Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Arne Güldener

SNMP-Überwachung eingebetteter Echtzeit-Ethernet-Systeme im Fahrzeug

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Arne Güldener

SNMP-Überwachung eingebetteter Echtzeit-Ethernet-Systeme im Fahrzeug

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Martin Becke

Eingereicht am: 3. Juli 2017

Arne Güldener

Thema der Arbeit

SNMP-Überwachung eingebetteter Echtzeit-Ethernet-Systeme im Fahrzeug

Stichworte

SNMP, Manager, Agent, Managed Device, Management Information Base, Netzwerk-Überwachung, Hardware-Board, Sensor-Board, Time-Triggered-Ethernet

Kurzzusammenfassung

Diese Bachelorarbeit befasst sich mit der Implementierung des Netzwerküberwachungsprotokolls ”Simple Network Management Protocol” in einem Echtzeit-Ethernet-Backbone eines Fahrzeugs der CoRE-Research-Group. Zur Überwachung eigendefinierter Werte wurde zur Management Information Base ein Modul definiert. Es werden folgende Werte überwacht: CPU-Temperatur, CPU-, Arbeitsspeicher- und Netzwerk-Auslastung sowie Strom- und Spannungswerte. Des Weiteren wurde an einem zu überwachendem Gerät ein zuätzliches Hardware-Board zur Nachrüstung zusätzlicher Sensoren für Strom- und Temperaturmessung gefertigt.

Arne Güldener

Title of the paper

SNMP-Monitoring of embedded Realtime-Ethernet-Systems in a vehicle

Keywords

SNMP, Manager, Agent, Managed Device, Management Information Base, Network Monitoring, Hardware-Board, Sensor-Board, Time-Triggered-Ethernet

Abstract

This bachelor thesis deals with the implementation of a network monitoring protocol called ”Simple Network Management Protocol” in a Real-Time-Ethernet-Backbone of a vehicle of the CoRE-Research-Group. A Management Information Base Module has been defined for monitoring user defined values such as: CPU-Temperature, CPU-, RAM-, and Network-Load as well as voltage and electric current on a Managed Device. An additional Hardware-Board was build to measure missing values such as energy and temperature.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

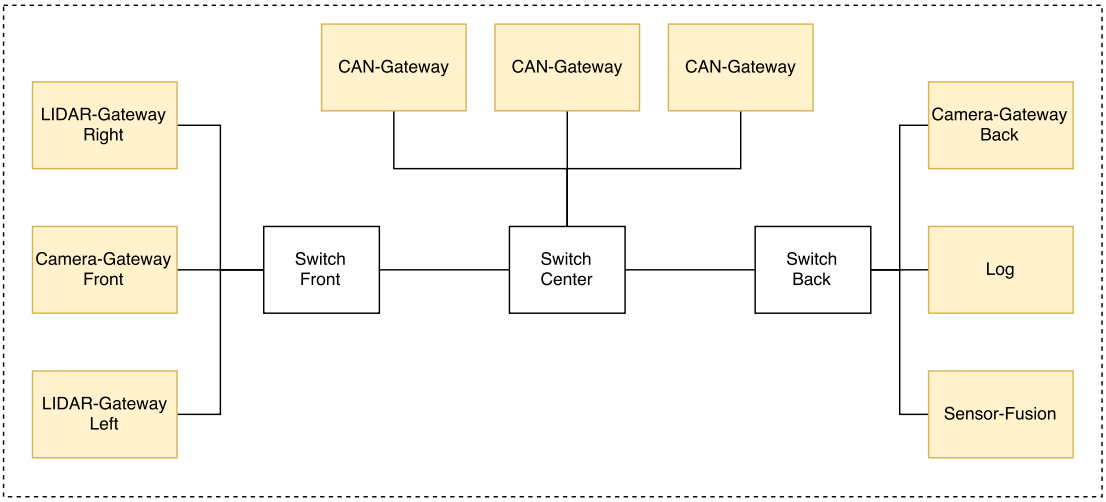


Abbildung 1.1.: Vereinfachte Echtzeit-Ethernet-Backbone-Architektur eines Fahrzeuges

Echtzeit-Ethernet-Systeme in modernen Fahrzeugen bekommen heutzutage immer mehr Bedeutung. Die dadurch stetige Zunahme von netzwerkfähigen Geräten, wird deren Überwachung zunehmend komplexer. Wie in Abbildung 1.1 zu sehen ist, besteht ein Echtzeit-Ethernet-Backbone im Fahrzeug aus mehreren Switches, Gateways und verschiedensten Computern. Um dessen sicherheitsrelevante Funktionen zu erhalten oder frühzeitig deren möglichen Ausfall vorzubeugen ist eine Überwachung solcher Geräte unabdingbar.

Um diese Geräte hinsichtlich der CPU-, Arbeitsspeicher- und Netzwerk-Auslastung sowie Temperatur und Stromverbrauch zu überwachen wird ein einfaches standardisiertes Netzwerk-Protokoll gesucht, welches diese Daten auf Abfrage an einen überwachenden Computer leitet. Dieser hat im Falle von Grenzwertüberschreitungen entsprechend zu reagieren, indem er gegebenenfalls im System automatisch eingreift und dem User/Administrator Bescheid gibt.

Es existieren bereits diverse Netzwerk-Überwachungs- und -Monitoring-Protokolle sowie auch -Tools. In dieser Arbeit wird ein Protokoll implementiert, welches aufgrund der begrenz-

ten Ressourcen der verschiedenen Geräte- ressourcenschonend hinsichtlich des Speichers, der Prozessor-, wie auch der Netzwerk-Auslastung operiert.

Diese Bachelorarbeit besteht aus zwei Teilgebieten. Der erste Teil beschäftigt sich mit der Implementierung vom Simple Network Management Protocol (SNMP) wie auch der Entwicklung einer eigenen Management Information Base (MIB, Datenbank, für die zu überwachenden Daten) in einem eingebettetem Echtzeit-Ethernet-System im Fahrzeug. Die Implementierung der zu überwachenden Geräte (Agenten) erfolgt auf verschiedenen Linux-Distributionen wie auch auf netzwerkfähigen Mikrocontrollern. Für die Agenten wie auch dem Manager (Überwacher) werden Software-Suites und Bibliotheken ausgewählt und analysiert, welche den Anforderungen an das System gerecht werden.

Im zweiten Teilgebiet wird für diese Arbeit ein Hardware-Board entwickelt, welches noch fehlende Sensor-Werte der zu überwachenden Geräten misst und sie diesen bereitstellt. Dies geschieht beispielhaft anhand eines der zu überwachenden Mikrocontrollern. Das Hardware-Board misst Werte wie Temperatur, Spannung und Strom. Mittels einer vorhandenen Schnittstelle werden diese Informationen an den Mikrocontroller geleitet und in dessen MIB eingetragen. Als Beispiel für die Umsetzbarkeit wurde diese Bachelorarbeit für den VW Golf 7 der CoRE-Research-Group geschrieben, das ein Echtzeit-Ethernet-Backbone mit Switches, ARM- und Linux-Boards integriert hat.

Aufbau der Arbeit

Im Kapitel 2 werden Grundlagen erläutert, die zum besseren Verständnis der folgenden Kapitel dienen. Es wird erläutert, was unter dem Simple Management Protocol zu verstehen ist. Es wird weiterhin erklärt, was unter dem Time-Triggered-Ethernet zu verstehen ist. Daraufhin wird festgelegt, welche Computer und Mikrocontroller sich im CoRE-Prototypfahrzeug befinden und welche Aufgabe diese haben. Kapitel 3 geht auf die Anforderungen des Systems ein. Es wird analysiert, wie SNMP-Nachrichten sich im Echtzeit-Ethernet des Prototypfahrzeuges zu typisieren sind, welche Informationen relevant zur Überwachung des Systems sind und wie der Überwacher und die zu überwachenden Geräte zu agieren haben. Des Weiteren wird festgestellt, welche der zu überwachenden Informationen bei den Geräten bereits implementiert sind beziehungsweise welche noch nachgerüstet werden müssen für das Hardware-Board. Das Kapitel 4 befasst sich mit den verschiedenen Konzepten hinsichtlich der möglichen Implementierungen der SNMP-Agenten und des -Managers, wie auch der Erstellung des Hardware-Boards. Letztendlich welches Konzept für die SNMP-Agenten, des -Managers und des Hardware-Board

genutzt wird, wird im Kapitel 5 erläutert. Daraufhin wird in Kapitel 6 das in dieser Arbeit entwickelte Endprodukt evaluiert und in Kapitel 7 zusammengefasst.

2. Grundlagen

Das Kapitel Grundlagen befasst sich mit den wesentlichen Punkten, die für das Verständnis dieser Bachelorarbeit beitragen. Im ersten Unterkapitel ”Simple Management Protocol” werden die Grundlagen des Protokolls erklärt. Daraufhin wird das genutzte Time-Triggered-Ethernet im Backbone des CoRE-Prototypfahrzeugs beleuchtet, woraufhin als letztes Unterkapitel die in diesem Backbone genutzte Hardware mit deren Aufgaben dargestellt werden.

2.1. Simple Network Management Protocol

Das Simple Network Management Protocol (SNMP) ist ein seit 1990 von der Internet Engineering Task Force (IETF) standardisiertes Protokoll (SNMPv1: RFC1098 (Case u. a., 1989)) zur Netzwetzwerküberwachung und Wartung von Geräten in einem IP-basierten Netzwerk. SNMP ist ein Protokoll auf der TCP/IP-Schicht 4 - Anwendung (RFC1122 (Braden, 1989)). Das Wort ”simple” im Namen kam daher, dass einerseits SNMP einen einfachen Satz an Befehlen hat und andererseits das verbindungslose Transport-Protokoll UDP (User Datagram Protocol: RFC768 (Postel, 1980), TCP/IP-Schicht 3 - Transport) nutzt. Dies bedeutet jedoch auch dementsprechend, dass es im Vergleich zu TCP (Transmission Control Protocol: RFC793 (Postel, 1981)) nicht garantiert wird, dass ein gesendetes SNMP-Pakete empfangen wird, dass es nur einmal empfangen wird oder dass mehrere Pakete in der gleichen Reihenfolge eintreffen, in der sie auch gesendet worden sind.

2.1.1. Akteure

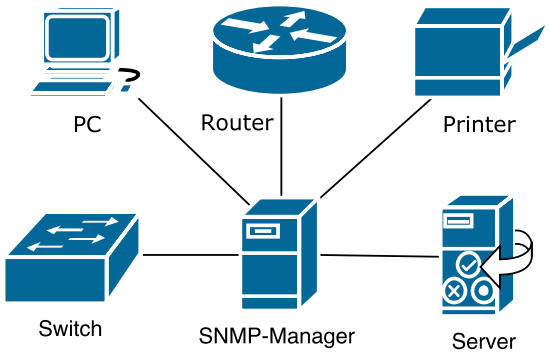


Abbildung 2.1.: Mögliches SNMP-Netzwerk mit einem Manager und mehreren Managed Devices

Manager

Die Aufgabe eines Managers (in Abbildung 2.1, Mitte) ist, die bei ihm konfigurierten *Managed Devices* zu überwachen. Ein SNMP-überwachtes System kann aus einen oder auch mehreren Managern bestehen.

Managed Device / Agent

Managed Devices sind die Geräte in einem SNMP-Netzwerk, die vom Manager überwacht werden. Dessen Aufgabe ist, die in deren Datenbank (Management Information Base) beschriebenen Informationen aktuell zu halten und diese auf Anfrage des Managers ihm zuzusenden. Wie in Abbildung 2.1 zu sehen ist, ist es nicht relevant, was für ein Geräte-Typ (ob PC, Switch, Drucker oder Router, ...) ein Managed Device ist, solange auf diesem ein SNMP-Agent läuft. Obwohl der Begriff ”Managed Device” lediglich das überwachte Gerät und ein ”Agent” das darauf laufende SNMP-Programm beschreibt, werden die Begriffe meist synonym behandelt.

2.1.2. SNMP-Versionen

Es existieren drei Versionen vom Simple Network Management Protocol: SNMPv1, SNMPv2 und SNMPv3. Jorrit Schippers (2012)) stellte in einer Bachelorarbeit der University of Twente fest, dass die Versionen SNMPv1 und SNMPv2c letztendlich am meisten genutzt werden.

SNMPv1

Die erste SNMP-Version ist 1988 definiert und 1990 mit dem RFC1098 (Case u. a., 1989) überholt und standardisiert durch das RFC1157 (Case u. a., 1990), jedoch inzwischen als historisch gekennzeichnet worden.
Die erste Version bietet keine Verschlüsselung der Daten. Die Zugriffsrechte auf die einzelnen zu überwachenden Objekte eines Gerätes werden über Communities gesteuert. Communities sind in gewissermaßen eine Mischung aus Nutzername und Kennwort. Jedoch werden diese Informationen über das Netzwerk als Klartext gesendet. Dies bedeutet jedoch somit, dass jeder im Netzwerk diese Daten auslesen kann (Sniffing-Gefahr). Diese Zugriffsrechte steuern mit *READ-ONLY* (nur lesen) und *READ-WRITE* (schreiben und lesen), welche Community welche Schreib- und Leserechte auf den MIB-Baum (siehe Unterkapitel 2.1.3) hat. 1992 erschien das RFC1351 (Davin u. a., 1992) und warb mit SNMPsec (Secure SNMP) für Sicherheit. Jedoch wurde dieses nie eingesetzt.

SNMPv2

Die ursprüngliche Version SNMPv2 beziehungsweise SNMPv2p arbeitete mit einem *Party-Based* Sicherheitsmechanismus (RFC1445 (Galvin und McCloghrie, 1993)), wurde aber als zu aufwendig angesehen. Stattdessen kamen mehrere Unterversionen (SNMPv2c, SNMPv2u und SNMPv2*) zum Einsatz, wobei sich SNMPv2c schließlich durchsetzte, jedoch wurde das RFC 1901 nie als Standard markiert, sondern lediglich als experimentell. SNMPv2c ist wie SNMPv1 noch Community-basiert und bietet auch hier keine erweiterte Sicherheit, da auch hier keine Verschlüsselung der Community-Strings existiert. Jedoch wurden mit SNMPv2 weitere Protokollnachrichten (siehe Unterkapitel 2.1.4) und mehr Datentypen für die Management Information Base definiert wie auch eine Manager-zu-Manager-Kommunikation eingerichtet.

SNMPv3

Mit den RFCs 3410 bis 3417, 3584, 3826 und 5343 (siehe Literaturverzeichnis) wurden aufgrund der immer noch mangelhaften Sicherheit der vorherigen Versionen wurde SNMPv3 definiert. Zu den neuen Sicherheitsmechanismen gehören Verschlüsselung der zu übertragenen Daten wie auch Authentifizierungen des Managers gegenüber den Agenten. Letztendlich wurde SNMPv3 wie oben erwähnt jedoch nie wirklich genutzt.

Kompatibilität zwischen SNMPv2 und SNMPv1

Es ist nicht direkt möglich, dass SNMPv2-Nachrichten von SNMPv1-Agenten verstanden werden können. Jedoch existieren sogenannte ”Proxi-Agenten”. Diese werden zwischen einem SNMPv2-Managern und einem SNMPv1-Agent geschaltet. Nachrichtentypen wie Set, Get und Repsonse werden direkt weitergeleitet, da diese sich nicht unterscheiden zwischen SNMPv1 und SNMPv2. Dementsprechend werden die Antworten vom SNMPv1-Agenten direkt an den Manager weitergeleitet. Lediglich SNMPv1-Traps werden in SNMPv2 umgewandelt, da diese anders aufgebaut sind (siehe Unterkapitel 2.1.4).

2.1.3. Management Information Base

Damit ein Manager auf die Informationen eines Agenten zugreifen kann, müssen diese in einer Datenbank definiert werden. Diese nennt sich Management Information Base (MIB, definiert in RFC1213 (McCloghrie und Rose, 1991)). Die darin überwachten Informationen (sogenannte ”Managed Objects”) einer MIB werden hierarchisch darstellt.
Der Aufbau einer MIB wird durch eine ”Structure of Management Information” (SMI, inzwischen SMIv2; RFC2578 (McCloghrie u. a., 1999)) festgelegt. Diese SMI ist eine Art Regelwert, wie die überwachten Werte (Managed Objects) zu definieren, welche Datentypen erlaubt und welche Operationen und Zugriffsrechte auf diesen Managed Objects erlaubt sind. Eine MIB wird in der Spezifikationssprache ASN.1 (Abstract Syntax Notation One) spezifiziert.

