Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Nils Parche

Sicherheitsserver für Navigationsinformationen in schiffsinternen Datennetzen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Nils Parche

Sicherheitsserver für Navigationsinformationen in schiffsinternen Datennetzen

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Robert Fitz

Abgegeben am 9. Oktober 2018

Nils Parche

Thema der Bachelorthesis

Sicherheitsserver für Navigationsinformationen in schiffsinternen Datennetzen

Stichworte

Softwareentwicklung, Server, Firewall, Application-Level-Gateway, Paketfilter, Ethernet, Multicast, C++

Kurzzusammenfassung

Im Rahmen dieser Arbeit wird ein Konzept erarbeitet mit dem Navigationsinformationen, die zwischen zwei IP-Segmenten kommunizieren, überwacht und analysiert werden. Auf Grund dieses Konzepts wird eine Anwendung programmiert und bewertet.

Nils Parche

Title of the paper

Security server for navigation information in vessel-internal data networks

Keywords

softwaredevelopment, server, firewall, Application-Level-Gateway, packet filter, Ethernet, Multicast, C++

Abstract

In the scope of this thesis, a concept is being developed to monitor an analyse navigational information between two IP-network communications. Based on this concept, an application will be programmed and assessed.

Inhaltsverzeichnis

1. Einführung

1.1. Motivation

Die Schifffahrt ist eine konservative Industrie die sich in einem schnellen digitalen Wandel vom bemannten zum autonomen Schiff hin entwickelt. Dabei ist die technische Realisierung durch die stark vernetzte Schiffssensorik bereits vorstellbar. Ein wichtiger Meilenstein ist die Klassifizierung der autonom betriebenen Schiffe. Klassifizierungsgesellschaften, wie beispielsweise der DNV GL, forcieren mit Forschungsprojekten in denen neuartige Antikollisionssysteme getestet werden die Entwicklung der benötigten Regularien. Für 2018 ist derzeit geplant in den norwegischen Fjorden autonom fahrende Fähren unter Kontrolle der Besatzung in Betrieb zu nehmen [25].

Mit der zunehmenden Anforderung an die Sensorik wurde im Jahr 2011 ein neuer Standard 61162-450 für die Kommunikation von Navigationsdaten von der IEC[1] ins Leben gerufen. Navigationsdaten wurden bis dahin nach dem NMEA[2]-0183 [20] Standard seriell von der Sensorik zu einer SPS[3] übermittelt, verarbeitet und in das Schiffssystem übertragen. Die neue Norm definiert ein Sensornetzwerk in dem Daten über ein Ethernet übertragen werden. Damit wird eine wesentlich höhere Datenrate und komplexere Vernetzung von Sensoren ermöglicht. Einhergehend mit den Vorteilen der Kommunikation über Ethernet ergeben sich auch neue Problemstellungen. Die Kommunikation kann leichter mitgeschnitten, abgegriffen und manipuliert werden.

Reedereien unterschätzen die Gefahr von Angriffen auf Schiffssysteme und die Versicherer befürchten Schäden in Milliardenhöhe bei Angriffen auf Navigationssysteme [19]. Tatsächlich häufen sich Vorfälle bei denen Schiffe als Ziel von Angriffen ausgewählt werden. Im Juli 2017 kam es im Mittelmeer zu einer GPS Spoofing[4] Attacke [12] bei der nachweislich die Positionsdaten von Schiffen in einem Sektor manipuliert wurden.

[1] International Electrotechnical Commission (IEC)
[2] National Marine Electronics Association (NMEA)
[3] Speicher Programmierbare Steuerung (SPS)
[4] Manipulation, Verschleierung oder Vortäuschen.

Durch den gezielten Einsatz einer Firewall, abgestimmt auf ein spezielles Protokoll, kann das Risiko unzulässiger Kommunikation reduziert werden. Diese Thematik der kontrollierten Datenkommunikation von Navigationsdaten wird in dieser Bachelorarbeit untersucht und Ansätze werden aufgezeigt.

1.2. Zielsetzung

Ziel der Arbeit besteht darin einen Sicherheitsserver zu entwickeln der Navigationsdaten, die nach dem IEC 61162-450 Standard übertragen und nach dem IEC 61162-1 Standard kodiert sind, auf Korrektheit und Konsistenz überprüft. Der Sicherheitsserver dient als Schnittstelle zwischen den zwei IP-Segmenten Sensornetz und Navigationsnetz. Die Sensordaten werden mittels Multicast-Nachrichten an alle Teilnehmer einer Gruppe gesendet. Der Server muss diesen Gruppen beitreten und die relevanten validen Nachrichten an das zweite IP-Netzsegment übertragen. Durch einen Mechanismus muss bestimmt werden welche Nachrichten als valide durch den Server geleitet werden. Ein Monitoring-Konzept dient der Visualisierung und stellt aufbereitete Informationen des Datenflusses durch den Gateway bereit.

1.3. Gliederung der Arbeit

Die Bachelorthesis ist in 8 Kapitel strukturiert. Das Kapitel 1 umfasst die Einleitung mit einer Beschreibung der Zielsetzung. Im Kapitel 2 wird auf die benötigten Grundlagen eingegangen die für das Verständnis der Arbeit notwendig sind. Anschließend werden in Kapitel 3 die Anforderungen analysiert. Dabei werden grundlegende Anforderungen aus der Zielsetzung abgeleitet. Weitere Anforderungen zur Fehlervermeidung werden auf Basis der FMEA[5] entwickelt. Die nachfolgenden Kapitel beschäftigen sich damit die erarbeiteten Anforderungen in einem Konzept- und Realisierungs-Prozess umzusetzen. Das Kapitel 4 beschreibt die Systemkomponenten und begründet die Entwurfsentscheidungen, die sich aus den Anforderungen und den Randbedingungen ergeben. Kapitel 5 beschäftigt sich mit der Konzeptionierung der Software. Aus den Entwurfsentscheidungen wird mittels Architekturkonzepten ein zu realisierendes System in Abstraktionsebenen entworfen. Die Realisierung erfolgt in Kapitel 6. Hier werden die im Konzept festgelegten Strukturen zur Umsetzung der Anforderungen angewendet. Die einzelnen

[5] Failure Mode and Effects Analysis (FMEA)

Komponenten werden mit geeigneten UML[6]-Diagrammen visualisiert und ausgearbeitet. Das Kapitel 7 widmet sich der Software Verifizierung. Es werden Test-Szenarien definiert an denen die Anforderungen überprüft, bewertet und dokumentiert werden. Das letzte Kapitel 8 fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf Erweiterungsmöglichkeiten des Sicherheitsservers.

[6] Unified Modeling Language (UML)

2. Theoretische Grundlagen

Das folgende Kapitel beschäftigt sich mit theoretischen Grundlagen, die für das technische Verständnis dieser Arbeit notwendig sind. Der Abschnitt 2.1 beschäftigt sich mit den Grundlagen zum Thema Ethernet, bevor der Abschnitt 2.2 die Kommunikation von Navigationsinformationen auf Schiffen behandelt. Der Abschnitt 2.3 beschäftigt sich mit der Prozessverwaltung und den Schnittstellen in einem Betriebssystem, die zur Kommunikation von Prozessen eingesetzt werden. Im Anschluss wird auf das grundlegende Funktionsprinzip einer Firewall in Abschnitt 2.4 eingegangen. Abschließend wird im Abschnitt 2.5 eine Methode zur Fehlervermeidung in Produkten vorgestellt.

2.1. Netzwerk

Im folgenden Abschnitt werden die wichtigsten Grundlagen der Ethernet-Kommunikation dargestellt.

2.1.1. OSI-Referenzmodell

Das OSI[1]-Referenzmodell wurde von der International Organization for Standardization (ISO) entwickelt und abstrahiert die verschiedenen Kommunikationssysteme in 7-Schichten um eine übersichtlichere Struktur der Kommunikationswege zu erhalten. Ausgenommen von der 7.-Schicht, stellt jede Ebene der darüberlegenden Schicht einen Dienst als Schnittstellenkommunikation zur Verfügung. Die Netzwerkdienste arbeiten auf den Schichten eins bis vier und die Anwendungsdienste auf den Schichten fünf bis sieben. Die einzelnen Schichten haben folgenden Aufgaben [18, S.14 ff]:

1. Die Bitübertragungsschicht definiert die physikalische Übertragung und ist für die Kodierung der Daten zuständig.

[1] OSI - Open System Interconnection

2. Die Sicherungsschicht segmentiert die Pakete in Frames mit einer physikalischen Adresse der MAC[2] und fügt eine Prüfsumme hinzu.
3. Die Vermittlungsschicht stellt netzübergreifende logische Adressen bereit.
4. Die Transportschicht ordnet mittels Ports den Datenpaketen eine Anwendung zu.
5. Die Kommunikationssteuerungsschicht verwaltet eine Sitzung zwischen zwei Kommunikationspartnern und regelt deren Dialog.
6. Die Datendarstellungsschicht dient als Übersetzer und regelt einen einheitlichen Transfersyntax zwischen zwei Systemen mit unterschiedlichen Datenformaten.
7. Die Anwendungsschicht stellt Anwendungen und Funktionen für Anwendungsfunktionalität zur Verfügung.

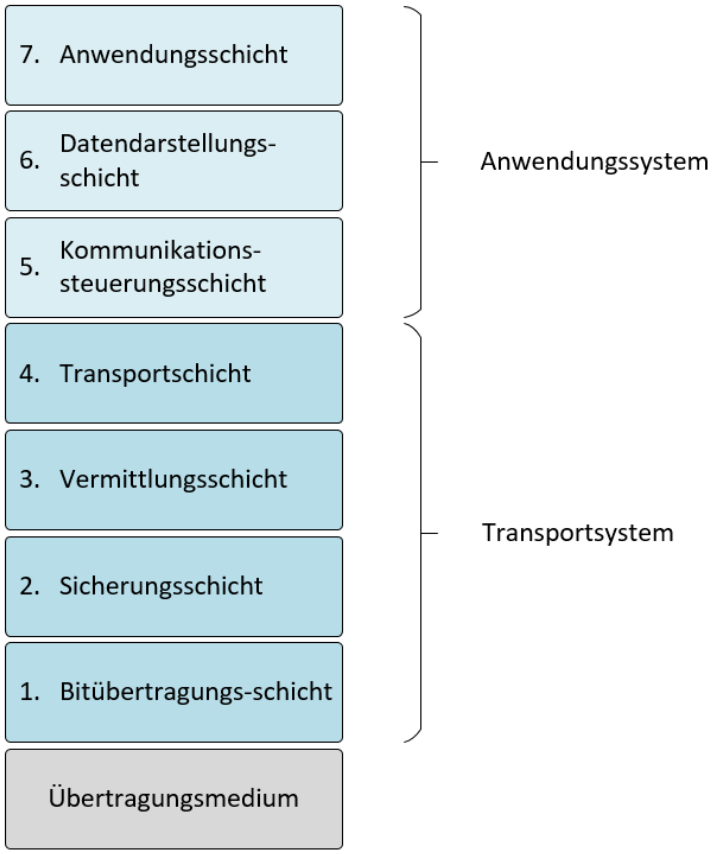


Abbildung 2.1.: OSI-7-Schichten-Modell

[2]MAC - Media Access Control

2.1.2. Ethernet

Bei Ethernet spricht man von einer paketvermittelnden Netzwerktechnik, deren Standards auf den Schichten 1 und 2 des OSI-Schichtenmodells die Adressierung und die Zugriffskontrolle auf unterschiedliche Übertragungsmedien definieren. Die Nutzdaten kommen bereits in Datenpaketen von den darüberliegenden Protokollen [26].

