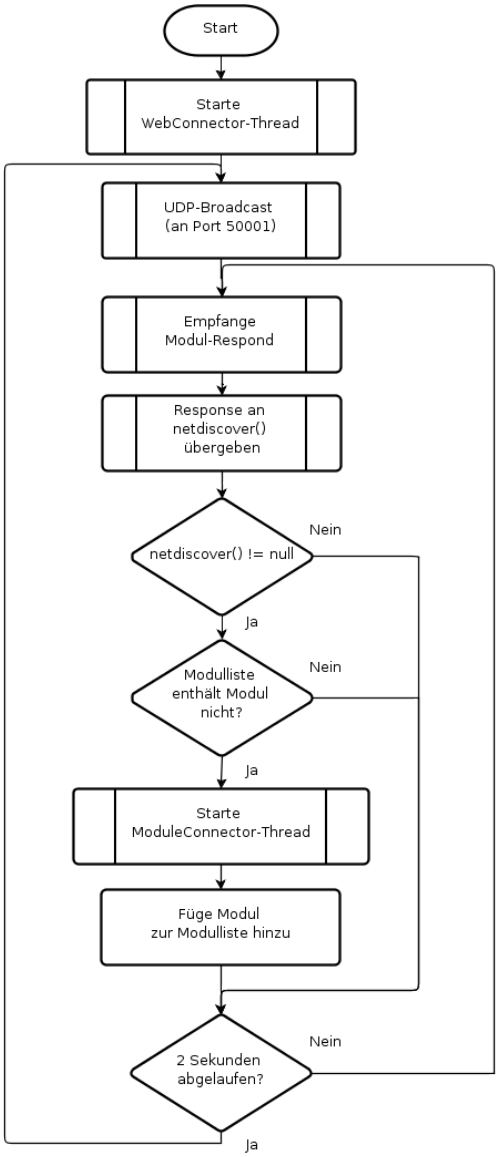


Abbildung 10: WebConnector

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

4.7 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. 11 zeigt je nach Kommunikation, an welchen Ports die Module liegen müssen bzw. von welchen Ports empfangen werden muss. Analog dazu Abb. 12 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 11: 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 12: Ports für die Kommunikation mit Anwendungen

4.8 Programmausgaben und Debugging

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
(IP)

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

I: ModulThread started #
(Hostname, IP)

E: Timeout #

E: ModulThread stopped #
```

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 der IP-Adresse von der die Anfrage kam.

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 Hostnamen und 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 eine zufällig generierte Zahl steht, die für weitere Fehlermeldungen hinsichtlich der TCP-Kommunikation eine Rolle spielt.

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

4.9 Website

4.10 Installation

4.10.1 JRE

4.10.2 Webserver

5 Projektorganisation