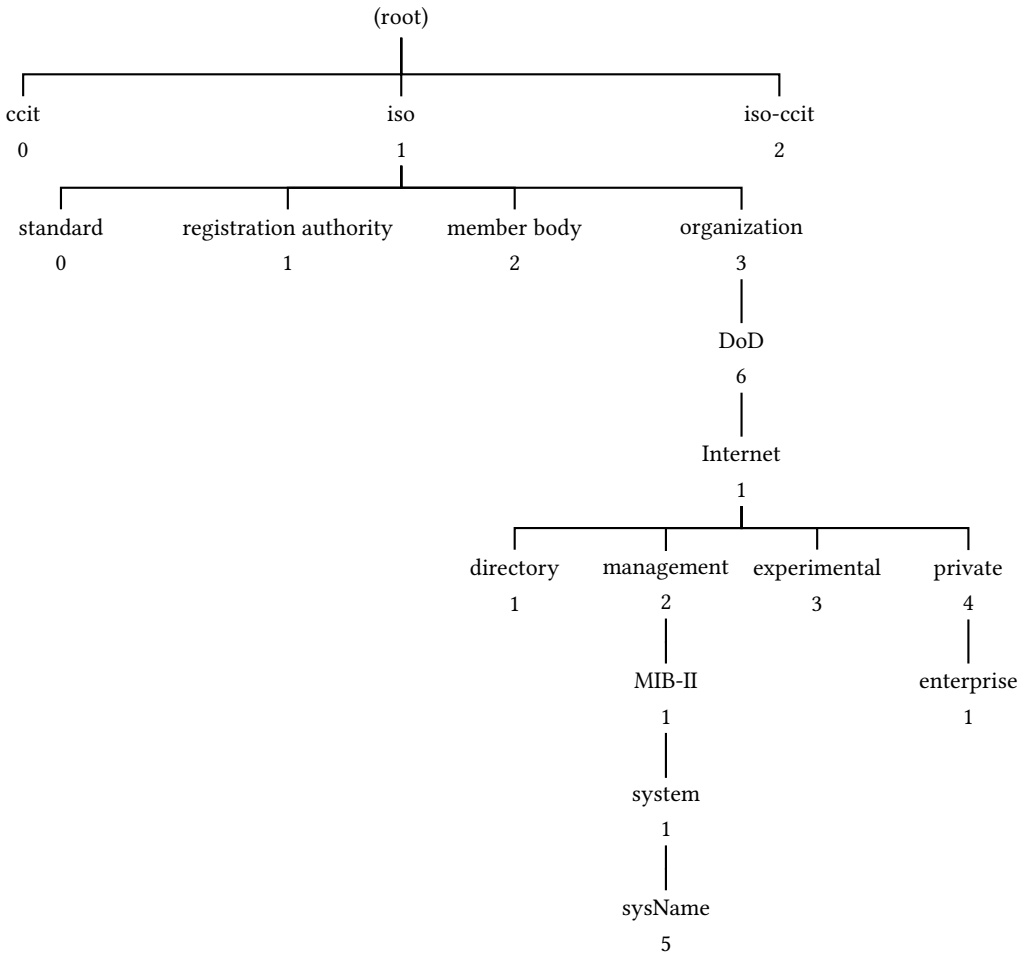


Abbildung 2.2.: Ausschnitt aus dem MIB-Baum

MIB-Modul

Wie in Abbildung 2.2 zu sehen ist, besteht der MIB-Baum aus einem namenlosen Wurzel ”(root)” mit mehreren Abzweigungen / Teilbäume (”subtrees). Diese Abzweigungen nennen sich MIB-Module. MIB-Module sind Container für die Blätter, die Managed Objects (zum Beispiel ”sysName”).
Die für SNMP relevanten Module sind einerseits die *MIB-II* wie auch das Modul *enterprise*: MIB-II ist eine durch das RFC1213 (McCloghrie und Rose, 1991) standardisierte Modul, welches eine einfache Architektur für TCP-IP-basierte Netzwerke bereitstellt. Das enterprise-Modul bietet Unternehmen Platz, um deren eigen definiertes MIB-Modul dort zu implementieren.
Damit Unternehmen dies machen dürfen, muss deren Modul bei der Internet Assigned Numbers

Authority (IANA) kostenlos registriert werden.
Meist werden MIB-Module auch einfach nur Management Information Base genannt.

Managed Object

Managed Objects (MOs) werden in den MIB-Modulen definiert. Diese beschreiben die zu überwachenden Werte, indem unter anderem der Datentyp, deren Lese- oder Schreibzugriff und dessen Beschreibung festgelegt wird. Eine Beispiel eines Managed Objects in Abbildung 2.2 wäre ”sysName”:

```
sysName OBJECT-TYPE
  SYNTAX  DisplayString (SIZE (0..255))
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
        "An administratively-assigned name for this
        managed node. By convention, this is the node's
        fully-qualified domain name."
::= { system 5 }
```

Listing 2.1: Auszug aus RFC 1213, ASN-1-Definition des Managed Objects ”sysName”

Dieses Managed Object (Listing 2.1) spiegelt den Domänennamen des überwachten Systems (Managed Device) wider, hat eine Zeichenkette von 0 bis 255 Zeichen und kann auch vom Manager aus gesetzt werden.

Object Identifier

Jede Abzweigung wie auch jedes Blatt im MIB-Baum besitzen eindeutige Bezeichner namens ”Object Identifier” (OID). Diese werden dem MIB-Baum nach hierarchisch bezeichnet. Wie in Abbildung 2.2 zu sehen ist, ist unter jedem Modul wie auch unter jedem Managed Object eine Nummer geschrieben. Würde ein SNMP-Manager zum Beispiel das oben erwähnte Managed Object ”sysName” eines Agenten erreichen wollen, so wäre dessen OID (von oben nach unten gelesen) *1.3.6.1.2.1.1.5*.

Skalare und tabulare Werte

Da MIBs nicht nur explizit für einen bestimmten Agenten definiert werden, sondern auch allgemein gültig und implementierbar sind, existieren skalare und tabulare Werte. Managed

Object wie ”sysName” sind skalare Werte, da zu diesem Typ nur eine Variable existiert (hier der Domänen-Namen des Gerätes). Tabulare Werte sind zum Beispiel Netzwerk-Interfaces eines Managed Devices. Verschiedene Managed Devices können eine variable Anzahl an Netzwerk-Interfaces haben. Ein Beispiel hierfür wäre die ”ifTable” der MIB-II.

2.1.4. SNMP-Protokollnachrichten

Die Kommunikation basiert auf dem simplen Prinzip des Request-Response. Der Manager fordert vom Agent etwas an (Request) und bekommt daraufhin vom Agent eine Antwort (Response).
Da SNMPv3 spärlich genutzt wird, werden im folgenden lediglich die SNMPv1- und SNMPv2-Nachrichten diskutiert (SNMPv2-Nachrichten sind spezifiziert RFC3416 (Presuhn, 2002b)):

Header		**PDU**				
Version	Community	Type	Request-ID	Error Status	Error Index	Variable-Binding(s)

Tabelle 2.1.: SNMP-Nachrichten bei Get, GetNext, Set, Response (Trap, Inform, Report ab v2)

- **Version** - Beinhaltet die genutzt SNMP-Version
- **Community** - Der Community-String zur Authentifikation
- **Type** - Nachrichtentyp (Get, GetNext, Set, Response, Trap, Inform, Report)
- **Request-ID** - Dient der Zuordnung von Request- zu den Response-Nachrichten
- **Error Status** - (Nur gesetzt in Response-Nachrichten) treten vor allen Dingen bei Set-Operationen auf; Vordefinierte Nummern geben den entsprechenden Fehler an
- **Error Index** - Zusätzliche Informationen zum Error Status, gibt den Index der auslösenden Variable an, bei der der Fehler auftrat
- **Variable-Binding(s)** - Managed Objects; Die Anzahl der Variable-Bindings ist nicht definiert; Entsprechend ist die Größe einer SNMP-Nachricht abhängig von den Variable Bindings

Bei SNMPv2 existieren sieben Nachrichtentypen:

- GetRequest
- GetNextRequest
- GetBulkRequest *(ab SNMPv2)*
- SetRequest
- Response
- Trap
- Inform *(ab SNMPv2)*

GetRequest

Get ist die Anfrage des Managers an einen Agenten für einen oder mehrere Managed Objects mittels angegebener OID (Abb. 2.3(a)). Der Agent verarbeitet daraufhin jedes einzelne angeforderte Managed Object, welches in der Variable-Binding-List der Nachricht enthalten ist. Besitzt ein Agent kein Managed Object zur angegebenen OID, so wird die Fehlermeldung *noSuchObject* zurückgesendet. Hat der Manager zum Beispiel mit dem angegebenen Community-String nicht genug Rechte um den Wert zu erfragen, so sendet der Agent die Fehlermeldung *nosuchInstance* zurück.

GetNextRequest

GetNext(OID) ist die Anfrage des Managers, mittels angegebener OID eine darauffolgendes Managed Object zu erfragen (Abb. 2.3(a)). Nach jedem Response(OID+1) des Agenten, kann ein weiteres GetNext auf diese OID gesendet werden, um das darauffolgende Managed Object zu erhalten (resultiert in einer hohen Netzwerk-Auslastung). Diese Methode ist praktisch, denn sie hat den Vorteil, dass der ganze MIB-Baum mit GetNext durchiteriert werden kann, ohne dass der Manager jede einzelne OID kennt. Dies wird auch als ”MIB-Walk” oder ”SNMP-Walk” bezeichnet. Ist das Ende des MIB-Baumes erreicht, so wird die Fehlermeldung *endOfMIBView* zurückgesendet. Es können auch lediglich einzelne MIB-Module iteriert werden.

GetBulkRequest

Ab SNMPv2 ist es möglich, mit GetBulk mehrere Managed Objects auf einmal abzufragen (Abb. 2.3(a)). Anstelle von GetNext, bei dem nach jedem Response des Agenten über ein

weiteres GetNext eine neue OID angefordert wird, werden bei GetBulk alle Managed Objects in eine Response-Nachricht geschrieben, was das Verhältnis der Nutzdaten (Payload) zu dem Header deutlich verbessert. Dies resultiert dann wiederum in weniger Fragmentierungen der Nachrichten.

SetRequest

Mit Set erfragt der Manager einen Agenten, ein oder mehrere Managed Objects zu ändern (Abb. 2.3(a)). Dies geschieht, indem der Manger dem Agenten in der Variable Bindings Liste die Manged Objects samt zu ändernde Werte mitsendet. Auch bei diesem Request können diverse Fehlermeldungen zurückgesendet werden (zum Beispiel aufgrund des falschen Datentyps, der falschen Größe oder ungenügender Rechte zum Ändern).

Response

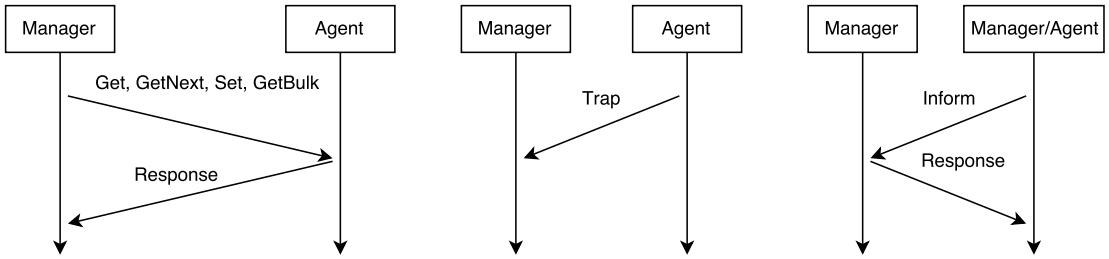
Dies ist die Antwort eines Agenten auf einen Manager-Request (Get, GetNext, Set, GetBulk oder Inform).

Trap

Traps (Abb. 2.3(b)) sind Ausnahmen in der typischen SNMP-Kommunikation. Diese basieren nicht auf den Request-Response-Prinzip. Traps werden ohne Request (unerwartet) von einem Agenten an einen Manager gesendet, um wichtige Ereignisse mitzuteilen. Dies müssen nicht unbedingt Fehler oder gar Ausfälle bedeuten. Aufgrund dessen, dass es relevant sein kann, wann ein bestimmtes Trap-Ereigniss aufgetreten ist und UDP-Pakete nicht unbedingt in der richtigen Reihenfolge oder überhaupt beim Manager eintreffen, wird die jeweilige *sysUpTime* des Agenten mitgesendet. Ein Agent wartet bei einer Trap-Nachricht nicht auf eine Antwort des Managers, somit kann nicht sichergestellt werden, dass diese Nachricht jemals vom Manager verarbeitet wurde.

InformRequest

Ab SNMPv2 gibt es den Nachrichtentypen Inform (Abb. 2.3(c)). Dieser Nachrichtentyp wurde ursprünglich für eine Manager-zu-Manager-Kommunikation eingesetzt, um gegenseitig sich Informationen zuzusenden. Da eine Inform-Nachricht sogar ein Response erwartet, wurde dies kurz darauf auch von Agenten als eine Trap-Erweiterung genutzt, da so aufgrund des Response sichergestellt werden konnte, dass diese Traps auch beim Manger eintreffen.



(a) Normale Nachrichten (b) Trap-Nachrichten (c) Inform-Nachrichten

Abbildung 2.3.: SNMP-Kommunikation bei verschiedenen Nachrichtentypen

2.2. Realtime Ethernet im Prototypfahrzeug

Die CoRE-Gruppe hat für das Forschungsprojekt RECBAR (Realtime Ethernet Backbone for Cars) in Zusammenarbeit mit der IAV GmbH Ingenieurgesellschaft Auto und Verkehr einen VW Golf 7 als Prototypfahrzeug erhalten. In Rahmen dieses Projektes wurde in das Auto ein Echtzeit-Ethernet-Backbone integriert, um das Potential und die Zukunftsfähigkeit von intelligenten Backbone-Infrastrukturen im Automotive-Bereich zu erforschen.

2.2.1. Echtzeit in Computersystemen

Echtzeit ist der Begriff, dass ein System in einer fest definierten Zeit auf ein Ereignis ein bestimmtes Ergebnis zu garantieren hat. Dabei existieren zwei verschiedene Arten von Deadlines:

Weiche Deadlines

Wenn ein Computer-System eine weiche Deadline (Soft Deadline) nicht einhalten kann, so ist zwar das Ergebnis unbrauchbar und kann verworfen werden, jedoch ist die Sicherheit des Systemes noch gewährleistet. Ein Beispiel im Auto könnte die Betätigung des Blinkers sein: Würde der Blinker erst nach einer Sekunde anfangen zu blinken, resultieren daraus keine für das System gefährlichen Folgen.

Harte Deadlines

Bei einer Nichteinhaltung einer harten Deadline kann dies für das System katastrophale Folgen haben. Beispielhaft wäre eine durch das System angesteuerte Fußbremse: Würde diese erst nach einer Sekunde nach Betätigung des Pedals bei einer Notbremsung reagieren, könnte daraus ein Unfall resultieren, was eine Gefahr für Mensch und Maschine bedeuten würde.

2.2.2. Time-Triggered Ethernet im CoRE-Backbone

Da das klassische Ethernet für die Kommunikation von Echtzeit-Systemen aufgrund diverser Einschränkungen (wie zum Beispiel keine garantierte Übertragung oder eine Übertragung in einem bestimmten Zeitraum) ungeeignet ist, wurde Echtzeit-Ethernet-Backbone im CoRE-Fahrzeuges das Time-Triggered Ethernet (TTE oder TTEthernet) implementiert. TTEthernet ist eine Erweiterung des Standard-Ethernets (TCP/IP-Schicht 1 - Netzzugang). Es wurde von der Firma TTTech Germany GmbH entwickelt und 2011 von der SAE International standardisiert (vgl. AS6802 (2011)). Zu diesem Standard existieren drei Nachrichtentypen:

Time-Triggered-Traffic (TT)

Time-Triggered-Traffic-Nachrichten haben die höchste Priorität im Netzwerk. Diese werden für den zeitkritischen Datenverkehr verwendet. Jedem Teilnehmer werden zuvor konfigurierte Zeitslots für deren TT-Nachrichten zugeordnet und somit eine Übertragung innerhalb des Slots garantiert.

Rate-Constrained-Traffic (RC)

Für diese Nachrichten wird mittels *Bandwidth Allocation Gap-Accounts* (Minimaler Zeitabstand zwischen zwei aufeinanderfolgenden Ethernet-Frames) sichergestellt, ob im Netzwerk genug Bandbreite zur Übertragung zu Verfügung stehen. Erst dann werden diese versendet. Das bedeutet auch, dass diese RC-Nachrichten verzögert werden können, wenn diese Bandbreite nicht zur Verfügung steht oder TC-Nachrichten durch ihre höhere Priorität Vorrang haben.

Best-Effort-Messages (BE)

Dieser Nachrichtentyp beinhaltet die Standard-Ethernet-Nachrichten (keine Sicherstellung, ob und wann die Nachrichten übertragen werden). Da dies die niedrigste Priorität hat, werden diese Nachrichten nur gesendet, wenn die Bandbreite nicht durch TE- oder RE-Nachrichten belastet ist.
Sind Teilnehmer in diesem Netz nicht TTE-fähig, so werden dessen Nachrichten automatisch vom TTE-Switch als Best-Effort-Nachrichten eingestuft und dementsprechend gesendet und empfangen.

2.3. Hardware im Prototypfahrzeug

Folgende Abbildung (Abb. 2.4) zeigt eine detaillierte Version der Abbildung 1.1. In den folgenden Unterkapiteln wird diese Hardware dargestellt und vereinfacht dessen Aufgabe im Auto beschrieben:

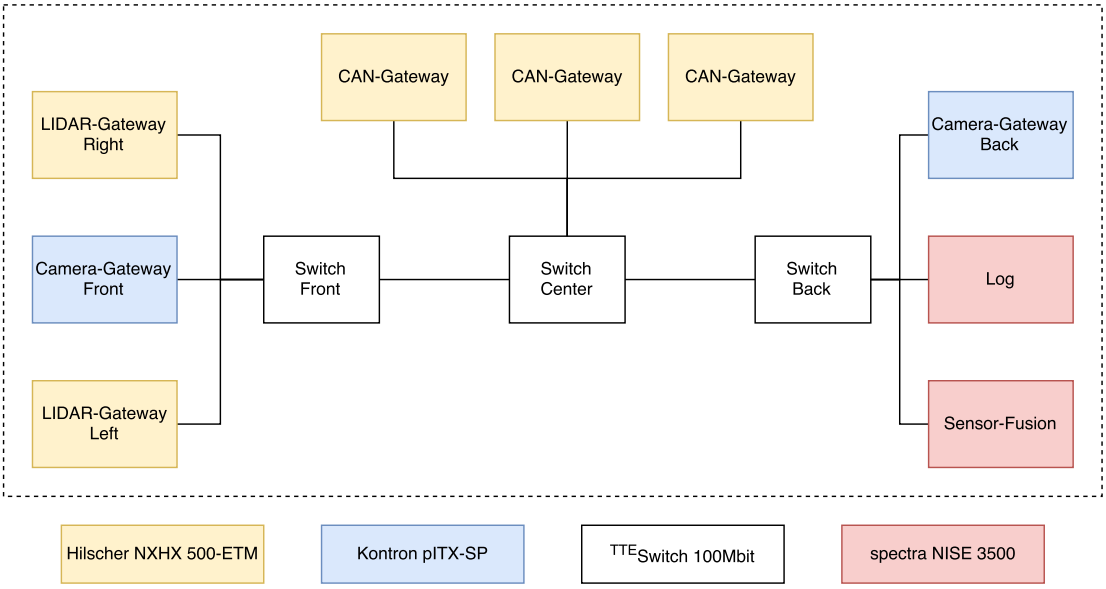


Abbildung 2.4.: Echtzeit-Ethernet-Backbone im VW Golf 7 der CoRE-Gruppe

2.3.1. Hilscher NXHX 500-ETM

Das NXHX 500-ETM (in Abbildung 2.4 gelb) der Hilscher Gesellschaft für Systemautomation mbH ist ein Software-Entwicklungsboard bestückt mit einem integriertem NETX 500 Netwerkcontroller basierend auf einen ARM 926EJ-S mit 200 Megahertz. Mit vier Kommunikationskanälen ist dieser konfigurierbar für Echtzeit-Ethernet oder Feldbus-Interfaces wie zum Beispiel CAN. Im VW Golf 7 der CoRE-Gruppe sind fünf dieser Entwicklungsboards in das Echtzeit-Ethernet-Backbone integriert:

CAN-Gateway

Drei Boards bieten die Schnittstelle zwischen dem eingebauten CAN-Netzwerk des Autos und dem Echtzeit-Ethernet-Backbone. Diese dienen von Standard-CAN-Packet-Analysen bis hin zu Forschungen, ob und inwiefern ein CAN-Netzwerk im Auto durch ein Echtzeit-Ethernet abgelöst werden könnte.