Ethernet-Frame

Der Ethernet-Frame baut auf der ersten und zweiten Schicht des OSI-Modells auf und ermöglicht es eine Kommunikation zwischen zwei Systemen im Ethernet mittels Ziel- und Quell-MAC-Adressen aufzubauen. Der in Abbildung 2.2 dargestellte Ethernet-Frame wird mit der Tabelle 2.1.2 in den die relevanten Informationen beschrieben [32].

Felder	Beschreibung
Ziel-MAC-Adresse	Die Hardware Zieladresse identifiziert den Netzwerkpunkt der die Daten empfangen soll mit der Größe 6 Byte.
Quell-MAC-Adresse	Die Hardware Quelladresse identifiziert den Sender mit der Größe 6 Byte.
VLAN-TAG (Optional)	Mit dem optionalen VLAN[3] kann ein Ethernet in Unternetze strukturiert werden. Feldbreite beträgt 4 Byte.
Typ	Der Typ gibt Auskunft über das nächst höhere verwendete Datenprotokoll. Als gängigster Typ soll hier das IPv4-Protokoll mit der Codierung 0x0800 angegeben werden. Feldbreite sind 2 Byte.
Daten (Payload)	Die Payload gibt die Menge der zu übertragenden Daten in der Schicht an. Mit höheren Kommunikationsschichten wird die Payload durch zusätzliche Header reduziert. Datenmenge 62 - 1500 Byte.
PAD	Das sogenannte „padding-field“ erscheint nur wenn die minimale Framelänge von 64 Byte nicht erreicht werden sollte. In dem Fall werden Bytes ergänzt.
Prüfsummenfeld	Die Prüfsumme ist eine 32-Bit-CRC[4]-Prüfsumme. Die Berechnung wird über den gesamten Frame, beginnend mit der Ziel-MAC-Adresse einschließlich PAD-Feld, bestimmt.

Tabelle 2.1.: Beschreibung des Ethernet-Frames

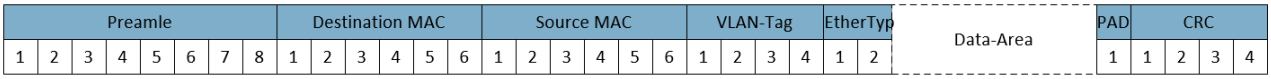


Abbildung 2.2.: Ethernet-Frame

2.1.3. Internet Protokoll

Das Internet Protocol (IP) ist eine Methode, beziehungsweise Protokoll, mit der Daten von einem IP-Netz in ein entferntes IP-Netz übertragen werden können. Das Protokoll setzt auf der dritten Schicht des OSI-Modells an. Jeder Computer in diesem Datennetz wird durch seine IP-Adresse eindeutig identifiziert. Aktuell existieren zwei Varianten in der Adressierung. Das IPv4[5] mit einem maximalen Adressbereich von $2^{32} \approx 4,3 \cdot 10^9$ Adressen und das IPv6[6] mit einem Adressbereich von maximal $2^{128} \approx 3,4 \cdot 10^{38}$ Adressen. Das IPv6 ist keine Erweiterung des IPv4 sonder ein neues Protokoll und wird in dieser Arbeit nicht näher betrachtet. Die nachfolgenden Beschreibungen und Erläuterungen beziehen sich ausschließlich auf das IPv4.

Mittels IP-Adresse und Subnetzmaske lassen sich IP-Netze in unterschiedliche Bereiche unterteilen und in Subnetze aufteilen. Das CIDR[7] beschreibt im RFC 6890 [5] die Nutzung des Adressbereiches im IPv4. Die Subnetzmaske dient der Separierung und bildet aus der IP-Adresse den Adress- und den Host-Bereich.

IPv4 Header

Der IPv4 Header setzt sich aus den in Abbildung 2.3 gezeigten und in Tabelle 2.1.3 beschriebenen Feldern zusammen.

[3] VLAN - Virtual Local Area Network
[4] CRC - Cyclic Redundancy Check
[5] IPv4 - Internet Protocol Version 4
[6] IPv6 - Internet Protocol Version 6
[7] CIDR - Classless Internet Domain Routing

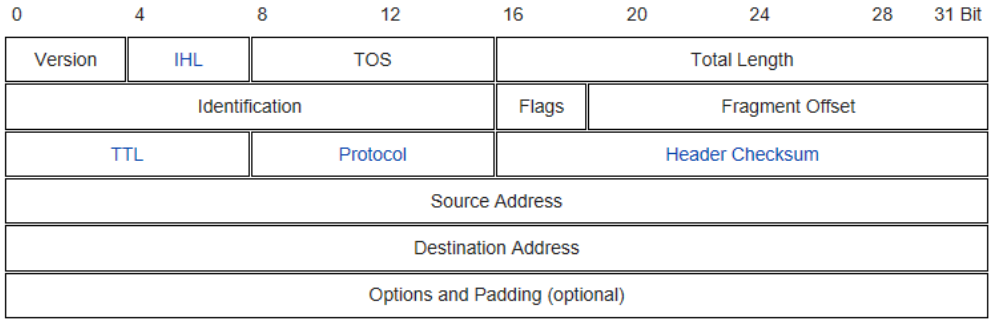


Abbildung 2.3.: IPv4 - Header [30]

Felder	Beschreibung
Version	Version des IP-Protokolls
IHL	Internet Header Length gibt die Länge des IP-Headers als ein Vielfaches von 32-Bit an.
TOS	Type of Service legt die Priorität des angeforderten Dienstes fest.
Total Length	Gibt die Gesamtlänge des IP-Paketes an.
Identification	Eine Nummerierung die fortlaufend hochgezählt wird und einer eindeutigen Reihenfolge der Pakete dient.
Flags	Bit-Position [0] = 0 / Bit-Position [1] = Don't Fragment → 1 Gibt an, dass ein Paket nicht zerlegt wurde. Bit-Position [2] More Fragment: → 0 Letzte Paket → 1 = weitere Fragmente folgen.
Fragment Offset	Gibt bei fragmentierten Paketen an aus wie vielen Elementen ein Paket besteht.
TTL	„Time of Live" gibt den Lebenszyklus eines Datenpaketes an. Dieser Wert wird bei jeder Router-Station verringert. Es soll damit verhindert werden, dass Pakete endlos durch ein Netzwerk geroutet werden.
Protocol	Diese Feld beschreibt das nachfolgende Protokoll. TCP-Paket = 6 und UDP = 17
Header Checksum	Die Checksumme wird nur auf dem IP-Header, nicht aus den Nutzdaten gebildet.
Source-Address	IP-Adresse der Station, die das IP-Paket abgeschickt hat.
Destination-Adress	IP-Adresse der Station, für die das IP-Paket bestimmt ist.
Options and Padding (optional)	Zusatzinformationen, werden meist für Diagnosezwecke verwendet.

Tabelle 2.2.: Beschreibung IPv4-Header

2.1.4. Transportprotokoll

Das Transportprotokoll ist in der 4. Schicht des OSI-Modells angesiedelt. Die zwei bekanntesten Protokolle sind TCP/IP und UDP. In dem nachfolgenden Abschnitt wird das UDP Protokoll kurz vorgestellt.

UDP

Das User Datagram Protocol (UDP) ist eine verbindungslose Datenkommunikation, was bedeutet, dass von dem Protokoll keine Empfangsbestätigung realisiert wird. Durch den geringen Protokoll-Header wird durch UDP die Bandbreite nicht so stark belastet wie dies bei einer TCP/IP Kommunikation auftreten würde. UDP verwendet Ports um Daten am Zielsystem dem richtigen Dienst zuzuweisen. Abbildung 2.4 zeigt den UDP-Header.

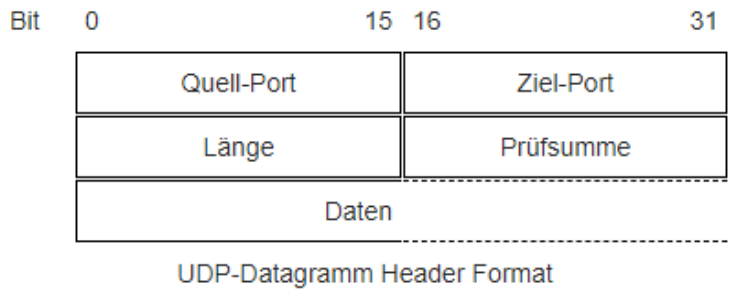


Abbildung 2.4.: UDP - Datagramm [31]

Felder	Beschreibung
Quell-Port	Gibt die Portnummer des zu sendenden Prozesses an.
Ziel-Port	Gibt an, welcher Prozess das Paket empfangen soll
Länge	Gibt die Länge der Daten an, bestehend aus den Daten und dem Header
Prüfsumme	Die Prüfsumme wird über den sogenannten Pseudo-Header, den UDP-Header und die Daten gebildet
Daten (Payload)	Nutzdaten

Tabelle 2.3.: Beschreibung IPv4-Header [31]

2.1.5. Übertragungstypen

In einem Ethernet gibt es drei Übertragungstypen:

- Unicast: Beschreibt eine Eins-zu-eins-Kommunikation zwischen einem Source- und Destinationdevice.
- Broadcast: Eine Übertragung von einem Device zu allen Endgeräten in einem Netz. Die spezielle IP-Adresse 192.168.1.255/24.
- Multicast: Beim Multicast wird eine Gruppe von Endgeräten mit einer speziellen IP-Adresse angesprochen. Im nachfolgenden Abschnitt wird detaillierter auf diese Übertragungsart eingegangen.

Multicast

IPv4-Multicast ist eine Punkt-zu-Mehrpunktverbindung die in einem speziellen IP-Adressraum (Tab. 2.4) nach der RTF3171[1] festgelegt ist.

Adressbereich	Beschreibung
224.0.0.0 - 239.255.255.255	Multicast Adressen
239.0.0.0 - 239.255.255.255	Administratively Scoped Address Block

Tabelle 2.4.: IPv4 Multicast-Adressen

Die Datenverteilung geschieht an den Netzknoten (Routern) auf dem Weg zu den Endknoten, sodass vom Sender keine höhere Netzauslastung verursacht wird. Um die Daten schon auf dem Ethernet-Level filtern zu können, werden die IP-Adressen auf bestimmte Pseudo-MAC-Adressen abgebildet. Für eine IPv4-Verbindung ist in Abbildung 2.5 die Generierung einer Pseudo-MAC-Adresse abgebildet. Zum koordinieren der Multicast-Nachrichten wird das IGMP[8] verwendet.