LIDAR-Gateway

Zwei Boards an der Front-Seite des Autos sind bestückt mit jeweils einem Laserscanner (LIDAR - Light detection and ranging) und sind Teil der Hindernis- und Gegenstandserkennung vor dem Auto. Für die Auswertung der Sensordaten werden diese Informationen über das Netzwerk an die Hardwarekomponente *Sensor-Fusion* gesendet.

2.3.2. Kontron pITX-SP

Zwei Kontron pITX-SP-Systeme (in Abbildung 2.4 blau) der Firma Kontron AG sind Camera-Gateways, an denen jeweils eine Kamera für vorn beziehungsweise hinten angeschlossen ist. Auf diesen Systemen ist die Linux-Distribution gentoo linux installiert.

Camera-Gateway Front

Zusammen mit den Sensordaten der beiden *LIDAR-Gateway*s werden diese Bild-Informationen zur Hindernis- und Gegenstandserkennung vor dem Auto genutzt. Auch diese Informationen werden für die weitere Verwertung der Daten an die Hardwarekomponente *Sensor-Fusion* gesendet.

Camera-Gateway Rear

Zusammen mit dem vorderen Gateway dienen diese zum Testen der Belastbarkeit des Echtzeit-Ethernet-Backbones. Jedoch dient das hintere Gateway im vergleich zum *Camera-Gateway Front* bislang nicht der Hindernis- und Gegenstandserkennung hinter dem Auto.

2.3.3. spectra NISE 3500

Die zwei spectra NISE 3500-Systeme der Firma Spectra GmbH & Co. KG haben jeweils als Betriebssystem lubuntu (Derivat der Linux Distribution Ubuntu) installiert und bilden die Plattform für die Komponenten *Log* und *Sensor-Fusion*:

Log

Diese Hardwarekomponente zeichnet jeglichen Netzwerkverkehr im Echtzeit-Ethernet-Backbone auf, um diesen zu analysieren und zu vergleichen mit den CAN-Paketen der *CAN-Gateway*s.

Sensor-Fusion

Für die Erstellung einer Karte für die Hindernis- und Gegenstandserkennung verwertet dieses System die Sensor-Informationen der *LIDAR-Gateway*s und des *Camera-Gateway Front*.

2.3.4. TTESwitch 100Mbit

Im Backbone sind drei TTEthernet-Switches der Firma TTTech Germany GmbH implementiert (in Abbildung 2.4 weiß). Diese regeln den priorisierten Verkehr der verschiedenen TTEthernet-Nachrichten mit einer Übertragungsrate von bis zu 100 Mbit/s.

3. Analyse / Anforderung an das System

Dieses Kapitel befasst sich mit vier Teilgebieten. Zuerst wird auf die Anforderungen von SNMP im Time-Triggered-Ethernet des CoRE-Backbones eingegangen. Daraufhin wird analysiert, welche Aufgaben die verschiedenen Akteure (Überwacher und zu Überwachende) in diesem Überwachungssystem haben und welche zeitliche Anforderungen an diese gestellt wird. Als nächstes werden die Werte beschrieben, die die zu überwachenden Geräte bereitzustellen haben. Zuletzt wird auf das noch zu entwickelnde Hardware-Board "Sensor-Board" eingegangen. Dort wird erläutert, wie dieses Board aufgebaut werden soll, welche Sensoren angeschlossen werden und welche Schnittstellen es bereitzustellen hat. Daraufhin werden die Anforderungen in einer Tabelle zusammengefasst.

3.1. Protokoll

Netzwerk

Wie im Kapitel 2 näher erläutert, ist in dem CoRE-Backbone ein Time-Triggered-Ethernet implementiert. Da durch dieses Netzwerk wichtige zeitkritische Frames fließen, sollen die Protokolldaten das Netzwerk nicht stören, solange höherpriorisiertere Nachrichten (Time-Triggered oder Rate-Constrained) das Netzwerk belegen. Dazu existiert der Nachrichtentyp "Best-Effort-Messages". Nachrichten mit diesem Typ haben niedrigste Priorität und bieten des weiteren Standard-Ethernet-gemäß keine Garantie, ob und wann die Daten versendet werden, beziehungsweise ankommen.
Da der Verlust von einzelnen Protokollnachrichten jedoch nicht kritisch hinsichtlich der Sicherheit des Systems ist, sollen die Nachrichten zur Überwachung dieses Systems über den Nachrichten Typ "Best-Effort-Message" versendet werden.

3.2. Akteure

Überwacher

Da wie oben erwähnt die Werte nicht zeitkritisch gesendet werden sollen und das Netzwerk nicht unnötig ausgelastet werden soll, hat der Überwacher die Werte der zu überwachenden Geräte in einem Zeitintervall von fünf Sekunden abzufragen.
Da die SNMP-Nachrichten per UDP versendet werden (siehe Kapitel 2.1), kann es vorkommen, dass einige Pakete nicht oder unvollständig übertragen werden. Das heißt, dass eine ausbleibende Antwort des zu überwachenden Gerätes nicht zwangsläufig dessen Ausfall bedeutet. Um dieser Fehlinterpretation entgegenzuwirken, wird eine Toleranz von fünf Anfragen (pro Sekunde eine) des Überwachers geschafft. Treffen weiterhin keine Antworten ein, wird ein Ausfall des Gerätes angenommen.
Bei einem Ausfall eines Gerätes oder wenn die Werte einen zu definierenden Grenzbereich überschreiten, soll der Überwacher dem Nutzer über eine Pop-Up-Nachricht informieren. Welches Gerät als Überwacher fungieren soll, ist noch zu bestimmen. Dies kann entweder ein bereits im Netz bestehendes Gerät oder ein durch einen Diagnose-Anschluss (weiterer verfügbarer Ethernet-Port zum Echtzeit-Ethernet-Backbone des CoRE-Prototypfahrzeuges) angeschlossener Laptop sein. Ein bereits bestehendes Gerät wäre sinnvoll hinsichtlich einer automatischen Überwachung des Systems. Da jedoch lediglich festgelegt ist, dass der Manager dem Benutzer bei Überschreitungen von Sensor-Werten über Pop-up-Nachrichten zu benachrichtigen hat, würde sich ein externes Gerät, bei dem der Nutzer mit diesem interagieren kann, anbieten.

Zu Überwachende

Da bei manchen Werten gegebenenfalls noch Berechnungen durchzuführen sind, sollen diese vorher und nicht erst auf Abfrage des Überwachers erfolgen, um möglich resultierende Time-Outs zu verhindern. Da der Überwacher alle fünf Sekunden die zu Überwachenden abfragt, sollen diese Berechnungen ebenfalls alle fünf Sekunden getätigt werden. Zu diesen Berechnungen zählt auch das Abfragen der Werte des Sensor-Boards. Des Weiteren haben die zu überwachenden Geräte die Werte auf Anfrage des Überwachers in einer Sekunde zu antworten. Diese eine Sekunde entspricht dem Time-out des Überwachers.
Da jedoch die Hauptaufgabe der jeweiligen Geräte Priorität hat, ist dies meist nicht möglich. Hat das jeweilige Gerät neben der Hauptaufgabe ein freies Zeitfenster der CPU, so hat dieses dann dem Überwacher zu Antworten.

3.3. Zu überwachende Werte

Die ausgewählten Werte zur Überwachung wurden hinsichtlich der Ausfallsicherheit eines jeweiligen Gerätes ausgesucht. Folgende Werte sollen zur Überwachung bereitgestellt werden:

- CPU-Auslastung
- CPU-Temperatur
- Arbeitsspeicher-Auslastung
- Netzwerk-Auslastung
- Strom-Verbrauch
- Anliegende Spannung

CPU-Auslastung

Die CPU-Auslastung ist der prozentuale Anteil, die der Prozessor tatsächlich an Prozessen arbeitet und sich nicht im Leerlauf (Idle-Zustand) befindet. Ist die CPU zu stark ausgelastet, werden Prozesse gegebenenfalls keine Rechenzeit zur Verfügung gestellt oder erst nach einer zu langen Wartezeit. CPU-Last ist vor allen Dingen in zeitkritischen Systemen ein wichtiger Faktor für dessen Sicherheit.

CPU-Temperatur

Wenn der Prozessor oder ein Mikrocontroller durch unzureichende Kühlung (aktiv mit Lüfter oder passiv mit lediglich einem Kühlkörper) eine zu hohe Temperatur erreicht, kann es gegebenenfalls zu einem Absturz des Gerätes führen.

Arbeitsspeicher-Auslastung

Je mehr der Arbeitsspeicher ausgelastet ist, desto höher ist die Wahrscheinlichkeit, dass mehr Seitenfehler (Page-Faults) auftreten. Seitenfehler treten auf, wenn auf Programmspeicher zugegriffen wird, dieser sich jedoch nicht im Arbeitsspeicher befindet, sondern ausgelagert wurde auf die Festplatte (vgl. Tanenbaum (2009)). Wenn so ein Fehler auftritt, kommt es zu einer Programmunterbrechung, damit der zu ladende Programmspeicher nachgeladen wird und gegebenenfalls anderer nicht genutzter Speicher ausgelagert. Page-Faults sind im

eigentlichem Sinne keine Fehler und treten häufig im alltäglichen Betrieb auf. Wenn jedoch durch die zu hohe Auslastung fortwährend Seitenfehler auftreten, so wird das System durch die Programmunterbrechungen und gegebenenfalls folgende langsame Festplattenzugriffe aufgehalten. Diese Anforderung bezieht sich lediglich auf das Kontron pITX-SP- und spectra NISE 3500-System, da diese im Gegensatz zu dem Hilscher NXHX 500-ETM-Board eine virtuelle Speicherverwaltung besitzen.

Netzwerk-Auslastung

Die Netzwerk-Auslastung spiegelt die eingehende wie auch ausgehende Last eines Interfaces wider. Eine hohe Netzwerk-Auslastung ist ein Risiko für die ein zeitkritisches System. Dies wird mittels dem TTEthernet kontrolliert durch Priorisierungen und Zeitslots unter Kontrolle gehalten. Jedoch ist auch so ein Netzwerk durch dessen Bandbreite (im CoRE-Backbone bis zu 100 MBit/s) limitiert und sollte hinsichtlich deren Netzwerk-Auslastung überwacht werden.

Strom-Verbrauch und anliegende Spannung

Die meisten Geräte verfügen über keine eigene Sensoren zur Strom- wie auch der Spannungsmessung. Dementsprechend soll dafür das Sensor-Board gebaut werden, damit der Strom-Verbrauch (in Ampere) wie auch die anliegende Spannung (in Volt) gemessen und dem Überwacher dann zur Verfügung gestellt wird.

3.4. Sensor-Board

Da den Hilscher NXHX 500-ETM-Boards Sensoren für CPU-Temperatur, Strom-Verbrauch und anliegende Spannung fehlen (vgl. Hilscher Gesellschaft für Systemautomation mbH (2005)), müssen diese Sensoren nachgerüstet werden. Um eine Erweiterbarkeit für andere Geräte im Netz zu ermöglichen und um zusätzliche Rechenleistung zur Auswertung der Daten zu ersparen, wird dazu ein weiteres Hardware-Board (”Sensor-Board”) entwickelt.

Universal anwendbar

Das Board soll hinsichtlich der Schnittstelle zur Datenübermittlung an das jeweilige zu überwachende Gerät wie auch der Anschlüsse für die verschiedenen Sensoren so gebaut werden, dass diese nicht nur speziell am Hilscher NXHX 500-ETM-Board Verwendung finden, sondern auch an andere Geräte angeschlossen werden können. Jegliche Sensoren sollen als Module an

die Steuereinheit angeschlossen werden. Dazu soll ein Busssystem genutzt werden, damit eine unnötige Verdrahtung erspart wird.

Steuereinheit

Die Steuereinheit des Sensor-Boards hat eine serielle Schnittstelle bereitzustellen. Diese Schnittstelle soll eine Kommunikation zwischen dem zu überwachenden Gerät ermöglichen, damit ein Austausch der Sensor-Informationen gelingt. Aufgrund der vergleichbar niedrigen Werte-Update-Intervalls des Überwachers (siehe Unterkapitel 3.2 "Zu Überwachende") soll der Mikrocontroller zugunsten des eigenen Strom-Verbrauches ebenfalls nur alle fünf Sekunden dessen eigene Module abfragen und gegebenenfalls diese Modul-Daten aufbereiten.
Um Energie zu sparen, soll sich diese Steuereinheit außerhalb des Messens und der Kommunikation zum überwachenden Gerät im Schlafmodus befinden.

Zeitliche Anforderungen

Das Board hat auf Anfrage des Überwachers umgehend (unter einer Sekunde) zu reagieren und die dementsprechenden Sensor-Werte zu übermitteln.

Größe des Boards

Da das Sensor-Board in dieser Arbeit für eingebettete Systeme im Auto gebaut wird, stellt sich aufgrund des Platzmangels zwangsläufig die Frage, wie groß das Sensor-Board werden soll. Die Hilscher NXHX 500-ETM-Boards im VW Golf 7 der CoRE-Gruppe sind hinten im Kofferraum und vorne im Handschuhfach des Autos verbaut. Beispielhaft wird nur für eines der im Kofferraum verbauten Hilscher NXHX 500-ETM-Boards das Sensor-Board entwickelt. Diese sind in einem Gehäuse eingebaut:

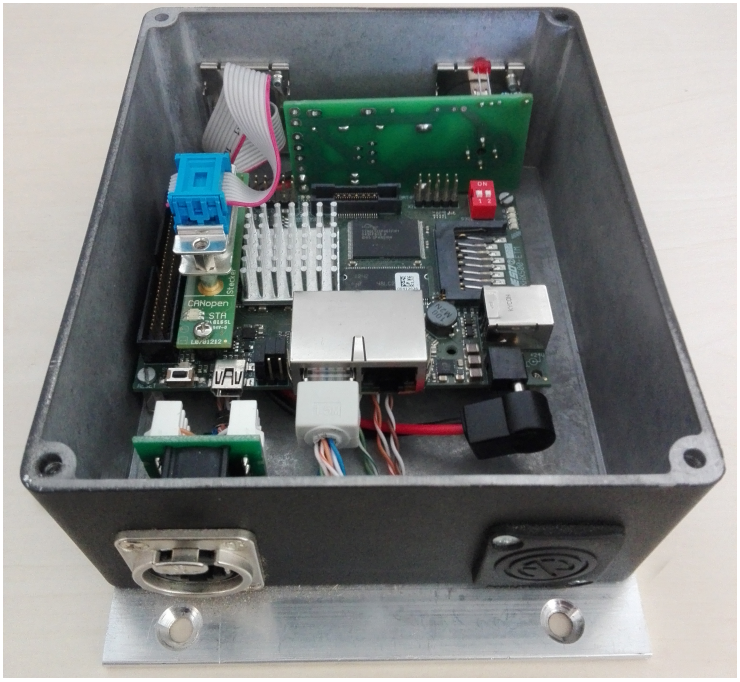


Abbildung 3.1.: Über dem Hilscher NXHX 500-ETM-Board im Gehäuse soll das Sensor-Board verbaut werden

Wie in Abbildung 3.1 zu sehen ist, ist im oberen Teil des Gehäuses ein Bereich über dem Hilscher NXHX 500-ETM-Board frei. Dieser Bereich wurde ausgemessen: 11 * 8 * 2 cm (Länge * Breite * Höhe). Diese Maße sind die maximale Größe des zu bauenden Sensor-Boards für die Montage über dem Hilscher NXHX 500-ETM-Board.

3.5. Zusammenfassung der Anforderungen

Die oben genannten Anforderungen werden nun in einer Tabelle zusammengefasst:

Betroffene Instanz	**Name**	**Wert / Einheit**
Protokoll	TTE-Nachrichtentyp	Best-Effort-Message
Überwacher	Abfrage-Intervall (Agenten)	5 Sekunden
	Time-out nach einer Abfrage	1 Sekunde
	Anz. Time-out bis angenom. Ausfall	5 Time-outs
Zu Überwachende	Abfrage-Intervall (zu überw. Werte)	5 Sekunden
	Abfrage-Intervall (Sensor-Board)	5 Sekunden
	Reaktionszeit zum Antworten	< 1 Sekunde
Sensor-Board	Max. Größe (L * B * H)	11 * 8 * 2 cm
	Schnittstelle zum zu überw. Gerät	Serielle Schnittstelle
	Reaktionszeit zum Antworten	< 1 Sekunde
	Schnittstelle zu Sensoren	Bussystem
	Anzuschließende Sensoren	Thermometer, Strommesser, Spannungsmesser
	Abfrage-Intervall (Sensoren)	5 Sekunden
Zu überwachende Werte	CPU-Auslastung	Prozent
	CPU-Temperatur	Grad Celsius
	Arbeitsspeicher-Auslastung	Prozent
	Netzwerk-Auslastung	Prozent
	Stromverbrauch	Ampere
	Anliegende Spannung	Volt

Tabelle 3.1.: Anforderungen an die verschiedenen Systeme

4. Konzept

Dieses Kapitel beschäftigt sich mit den verschiedenen Möglichkeiten, wie SNMP in den überwachenden und den unterschiedlichen zu überwachenden Geräten implementiert werden kann. Des Weiteren wird untersucht, wie die zu überwachenden Informationen MIB-technisch aufbereitet und bewertet werden können. Im letzten Unterkapitel wird das Sensor-Board in Komponenten und Modulen aufgeteilt und analysiert, wie diese miteinander interagieren und letztendlich die Informationen vom Sensor-Board zum überwachenden Gerät gelangen.

Da SNMP als Überwachungs-Protokoll genutzt wird, wird dementsprechend der Überwacher nun ”Manager”, die zu überwachenden Geräte ”Agenten” genannt, um Verwechslungen zu vermeiden und eine Eindeutigkeit beizubehalten. Zudem wird aufgrund, dass MIB-Module meist nur MIB genannt werden (siehe Kapitel 2.1.3), diese Gebräuchlichkeit beibehalten.

4.1. SNMP

4.1.1. Agenten

Kontron pITX-SP und spectra NISE 3500

Es würde sich anbieten, unabhängig vom System die selbe Software zu nutzen, welche die zu überwachenden Daten in die Management Information Base einträgt und diese auf Anfrage des Managers ihm zusendet. Für das Betriebssystem gentoo (Kontron pITX-SP) wie auch lubuntu (spectra NISE 3500) bietet sich die Software-Suite ”Net-SNMP” an. Diese läuft unter BSD-Lizenz (die Software kann unter anderem frei verwendet, geändert und verbreitet werden, vgl. The FreeBSD Project). Da Net-SNMP Open-Source ist, ist es einfach möglich, das Programm nach eigenen Anforderungen anzupassen. Vor allen Dingen kann in einem Echtzeit-System diese Dynamik zum Vorteil der Ressourcenschonung verwendet werden. Des Weiteren ist Net-SNMP durch seine große Verbreitung und Unterstützung für viele unixoide Betriebssysteme wie auch verschiedene Windows-Systeme.

Hilscher NXHX 500-ETM

Die Geräte, die im CoRE-Backbone integriert sind, arbeiten durch das Real-Time-Ethernet lediglich auf TCP/IP-Schicht 1 - Netzzugang.
Um SNMP und somit UDP zu realisieren, bieten sich zwei Möglichkeiten an:
Eine weitere Bachelorarbeit der CoRE-Gruppe, die von Piotr Konieczny parallel zu dieser entwickelt wurde, befasst sich mit der Implementierung eines TCP/IP-Stacks auf den Hilscher NXHX 500-ETM-Boards. Dieser Stack heißt lwIP (lightweight IP) und ist ein Open-Source-Stack, welcher speziell für eingebettete Systeme (besonders Mikrocontroller) entwickelt worden ist, um Protokolle über der TCP/IP-Schicht 2 (speziell TCP, aber auch UDP) zu nutzen. lwIP bietet auch eine Implementation von SNMP an.
Die zweite Möglichkeit wäre, die sechs verschiedenen SNMP-Nachrichtentypen selber zu schreiben, anstelle ein TCP-IP-Stack zu nutzen. Da für SNMP das Protokoll UDP verwendet wird, fallen netzwerktechnische Aktionen wie Verbindungsaufbau, Congestion-Control, Flowcontrol etc. nicht an. Dies bietet den Vorteil, dass somit nur das nötigste an SNMP und keine weiteren nicht benötigten Protokolle implementiert werden. Dennoch ist dadurch die Erweiterbarkeit sehr eingeschränkt.

Aktualisierung der MIB

Da sich die zu überwachenden Informationen stetig verändern, müssen diese Änderungen in die MIB eingetragen werden. Dazu gibt es zwei Möglichkeiten: Entweder werden die Informationen beim Agenten bei jeder Abfrage vom Manager aktualisiert oder - unabhängig vom Manager - in einem bestimmten Zeitintervall. Die erste Möglichkeit bietet den Vorteil, dass somit die CPU nicht unnötig belastet wird, wenn der Manager keine Anfragen stellt. Des Weiteren ist eine komplexere Konfiguration des Agenten nicht notwendig für die Intervall-Einstellungen und die gegebenenfalls notwendigen Einstellungen für den wechselseitigen Ausschluss bei den Variablen-Manipulationen. Die zweite Möglichkeit wird jedoch ausgeschlossen, da die Anforderungen spezifizieren, dass ein Agent aufgrund der möglich entstehenden Manager-Timeouts die Managed Objects vorher aktualisiert haben soll.

4.1.2. Manager

Da bei den Kontron pITX-SP- und spectra NISE 3500-Agenten die Software-Suite Net-SNMP verwendet wird, wäre es vorteilhaft, bei dem Manager ebenfalls diese Suite zu nutzen. Da ein Net-SNMP-Manager lediglich konfiguriert, jedoch nicht weiter programmiert werden muss, ist dessen Einrichtung simpler gegenüber den Net-SNMP-Agenten.

4.1.3. Management Information Base

Letztendlich stehen drei Möglichkeiten der Informationsaufbereitung für die zu überwachenden Informationen an dem SNMP-Manager zur Diskussion:

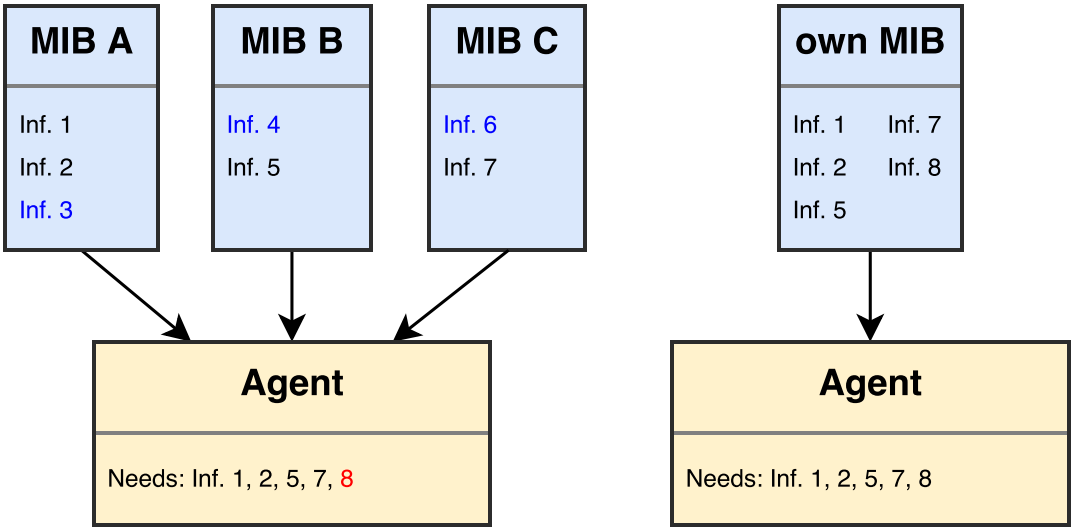


Abbildung 4.1.: Mögliche MIB-Informationsaufbereitungen

Bestehende Information aus anderen MIBs beschaffen

Da bereits viele Standard-MIBs existieren, um Systeme zu überwachen, können notwendige Informationen (Managed Objects) aus bereits bestehenden MIBs abgefragt werden (vgl. Abb. 4.1 Agent links). Dies würde den Vorteil bringen, dass diese Daten und Sensor-Informationen nicht manuell nachimplementiert werden müssen. Damit würde ein erheblicher Arbeitsaufwand wegfallen, da so nur von der Manager-Seite aus die Informationen abgegriffen werden müssten. Dies setzt jedoch voraus, dass die notwendigen Informationen in den MIBs bereits bestehen. Ist dies nicht der Fall (siehe fehlende rote Information 8), gibt es keine Möglichkeit der Erweiterbarkeit. Des Weiteren werden dennoch nicht benötigte Informationen (siehe blaue Informationen 3, 4, 6) implementiert.

Eigene MIB definieren

Eine Alternative zu Daten aus verschiedenen Quellen abzurufen, wäre, diese Daten selber in einer eigenen Management Information Base zu integrieren (vgl. Abb. 4.1 Agent rechts).

Eine eigene MIB bietet eine erhebliche Erweiterbarkeit gegenüber dem obengenannten Konzept. Jedoch ist somit ein vergleichsweise hoher Implementierungsaufwand notwendig für die Informationsbeschaffung, welche gegebenenfalls bereits gegeben wäre. Je nach Implementierung kann so auch ein großer Teil an Speicherplatz eingespart werden, wenn andere nicht notwendige MIBs nicht mit eingebunden werden.

Hybrid

Eine weitere Möglichkeit wäre ein Hybrid aus beiden obengenannten Konzepten. Diejenigen Informationen, welche noch nicht in anderen MIBs vorhanden sind, müssen in einer neuen MIB definiert werden. Würde man diesem Konzept folgen, so müsste die in Abbildung 4.1 genannte Information 8 alleine in einer eigenen MIB definiert werden.
Im Vergleich zu den oberen Konzepten wäre der Arbeitsaufwand am geringsten, jedoch die Speicher-Auslastung am höchsten.

4.2. Sensor-Board

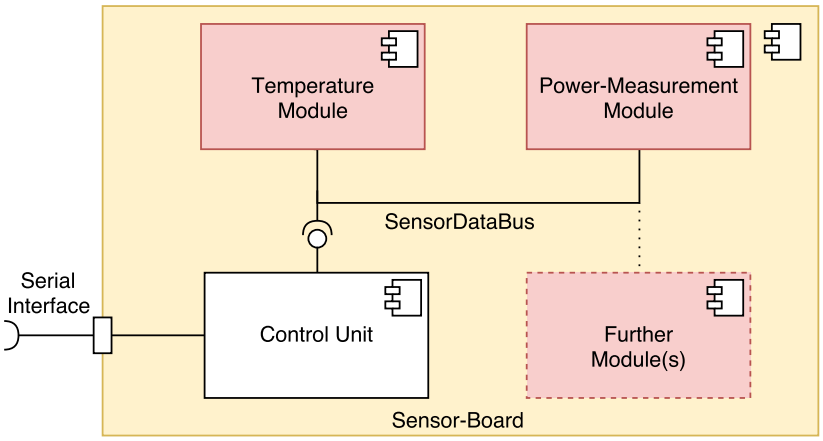


Abbildung 4.2.: Modulare Beschreibung des Sensor-Boards

Wie in Abbildung 4.2 zu sehen ist, besteht das Sensor-Board aus mehreren Komponenten. Die Hauptkomponente ”Control Unit” (Steuereinheit) stellt die Sensor-Daten bereit und sendet diese dem SNMP-Agenten auf dessen Abfrage über eine Schnittstelle zu. Es ist zu überlegen, dieses Board als Sub-SNMP-Agenten zu implementieren. Somit müsste das Sensor-Board auch netzwerkfähig sein und ein Stack implementiert bekommen, damit das Protokoll SNMP genutzt werden kann. Jedoch müsste dann dieses Board auch mit am Echtzeit-Ethernet-Backbone

angeschlossen werden. Entsprechend simpler wäre eine serielle Schnittstelle (RS-232 oder UART) zu betrachten, um die Sensor-Informationen zu übermitteln. Des Weiteren muss, da es keine Standards für Protokollnachrichten für serielle Schnittstellen gibt, ein entsprechendes Protokoll genutzt werden. Die Sensoren werden mittels Modulen (in Abb. rot) über ein Bussystem an die Steuereinheit angeschlossen. Dieses Bussystem bietet den Vorteil, dass noch weitere Module angeschlossen werden können, ohne dass eine weitere Verdrahtung an der Steuereinheit notwendig ist.

Die Komponente ”Temperature Module” misst mittels eines anzubringenden Temperatur-Sensors die CPU-Temperatur des Agenten. Das Modul ”Power-Measurement Module” misst die anliegende Spannung und den durchfließenden Strom an den Agenten. Diese Daten werden dann wie oben erwähnt über ein Bussystem an die Steuereinheit gesendet. Da ein Bussystem genutzt wird, können auch somit weitere Module (”Further Modules”) hinzugefügt werden.

4.2.1. Steuereinheit

Es zu entscheiden, ob ein Entwicklungsboard oder ein einzelner Mikrocontroller als Steuereinheit genutzt werden soll:

Entwicklungsboard

Entwicklungsboards haben den Vorteil, dass auf denen ein Mikrocontroller und dessen notwendige Elektrobauteile bereits vorverbaut sind. Des Weiteren haben viele Entwicklungsboards vom Hersteller bereits eine HAL (Hardware Abstraction Layer) integriert, sodass damit die Implementierung erleichtert wird.

Einzelner Mikrocontroller

Da ohnehin ein Board entwickelt werden muss, könnte es aus Platzgründen vorteilhafter sein, einen einzelnen Mikrocontroller zu nutzen anstelle eines großem Entwicklungsboards. Auch die Kosten eines Mikrocontrollers sind erheblich geringer als die eines Entwicklungsboards. Jedoch wird zu einem Mikrocontroller meist auch ein teurer Programmer benötigt.

4.2.2. Wahl der Sensoren

Strom- und Spannungsmessung

Spannungsmessung ist bei Mikrocontrollern auf erste Sicht relativ simpel. Diese haben analoge Eingänge (Analog-Digital-Converter), dessen Wertebereich beispielhaft von 0 bis 1023 eine jeweilige Spannung von 0 bis 5 Volt widerspiegeln können. Da die Geräte jedoch von einer Autobatterie betrieben werden, welche eine Spannung von 12 Volt ausgibt, muss somit eine Alternative geschaffen werden wie mit Strom- und Spannungsteilern, welche mithilfe von Widerständen zu lasten des Stromverbrauchs Strom in Wärme umwandeln.
Eine weitere Methode um Strom zu messen, wäre einen zusätzliches Modul zu nutzen. Dies erspart einerseits Rechenoperationen auf der Seite der Steuereinheit und bietet seine höhere Genauigkeit hinsichtlich der Spannungs- und Stromteiler.

Temperatur

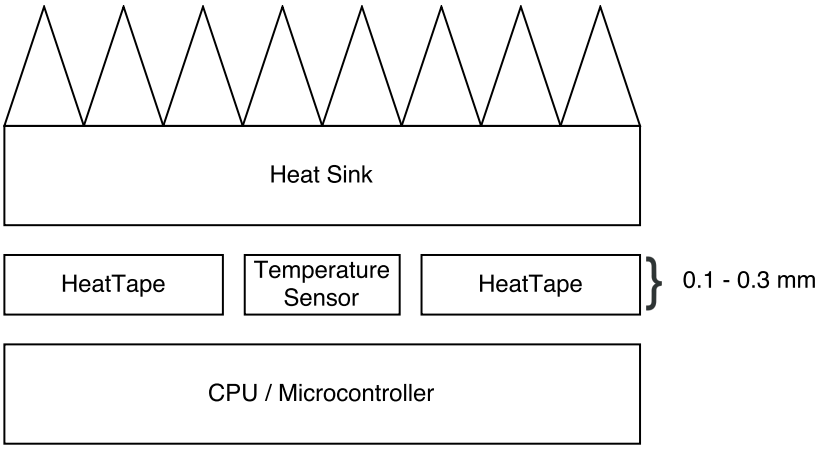


Abbildung 4.3.: Temperatur-Sensor zwischen CPU/Mikrocontroller und Kühlkörper

Wie in Abbildung 4.3 zu sehen ist, muss der Temperatur-Sensor flach genug sein, damit dieser bei dem zu messenden Gerät zwischen der CPU (beziehungsweise Mikrocontroller) und dem sich darauf befindlichen Kühlkörper (”Heat Sink”) passt. Der netX50-Controller und dessen Kühlköperer sind dem Hilscher NXHX 500-ETM-Board mit wärmeleitfähigem Klebeband (”Heat Tape”) befestigt. Dieses Klebeband hat eine Breite von 0,1 bis 0,3 Millimeter. Dementsprechend muss der Temperatur-Sensor dieser Breite entsprechen.
Somit sind viele Temperatur-Sonden ausgeschlossen, da diese häufig einen Durchmesser von

einem bis zwei Millimeter haben. Zusätzlich sind diese meist zylinderförmig und würden bei Befestigung Probleme bereiten und durch ein eventuell dadurch entstehenden Freiraum, würde der Kühlkörper an Wirkung verlieren und gegebenenfalls die CPU überhitzten.
Alternativ gibt es Thermistoren. Dies sind Widerstände, welche je nach Temperatur entweder hochohmiger werden (Kaltleiter bzw. PTC) beziehungsweise je wärmer diese werden, mehr elektrischen Strom leiten (Heißleiter, NTC). Folglich wird somit nicht die Temperatur gemessen, sondern der Widerstand. Mit dem gemessenen Wert wird daraufhin die entsprechende Temperatur entweder mittels einer Look-Up-Table oder verschiedener Formeln bestimmt. Eine Look-Up-Table bietet den Vorteil, dass nur der Wert gemessen werden muss und in der Tabelle einen entsprechenden Temperaturwert entnommen werden kann. Jedoch wird je nach Genauigkeit mehr Speicher benötigt. Durch die Nutzung einer Formel wird weniger Speicher genutzt auf Kosten des Rechenaufwandes und der Rechenzeit.

5. Realisierung

Auf der Grundlage der in Kapitel 4 entworfenen Konzepte, steht dieses Kapitel nun im Fokus der Umsetzung. Der erste Teil widmet sich der anhand der Anforderungen erstellten Management Information Base. Diese wurde daraufhin bei dem Manager und den Agenten implementiert. Danach wird auf die genutzte SNMP-Version und später auf die Fertigstellung des Sensor-Boards und dessen Kommunikation mit dem zu überwachenden Gerät eingegangen.