[8] IGMP - Internet Group Management Protocol

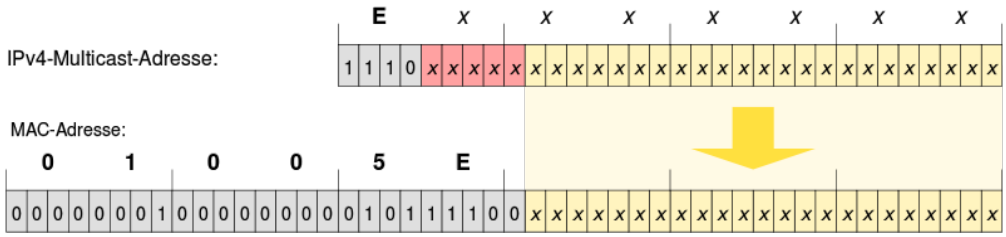


Abbildung 2.5.: IPv4 Multicast Abbildung der MAC-Adresse [22]

2.2. Kommunikation von Navigationsinformationen auf Schiffen

Der Datenaustausch von Navigationsinformationen auf Schiffen wird von der NMEA in dem Standard NMEA-0183 [20] definiert. Der Aufbau und die Struktur eines NMEA-0183 Datentelegramms ist dort spezifiziert und in dem Abschnitt 2.2.1 näher erläutert.
Die IEC überführte den nationalen Standard NMEA-0183 in die internationale Norm der IEC-61162 und erweiterte diesen nach den technologischen Anforderungen. In der aktuellen Fassung ist der Standard in fünf Teile gegliedert. Die relevanten Bereiche aus den Standards werden nachfolgend aufgelistet.

2.2.1. IEC 61162-1: Serielle Kommunikation und Datenprotokoll

Die IEC 61162-1 beschreibt die serielle Kommunikation, die in der Ausführung unidirektional zwischen einem Sender und mehreren Endgeräten stattfindet. Die Übertragungsrate ist fest spezifiziert. Die Daten werden, um Störeinflüsse zu verringern, differenziell mit der RS422-Schnittstelle übertragen. In den Teilen 2 und 3 wird eine bidirektionale Kommunikation mit höheren Übertragungsraten eingeführt.

Serielle Übertragung

Die serielle asynchrone Datenübertragung ist nach den in Tabelle 2.5 gelisteten Parametern definiert. Die Daten werden in einem Datenframe, der in Abbildung 2.6 dargestellt ist, beginnend mit dem least-significant-bit übertragen.

Baudrate	Datenbits	Parität	Stopbit
4800	8 (D7=0)	keine	1

Tabelle 2.5.: Serielle Kommunikation Parameter nach 61162-1

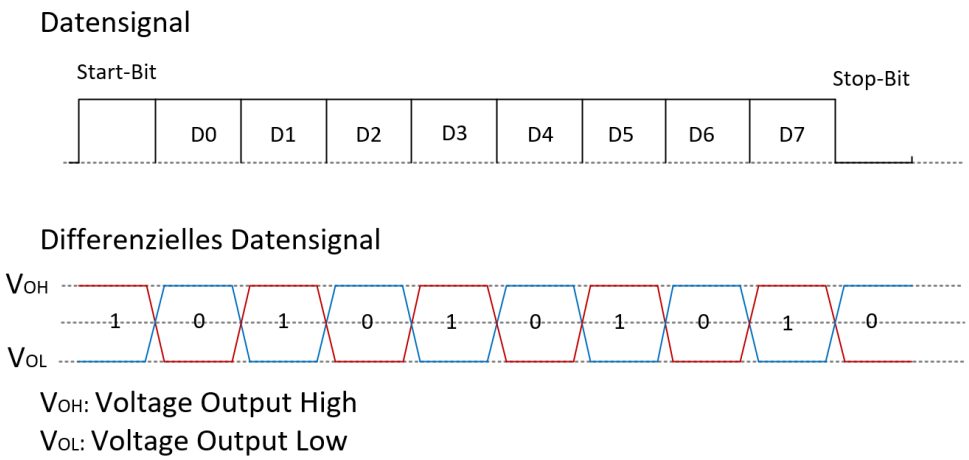


Abbildung 2.6.: NMEA Datenframe nach 61162-1

Struktur des Datenprotokolls

Die Struktur des NMEA-Datenprotokolls ist der essentielle Teil des Standards. Ein Datensatz, namentlich „**Sentence**[9]“ genannt, setzt sich immer aus einem Adress-, einem Daten- und einem Checksummen-Feld zusammen. Ein detaillierter Aufbau ist der Abbildung 2.7 zu entnehmen.
Alle zu übertragenden Daten werden als Char[10]-Zeichen, entsprechend einem Byte, kodiert. Ein Sentence beginnt mit dem “$“ als Start Symbol, gefolgt von einem 5-Byte breiten Adressfeld in dem der Talker-Identifier[11] und der Sentence-Formatter[12] deklariert werden. Das Adressfeld wird durch das definierte Trennzeichen “,“ von dem darauffolgenden Datenfeld abgeschlossen. In einem Datenfeld können mehrere Datensätze enthalten sein, die durch das Trennzeichen “,“ von einander separiert werden. Um Fehlinterpretationen zu vermeiden dürfen fehlende Datenwerte nicht als Null dargestellt werden und müssen durch ein doppeltes Trennzeichen, für keine Daten, signalisiert

[9] Sentence ist ein Datenframe nach IEC 61162-1
[10] Char ist ein Datentype der Breite 8-Bit. Mit jedem Byte kann ein ASCII Zeichen dargestellt werden.
[11] Der Talker-Identifier gibt an um welches Sensorgerät es sich handelt.
[12] Der Sentence-Formatter beschreibt das Datenprotokoll.

werden. Das numerische Trennzeichen bildet der dezimale Punkt. Ein Datenfeld wird mit dem ASCII-Zeichen “*“ beendet und verweist auf das folgende Checksummen-Feld. Resultierend aus der Codierung der Daten ergibt sich ein Datenwert der Breite von 2-Byte, entsprechend zwei Char.

Die Checksumme wird aus den Datenwerten beginnend mit dem Adressfeld bis Ende des Datenfeldes, einschließlich aller Trennzeichen, mit einer xor-Verknüpfung Symbolweise berechnet, wie in Abbildung 2.7 gezeigt. Der zu übertragende Sentence wird durch die Folge von zwei Char-Symbole “<CR><LF>“, entsprechend 2-Byte abgeschlossen.
Die maximale Länge eines Sentence ist mit 82-Byte einschließlich Start- und Ende-Signalisierung spezifiziert. In den Abbildungen 2.8, 2.9 und 2.10 sind die drei Sensorte-

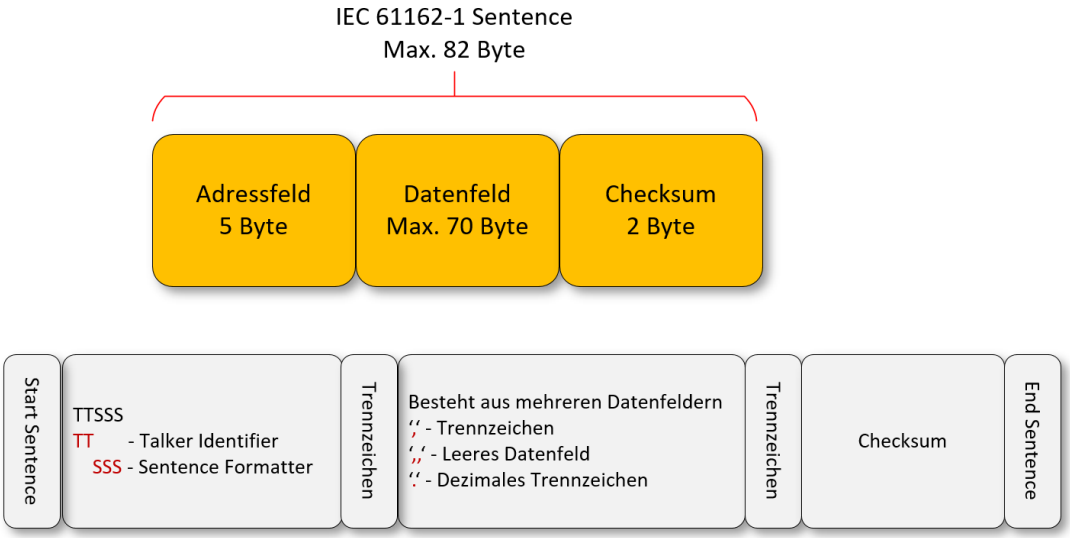


Abbildung 2.7.: Aufbau des Datenprotokolls nach IEC 61162-1

legramme dargestellt, die im Kapitel 6 für eine weitere Betrachtung benötigt werden.

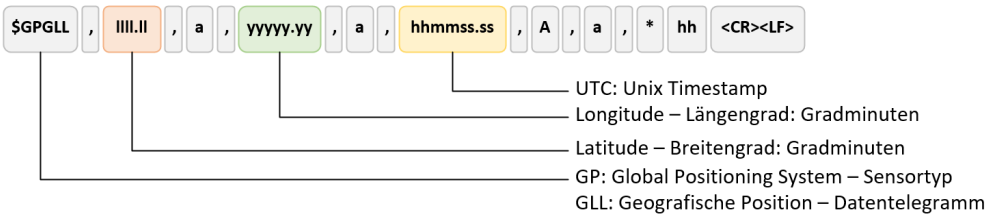


Abbildung 2.8.: NMEA-Sentence GPGLL, detailliert

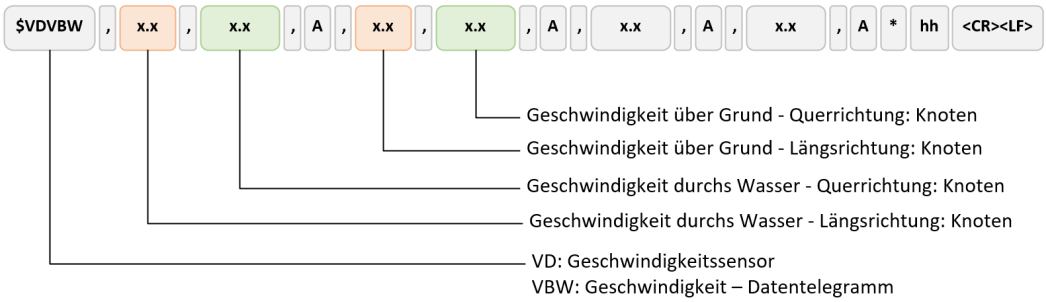


Abbildung 2.9.: NMEA-Sentence VDVBW, detailliert

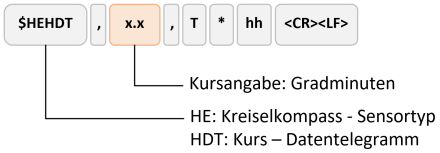


Abbildung 2.10.: NMEA-Sentence HEHDT, detailliert

2.2.2. IEC 61162-450: Ethernet Kommunikation

Die Erweiterung des Standards IEC 61162-450 beschäftigt sich mit der Ethernet Kommunikation von Daten die im Teil 1 des Standards definierten wurden. Als Netztopologie ist ein IP-basiertes Local Area Network (LAN) definiert in dem sich verschiedene Endgeräte befinden die bidirektional untereinander kommunizieren. Die Datenkommunikation wird auf dem Transportprotokoll UDP auf der 4. Schicht des OSI-Modells umgesetzt und als Kommunikationsmittel werden spezielle Multicastgruppen spezifiziert.
Abbildung 2.11 zeigt beispielhaft ein Datennetz nach 61162-450. In diesem Datennetz sind 5 verschiedene Funktionseinheiten definiert, siehe Tabelle 2.6, die unterschiedliche Funktion einnehmen. Als wichtiger Bestandteil dieser Arbeit zählt der SNGF[13]. In den anschließenden Kapiteln wird ein serielles NMEA-Telegramm aus dem 61162-1 in den 61162-450 Standard konvertiert.