5.1. Management Information Base

Aus folgenden Entwurfskriterien wurde letztendlich wurde eine eigene MIB mit dem Namen *CoRE-MIB* (siehe Anhang A) definiert:

- Beliebige Erweiterbarkeit
- Geringe Speicher-Auslastung
- Unüberschaubarkeit

Da bei der MIB-II die Managed Objects CPU-Temperatur, Stromverbrauch und anliegende Spannung nicht implementiert sind (siehe RFC123 (McCloghrie und Rose, 1991)), war eine neue MIB unabdingbar. Ein daraus folgender Hybrid (siehe Kapitel 4.1.3) wäre möglich, ist jedoch aufgrund einer resultierenden Unüberschaubarkeit, an welcher Stelle welche Managed Objects entnommen werden und einer zusätzlichen Speicher-Auslastung ausgeschlossen worden.
Weil dennoch Informationen genutzt werden, die ebenso in der Standard-MIB-II existieren (wie zum Beispiel ”sysName”), müssen diese somit in der CoRE-MIB neu definiert werden, behalten aber die ursprüngliche Definition mit deren ursprünglich zugehörigen OID (siehe Anhang A). Werte, die aus unten erwähnten Gründen dennoch nicht bei einigen Agenten implementiert werden können, bekommen einen Standardwert (siehe ”DEFVAL” in Anhang A). Daraus folgt diese Management Information Base:

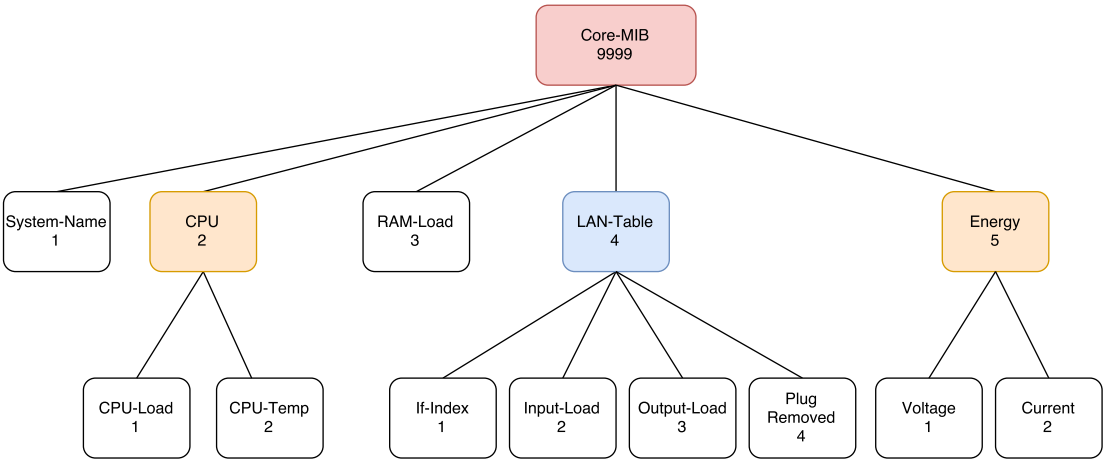


Abbildung 5.1.: Management Information Base für CoRE

Wie in Abbildung 5.1 zu sehen ist, besteht der Baum der eigen definierten Management Information Base aus neun Managed Objects (weiße Blätter). Diese wurden zum Teil je in Gruppen (orangefarbene Knoten) zusammengefasst. Der blaue Knoten stellt eine MIB-Tabelle dar.
Aus Gründen der Übersichtlichkeit und einfacheren Lesbarkeit entsprechen die in dieser Abbildung angegebenen Namen nicht unbedingt den Namen, wie diese letztendlich definiert wurden (siehe Anhang A).
Da diese MIB für eine Forschungs- und Testzwecken definiert wurde, wurde hierfür keine ”Private Enterprise Number” bei der IANA beantragt. Dementsprechend wurde der Core-MIB eine noch freie private Enterprise-Nummer 9999 genommen und im enterprise-Baum (siehe Abbildung 2.2) eingehängt. Somit ist diese MIB unter der OID *{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) coreMIB(9999)}* erreichbar.
Wie in Kapitel 2 erläutert wurde, existiert nur eine kleine Auswahl an verfügbaren Datentypen für Management Information Bases. Somit wurde der Datentyp *Unsigned32* für jegliche positive variable Zahl und *Integer32* für jede Zahl, die auch einen negativen Wert erreichen kann, genutzt. Die genutzten Datentypen und eine gegebenenfalls genutzte Bereichseinschränkung sind hinter die jeweiligen OID-Namen geschrieben.

System-Name - *OCTET STRING* (Size (0..255))
Der System-Name gibt den Domänen-Namen des jeweiligen Gerätes an. Diese OID existiert bereits in der MIB-II mit dem Namen ”sysName”. In beiden MIBs wird dieser Name als ein von 0 bis 255 Zeichen definiert. Es wurde wie oben erwähnt die dafür originale Definition übernommen und dessen Ursprungs-OID angegeben.

CPU

CPU-Load - *Unsigned32 (0..100)*
Die CPU-Load gibt die jeweilige Prozessor-Auslastung eines Gerätes in Prozent an. Dies ist die Zeit innerhalb einer Sekunde, die der Prozessor nicht im Idle-Zustand verbringt. Hat der Prozessor mehrere Kerne, so wird jede einzelne Auslastung berechnet und aus denen ein Mittelwert gebildet.

CPU-Temp - *Integer32*
Die Temperatur der CPU wird in Grad Celsius angegeben.

RAM-Load - *Unsigned32* (0..100)
Die Arbeitsspeicher-Auslastung berechnet sich aus dem Verhältnis vom genutzten zum gesamten Arbeitsspeicher an:

$$RAM_{Load} = \frac{100}{RAM_{Total} * RAM_{Used}}$$

LAN-Table

If-Index - (*Integer32* (1..)
If-Index (Interface-Index) bezeichnet einen einzigartigen Identifier für jedes Interface eines Gerätes.

Input und Output-Load - *Unsigned32* (0..100)
Die Input-Auslastung bezeichnet das Verhältnis der seit einer Sekunde eingehenden Bytes (”ifInOctets”) zur jeweiligen Geschwindigkeit des Interfaces (”ifSpeed”). Diese Berechnung basiert auf einer Methode der Firma Cisco Systems GmbH, um Netzwerk-Auslastungen über SNMP zu berechnen:

$$Input\,utilization = \frac{\Delta ifInOctets * 8 * 100}{(number\,of\,seconds\,in\,\Delta) * ifSpeed}$$

Analog zur Input-Load berechnet sich die Output-Load anstatt der eingehenden mit den ausgehenden Octets (”ifOutOctets”) :

$$Output\,utilization = \frac{\Delta ifOutOctets * 8 * 100}{(number\,of\,seconds\,in\,\Delta) * ifSpeed}$$

Ist das Interface nicht Full-Duplex-betrieben (Half-Duplex), so werden die eingehenden wie auch ausgehenden Octets summiert und so zeitlich ins Verhältnis zur Interface-Geschwindigkeit gesetzt:

$$Utilization = \frac{(\Delta ifInOctets + \Delta ifOutOctets) * 8 * 100}{(number\,of\,seconds\,in\,\Delta) * ifSpeed}$$

Dieser Wert wird daraufhin beiden Managed Objects dann zugeordnet.

Plug Removed - *Unsigned32* (0..1)
Plug Removed gibt mit entweder 0 oder 1 an, ob sich der Ethernet-Stecker physisch noch im Port des Interfaces befindet oder ob das Interface initialisiert ist.

Energy

Voltage - *Integer32*
Dieser Wert zeigt die an dem Agenten anliegende Spannung in Volt an.
Current - *Integer32*
Current gibt zeigt den anliegenden Strom in Milliampere am Agenten an.

5.2. Implementierung der MIB

Im Folgenden wird erläutert, wie auf der Seite des Managers (Net-SNMP) und auf den Seiten der Agenten (Net-SNMP und lwIP) diese MIB implementiert ist.

5.2.1. Net-SNMP-Manager

Da der Manager die verschiedenen Managed Objects der Agenten auch nur numerisch über deren OID ansprechen kann (wie zum Beispiel mit 1.3.6.1.4.1.9999.2 für die CPU-Auslastung, siehe Kapitel 2.1.3), ist es nicht unbedingt notwendig, die MIB nachzuimplementieren. Da Net-SNMP jedoch auch die Funktion anbietet, zu verifizieren, ob die empfangenen Daten auch zum Beispiel dem richtigen Datentypen entsprechen und automatisch die jeweiligen Namen den OIDs zuordnet, ist es doch zu empfehlen, die MIB auch auf bei dem Manager zu implementieren. Dazu muss lediglich die MIB-Datei in den Order *~/.snmp/mibs/* kopiert werden und in der Net-SNMP-Konfigurationtei (*/etc/snmp/snmp.conf/*) die Textzeile "*mibs +CORE-MIB*" hinzugefügt werden.

Damit die Agenten überwacht werden, wurde ein Shell-Skript entworfen, welches alle fünf Sekunden (siehe Kapitel 3.2: Überwacher) jeden Agenten mittels dem Net-SNMP-Befehl *getBulkWalk* dessen *CoreMIB* abfragt.

5.2.2. Net-SNMP-Agenten

Es gibt mehrere Möglichkeiten bei Net-SNMP eine eigene MIB zu implementieren. Die erste simplere Methode ist, dies über einen sogenannten Sub-Agenten zu realisieren. Dies ist ein weiterer Agent (Daemon), welcher auf einem Managed Device arbeitet. Der Sub-Agent kann in diesem Fall die CoRE-MIB implementieren. Daraufhin würde diese Managed Objects an den Haupt-Agenten weiterreichen, der diese dann an den Manager senden würde. Jedoch wird so auch die MIB-II mit implementiert.
Alternativ dazu kann auch der Haupt-Agent geändert werden. Dafür wird der Quellcode der Net-SNMP-Software-Suite heruntergeladen und nur die eigene MIB implementiert, ohne dass ein weiterer Sub-Agent parallel zum Hauptagenten arbeitet.
Viele Informationen, welche zur Berechnung der folgenden Managed Objects benötigt werden, werden aus Dateien ausgelesen. Lese- oder Schreibzugriffe auf Dateien dauern in der Regel lange. Da jedoch die genutzten Dateien lediglich virtuelle Dateien sind, welche Kernel-Informationen enthalten und nicht auf der Festplatte gespeichert werden (vgl. Alessandro Rubini), sind die Zugriffszeiten unbedenklich.

Die Vorgehensweise zur Implementierung der CoRE-MIB (siehe Anhang A) war folgende:

1. Durch *Configure* alle nicht benötigten MIBs ausschließen und CoRE-MIB einbinden
2. Code-Skeletons für die CoRE-MIB durch MIB-Parser erstellen lassen
3. Skeletons in Net-SNMP-Quellcode einfügen und entsprechend der CoRE-MIB implementieren
4. Kompilieren und anschließend installieren

Zuerst werden mit *Configure* alle nicht benötigten MIBs ausgeschlossen und nur die CorRE-MIB eingebunden (1).
Net-SNMP stellt einen eigenen MIB-Parser ”mib2c” bereit, der zu einer gegeben MIB-Datei Code-Skeletons erstellt (2).
Dieser erzeugte Code muss daraufhin entsprechend den eigenen Anforderungen angepasst werden (3), dass bei eintreffenden Manager-Requests die richtigen Werte gesendet und dass die

Werte aktualisiert werden. Fast alle CoRE-MIB-Werte müssen regelmäßig aktualisiert werden. Net-SNMP bietet eine Funktion an, die nach einer gegebenen Zeit eine angegebene Callback-Funktion auslöst, um zum Beispiel die Aktualisierungen durchzuführen.
Nachdem der Code entsprechend angepasst wurde, kann der Agent nun kompiliert und anschließend auf dem Managed Device installiert werden (4).

Die verschiedenen Managed Objects der CoRE-MIB wurden folgendermaßen implementiert:

System-Name
Mit der POSIX-Schnittstelle lässt sich der Host-Name durch die Funktion ”gethostname()” aus der Bibliothek ”unistd.h” (vgl. Michael Kerrisk (a)) auslesen. Da der System-Name nicht geändert wird während des Betriebes, wird dieser Wert lediglich einmalig bei der Initialisierung (beim Start) vom Net-SNMP-Daemon ausgelesen und nicht bei der oben genannten Callback-Funktion.

CPU

CPU-Load
Die Last wird anhand der sich aus der Zeit, die die CPU sich nicht im Idle-Zustand befindet. Bei Unix-Systemen findet man diese Informationen unter dem Pfad *proc/stat* (vgl. Michael Kerrisk (b)). In der ersten Zeile dieser Datei befinden sich unter anderem die Werte, die die CPU sich im ”User”- für ”Nice”- wie auch für ”System”-Prozesse arbeitet. Hinzu kommt der Wert, die Zeit, die die CPU im Idle-Zustand war. Somit lässt sich die Auslastung folgendermaßen ausrechnen:

$$CPU_{Load} = 100 * \frac{t_{User} + t_{Nice} + t_{System}}{t_{User} + t_{Nice} + t_{System} + t_{Idle}}$$

CPU-Temp
Bei lubuntu wie auch gentoo lässt sich die CPU-Temperatur aus einer Datei im System auslesen. Da diese jedoch sehr variieren, wird der Pfad zur Datei manuell in einer Konfigurationsdatei geschrieben und beim Start von Net-SNMP ausgelesen.

RAM-Load
Für die Arbeitsspeicher-Auslastung,wird die Datei ”/proc/meminfo” (vgl. Michael Kerrisk (b)) ausgelesen. Die absolute Anzahl an physikalisch verfügbarem RAM befindet sich in der Zeile ”MemTotal”. Da ”meminfo” den genutzten Arbeitsspeicher sehr detailliert untergliedert, müssen

dafür die einzelnen Werte addiert werden. Um Zeit und Rechenoperationen zu sparen wird daher zuerst nicht der prozentuale Anteil das genutzten Speichers zum gesamten gerechnet, sondern den des freien Speichers:

$$RAMLoad = 100 - \frac{100}{RAM_{Total} * RAM_{Free}}$$

Somit ist das Ergebnis äquivalent.

LAN-Table

Da diese MIB-Tabelle abhängig von der Anzahl der Netzwerk-Interfaces ist, müssen bei der Initialisierung diese identifiziert werden. Wie auch bei der CPU-Auslastung und -Temperatur wird hier auf das Dateisystem zugegriffen. Jegliche Netzwerk-Informationen bei Unix-Systemen befinden sich im Verzeichnis "/sys/class/net/" (vgl. Linux Kernel Organization). In diesem Verzeichnis sind die jeweiligen Interfaces aufgelistet.

If-Index

Die Interface-Nummer befindet sich in jedem jeweiligen Interface-Unterordner in der Datei "/<iface>/ifIndex". <iface> ist der jeweilige vom System vergebene Name des Interfaces.

Input und Output-Load

Wie im Kapitel 4 "Konzept" angegeben, werden für die Berechnung der Netzwerk-Auslastungen die Variablen *Δ ifInOctets*, *Δ ifOutOctets* und *ifSpeed* benötigt.
Die Variablen *ifInOctets* wie auch *ifOutOctets*, befinden sich bei dem jeweiligen Interface im Unterordner "/<iface>/statistics/rx_bytes" beziehungsweise "/<iface>/statistics/tx_bytes".
Die Interface-Geschwindigkeit *ifSpeed* ist in der Datei "speed" im jeweiligen Interface-Ordner auslesbar.
Da die Berechnung abhängig davon ist ob das Interface Full- oder nur Half-Duplex-betrieben ist, muss diese identifiziert werden. In der Datei "duplex" steht dementsprechend entweder der Wert "full" oder "half".
Ist das jeweilige Interface nicht initialisiert, so sind beide Werte 0.

Plug Removed

Unter der Datei "/<iface>/carrier" ist der "physical link state" angegeben. Dieser gibt mit "1" den Wert an, ob das Interface physikalisch verbunden ist. Da bei "Plug Removed" jedoch der gegenteilige Wert genutzt werden soll, wird der Wert invertiert.
Ist das jeweilige Interface nicht initialisiert, so ist der Wert 0.

Energy

Voltage und Current

Weder das spectra NISE 3500, noch das Kontron pITX-SP haben dafür Hardware eingebaut, die diese Werte berechnen und wurden somit nicht implementiert und besitzen den Standard-Wert 0.
Alternativ könnte jedoch das Sensor-Board dafür verwendet werden.

5.2.3. lwIP-Agenten

Letztendlich wurde der lwIP-Stack zur Nutzung von SNMP genutzt, da dessen Implementierung und bereitgestellte Tools für das Einrichten der Management Information Base sehr erleichtert wird. Da vermutlich in Zukunft auch weitere Protokolle genutzt werden sollen, die über TCP/IP-Schicht 2 arbeiten sollen, wäre somit auch eine hohe Erweiterbarkeit gegeben.

Die Kommunikation mit dem Sensor-Board wurde über eine serielle Schnittstelle (UART) realisiert.

Wie bereits erwähnt, hat Piotr Konieczny parallel zu dieser Bachelorarbeit lwIP für das Hilscher NXHX 500-ETM-Board eine Implementierung entwickelt. Dieser Stack wird genutzt und entsprechende angepasst, dass das Protokoll SNMP genutzt werden kann. Damit dieser Agent die eigene Management Information Base implementieren kann, existiert (wie auch bei Net-SNMP) ein Tool "LwipMIBCompiler", welches die MIB-Datei parst und daraus ein C-Code-Skeleton entwickelt. Die CoRE-MIB (siehe Anhang A) wurden daraufhin folgendermaßen implementiert:

System-Name

Der Hostname wird bei lwIPin der Datei "/LWIP/netif/ethernetif.c" bei der Initialisierung der Netzwerk-Interfaces initialisiert. Wie auch bei den Net-SNMP-Agenten wird dieser Wert nur einmal in die CoRE-MIB dieses Agenten eingetragen und nicht aktualisiert, da dies auch nicht zur Laufzeit möglich ist.

CPU

CPU-Load

Da dies ein Mikrocontroller ist, existiert hierfür keine Auslastung, da diese theoretisch bei 100 Prozent liegt. Als Alternative könnte jedoch eine Auslastung gemessen werden mit dem Anteil der Zeit, die in der while(1)-Schleife verbracht wird, zu den Interrupt-

Service-Routinen oder die Zeit, die der Mikrocontroller sich im Schlafmodus befindet, falls dieser das unterstützt.

CPU-Temp
Die Temperatur des Hilscher NXHX 500-ETM-Boards wird von dem Sensor-Board gemessen.
Das Hilscher NXHX 500-ETM-Board sendet über die serielle Schnittstelle als Request den Buchstaben ”t” (**t**emperature) und als Abschlusszeichen einen Zeilenumbruch (Steuerzeichen mit ASCII-Code: 0xA). Daraufhin wird aktiv auf eine Antwort des Sensor-Boards gewartet (Polling). Bei erfolgreichem Empfang ist die Antwort ’t’ und die darauffolgenden zwei Bytes der jeweilige numerische Wert für die Temperatur.
Kommt innerhalb einer Sekunde dennoch keine Antwort oder enthält der Empfangspuffer nicht die erwartete Antwort, so wird der Wert auf 0 gesetzt.

RAM-Load
Da dieses Entwicklungsboard über keine virtuelle Speicherverwaltung verfügt, existiert dementsprechend auch keine Arbeitsspeicher-Auslastung. Somit ist der Wert 0.

LAN-Table
Für die Erfassung der vier Werte der LAN-Table existierten zwei relevante Daten-Strukturen:

```
typedef struct ETHERNET_CONNECTION_STATE_Ttag {
  unsigned int uSpeed;
  unsigned int fIsLinkUp;
  unsigned int fIsFullDuplex;
} ETHERNET_CONNECTION_STATE_T
```

Listing 5.1: Ethernet-Verbindungsstatus-Datenstruktur aus der HAL des Hilscher NXHX 500-ETM

Diese Datenstruktur aus der Datei ”hal_eth_std_mac.h” (Listing 5.1, aus der HAL des Herstellers) lässt sich mittels der Funktion ”EthStdMac_GetConnectionState” für ein jeweiliges Interface abfragen.
Die Variable **uSpeed** gibt die Interface-Geschwindigkeit in Megabits pro Sekunde (10 oder 100) an.
Die Variable **fIsLinkUp** gibt den Link-Status an. Der Link-Status ist abhängig von der Initialisierung des Interfaces und ob der physische Stecker sich im jeweiligen Port befindet. Ist der

Link ”up”, so ist der Wert nicht 0.
Die Variable **fIsFullDuplex** gibt mit einem Wert ungleich 0 an, ob das Interface Full-Duplex sendet und empfängt. Ist der Wert 0, wird das Interface nur Half-Duplex-betrieben.

```
struct stats_mib2_netif_ctrs {
        u32_t ifinoctets;
        u32_t ifinucastpkts;
        u32_t ifinnucastpkts;
        u32_t ifindiscards;
        u32_t ifinerrors;
        u32_t ifinunknownprotos;
        u32_t ifoutoctets;
        u32_t ifoutucastpkts;
        u32_t ifoutnucastpkts;
        u32_t ifoutdiscards;
        u32_t ifouterrors;
};
```

Listing 5.2: Zähler-Datenstruktur für Netzwerk-Interfaces aus der Datei ”stats.h” des lwIP-Stacks

Die Datenstruktur ”stats_mib2_netif_ctrs” (Listing 5.2) des lwIP-Stacks enthält viele Zähler, welche beim Empfangen und Senden von Frames inkrementiert werden. Diese sind für die Standard-MIB-II ursprünglich implementiert worden. Da jedoch wie oben erwähnt nur eine eigene MIB genutzt wird (und somit nicht MIB-II), werden diese Zähler deswegen eigentlich nicht genutzt.

If-Index
Das Hilscher NXHX 500-ETM-Board besitzt zwei Netzwerk-Interfaces. Diese besitzen durch die HAL des Herstellers fix den jeweiligen Wert 0 und 1. Da jedoch die Interface-Nummern mit 1 beginnen, werden diese um 1 inkrementiert. Bei der Initialisierung der Interfaces mit lwIP werden diese inkrementierten Port-Nummern mit übergeben und eingetragen.

Input und Output-Load
Wie im Kapitel 4 ”Konzept” die Formel zur Netzwerk-Auslastung angegeben, werden für die Berechnung der Netzwerk-Auslastungen die Variablen $\Delta ifInOctets$, $\Delta ifOutOctets$ und *ifSpeed* benötigt.
Die gleichbenannten Variablen *ifInOctets* wie auch *ifOutOctets*, befinden sich in der oben

erwähnten Datenstructur ”stats_mib2_netif_ctrs”. Die Interface-Geschwindigkeit *ifSpeed* lässt sich aus der Datenstruktur ”hal_eth_std_mac.h” unter der Variable *uSpeed* auslesen. Um zu identifizieren, ob das Interface Full-Duplex-betrieben wird„ wird die Variable *fIsFullDuplex* aus der Datenstruktur ”ETHERNET_CONNECTION_STATE_Ttag” ausgelesen.
Ist das Interface nicht initialisiert (siehe ”Plug Removed”), so ist die Netzwerk-Auslastung (ein- wie auch ausgehend) 0.

Plug Removed
Die Variable *ifIsLinkUp* gibt an, ob das Interface initialisiert ist. Ist jedoch physikalisch der Stecker nicht im Port, so ist die Variable auch 0 und kann somit für dieses Managed Object genutzt werden, muss jedoch dafür invertiert werden.

Energy

Voltage
Der verbrauchte Strom am Hilscher NXHX 500-ETM-Board ist eines der drei Sensor-Werte, die vom Sensor-Board gemessen und auf Anfrage übermittelt werden.

Current
Auch die anliegende Spannung wird vom Sensor-Board gemessen und über die serielle Schnittstelle auf Anfrage übermittelt.

5.3. Genutzte SNMP-Version

Es standen drei verschiedene SNMP-Versionen zur Auswahl. Letztendlich wurde SNMPv2c genutzt, da im Vergleich zu SNMPv1 mehr Datentypen für die MIB zur Verfügung stehen, dass durch *GetBulkRequest* anstatt *GetNextRequest* weniger Netzwerk-Auslastung durch weniger Overhead entsteht. SNMPv3 wurde nicht genutzt, da einerseits die Verbreitung sehr gering ist (siehe Unterkapitel 2.1.2) und dass zum Beispiel lwIP kein SNMPv3 bisher unterstützt. Eine Verschlüsselung der Daten ist bisher in diesem Netzwerk nicht notwendig, da einerseits durch die Ver- und Entschlüsselung Ressourcen und Rechenzeit benötigt und andererseits auf die zu überwachenden Werte nur lesend zugegriffen wird. Des Weiteren müsste für SNMPv3 das Transport-Protokoll TCP (Transmission Control Protocol) genutzt werden.

5.4. Sensor-Board

Dieses Unterkapitel befasst sich mit der hardware- wie auch softwareseitigen Fertigstellung des Sensor-Boards.

5.4.1. Hardware

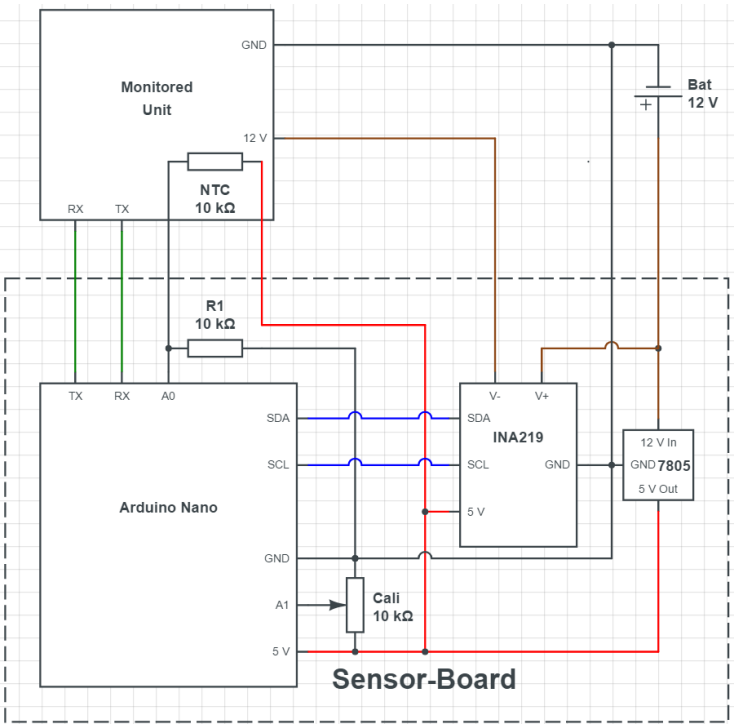


Abbildung 5.2.: Schaltplan des Sensor-Boards

Wie in Abbildung 5.2 zu sehen ist, wurden folgende elektronischen Komponenten verwendet:

Arduino Nano (Steuereinheit)

Für das erste Sensor-Board wurde als Entwicklungsboard ein Arduino Nano genutzt. Arduino-Entwicklungsboards bieten durch die große Verbreitung, durch viele Hardware-Schnittstellen (wie zum Beispiel I^2C, SPI, UART) und umfangreiche Bibliotheken eine große Auswahl an Möglichkeiten für Projekte.
Um die Sensordaten zum zu überwachten Agenten zu senden, wurde als serielle Schnittstelle UART (Universal Asynchronous Receiver Transmitter, grüne Pins für jeweilige Datenleitungen: ”TX” und ”RX”) genutzt.

Wie an den blauen Pins ”SDA” (Datenleitung) und ”SCL” (Taktleitung) zu sehen ist, wurde als Bussystem I^2C (Inter-Integrated Circuit) verwendet, um die Sensordaten abzufragen.

INA219 (Strom-Sensor)

Um die anliegende Spannung und den Strom zu messen, wurde ein Sensor benötigt, der einerseits I^2C unterstützt und andererseits eine Spannung bis 12 Volt (Auto-Batterie-Spannung) und eine Stromstärke von mindestens 150 Ampere (maximaler Stromverbrauch laut Datenblatt des Hilscher NXHX 500-ETM-Boards, vgl. Hilscher Gesellschaft für Systemautomation mbH (2005)) messen kann. Dazu wurde der Chip INA219 von Texas Instruments ausgewählt. Dieser kann eine Spannung bis 24 Volt und eine Stromstärke bis 3,5 Ampere messen.

NTC (Temperatur-Sensor)

Zur Temperaturmessung wurde ein NTC-Sensor (Heißleiter, siehe Unterkapitel 4.2.2) MF5B mit einem Nennwiderstand von 10 Kiloohm ausgesucht. Dieser kann in einem Temperatur-Bereich von -50° bis +150° Celsius messen. Dieser Widerstand hat eine Breite von nur 0,1 Millimeter. Jedoch wurde dieser nicht an den I^2C-Bus mit angebunden, da dafür noch ein weiterer I^2C-fähiger Mikrocontroller angeschlossen werden müsste, der den Wert des Heißleiters misst, in eine Temperatur umrechnet und dann an die Steuereinheit dieses Sensor-Boards senden würde. Stattdessen wurde der Heißleiter in Reihe mit einem weiteren 10-Kiloohm-Widerstand (R1) angeschlossen. Zwischen den beiden Widerständen wurde der Analogpin *A0* des Arduinos angeschlossen um die abgefallene Spannung durch den erzeugten Spannungsteiler zu messen.

7805 (Spannungsregler)

Da der Arduino Nano und der Strom-Sensor eine Versorgungsspannung von 5 Volt braucht, jedoch die Autobatterie 12 Volt hat, wird mittels eines Spannungsreglers diese Batteriespannung von 12 auf 5 Volt reduziert.

Calibrator

Um Messungenauigkeiten vom Temperatur-Sensor auszugleichen (zum Beispiel durch minimale Spannungsunterschiede), wurde ein weiteres Potentiometer angebracht. Mit diesem Potentiometer kann ein negativer wie auch positiver Offset zur gemessenen Temperatur eingestellt werden.

5.4.2. Software

Sensordaten-Aufbereitung

Die Sensordaten werden jede Sekunde neu aufbereitet:
Für das INA219-Strommessmodul existieren für Arduinos bereits Bibliotheken, die das Konfigurieren und Anfragen von Messdaten sehr vereinfacht. Der Arduino fungiert als I^2C-Master, der INA219-Sensor (und jegliches mögliches weitere Modul) als I^2C-Slave. Der Arduino fragt in diesem Intervall zuerst die anliegende Spannung und daraufhin die Stromstärke ab. Diese werden dann in die jeweiligen Variablen geschrieben, die dem Agenten dann für die serielle Schnittstelle bereitgestellt werden.

Der Temperatur-Sensor ist wie oben beschrieben nicht am I^2C-Bus angeschlossen. Daher ist das Aufbereiten der Daten etwas aufwendiger als das über die I^2C-Module. Da dieser Sensor ein heißleitener Widerstand ist, muss somit der Widerstand gemessen werden. Wie im vorherigen Unterkapitel beschrieben worden ist, ist ein weiterer Widerstand in Reihe zu dem Heißleiter angeschlossen (Spannungsteiler). Da ein Mikrocontroller keinen Widerstand messen kann, muss somit die Spannung bestimmt werden und anhand der *Steinhart-Hart-Gleichung* (vgl. Steinhart und Hart (1968)) die entsprechende Temperatur ermittelt werden:

$$\frac{1}{T} = A + B * ln(R) + C * [ln(R)^3]$$

- T ist die resultierende Temperatur in Kelvin (da °Celsius erfordert ist, muss noch 273,15 dazu addiert werden).
- A, B, C sind Steinhart-Hart-Koeffizienten, die sich anhand der sich im Datenblatt befindlichen Temperatur-Widerstand-Überleitungstabelle berechnen lassen.
- R ist der gemessene Widerstand in Ohm.

Je nachdem, wie das Sensor-Board später erweitert werden soll, kann entschieden werden, ob wie oben eine aufwendige Berechnung durchgeführt werden soll oder eine Look-Up-Table genommen wird, bei der ein jeweils gemessener Spannungswert einer Temperatur gegenübergestellt wird.
Wenn die Sensordaten vollständig aufbereitet worden sind, jedoch bis zum nächsten Intervall zum Messen Zeit verbleibt, so ist die Steuereinheit diese Zeit im Schlafmodus.

Kommunikation mit dem Agenten

Das Protokoll an der seriellen Schnittstelle ist simpel gehalten:

Value	Agent-Request	Sensor-Board-Response
Temperature	t \n	t α_1 α_2 \n
Voltage	v \n	v α_1 α_2 \n
Current	c \n	c α_1 α_2 \n

Tabelle 5.1.: Protokoll zwischen Agent und Sensor-Board

Wie in der Tabelle 5.1 zu sehen ist, existieren sechs verschiedene Nachrichten mit jeweilis einem Request zu einem Response für den jeweiligen Sensor-Wert:

- Die Zeichen 't', 'v' und 'c' sind die Identifier für die jeweilig geforderten Werte.
- Das Zeichen \n ist ein Steuerzeichen (Line Feed bzw. Zeilenumbruch, ASCII-Code: 0xA) und gibt somit das Ende einer Nachricht an.
- Da Werte wie die Stromstärke laut den Anforderungen im Milliampere gemessen werden sollen, wird es sehr wahrscheinlich sein, dass dafür ein Byte (Wertebereich vorzeichenlos: 0 bis 255) zum Senden des Wertes nicht ausreichen wird. Daher wurden für jeden Wert in diesem Protokoll zwei Byte (0 bis 65535 - little Endian) zur Verfügung gestellt mit α_1 und α_2.

Somit sind die Request-Nachrichten zwei Byte groß, während die Response-Nachrichten vier Byte groß sind.

Bei jedem eintreffenden Request über die serielle Schnittstelle wird eine Interrupt-Service-Routine (ISR) gestartet (welche die Sensoraufbereitung unterbricht). Diese ISR speichert die eintreffende Nachricht in einen Puffer. Nach vollständigem Empfang wird der Puffer durchiteriert und auf Requests überprüft. Ist dies der Fall, so wird direkt darauf geantwortet mit dem zuletzt gemessenen geforderten Wert.

5.4.3. Fertigstellung

Abbildung 5.3 zeigt das gelötete Sensor-Board:

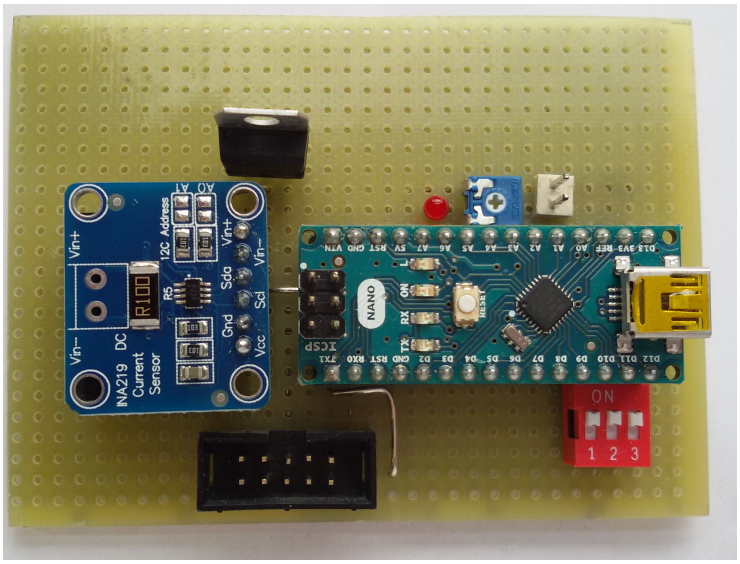


Abbildung 5.3.: Realisiertes Sensor-Boards

Das blaue Modul ist das Strom- und Spannungsmessungsmodul INA219. Direkt unter diesem Modul (nicht im Bild zu erkennen) befindet sich der Stromeingang von der Batterie wie auch der Stromausgang zum Hilscher NXHX 500-ETM-Board.
Über INA219 ist der im Schaltplan (Abbildung 5.2 zu sehende lineare Spannungsregler 7805, welcher die Batteriespannung (12 Volt) auf die 5 Volt herunterregelt, die für die Betriebsspannung des Sensor-Boards notwendig sind.
Rechts daneben befindet sich eine rote Status-LED (nicht im Schaltplan zu sehen).
Direkt daneben befindet sich das Potentiometer für die Kalibrierung des Temperaturwiderstandes (MF5B).
Der Anschluss dafür befindet sich rechts daneben (zwei weiße Pins). Der Arduino (Steuereinheit) ist das große türkisfarbene Modul.
Des Weiteren wurden drei zusätzliche DIP-Schalter angebracht, um einzustellen, welche der drei Werte gemessen werden sollen (Schalter 1 für Temperatur, 2 für Spannung, 3 für Strom).
Unter dem INA219 und dem Arduino befindet sich der Anschluss für die serielle Schnittstelle zum Hilscher NXHX 500-ETM-Board.

Die Größe der Fläche wurde absichtlich etwas höher gehalten, damit bei der Montage an das Gehäuse die bereitgestellten Schraubenlöcher am Hilscher NXHX 500-ETM-Board genutzt werden können (siehe Abbildung 5.4).

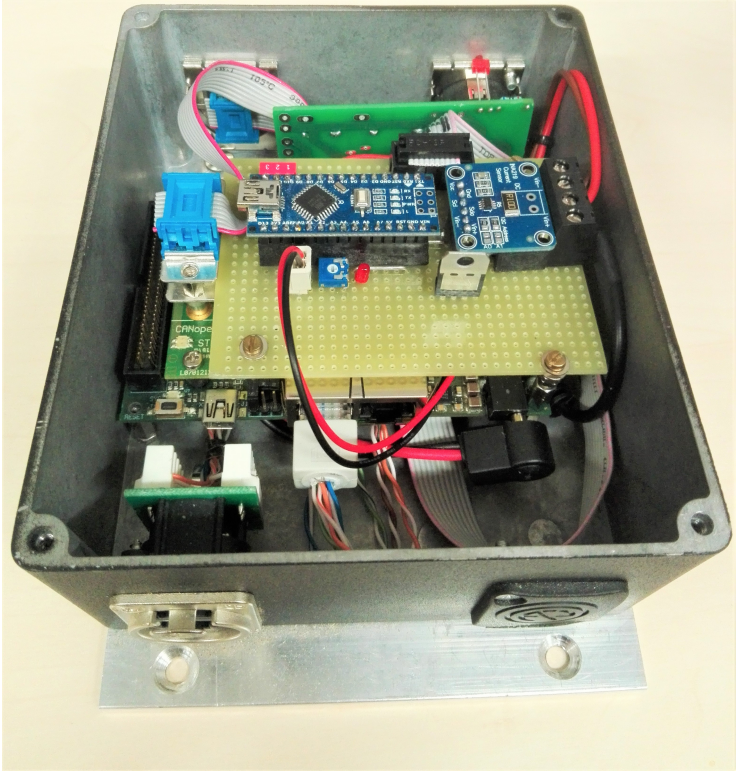


Abbildung 5.4.: Montiertes Sensor-Boards

6. Evaluation und Qualitätssicherung

Dieses Kapitel widmet sich der Beurteilung, ob die Anforderungen an den verschiedenen Instanzen eingehalten worden sind. Dazu wird auf jede einzelne Instanz, wie sie in der Tabelle 3.1 aufgelistet worden sind, eingegangen und dessen Ist- und Sollwerte verglichen. Die Netzwerk- und Paketanalysen wurden mit dem Tool Wireshark durchgeführt.