[13]Serial to Network Gateway Function Block

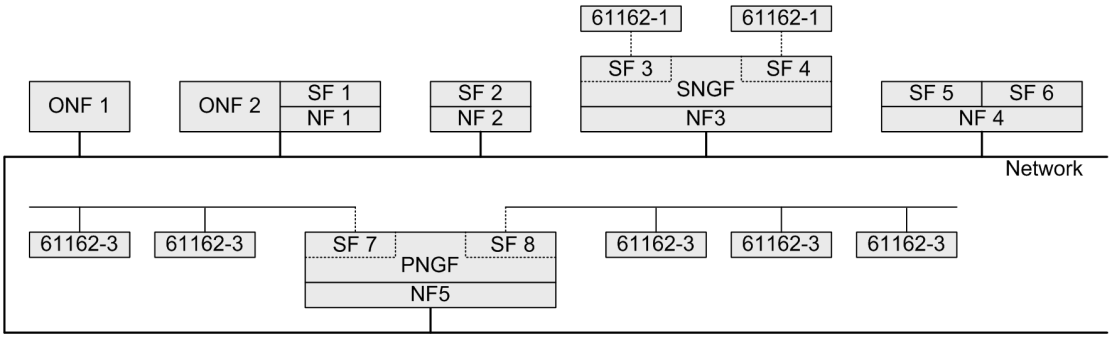


Abbildung 2.11.: Netzwerktopologie Beispiel nach 61162-450 [13, S.21]

Netzknoten	Beschreibung
ONF	Other Network Function Block
SF	System Function Block
NF	Network Function Block
SNGF	Serial to Network Gateway Function Block
PNGF	Parameter Group Number (PGN) to Network Gateway Function Block

Tabelle 2.6.: Netzknoten im 61162-450 Ethernet

Ethernet Frame

Der Ethernet Frame mit IEC 61162-450 Sentence ist der Abbildung 2.12 zu entnehmen. Der Sentence setzt sich aus einem Header und dem Payload-Bereich zusammen. Im Header wird festgelegt wie die Daten im Payload-Segment strukturiert sind. Eine detailliertere Darstellung der Daten ist in Abbildung 2.13 zu sehen. In dieser Arbeit wird nur auf den Header mit der Signatur „UdPbC“ eingegangen der eine Datenübertragung nach IEC 61162-1 spezifiziert. Der Vollständigkeit halber sind alle möglichen „Valid Header“ aufgeführt. In der Payload einer IEC 61162-1 Übertragung ist ein TAG Block-Feld, eine Checksumme und der Dateninhalt der IEC 61162-1 enthalten.

Die genaue Struktur des Ethernet-Protokolls wird nachfolgend detaillierter beschrieben. Das erste Element ist der Protokoll Header. Wie bereits zuvor erwähnt, gibt er an wie die Daten in den nachfolgenden Feldern strukturiert sind. Bei einem „UdPbC“ wird eine NMEA Übertragung nach dem IEC-61162-1 erwartet. Der Header wird mit dem Trennzeichen ‘\‘ von dem darauffolgenden TAG Block getrennt. Der TAG Block setzt sich aus folgenden Parametern zusammen:

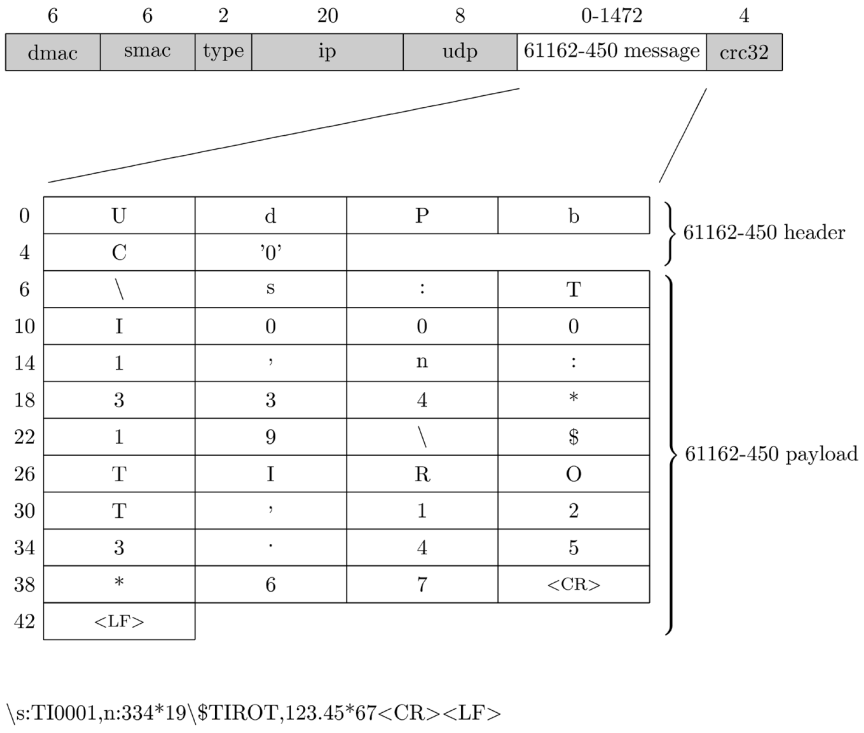


Abbildung 2.12.: Ethernet Frame nach IEC 61162-450 [13, S.32]

- s: Gibt die Quelle an von der Daten gesendet werden.
- d: Hier kann ein spezielles Ziel als Adresse definiert werden.
- n: Sequenz Nummer die fortlaufend inkrementiert wird.
- g: Nachrichten können in Gruppen zusammengefügt werden, wenn die Information über mehrere Nachrichten gesendet wird.
- c: UNIX-Zeitstempel
- t: Frei wählbares Textfeld.

Der TAG Block ist durch das Trennzeichen '*' von dem anschließenden Checksum-Feld getrennt. Die Checksum wird nach den gleichen Kriterien wie in Abschnitt 2.2.1 berechnet. Als Datengrundlage wird der komplette TAG Block in Abbildung 2.13 verwendet. Das Trennzeichen '\' schließt das Checksum-Feld ab. Angehängt folgt dann der in Abschnitt 2.2.1 ausführlich beschriebene NMEA IEC 61162-1 Sentence.

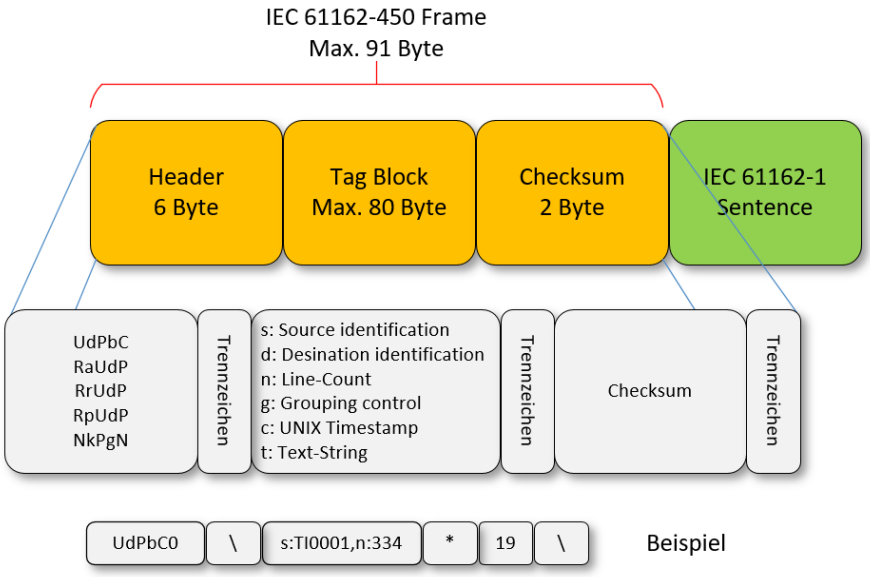


Abbildung 2.13.: UDP Payload einer IEC 61162-450 Übertragung

Multicastgruppen

Die Datenverteilung geschieht über spezielle Multicastgruppen denen Übertragungsgruppen zugeordnet sind. In Tabelle 2.7 sind die relevanten Multicast-Adressen aufgelistet. In dieser Arbeit werden Navigationsdaten der Übertragungsgruppe NAVD betrachtet.

Übertragungs-gruppen	Kategorien	Multicast-Adressen	Ziel-Port
MISC	Nicht näher spezifiziert	239.192.0.1	60001
TGTD	AIS Daten, Radar Ziel Nachrichten	239.192.0.2	60002
SATD	Navigationsdaten mit hoher Aktualisierungsrate	239.192.0.3	60003
NAVD	Navigationsdaten, die nicht in TGTD und SATD enthalten sind	239.192.0.4	60004
VDRD	VDR erforderliche Daten	239.192.0.5	60005
RCOM	Funkkommunikationsequipment	239.192.0.6	60006
TIME	Zeitübertragende Ausrüstung	239.192.0.7	60007

Tabelle 2.7.: Übertragungsgruppen mit Zuweisung von Multicast-Adressen und Ziel-Ports

2.3. Betriebssysteme

Die grundlegenden Funktionen eines Betriebssystems sind die Prozessverwaltung, die Benutzerverwaltung, der Speicherzugriff und die Verwaltung von Hardware-Ressourcen. In den folgenden Abschnitten wird die Linux-Architektur betrachtet.

2.3.1. Prozessmodell

Um in einem Mehrprogrammsystem die Abfolge zu steuern und Prioritäten festlegen zu können existiert ein Prozessmodell. Die Prozesse können in einem Linux-System acht Zustände (Abbildung 2.14) annehmen. Ein Prozess beginnt mit dem Zustand **neu**, durchläuft das Zustandsmodell bis der Prozess abgearbeitet und **beendet** wird. Ein fertig abgearbeiteter Prozess (Systemaufruf 'exit') wird in der Prozessliste als Zombie geführt, bis der Elternprozess den Rückgabewert (Systemaufruf 'wait') abgefragt hat.