6.1. Protokoll

Da die Anforderungen an das Protokoll lediglich waren, dass dessen Nachrichten als Best-Effort-Messages versendet werden, mussten die TTEthernet-Frames nicht angepasst werden. Wie in den Grundlagen erwähnt, werden alle Standard-Ethernet-Frames als Best-Effort-Messages versendet.

6.2. Test-Aufbau

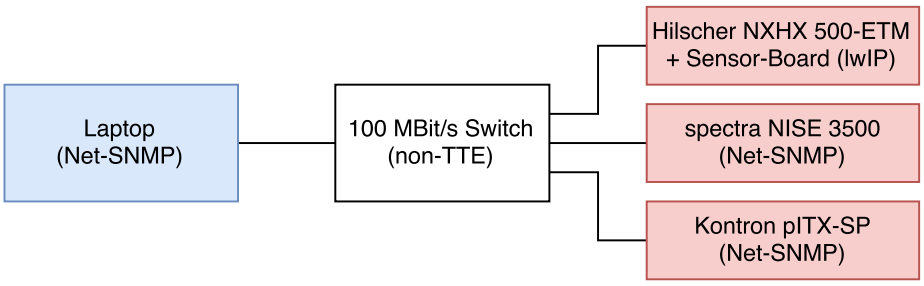


Abbildung 6.1.: Testaufbau mit Manager (blau), Switch (weiß) und Agenten (rot)

Wie in Abbildung 6.1 zu sehen ist, wurde zum Testen der Funktionsfähigkeit ein kleineres Netzwerk aufgebaut. Aufgrund der momentanen Umbauten am Fahrzeug konnte leider kein TTEthernet-Switch genutzt werden. Somit wurde lediglich der SNMP-Verkehr im Netzwerk getestet und die Funktionsfähigkeit und Richtigkeit der Implementation des Simple-Network-Managements-Protokolls wie auch der dazugehörigen Management Information Base bewiesen, jedoch nicht die Integration im TTEthernet.

Im Bild links wurde als Manager (blau) ein Laptop (mit dem Betriebssystem Ubuntu) genutzt, auf dem das in Kapitel 5.2.1 beschriebene Net-SNMP für den Manager installiert und konfiguriert wurde.
In der Mitte des Bildes (weiß) befindet sich ein Standard-TCP/IP-Layer-2 Switch mit einer Übertragungsrate von bis zu 100 MBit/s (entprechend der Switches im CoRE-Backbone).
Auf der rechten Seite des Bildes befinden sich für je eine Hardwarekomponente repräsentative Agenten (rot), auf denen Net-SNMP oder lwIP installiert wurden. Bei dem Hilscher NXHX 500-ETM-Board wurde zusätzlich das Sensor-Board angeschlossen.

6.3. SNMP

Anhand der beispielhaften Abbildung 6.2 wurden die drei Agenten im Netz abgefragt. Mittels *GetBulk* fragt der Manager jeden Agenten nach dessen CoRE-MIB ab.

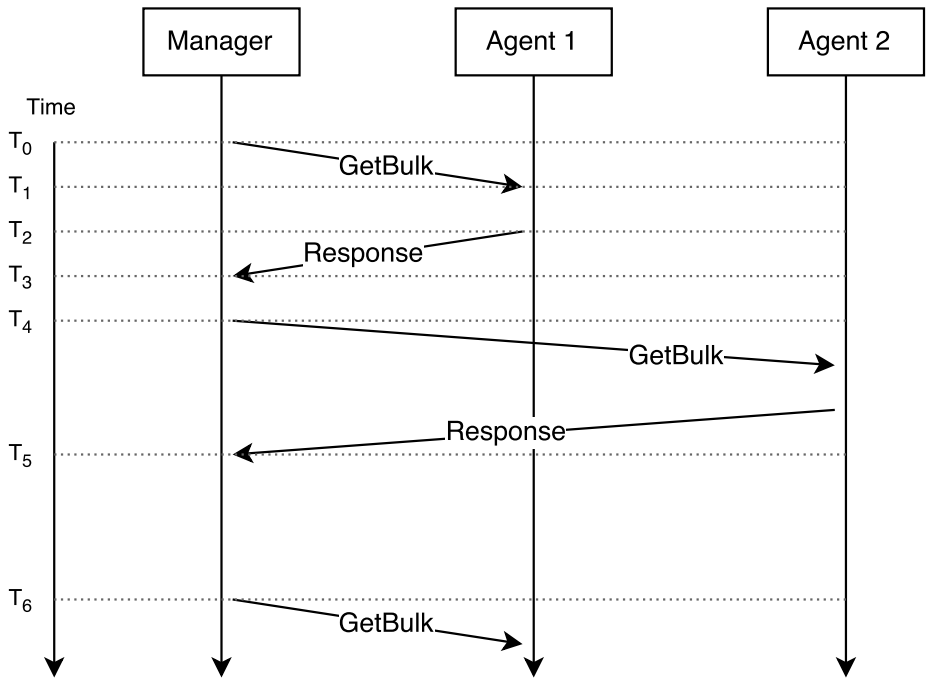


Abbildung 6.2.: Gemessene Zeitpunkte beim Abfragen der Agenten

6.3.1. Vergleich *GetNextRequest* zu *GetBulkRequest*

Zum Testen der Netzwerkbelastung von SNMPv1 zu SNMPv2, wurde ein Vergleich zwischen Protokollnachrichten *GetBulkRequest* (SNMPv2) und *GetNextRequest* (SNMPv1) an einem Kon-

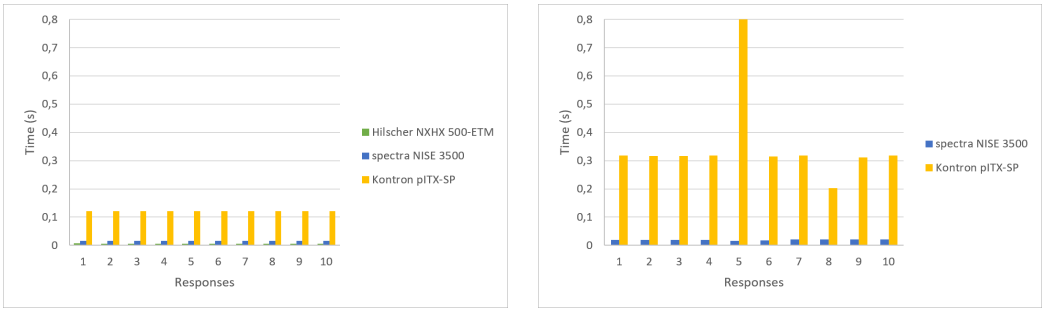
tron pITX-SP-System gemacht. Da durch die Tabelle der CoRE-MIB die Managed Objects pro Agent variieren können, sei hier erwähnt, dass dieses System insgesamt zwölf Managed Objects überwacht.

Da die CoRE-MIB tabulare (LAN-Table), aber auch skalare Werte enthielt, entstanden zwei *GetBulkRequest*s wie auch die zwei zugehörigen Responses. So wurden über das (ungestörte) Netzwerk 696 Bytes (0,68 Kilobytes) gesendet und empfangen.
Dies wurde mit dem *GetNextRequest* verglichen. Da diese Protokollnachricht pro Request nur ein Managed Object abfragt und dafür je ein *Response* erhält, wurde so das Netzwerk deutlich mehr belastet. Es wurden insgesamt 26 SNMP-Nachrichten über das Netz gesendet. Dies sind die jeweils zwölf Requests wie auch Responses auf dem Kontron pITX-SP-System existierendes Managed Object (insgesamt 24). Daraufhin erfolgt ein letzter *GetNextRequest* (Nachfrage auf einen Wert nach dem Managed Object ”Current”). Da nach ”Current” kein weiteres Managed Object für die CoRE-MIB auf diesem System existert, wurde die Fehlermeldung *endOfMIBView* zurückgesendet, da kein weiteres MIB-Modul implementiert wurde. Diese 26 Nachrichten waren insgesamt 2248 Bytes (2,12 Kilobbytes) groß.
Daraus ergibt sich, dass textitGetNextRequest mehr als das dreifache (3,23-fache) der Netzwerklast nutzt für den selben Informationsgehalt. Würden weitere MIB-Module hinzugefügt werden, so wäre die Auslastung entsprechend höher.

6.4. Überwacher / Manager

6.4.1. Abfrage-Intervall des Managers zu den Agenten

Das Abfrage-Intervall zu den Agenten wurde getestet, indem das Standard-Skript zur Überwachung die Agenten alle fünf Sekunden (in Abbildung 6.2 Zeitpunkt zwischen T_5 und T_6) deren MIB abfragte. Analysiert wurden daraufhin die ausgehenden Request-Pakete des Managers. Verglichen wurden die zeitlichen Unterschiede zwischen einem Request an einen Agenten zu dem nächsten Request an denselben Agenten (T_0 zu T_6).
Da das Skript die Agenten nicht gleichzeitig, sondern sukzessiv abfragt, entsteht entsprechend eine Verzögerung (T_0 zu T_5) von durchschnittlich 0,32 Sekunden bis zur nächsten Abfrage an denselben Agenten.



(a) Unausgelastetes Interface

(b) Ausgelastetes Interface

Abbildung 6.3.: Responsezeiten je nach Interface-Auslastung eines Agenten

6.4.2. Time-out nach einer Abfrage

Ob ein Time-out eines Agenten durch ein ausgelastetes Interface des Agenten entsteht, wurde anhand einer Dateiübertragung an den Agenten getestet. Mittels SCP (Secure Copy) wurde parallel die Managed Objects der Agenten abgefragt.

Unausgelastetes Interface

Wie in Abbildung 6.3(a) zu sehen ist, sind die Responsezeiten bei Hilscher NXHX 500-ETM und spectra NISE 3500 bei einem unausgelasteten Interface im Hundertstelsekunden-Bereich. Jedoch sticht die hardwarebedingte vergleichsweise hohe Latenz von dem Kontron pITX-SP-System heraus.

Ausgelastetes Interface

In Abbildung 6.3(b) ist zu sehen, dass die Responsezeit vom spectra NISE 3500-System unmerklich höher wurde (durchschnittlich 0,01 Sekunden mehr). Lediglich das Kontron pITX-SP-System sticht mit einer Spitze von 0,8 Sekunden heraus.
Aufgrund von Überlaufproblemen bei lwIP (da dieser Stack in einer weiteren Bachelorarbeit noch in der Implementierungsphase ist) konnten leider keine Auslastungstests beim Hilscher NXHX 500-ETM-System durchgeführt werden.

System	Unausgelastet	Ausgelastet
spectra NISE 3500	0,01	0,02
Kontron pITX-SP	0,12	0,35
Hilscher NXHX 500-ETM	0,01	-

Tabelle 6.1.: Durchschn. Responsezeiten (in Sekunden) je nach Interface-Auslastung

Die durchschnittlichen Responsezeiten der einzelnen Agenten wurde in Tabelle 6.1 dargestellt. Da selbst die Spitze des Kontron pITX-SP-Systems bei dessen ausgelastetem Interface unter einer Sekunde ist, ist damit die Anforderung erfüllt, dass die SNMP-Response-Pakete innerhalb einer Sekunde bei dem Manager eintreffen.

6.4.3. Angenommener Ausfall nach Time-outs

Der Ausfall eines Agenten wurde auf drei verschiedene Arten getestet:

- Netzwerk-Steckverbindung des Agenten am Switch physisch entfernt
- Netzwerk-Interface des Managers deaktiviert
- Netzwerk-Interface des Agenten deaktivert

Unabhängig von den oben genannten Arten verhielt sich der Manager gleich. Nach einer Sekunde (durchschnittlich 1,02 Sekunden) wertete der Manager dies als Time-out und sendete ein neuen Request an den Agenten. Da dieser nicht antwortete, wurden vier weitere Requests gesendet, bis daraufhin schließlich ein Geräte-Ausfall angenommen wurde.

6.5. Agenten

6.5.1. Abfrage-Intervall zu den zu überwachenden Werten

Ob das Mess-Intervall exakt eingehalten wurde, wurde nicht getestet. Es wurde jedoch mit dem Manager getestet, dass sich ein Wert innerhalb einer gewissen Zeit entsprechend ändert, indem bei den Agenten die CPU-Temperatur abgefragt wurde. Bei den Net-SNMP-Agenten wurde die Temperatur mittels dem Tool cpuburn erhöht. Dieses Tool versucht den Prozessor bis zu 100 Prozent auszulasten.
Entsprechend diesem gestarteten Tool stieg die Temperatur wie auch die Auslastung der CPU an. Nach einer erneuten Abfrage (nach Anforderungen alle fünf Sekunden) des Managers haben sich diese Managed Objects geändert.

6.5.2. Abfrage-Intervall zum Sensor-Board

Das Abfrage-Intervall zum Sensor-Board wurde mit einem für den Arduino geschriebenen Testprogramm getestet. Dieses Intervall misst die Zeit mittels eines Softwaretimers, wie oft der Agent über die serielle Schnittstelle ein Request an das Sensor-Board erstellt. Diese Werte blieben bei durchschnittlich 6,5 Sekunden. Die Überschreitung dieser Werte lag daran, dass die abgefragten Werte verarbeitet werden mussten und dass der Manager den Agenten auch zwischendurch abfragte.

6.6. Bemerkung zu *GetBulkRequest* bei lwIP und Net-SNMP

Während des Testens ist aufgefallen, dass sich lwIP gegenüber Net-SNMP hinsichtlich der Nachricht *GetBulkRequest* unterschiedlich verhält. Die Agenten wurden anhand der CoRE-MIB mittels der SNMP-Nachricht *GetBulkRequest* vom Net-SNMP-Manager abefragt. Aufgefallen ist, dass die lwip-Agenten auf *GetBulkRequest* der CoRE-MIB mit nur einem *Response* antworten, welches wie gewünscht die Daten der ganzen CoRE-MIB enthält. Wird hingegen *GetBulkRequest* bei Net-SNMP-Agenten genutzt, antworten die Agenten zuerst mit einer Nachricht, welche die CoRE-MIB-Werte vom Systemnamen bis zur IP-Table enthielten. Daraufhin sendete der Manager ein weiteres *GetBulkRequest* für die Spannungs- und Stromwerte. Bisher konnte jedoch nicht die Ursache dieses Unterschiedes geklärt werden. Da das gewünschte Ergebnis (Erhalt der Managed Objects) jedoch das gleiche ist, ist dies kein großer Nachteil. Jedoch erhöht sich durch die höhere Netzwerknutzung das Risiko, dass einerseits aufgrund des TTEthernet-Typen *Best-Effort* und andererseits aufgrund des Transportprotokolls UDP manche Nachrichten gegebenenfalls nicht gesendet beziehungsweise empfangen werden.

6.7. Sensor-Board

6.7.1. Größe des Boards

Das Sensor-Board soll eine maximale Größe von 11 * 8 * 2 cm haben, damit dies in das in den Anforderungen erwähnte Gehäuse passt. Letztendlich hat das Board eine Größe 8,5 * 6,4 * 2 cm. Das Board wurde über den Montage-Schraublöchern des Hilscher NXHX 500-ETM-Boards montiert (siehe Abbildung 5.4).

6.7.2. Schnittstelle zum zu überwachenden Gerät

Als serielle Schnittstelle zum Hilscher NXHX 500-ETM-Board wurde UART genutzt. Es wäre auch möglich gewesen, RS-232 zu nutzen. Da aber weder das Hilscher NXHX 500-ETM-Board noch der auf dem Sensor-Board als Steuereinheit genutzte Arduino RS-232 ohne einen entsprechenden Wandler (wie der MAX232) diese Schnittstelle nutzen können, wurde dies nicht in Betracht gezogen.

6.7.3. Reaktionszeit zum Antworten

Ob das Sensor-Board dem Hilscher NXHX 500-ETM-Board rechtzeitig antwortet, wurde mittels eines kleinem Testprogramm geprüft, welches das Sensor-Board alle fünf Sekunden (wie in den Anforderungen beschrieben) über die serielle Schnittstelle die Temperatur, anliegende Spannung und Strom abfragt. Das Sensor-Board antwortete daraufhin unter einer Sekunde.
Des Weiteren wurde (im normalen Betrieb) getestet, wie das Hilscher NXHX 500-ETM-Board auf Nicht-Empfang von Daten reagiert. Nachdem das Sensor-Board nach einer Sekunde nicht reagiert, wird der Wert auf den Standardwert gesetzt, wie dieser in der CoRE-MIB beschrieben wurde (siehe Anhang A Felder ”DEFVAL”) .

6.7.4. Schnittstelle zu Sensoren

Als serielle Schnittstelle der Sensoren wurde nur teilweise ein Bussystem genutzt:

Strom- und Spannungsmessung

Zur Strom- und Spannungsmessung wurde Modul INA219 verwendet. Dieses ist über das Bussystem I^2C angeschlossen worden.
Die Korrektheit der über I^2C-übertragenen Sensor-Werte wurde mit einem Multimeter überprüft. Für die Spannungsmessung wurde dazu das Multimeter parallel zu dem INA219 geschaltet. Die Spannungsmessung unterschied sich im Zehntelvolt-Bereich.
Für die Messung des Stromes, wurde das Multimeter entsprechend in Reihe zu diesem Strommessmodul geschaltet, auch hier entstanden Differenzen von durchschnittlich 200 Milliampere. Diese Unterschiede können jedoch auch auf Messungenauigkeiten auf Seiten des Moduls wie auch des Multimeters zurückzuführen sein, sind aber auch nicht relevant hinsichtlich der Sicherheit des Systems.