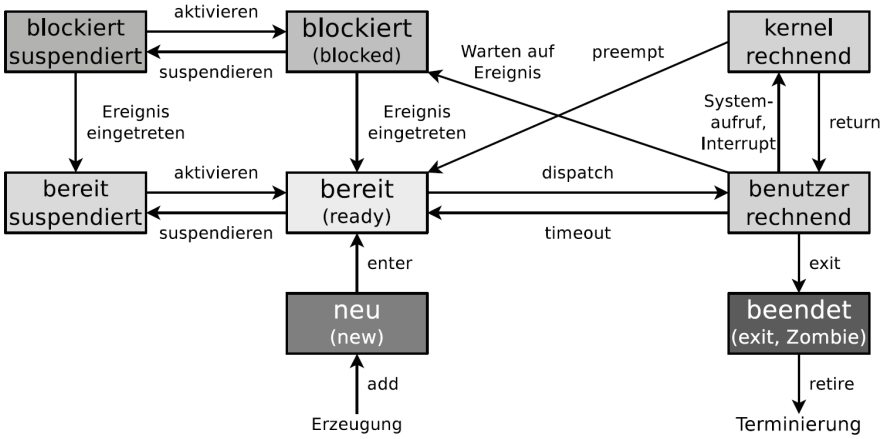


Abbildung 2.14.: Linux-Prozessmodell mit acht Prozesszuständen [3, S.150]

2.3.2. Scheduler

Der Scheduler gibt die Reihenfolge der Prozesse im Zustand 'bereit' für den Dispatcher vor. Die Aufgabe des Schedulers ist es unter Anderem die CPU möglichst effizient auszulasten und die Prozesse nach einem fairen Prinzip nach Prioritäten in der Warteschlange zu strukturieren. Der Dispatcher weist dann dem Prozessor den aktuell anstehenden Prozess zu und organisiert die Registerinhalte.

2.3.3. Prozesse und Threads

Prozesse bestehen aus einem Prozesskontrollblock und einem Speicherblock. Der Prozess-Kontrollblock enthält Informationen wie die Prozessidentifikation (PID) und Prozesszustandsinformation, die das Betriebssystem für die Steuerung benötigt. Der Speicherblock teilt sich in einen dynamischen Stack- und Heap-Bereich mit Registerblöcken und in ein statisches Textsegment, das den Programmcode beinhaltet, auf. Der Speicherbereich eines Prozesses ist nach außen geschützt. Es ist somit nicht möglich auf den internen Speicher eines Prozesses von außen zuzugreifen. Prozesse können Ressourcen zugeteilt werden.

Threads sind immer einem Prozess zugeteilt. Dabei können mehrere Threads demselben Prozess angehören und somit den gleichen Speicherbereich adressieren. Sie besitzen aber ihren eigenen Stack, Programmzeiger und Registersatz [6, S. 73 ff].

2.3.4. Interprozesskommunikation

Wegen des geschützten Speicherbereiches eines Prozesses wird eine Technik benötigt um Daten zwischen Prozessen austauschen zu können. Die Interprozesskommunikation bietet eine Auswahl verschiedener Methoden um den Datenaustausch zu ermöglichen. Nachfolgend werden die zwei relevanten Mechanismen ausführlicher beschrieben.

Message-Queue

Eine Message-Queue (Nachrichtenschlange) ist eine verkettet Liste, die nach dem Prinzip des FIFO-Speichers Daten in eine Liste schreibt und ausließt. Durch eine Priorisierung von Nachrichten kann das Auslesen von Daten neu geordnet werden. Die Message-Queue kann in einer nicht blockierenden Konfiguration betrieben werden. In diesem Zustand wird bei einem gefüllten Speicher das letzte Datensegment aus der Kette geschoben und es tritt ein Datenverlust ein. Alternativ dazu verhält es sich bei einer blockierenden Konfiguration. Dort wird bei gefülltem Speicher der aktuelle Datenwert verworfen. Die Steuerung der Message-Queue übernimmt das Betriebssystem [3, S.195 ff].

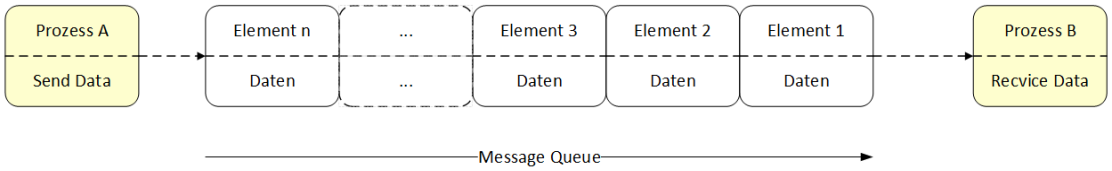


Abbildung 2.15.: Message-Queue

Sockets

Ein Socket ist ein Kommunikationsendpunkt. Er stellt eine Schnittstelle auf der Transportschicht (4. OSI-Schicht) zu einem Prozess her. Damit ist es möglich Prozesse über ein Ethernet miteinander zu verbinden [6, S. 221 ff].

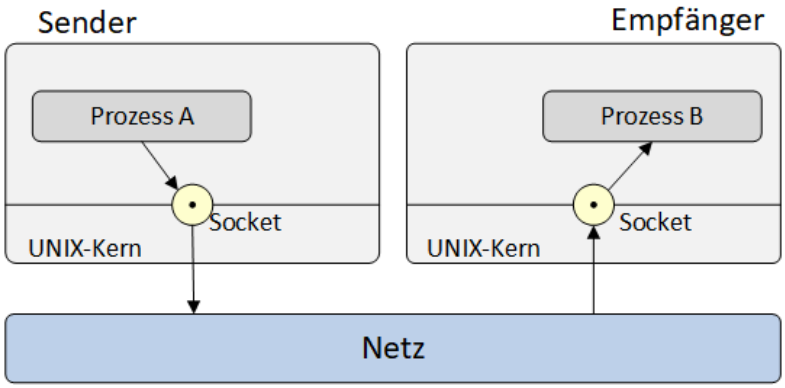


Abbildung 2.16.: UNIX - Socket

2.4. Firewall

Eine Firewall besteht aus mehreren Säulen. Die wichtigsten repräsentiert der **Paketfilter** und der **Application Level Gateway**. Der Paketfilter selektiert auf der 3. und 4. OSI-Schicht und der dahinter geschaltete Application Level Gateway führt eine Datenanalyse auf der 7. OSI-Schicht durch.

2.4.1. Paketfilter

Eine Paketfilterung wird auf der dritten und vierten Schicht des OSI-Modells durchgeführt. Mit den Header-Informationen aus der Vermittlungs- und Transportschicht können Pakete nach IP-Adressen und Portnummern oder Transportprotokoll selektiert werden. Bei der Paketfilterung wird zwischen statischen und dynamischen Filtern unterschieden. Die statischen Filter bearbeiten jedes Paket nach einem einheitlichen Schema und unterscheiden nicht nach vorhergehenden Kommunikationsabläufen. Dynamische Filter können für einen speziellen Kommunikationsablauf konfiguriert werden [16, S.161 ff].

Linux bietet eine Paketfilterfunktion, die sogenannte Netfilter - Architektur [2] (Abbildung 2.17), die im Kernel[14] integriert ist. Es existieren drei Regelketten (Chains). Eingehende Pakete, die einem Socket zugeordnet werden sollen, durchlaufen die INPUT-Regelketten und werden dann einem lokalen Prozess zugewiesen. Es ist dabei möglich ein spezielles Interface anzugeben durch das die Pakete gelangen müssen. Ausgehende Pakete werden durch die OUTPUT-Regelkette geleitet. Auch in diesem Fall muss ein Interface angegeben werden. Pakete bei dem das Routing erfolgt, durchlaufen die FORWARD-Regelkette und werden von einem Interface zum anderen übertragen.

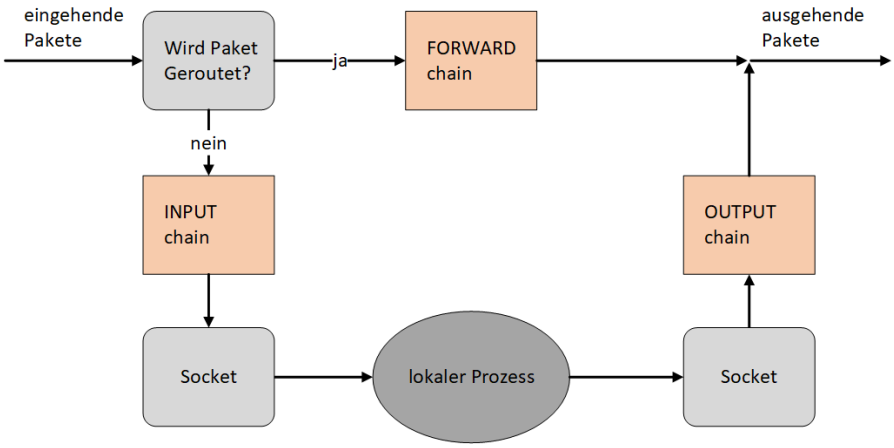


Abbildung 2.17.: Linux Netfilter-Architektur

[14] Als „Kernel“ wird der „Kern“ eines Betriebssystems bezeichnet, der grundlegende Funktionen wie Prozessverwaltung und Speicherzugriffe zur Verfügung stellt.

2.4.2. Application-Level-Gateways

Die Application-Level-Gateways (ALGs), auch umgangssprachlich Proxy-Server, arbeiten auf der siebten Schicht des OSI-Modells. Bei Proxys die für ein spezielles Protokoll konfiguriert wurden spricht man auch von einem dedicated Proxy. Die ALGs haben die Aufgabe Daten zwischen zwei Netzen zu filtern. Auf der Applikationsebene können Datenprotokolle analysiert und gefiltert werden, sodass Daten nicht direkt zwischen Client und Server ausgetauscht werden, sondern über die ALGs laufen müssen. So kann sichergestellt werden, dass der Dateninhalt im definierten Format vorliegt wenn es dem Zielsystem vom ALG zugewiesen wird.

Allgemeiner formuliert kann ein ALG damit folgende Funktionen übernehmen [16, S.184 ff]:

- Beschränkung der Kommunikation und der Applikationsfunktionen
- Umsetzen von benutzerspezifischen Regeln
- Überprüfung von Inhalten
- Es wird keine direkte Verbindung zwischen Sender und Empfänger ermöglicht
- Kommunikation wird auf der Applikationsebene aufgezeichnet.

2.5. Failure Mode and Effects Analyse

Die Produkt-FMEA ist eine spezielle Form der FMEA und wird vorzugsweise in der Planungsphase und während der Entwicklung von Produkten eingesetzt [29, S.14 ff]. Die aus den möglichen Fehlerszenarien erarbeiteten hypothetischen Fehlerfälle werden mit diesem Tool protokolliert, analysiert und ausgewertet. Die nachfolgenden Arbeitsschritte beschäftigen sich damit den Fehlerfolgen ein Bedeutungskriterium (**B**[15]) zuzuweisen und Vermeidungsmaßnahmen zu ermitteln, die gegen den Eintritt des Fehlerfalles wirken sollen. Aus dieser Bewertung wird eine Auftrittswahrscheinlichkeit (**A**[16]) bestimmt. Die Entdeckungswahrscheinlichkeit (**E**[17]) muss aus der gleichnamigen Maßnahme abgeschätzt werden. Als Ergebnis wird die Risiko-Prioritätszahl (**RPZ**) bestimmt, die sich aus dem Produkt der Faktoren (B,A,E) bestimmen lässt. Die RPZ ist ein Maß für die Systembedeutung eines Fehlers und kann als Indikator eingesetzt werden um eine priorisierte Bearbeitung daraus abzuleiten.

[15] $B \in \mathbb{N}[1, 10]$, je höher der Faktor, desto höher die Bedeutung des Fehlers auf die Systemfunktion.
[16] $A \in \mathbb{N}[1, 10]$, je kleiner der Faktor, desto geringer ist die Eintrittswahrscheinlichkeit.
[17] $E \in \mathbb{N}[1, 10]$, je höher der Faktor, desto geringer ist die Entdeckungswahrscheinlichkeit.

3. Anforderungsanalyse

In der Anforderungsanalyse werden die Anforderungen an das System erarbeitet. Die Anforderungen werden in zwei Bereiche gegliedert. Zum Einen den grundlegenden Anforderungen die sich aus der Zielsetzung, beschrieben im Abschnitt 1.2, ableiten lassen. Zum Anderen den Anforderungen zur Fehlervermeidung die aus einer Analyse der möglichen Fehlerarten, die in einer „Failure Mode and Effects Analyse“ erarbeitet wurden, bestimmt werden.

3.1. Grundlegende Anforderungen

Die grundlegenden Anforderungen leiten sich aus den im Abschnitt 1.2 festgesetzten Zielsetzungen unter Berücksichtigung der Randbedingungen ab und sind in Tabelle 3.1 aufgelistet. Die Kennung G_X beschreibt eine **grundlegende** Anforderung.

Kennung	Bezeichnung	Beschreibung
G1	Wartungs- und Erweiterbarkeit	Das Programm soll so strukturiert sein, dass es ohne Aufwand auf andere Systeme angepasst und erweitert werden kann.
G2	Syntax nach IEC 61162-1	Der Sicherheitsserver soll Sensortelegramme nach IEC 61162-1 spezifizierten „approved-Sentece“ auf ihren Syntax überprüfen. Überprüft werden soll auf zulässige Symbole und Datenlänge.
G3	Syntax nach ICE 61162-450	Die Header-Informationen des Sensors sollen auf ihren Syntax überprüft werden. Überprüft werden soll auf zulässige Symbole und Datenlänge.
G4	Plausbile Sensordaten	Die drei Sensortelegramme „GLL, HDT und VWB“ , beschrieben in Abschnitt 2.2.1, sollen auf ihre Konsistenz und Korrektheit geprüft werden.
G5	Visualisierung	Der Datenfluss durch den Server soll mit einem Monitoring visualisiert werden.

Tabelle 3.1.: Grundlegende Anforderungen

3.2. Analyse der Fehlerszenarien

Erweitert werden die Anforderungen durch eine Analyse der Fehlerszenarien. Der Schwerpunkt steht in einer fachlichen Einschätzung von möglichen vorstellbaren Szenarien und den damit verbundenen Fehlermöglichkeiten.

3.2.1. Szenarien

Die Szenarien beschreiben abstrakt einen Zustand, den der Autor für realistisch eintreffende Betriebsfälle in einem IP-Datennetz erachtet.

1 Ein Sensor arbeitet nicht ordnungsgemäß und sendet falsche Daten in eventuell zu hohem Intervall. Typischer Sendeintervall wäre einmal die Sekunde. Die dadurch entstehende erhöhte Netzauslastung führt zu einer Überlastung des Zielsystems und verhindert die reguläre Verarbeitung. Ein Verlust von Datenpaketen im Netzwerksocket könnte die Folge sein.

2 Sensordaten werden nicht in dem Normen beschrifteten Format gesendet. Ein Fehler in der Konfiguration der Quelle könnte die Ursache sein.

3 Induktive Störungen durch fehlerhafte Abschirmung des Übertragungsmediums führen zu Bitfehlern in der Datenübertragung.

4 Manipulation von Sensorwerten durch den „Man-in-the-Middle-Angriff“: Es wird versucht sich unbemerkt in die Kommunikation zwischen zwei Partner einzuschleusen um Daten mitzulesen oder zu verändern.

3.2.2. FMEA Matrix - Sicherheitsserver

Anlehnend an das FMEA-Formblatt der VDA[1] [29, S. 233] ist ein eigenes Formblatt 3.3 entstanden. Die allgemeine Struktur entspricht der Vorlage die mit der signifikanten Änderung eines Klassifizierungsmerkmals ergänzt wurde. Die Klassifizierung ist ein Auswahlschlüssel mit dem die Bearbeitungspunkte der FMEA strukturiert nach Bearbeitungspriorität und Machbarkeit unter Berücksichtigung des RPZ-Wertes eingestuft werden. Eine ausführlichere Erläuterung der Klassifizierung kann dem Abschnitt 3.2.3 entnommen werden.

[1] Verband der Automobilindustrie

3.2.3. Klassifizierungskriterien

Die Auswahlkriterien sind in zwei Hauptbereiche gegliedert. In die allgemeinen Fehlerszenarien(**K**) und in die Angriffsszenarien (**A**), die eine bewusste Fehlerfunktion hervorrufen sollen um das System zu schwächen oder auszuhebeln.

Indizes	Klassifizierung	Beschreibung
A1	Angriffsfälle	Im Allgemeine handelt es sich um aktive Fehler die es zum Ziel haben dem System in jeglicher Art zu schaden.
K1	Low Hanging Fruits	Low Hanging Fruits sind Aufgaben die schnell erledigt werden können. Es handelt sich dabei um eine wenig aufwendig und schnell zu erledigende Aufgabe die zu einem relevanten Ergebnis führt.
K2	Hohe RPZ	Durch eine hohe RPZ kennzeichnen sich Fehler die eine hohe Bedeutung für das System haben.
K3	Kleine RPZ	Durch eine geringe RPZ gekennzeichnete Fehler bedeuten eine geringere Gewichtung für das System.
K4	Nicht umsetzbar	Fehlerquellen die sich nicht durch die Systemkomponente beheben lassen. Der Ansatz für die Fehlerbehebung liegt außerhalb der hier bearbeiteten Komponente.
K5	Hoher Aufwand	Die Behebung der Fehlerart ist im Verhältnis zum Nutzten gering.
K6	Hohe Relevanz	Bedeutet, dass trotz eventuell geringer Auftrittswahrscheinlichkeit, der Eintritt eines Fehlers eine hohe Bedeutung auf die Sicherheit des Systems hat.
K?	Nicht entscheidbar	Fehlerarten, denen keine eindeutige Maßnahme ohne weitere Analyse zugeteilt werden kann.

Tabelle 3.2.: Auswahlkriterien - FMEA

Bachelorarbeit Sicherheitsserver für Navigationsinformationen in schiffsinternen Datennetzen	FMEA Failure Mode and Effects Analyse	Erstellt: Name: Nils Parche Datum: 13.07.2018 Seiten: 3
Typ/Modell/Fertigung Netzwerksystemkomponente	Produkt-FMEA/Systemelement: Sicherheitsserver	Bearbeitet: Name: Nils Parche Datum: 16.08.2018

mögliche Fehlerfolge	B	mögliche Fehlerart	mögliche Fehlerursache	A	Entdeckungs-maßnahme	E	RPZ	Vermeidung-maßnahme	Klassifi-zierung
Systemelement: Sicherheitsserver									
Funktion: NMEA Datensätze nach IEC 61162-1 einlesen / auswerten / analysieren									
Fehlermeldung an das Monitoring-System: Warninglevel. Verwerfen der Daten.	7	Checksumme des NMEA Sentence nach IEC 61161-1 stimmt nicht mit den Daten überein.	Fehler in der Datenübertragung.	5	Mitlaufende Checksumme in der Datenanalyse.	2	70	Plausibilitätsüberprüfung im ALG.	K1
Fehlermeldung an das Monitoring-System: Alarmlevel. Verwerfen der Daten.	10	Allgemeiner Aufbau des Datenframes entspricht nicht dem Syntax siehe Abbildung 2.7.	Fehlerhaft konfigurierter Sensor.	5	Parsen des Datenframes nach Syntax und zulässigen Symbolen.	3	150	Plausibilitätsüberprüfung im ALG.	K2
Fehlermeldung an das Monitoring-System: Alarmlevel. Verwerfen der Daten.	10	Sensorwerte liegen außerhalb physikalischen Grenzen.	Parametrierungsfehler der Sensoren. Modifizierung der Sensordaten von Extern.	4	Prüfen der Sensordaten nach dem Qualitätskriterium: Plausibilität [15].	2	80	Plausibilitätsüberprüfung im ALG.	A1,K6
Fehlermeldung an das Monitoring-System: Alarmlevel. Verwerfen der Daten.	7	Sensorwerte verhalten sich nicht nach dem Schiffsmodell	Parametrierungsfehler der Sensoren	2	Prüfen der Sensordaten nach dem Qualitätskriterium: Konsistenz [14].	2	28	Ermitteln von Datenwerten über ein Vorhersagemodell und verifizieren auf Konsistenz der Messwerte: Sensor Fusion.	A1,K5

mögliche Fehlerfolge	B	mögliche Fehlerart	mögliche Fehlerursache	A	Entdeckungs-maßnahme	E	RPZ	Vermeidung-maßnahme	Klassifi-zierung
Funktion: NMEA Datensätze nach IEC 61162-450 einlesen / auswerten / analysieren									
Fehlermeldung an das Monitoring-System: Warninglevel. Verwerfen der Daten.	7	Checksumme des NMEA Headers nach IEC 61161-450 stimmt nicht mit den Daten überein.	Fehler in der Daten-übertragung.	5	Mitlaufende Checksum-me in der Datenanaly-se.	2	70	Plausibilitätsüberprüfung im ALG.	K1
Fehlermeldung an das Monitoring-System: Alarmlevel. Verwerfen der Daten.	10	Allgemeiner Aufbau des Datenframes entspricht nicht dem Syntax siehe Abbildung 2.12.	Fehler in der Daten-übertragung.	5	Parsen[2] des Datenfra-mes nach Syntax und zulässigen Symbolen.	3	150	Plausibilitätsüberprüfung im ALG.	K2
Fehlermeldung an das Monitoring-System: Alarmlevel. Verwerfen der Daten.	10	Datensätze in falscher Reihenfolge.	Fehler in der Daten-übertragung. Manipulation des Da-tenheaders.	4	Überprüfen der Sequenz-Nummer im Datenheader.	2	80	Plausibilitätsüberprüfung im ALG.	A1, K6
Fehlermeldung an das Monitoring-System: Warninglevel. Verwerfen der Daten.	7	Verzögerung von Da-tensätzen.	Fehler in der Daten-übertragung. Manipulation der Daten.	5	Überprüfen des UNIX-Zeitstempel im Daten-header.	2	70	Plausibilitätsüberprüfung im ALG.	A1, K6
Funktion: Datenverareitung									
Fehlermeldung an das Monitoring-System: Alarmlevel. Verwerfen der Daten.	10	Datenframes von nicht registrierten Sensoren im Sensor-Netzwerksegment.	Konfigurationsdatei un-vollständig oder Sensor fehlerhaft eingestellt. Angriffsfall: Senden von Pseudodaten.	3	Kommunikationsprotokoll (Ethernet- und IP-Header) der Daten verifizieren nach IP und MAC Adressen.	6	180	Plausibilitätsüberprüfung im ALG. Konfigurieren der „iptables“ im Packet-filter: Blacklisting der IP-Adresse.	A1, K2, K5
Verwerfen der Daten.	10	Verbindungsaufbau aus dem Sensornetz auf nicht zulässige Ports-Anwendungen.	Angriffsfall: Lücken in der Firewall ausfindig machen	2	Auswerten der „ipta-bles“ Log Datei.	6	120	Konfigurieren der Pa-ketfilter: Input-Chain Regel anlegen und nicht zulässige Anwen-dungen/Portnummern sperren.	A1