Temperaturmessung

Wie bereits im Kapitel ”Realisierung” erwähnt, wurde aufgrund der zusätzlichen Hardwarekosten zur Temperaturmessung der analoge Anschluss des Arduinos genutzt anstelle des I^2C-Busses.
Der Temperatur-Sensor MF5B wurde anhand eines weiteren Thermometers und der des Herstellers bereitgestellte Überleitungstabelle überprüft. Als Referenztemperaturen wurden drei verschiedene Temperaturen aufgenommen: 20 ° Celsius (Raumtemperatur)), 25 ° Celsius (Außentemperatur) und 8 ° Celsius (Kühlschranktemperatur). Aufgrund der Spannungsschwankungen unterschiedlicher Stromquellen wurde bei der ersten Messung der Kalibrierungs-Widerstand eingestellt, um ein entsprechenden Offset einzustellen.

6.8. Zu überwachende Werte / MIB

Folgende Tabelle (6.2) zeigt zusammengefasst, woher die Net-SNMP- und bei den lwIP-Agenten die zur Berechnung der einzelnen zu überwachenden Werte benötigten Informationen extrahierten:

Werte	**Net-SNMP**	**lwIP**
CPU-Auslastung	/proc/stat	n.A.
CPU-Temperatur	Konfigurationsdatei	Über Sensor-Board
Arbeitsspeicher-Auslastung	/proc/meminfo	n.A.
Netzwerk-Auslastung	/sys/net (Verzeichnis)	”stats_mib2_netif_ctrs”
Strom-Verbrauch	n.A.	Über Sensor-Board
Anliegende Spannung	n.A.	Über Sensor-Board

Tabelle 6.2.: Benötigte Informationen zur Berechnung der Managed Objects der Agenten

7. Zusammenfassung

Ziel dieser Bachelorarbeit war, das Simple Network Management Protocol (SNMP) für das Echtzeit-Ethernet-Backbone im VW Golf 7 der CoRE-Gruppe zu implementieren und dazu ein weiteres Hardware-Board (”Sensor-Board”) für notwendige fehlende Sensor-Werte zu fertigen.

7.1. Time-Triggered-Ethernet

Das CoRE-Backbone hat als Erweiterung des Standard-Ethernets ein Time-Triggered-Ethernet (TTE oder TTEthernet) implementiert. Den Anforderungen entsprechend werden SNMP-Nachrichten dem TTE-Nachrichtentypen ”Best-Effort-Messages” erhalten und so mit der niedrigsten Priorität durch das Netzwerk gesendet. Dies bedeutet jedoch auch, dass dessen Übertragung nicht garantiert wird.

7.2. SNMP

Die letztendlich genutzte Protokollversion war SNMPv2c aufgrund der großen Verbreitung, der größeren Auswahl an Datentypen und dem Nachrichtentypen *GetBulk*. Damit die Werte CPU-Temperatur, CPU-, Arbeitsspeicher- und Netzwerk-Auslastung sowie Strom- und Spannungswerte abgefragt werden konnten, wurde dafür eine entsprechende Datenbank (MIB-Modul oder auch nur MIB) definiert (siehe Anhang A).
Diese Management Information Base wurde auf dem Überwacher (Manager) und den zu überwachenden Geräten (Managed Devices) als Agenten implementiert.
Es gab drei verschiedene zu überwachende Hardware-Typen, wobei zwei der Managed Devices Computer (Kontron pITX-SP und spectra NISE 3500) mit Linux-Distributionen waren und einer ein Entwicklungsboard mit Mikrocontroller (Hilscher NXHX 500-ETM). Auf den Computern wurde für die Implementierung von SNMP die Software-Suite Net-SNMP und auf dem Entwicklungsboard der Stack ”lwIP” genutzt.
Ein Laptop, der ebenfalls Software-Suite Net-SNMP nutze, diente als Manager. Dieser wurde über ein Diagnose-Anschluss an das CoRE-Backbone angeschlossen.

7.3. Sensor-Board

Da dem Hilscher NXHX 500-ETM-Board Werte für die CoRE-MIB wie CPU-Temperatur, Stromverbrauch und anliegende Spannung fehlten, wurde dafür das Sensor-Board über die serielle Schnittstelle angeschlossen.
Das Sensor-Board ist eine kleine Platine, die ein Strom- und Spannungsmessmodul über den seriellen Datenbus I^2C angeschlossen hat. Des Weiteren ist an dem Board über einen analogen Eingang ein Thermistor (temperaturabhäniger Widerstand) zur Messung der Mikrocontroller-Temperatur angeschlossen worden.
Als Steuereinheit des Sensor-Boards wurde ein Arduino Nano verwendet.

7.4. Ausblick

Die Arbeit hat hinsichtlich der SNMP-Implementierung einen großen Fokus auf die Implementierung der eigenen CoRE-MIB gelegt. Momentan werden Werteüberschreitungen vom Manager selber erkannt. Alternativ könnte dies in Zukunft auch mit Trap-Nachrichten implementiert werden. Da diese Trap-Nachrichten sicherheitsrelevant sind, sollten diese dann Time-Triggered-Ethernet nicht als Best-Effort-, sondern höherpriorisiert als Rate-Constrained- oder Time-Triggered-Nachrichten gesendet werden. Dies bedeutet auch, dass der lwIP-Stack noch mit dem TTE-Stack für das Hilscher NXHX 500-ETM-Board verheiratet werden müsste.
Des Weiteren könnte überlegt werden, ob anstatt dem Nachrichtentypen Trap, der Nachrichtentyp Inform genutzt werden, da so auch sichergestellt wird, ob die Nachricht angekommen ist.
Weiterhin könnte als Manager anstatt eines Laptops im Cockpit des Autos ein weiteres oder bestehendes Gerät im Auto dafür verwendet werden. Es müsste daraufhin überlegt werden, wie der User auf Werteüberschreitungen zu reagieren hat oder ob das System daraufhin automatisch reagiert.

A. Core-MIB

```
CORE-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Integer32, Unsigned32,
    enterprises                          FROM SNMPv2-SMI
;

coreMIB MODULE-IDENTITY
    LAST-UPDATED "201705290000Z"  -- 29th May 2017
    ORGANIZATION "http://core.informatik.haw-hamburg.de/"
    CONTACT-INFO
        "postal:    Hochschule fuer Angewandte Wissenschaften Hamburg
                    CoRE-Group
                    Berliner Tor 5
                    20099 Hamburg
        email:      arne.gueldener@haw-hamburg.de"
    DESCRIPTION
                "MIB for embedded Systems of core group"
    ::= { enterprises 9999 }

sysName OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "An administratively-assigned name for this managed
                node. By convention, this is the node's fully
                qualified domain name.
                (original MIB-II 1.3.6.1.2.1.1.5 - sysName)"
    DEFVAL { "unnamed device"}
    ::= { coreMIB 1 }
```

```
cpuSensors          OBJECT IDENTIFIER ::= { coreMIB 2 }

cpuLoad OBJECT-TYPE
    SYNTAX      Unsigned32 (0..100)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "This identifier shows the average load of all CPU
                cores expressed as a percentage within one second.
                If the CPU has more cores, the average value will be
                formed from all cores"
    DEFVAL { 0 }
    ::= { cpuSensors 1 }

cpuTemperature OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "This identifier shows the temperature of the CPU in
                degree Celsius."
    DEFVAL { 0 }
    ::= { cpuSensors 2 }

ramLoad OBJECT-TYPE
    SYNTAX      Unsigned32 (0..100)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "Percentage value of the current RAM load."
    DEFVAL { 0 }
::= { coreMIB 3 }

lanTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF LanEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
                "The lanTable listing each down below defined
                Managed Objects. This size of this table
```

```
            depends on the amount of interfaces on the
            device."
::= { coreMIB 4 }

lanEntry OBJECT-TYPE
    SYNTAX      LanEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
                "An entry (conceptual row) in the lanTable."
    INDEX      { lanIndex }
    ::= { lanTable 1 }

LanEntry ::= SEQUENCE {
   lanIndex     INTEGER,
   lanLoadOut   Unsigned32,
   lanLoadIn    Unsigned32,
   plugRemoved  Integer32
}

lanIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
                "The index value that uniquely identifies the
                interface to which this entry is applicable. The
                interface identified by a particular value of
                this index is the same interface as identified
                by the same value of the IF-MIB's ifIndex.
                (original 1.3.6.1.2.1.4.20.1.2)"
::= { lanEntry 1 }

lanLoadOut OBJECT-TYPE
    SYNTAX      Unsigned32 (0..100)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "Percentage of last outgoing Octets compared to
                the interface's maximum speed. If the interface
```

```
                    is not initialized, this value is 0."
        DEFVAL { 0 }
    ::= { lanEntry 2 }

    lanLoadIn OBJECT-TYPE
        SYNTAX      Unsigned32 (0..100)
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                    "Percentage of last ingoing Octets compared to
                    the interface's maximum speed. If the interface
                    is not initialized, this value is 0."
        DEFVAL { 0 }
    ::= { lanEntry 3 }

    plugRemoved OBJECT-TYPE
        SYNTAX      Integer32 (0..1)
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                    "Indicates whether the ethernet plug is
                    removed or not. If the interface
                    is not initialized, this value is 0."

        DEFVAL { 0 }
    ::= { lanEntry 4 }

energy OBJECT IDENTIFIER ::= { coreMIB 5}

voltage OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "Applied Voltage (in volts) on this computer"
    DEFVAL { 0 }
::= { energy 1 }

currentAmpere OBJECT-TYPE
    SYNTAX      Integer32
```

```
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "Electric Current (in milli amperes) this computer
                is currently consuming"
    DEFVAL { 0 }
::= { energy 2 }

END
```

Literaturverzeichnis

[CoRE-Research-Group] *CoRE Research Group*. http://core.informatik.haw-hamburg.de/en/. – Zugriffsdatum: 2017-03-26

[INA219] *INA219*. http://www.ti.com/lit/ds/symlink/ina219.pdf. – Zugriffsdatum: 2017-04-09

[lwIP] *A Lightweight TCP/IP stack*. https://savannah.nongnu.org/projects/lwip/. – Zugriffsdatum: 2017-03-26

[Alessandro Rubini] Alessandro Rubini: http://www.linux.it/~rubini/docs/vfs/vfs.html. – Zugriffsdatum: 2017-05-15

[Arduino Nano] Arduino Nano: https://www.arduino.cc/en/Main/ArduinoBoardNano/. – Zugriffsdatum: 2017-06-13

[AS6802 2011] AS6802, SAE: Time-triggered ethernet. In: *SAE International* (2011)

[Blumenthal u. a. 2004] Blumenthal, U. ; Maino, F. ; McCloghrie, K.: The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model / RFC Editor. RFC Editor, June 2004 (3826). – RFC. – URL http://www.rfc-editor.org/rfc/rfc3826.txt. – ISSN 2070-1721

[Blumenthal und Wijnen 2002] Blumenthal, U. ; Wijnen, B.: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) / RFC Editor. RFC Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3414.txt. – ISSN 2070-1721

[Braden 1989] Braden, Robert: Requirements for Internet Hosts - Communication Layers / RFC Editor. RFC Editor, October 1989 (3). – STD. – URL http://www.rfc-editor.org/rfc/rfc1122.txt. – ISSN 2070-1721

[Case u. a. 2002a] Case, J. ; Harrington, D. ; Presuhn, R. ; Wijnen, B.: Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) / RFC Editor. RFC

Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3412.txt. – ISSN 2070-1721

[Case u. a. 2002b] Case, J. ; Mundy, R. ; Partain, D. ; Stewart, B.: Introduction and Applicability Statements for Internet-Standard Management Framework / RFC Editor. RFC Editor, December 2002 (3410). – RFC. – URL http://www.rfc-editor.org/rfc/rfc3410.txt. – ISSN 2070-1721

[Case u. a. 1989] Case, Jeffrey D. ; Fedor, Mark ; Schoffstall, Martin L. ; Davin, Chuck: Simple Network Management Protocol (SNMP) / RFC Editor. RFC Editor, April 1989 (1098). – RFC. – URL http://www.rfc-editor.org/rfc/rfc1098.txt. – ISSN 2070-1721

[Case u. a. 1990] Case, Jeffrey D. ; Fedor, Mark ; Schoffstall, Martin L. ; Davin, James R.: Simple Network Management Protocol (SNMP) / RFC Editor. RFC Editor, May 1990 (1557). – STD. – URL http://www.rfc-editor.org/rfc/rfc1157.txt. – ISSN 2070-1721

[Cisco Systems GmbH] Cisco Systems GmbH: www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/8141-calculate-bandwidth-snmp.html/. – Zugriffsdatum: 2017-05-14

[cpuburn] cpuburn: https://patrickmn.com/projects/cpuburn/. – Zugriffsdatum: 2017-05-12

[Davin u. a. 1992] Davin, J. ; Galvin, J. ; McCloghrie, K.: SNMP Administrative Model / RFC Editor. RFC Editor, July 1992 (1351). – RFC. – URL http://www.rfc-editor.org/rfc/rfc1351.txt. – ISSN 2070-1721

[Frye u. a. 2003] Frye, R. ; Levi, D. ; Routhier, S. ; Wijnen, B.: Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework / RFC Editor. RFC Editor, August 2003 (74). – BCP. – URL http://www.rfc-editor.org/rfc/rfc3584.txt. – ISSN 2070-1721

[Galvin und McCloghrie 1993] Galvin, J. ; McCloghrie, K.: Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2) / RFC Editor. RFC Editor, April 1993 (1445). – RFC. – URL http://www.rfc-editor.org/rfc/rfc1445.txt. – ISSN 2070-1721

[gentoo linux] GENTOO LINUX: https://www.gentoo.org/. – Zugriffsdatum: 2017-06-02

[Harrington u. a. 2002] HARRINGTON, D. ; PRESUHN, R. ; WIJNEN, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks / RFC Editor. RFC Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3411.txt. – ISSN 2070-1721

[Hilscher Gesellschaft für Systemautomation mbH] HILSCHER GESELLSCHAFT FÜR SYSTEMAUTOMATION MBH: http://www.hilscher.com/. – Zugriffsdatum: 2017-05-10

[Hilscher Gesellschaft für Systemautomation mbH 2005] HILSCHER GESELLSCHAFT FÜR SYSTEMAUTOMATION MBH: *Device Description, NXHX50-ETM - Software Development Board*, 2005. – 8 S

[IAV GmbH Ingenieurgesellschaft Auto und Verkehr] IAV GMBH INGENIEURGESELLSCHAFT AUTO UND VERKEHR: https://www.iav.com/. – Zugriffsdatum: 2017-07-01

[Jorrit Schippers 2012] JORRIT SCHIPPERS: *Analysis of SNMP usage in the real world.* https://www.utwente.nl/ewi/dacs/assignments/completed/bachelor/reports/B_assignment_Schippers.pdf. 2012. – Zugriffsdatum: 2017-06-12

[Kontron AG] KONTRON AG: http://www.kontron.de/. – Zugriffsdatum: 2017-05-10

[Levi u. a. 2002] LEVI, D. ; MEYER, P. ; STEWART, B.: Simple Network Management Protocol (SNMP) Applications / RFC Editor. RFC Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3413.txt. – ISSN 2070-1721

[Linux Kernel Organization] LINUX KERNEL ORGANIZATION: https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-class-net/. – Zugriffsdatum: 2017-04-30

[lubuntu] LUBUNTU: https://www.lubuntu.net/. – Zugriffsdatum: 2017-06-02

[McCloghrie u. a. 1999] MCCLOGHRIE, Keith ; PERKINS, David ; SCHOENWAELDER, Juergen: Structure of Management Information Version 2 (SMIv2) / RFC Editor. RFC Editor, April 1999 (58). – STD. – URL http://www.rfc-editor.org/rfc/rfc2578.txt. – ISSN 2070-1721

[McCloghrie und Rose 1991] McCloghrie, Keith ; Rose, Marshall T.: Management Information Base for Network Management of TCP/IP-based internets:MIB-II / RFC Editor. RFC Editor, March 1991 (17). – STD. – URL http://www.rfc-editor.org/rfc/rfc1213.txt. – ISSN 2070-1721

[Michael Kerrisk a] Michael Kerrisk: http://man7.org/linux/man-pages/man2/gethostname.2.html/. – Zugriffsdatum: 2017-04-30

[Michael Kerrisk b] Michael Kerrisk: http://man7.org/linux/man-pages/man5/proc.5.html/. – Zugriffsdatum: 2017-04-30

[Postel 1980] Postel, J.: User Datagram Protocol / RFC Editor. RFC Editor, August 1980 (6). – STD. – URL http://www.rfc-editor.org/rfc/rfc768.txt. – ISSN 2070-1721

[Postel 1981] Postel, Jon: Transmission Control Protocol / RFC Editor. RFC Editor, September 1981 (7). – STD. – URL http://www.rfc-editor.org/rfc/rfc793.txt. – ISSN 2070-1721

[Presuhn 2002a] Presuhn, R.: Transport Mappings for the Simple Network Management Protocol (SNMP) / RFC Editor. RFC Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3417.txt. – ISSN 2070-1721

[Presuhn 2002b] Presuhn, R.: Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP) / RFC Editor. RFC Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3416.txt. – ISSN 2070-1721

[SAE International] SAE International: https://www.tttech.com/. – Zugriffsdatum: 2017-06-15

[Schoenwaelder 2008] Schoenwaelder, J.: Simple Network Management Protocol (SNMP) Context EngineID Discovery / RFC Editor. RFC Editor, September 2008 (78). – STD. – URL http://www.rfc-editor.org/rfc/rfc5343.txt. – ISSN 2070-1721

[Spectra GmbH & Co. KG] Spectra GmbH & Co. KG: https://www.spectra.de/. – Zugriffsdatum: 2017-05-14

[Steinhart und Hart 1968] Steinhart, John S. ; Hart, Stanley R.: Calibration curves for thermistors. In: *Deep Sea Research and Oceanographic Abstracts* 15 (1968), Nr. 4, S. 497 – 503. – URL http://www.sciencedirect.com/science/article/pii/0011747168900570. – ISSN 0011-7471

[Tanenbaum 2009] Tanenbaum, Andrew S.: *Moderne Betriebssysteme*. 3., aktualisierte Auflage. Pearson Studium, 2009. – ISBN 3827373425

[The FreeBSD Project] The FreeBSD Project: https://www.freebsd.org/copyright/freebsd-license.html/. – Zugriffsdatum: 2017-06-01

[TTTech Germany GmbH] TTTech Germany GmbH: https://www.tttech.com/de/. – Zugriffsdatum: 2017-06-30

[Wijnen u. a. 2002] Wijnen, B. ; Presuhn, R. ; McCloghrie, K.: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) / RFC Editor. RFC Editor, December 2002 (62). – STD. – URL http://www.rfc-editor.org/rfc/rfc3415.txt. – ISSN 2070-1721

[Wireshark] Wireshark: https://www.wireshark.org/. – Zugriffsdatum: 2017-06-13

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 3. Juli 2017 Arne Güldener