[2] Analysieren, segmentieren und codieren von Daten.

mögliche Fehlerfolge	B	mögliche Fehlerart	mögliche Fehlerursache	A	Entdeckungs-maßnahme	E	RPZ	Vermeidung-maßnahme	Klassifi-zierung
Verwerfen der Daten.	10	Verbindungsaufbau von außerhalb des Sensor-Netzwerksegmentes.	Angriffsfall:	2	Auswerten der „iptables“ Log Datei.	6	120	Konfigurieren der Packetfilter: Input-Chain Regel anlegen und nicht zulässige Netzwerksegmente sperren.	A1
Verwerfen der Daten.	10	Verlust von Sensordaten im Eingangsspeicher des Sockets durch zu hohe Bearbeitungszeit im verbundenen Prozess.	Fehlerhafte Sensorkonfiguration Angriffsfall: DDOS-Attacke[3]	3	Auswerten der Verbindungsparameter im ALG.	5	150	Multi-Prozess-Server implementieren und blacklisting von IP-Adressen.	A1, K5
Deadlock-Zustand des Systems, dadurch dass keine Prozess-IDs mehr vergeben werden können.	10	Zombie-Bombe im Server-Prozess.	Fehlerhafte Beendung von Child-Prozessen und keine Abfrage der Rückgabe-Werte.	6	Meldungen in den System-Log-Dateien. Abfrage der im System laufenden Prozesse.	5	300	Signalhandler für das SIGCHLD Signal implementieren und Status der Child-Prozesse abfragen und beenden.	K1, K2
System nicht Funktionsfähig.	6	Interprozess-Kommunikationskanäle nicht bereit.	Nicht definierte Startbedingungen.	6	Meldungen in den System-Log-Dateien. Abfrage der im System laufenden Prozesse.	5	180	Implementierung eines Initialisierungsprozesses, der eine definierte Reihenfolge der zu startenden Prozesse festlegt.	K1

Tabelle 3.3.: FMEA - Sicherheitsserver

[3] Distributed-Denial-of-Service

3.3. Anforderungen der Fehlervermeidung

Die in der FMEA-Matrix 3.3 bestimmten Entdeckungs- und Vermeidungsmaßnahmen werden in Anforderungen an das System überführt und sind in der Tabelle 3.4 aufgelistet. Anforderungen, die sich mit den aus Abschnitt 3.1 überdecken werden hier nicht extra aufgeführt. Die Kennung F_X beschreibt eine Anforderung zur **Fehlervermeidung**.

Kennung	Bezeichnung	Beschreibung
F1	Checksummen Verifizierung	Berechnung der Checksumme aus dem Datenframe des NMEA-Sentence nach IEC 61162-1 und verifizieren mit der empfangenden Checksumme im Datenfeld.
F2	Checksummen Verifizierung	Berechnung der Checksumme aus dem Tag-Block des IEC 61162-450 Datenframes und verifizieren mit der empfangenden Checksumme im Datenfeld.
F3	Sequenz-Nummer	Die im Datenheader des IEC 61162-450 vorhandene Sequenz-Nummer soll ausgewertet werden um die Reihenfolge der Datenframes zu analysieren.
F4	Unix Timestamp	Die im Datenheader des IEC 61162-450 vorhandene Zeitbasis soll ausgewertet werden um eine Information über die Übertragungsdauer der Pakete zu erhalten.
F5	Registrierte Sensoren	Das System muss in der Lage sein zwischen registrierten und fremden Sensoren zu unterscheiden. Verifizieren der IP und MAC-Adresse mit einer Liste.
F6	Kommunikationskanäle	Das System lässt Kommunikation nur auf den zu erwartenden Port zu. Informationen dazu sind dem Abschnitt 2.2.2 zu entnehmen.
F7	Netzzugriff	Kommunikation von außerhalb des Sensornetzwerkes soll geblockt werden und nicht zur Verarbeitung an die Prozesse weitergeleitet werden.
F8	Multiprozess Server	Der Server soll in der Lange sein mehrere Clients gleichzeitig bearbeiten können.
F9	Zombie-Prozesse	Bei einem Multiprozess-Server dürfen keine Zombie-Prozesse entstehen.
F10	Initialisierung	Der Sicherheitsserver muss einen festen definierten Initialisierungszustand ablaufen.
F11	Sensor Fusion	Bestimmen von fehlerhaften Sensorwerten eines bestimmten Sensortypes, GLL, HDT oder VBW mit dem Verfahren „Koppelnavigation“, beschrieben im Abschnitt 5.3.3.
F12	Man-in-the-Middle	System soll in der Lage sein veränderte Daten zu erkennen.

Tabelle 3.4.: Anforderungen der Fehlervermeidung

4. System

In diesem Kapitel wird die Hardware bestimmt mit der die Systemkomponente „Sicherheitsserver“ und die notwendige Testumgebung ausgestattet werden sollen.

4.1. Randbedingungen

Aus der Aufgabenstellung lässt sich das in Abbildung 4.1 gezeigt globale Konzept darstellen. Der Sicherheitsserver stellt die zu entwickelnde Systemkomponente dar. Im Wesentlichen sollen zwei IP-Netzsegmente mittels Proxy-Server über eine Ethernet-Verbindung miteinander verbunden werden.

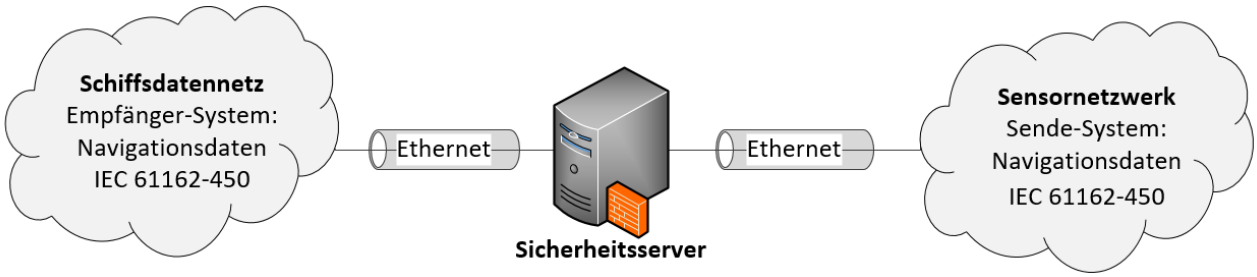


Abbildung 4.1.: Globale Systembeschreibung

In der Zielsetzung wird eine Nachrichtenkommunikation vom Sensor- in das Schiffsdatennetz beschrieben. Eine Abschätzung der zu erwartenden Datenlast besteht auf Grundlage einer typischen Ausstattung von Navigationssensorik auf einem Schiff (siehe Tabelle 4.1).

4.1.1. Datenübertragung

Zunächst wird bestimmt mit welcher Datenmenge für ein in Abbildung 4.1 dargestelltes Datennetz gerechnet werden kann. Dazu wird eine Standard Navigationsanlage betrachtet, die mit der in Tabelle 4.1 typischen Sensorik ausgestattet ist.

Sensorik	Talker-Identifier	Baudrate	k
GPS1	GP	4800	7,317
GPS2	GP	4800	7,317
Gyro1	HE	38400	58,537
Gyro2	HE	38400	58,537
Speedlog1	VD	4800	7,317
Speedlog2	VD	4800	7,317
Wind	WI	4800	7,317
Depth	SD	4800	7,317
AIS	AI	38400	58,537
NAVTEC	IN	4800	7,317

Tabelle 4.1.: Sensorik eines Navigationsystems

Um das zu erwartende Datenvolumen zu bestimmen, wird in grober Näherung angenommen, dass die Baudrate mit der maximalen Größe an NMEA-Datenframe ausgenutzt wird. Da ein Baud einem Bit/s entspricht, kann man mit dem Faktor k die zu übertragenden NMEA-Sentence pro Sekunde mit einer Länge von 82 Byte angeben.

$$\begin{aligned} k_x &= \frac{Baudrate_x}{8\,Bit} \cdot \frac{1}{Sentence_{max}} \\ &= \frac{Baudrate_x}{8\,Bit} \cdot \frac{1}{82\,Byte} \end{aligned}$$

Das Datenvolumen setzt sich aus den Sensordaten plus den Ethernet-, IP- und UDP-Header zusammen. Daraus ergibt sich ein maximaler DatenFrame D_x:

$$\begin{aligned} D_x &= Ethernet\,Packet\,Header + IP\,Header + UDP\,Header + Payload \\ &= 26Byte + 20Byte + 8Byte + (91 + 82)\,Byte \\ &= 227Byte \end{aligned}$$

Die Datenübertragungsrate lässt sich allgemein mit der Formel 4.1 beschreiben. C ent-

spricht der Datenübertragungsrate, D gibt die Datenmenge in Byte und t die Zeit in Sekunden an.

$$C = \frac{D}{t} = \frac{Byte}{sec} \tag{4.1}$$

Damit lässt sich eine grobe Abschätzung der maximalen Datenübertragungsrate C_{max} der in Tabelle 4.1 aufgeführten Sensoren nach Formel 4.2 berechnen. Hinweis: Der Faktor k wird zur Bestimmung der Datenmenge aufgerundet, da es keine gebrochenen Datenframes gibt und damit eine schlechtere Abschätzung gewählt wird. Der Faktor k hat die Einheit $\frac{1}{sec}$.

$$C_{max} = \sum_{n=0}^{N} = D_x \cdot k_n = 52,891\ kByte \tag{4.2}$$

Bei der in Tabelle 4.1 beschriebenen Navigationssensorik würde sich eine Übertragungsrate von 52,831 kByte/s ergeben. Gängige Ethernet-Interfaces mit einer geringen Übertragungsrate von $10\frac{MBit}{sec} \approx 1,25\ \frac{MByte}{sec}$ wären noch ein Vielfaches schneller als die benötigte Übertragungsrate.

4.2. Komponentenauswahl

Nachfolgend wird unter Einbeziehung der Randbedingungen eine Auswahl an Komponenten vorgestellt, die für den Einsatz geeignet erscheinen.

Sicherheitsserver

Für die Bestimmung der Hardwareauswahl wurde eine Recherche der zu Verfügung stehenden Alternativen durchgeführt. Die drei relevanten Systeme aus der Recherche sind folgende: Raspberry Pi mit externen Ethernet-Interface, BeagleBoard X15 und eine Firewall der Firma adstec vom Typ IRF2000.

Als **erste Variante** ist der Raspberry Pi Model 3 B+ vorgesehen. Er muss zur Verwendung mit einem erweiterten Netzwerkinterface versehen werden. Hier kommt das Adaptermodul von TP-Link Typ UE300 zum Einsatz. Die Abbildung 4.2 zeigt die beiden

Komponenten. Eine Auflistung und Gegenüberstellung der technisch relevanten Positionen wird für alle drei System in der Tabelle 4.2 dargestellt. Neben den reinen technischen Aspekten wird der fachliche Kontext ebenfalls mit einbezogen. Der Raspberry Pi ist ein weitverbreitetes Produkt, dass mit vielen Treibern von Peripherie unterstützt wird. Als Betriebssystem werden diverse Linux-Distributionen und Windows 10 IoT[1] angeboten.

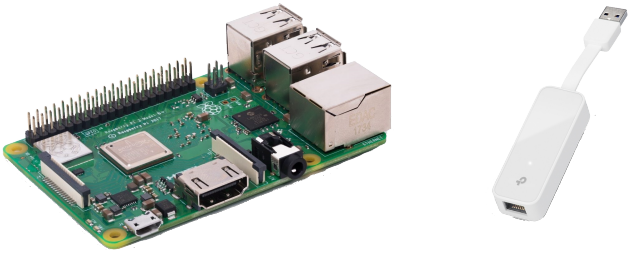


Abbildung 4.2.: Raspberry Pi Model 3 B+ [9] und Ethernet-Adapter TP-Link UE300 [28]

Die **zweite Variante** bildet das BeagleBoard X15 von der Firma Texas Instruments, in Abbildung 4.3 dargestellt. Auf der Platine sind bereits zwei Netzwerk-Interfaces vorhanden. Als Betriebssystem wird ein vorkonfiguriertes Debian der Version 9.4 angeboten.

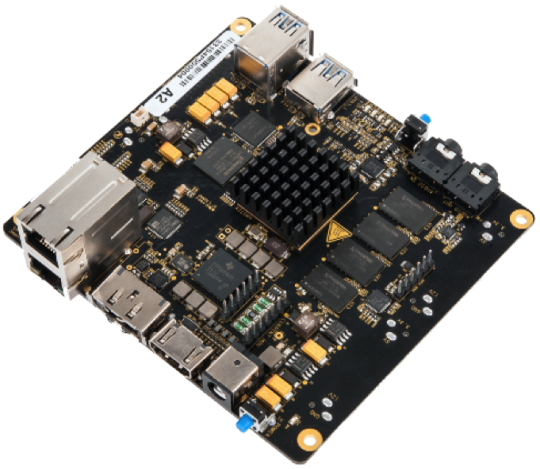


Abbildung 4.3.: BeagleBoard X15 [8] - Texas Instruments

Als **dritte Variante** wird die Firewall IRF2000 der Firma adstec aufgeführt. Eine Darstellung der Komponente ist in Abbildung 4.4 zu entnehmen. Dabei handelt es sich um

[1] Internet of Things

eine bereits vorkonfigurierte Firewall mit einem Embedded Linux[2] als Betriebssystem. Als Schnittstelle für die Programmierung eigener Applikationen wird das Java und ein OSGI-Framework angeboten.

Abbildung 4.4.: Firewall IRF2000 [11] der Firma adstec

In der Tabelle 4.2 sind die relevanten technischen Eigenschaften der Komponenten zusammengefasst aufgelistet. Anhand dieser Informationen wird eine Abwägung zur Entscheidungsfindung im Abschnitt 4.4 getroffen.

Technische Details	**Komponenten**		
	Raspberry-Pi 3 B+ UE300	**BeagleBoard X15**	**IRF2000**
CPU	ARM-Cortex-A53 1.5 GHz 64-Bit Quad-Core	ARM-Cortex-A15 1.5 GHz 32-Bit Dual-Core	Embedded-System Keine Informationen
Ethernet-Ports	intern 1xGbit extern 1xGBit/s[3]	2x Gbit/s	2x Gbit/s
USP[4]	• Große Auswahl an unterstützenden Betriebssystemen.	• Leistungsstarke CPU.	• Zulassung nach DNV GL[5]. • Vorkonfigurierte Firewall.
Contra	• Kein zweites integriertes Netzwerk-Interface.	• Nur ein verfügbares Betriebssystem.	• Java als einzige Programmiersprache für Applikationen.

Tabelle 4.2.: Vergleich der Hardwarekomponenten

[2] Ein System mit einem auf dem Linux-Kernel basierenden Betriebssystem.
[3] max. 310 MBit/s
[4] Unique Selling Point, Alleinstellungsmerkmal
[5] Internationale Klassifikationsgesellschaft

4.3. Sensornetzwerk

Um die im Szenarium Abbildung 4.1 dargestellten Bedingungen umzusetzen, wird eine Sensorik benötigt. Die damit erzeugten Daten werden für spätere Tests und Analysen benötigt um die Software zu validieren. Um die Sensordaten zu erhalten bietet sich eine Simulationsumgebung an mit der es möglich ist eine Vielzahl an Sensordaten zu generieren.

4.3.1. Simulation

Bei der Simulationsumgebung wird auf eine schon existierende Software zurückgegriffen, die NMEA-Telegramm in serieller RS422 Kodierung ausgeben. Um die Daten in das IEC 61162-450 Format zu kodieren wird ein neues Modul benötigt das die Anforderungen erfüllt.

NMEA Studio

Mit dem NMEA Studio [24] können Sensoren angelegt und miteinander verknüpft werden. So ist es möglich eine Schiffssimulation zu entwerfen, die alle verfügbaren Navigationsdaten auf die seriellen COM-Schnittstellen liefert. Jeder Sensor wird dabei einem COM-Anschluss zugewiesen. Dem Anhang auf der CD kann die Konfiguration der Software entnommen werden. Da es sich hierbei nur um ein Hilfsmittel handelt wird in der Arbeit nicht weiter auf das NMEA Studio eingegangen.

MOXA - Konverter

Als serielle Schnittstelle wird ein Konverter der Firma MOXA vom Typ UPORT 1450 eingesetzt. Das Gerät erweitert die verfügbaren RS422 Schnittstellen auf vier und dient dem NMEA Studio als Ausgangsschnittstelle.

MagicPlex - Veinland

Der MagicPlex der Firma Veinland ist ein Konverter, der aus seriellen NMEA Telegrammen auf Standard des IEC 61162-1 ein nach der Norm IEC 61162-450 konformes Ethernet Telegramm konvertiert. Der in Abbildung 4.5 dargestellte Konverter ist in der Lage

zehn serielle Eingänge in das geforderte Ethernet-Format zu übertragen. Auf die Geräte-Konfiguration wird im Umfang dieser Arbeit nicht weiter eingegangen.

Abbildung 4.5.: Veinland MagicPlex - Matrix: Seriell zu Ethernet

Die Abbildung 4.6 zeigt die grafische Benutzeroberfläche, in der die Konfiguration der seriellen Schnittstellen und die TAB-Block, beschrieben in Abschnitt 2.2.2, spezifiziert werden.

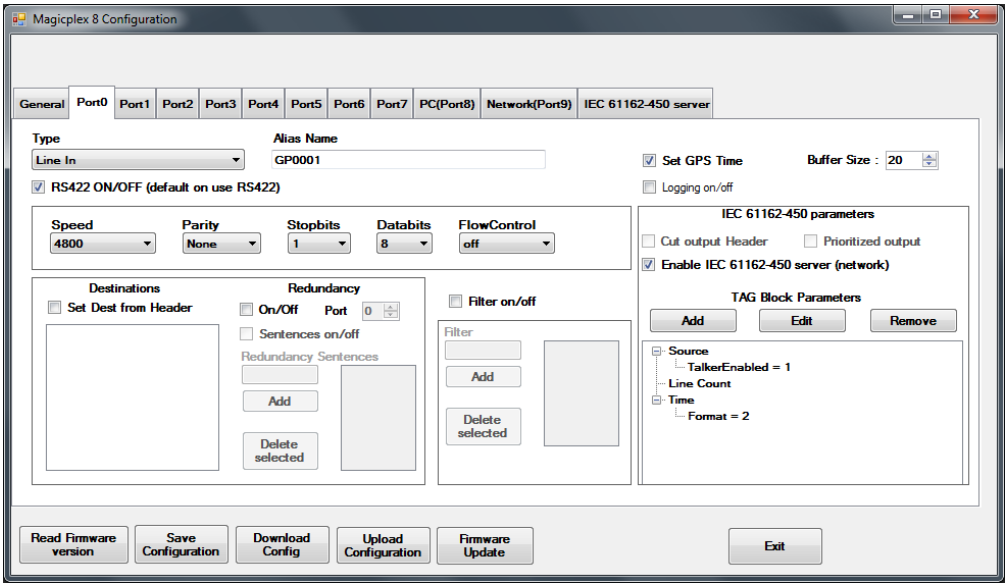


Abbildung 4.6.: MagicPlex - Grafische Benutzeroberfläche [10]

4.4. Entwurfsentscheidungen

Die in diesem Abschnitt begründeten Entwurfsentscheidungen leiten sich von den in Kapitel 3 beschriebenen Anforderungen und aus den in Abschnitt 4.1 dargestellten Randbedingungen, ab.

1 Bei der **Systemkomponente** wurde sich als Prototyp für ein Raspberry Pi der dritten Generation mit einem zweiten externen Ethernet-Interface, der mit dem Open-Source Betriebssystem Raspbian betrieben wird, entschieden. Ein wesentlicher Entscheidungspunkt für die Wahl eines Ein-Platinenrechners mit einem Linux-System stellte die gute Entwicklungsunterstützung sowie die Leistungsperformance des ARM-Cortex-A53 mit einer Taktfrequenz von 1,4 MHz dar.

2 Die Wahl der unterstützenden **Programmiersprachen** fiel auf die hardewarenahe objektorientierte Sprache C++. Mit ihr lässt sich eine hardwarenahe Programmierung an den Sockets effizient umsetzen und sie bietet dazu eine bessere Code-Strukturierung durch den Einsatz von Objekten, als in C. Des Weiteren bietet C++ die Möglichkeit mit der String-Klasse Charakter Operationen einfacher in der Entwicklung umzusetzen.

3 Die **Hauptanwendung** des Systems wird in die zwei Teilbereiche Paketfilter und Application Level Gateway unterteilt. In dieser Arbeit wird nur eine unidirektionale Datenkommunikation vom Sensordatennetz in das Schiffsdatennetz betrachtet, sodass der Paketfilter in dieser Konstellation nur für das Ethernet-Interface eth0 konfiguriert wird. Wie in Abschnitt 2.4.1 beschrieben arbeitet der Paketfiler, die sogenannten iptables, auf der dritten und vierten Schicht des OSI-Referenzmodell und stellt die selektierten Daten den Sockets bereit. Die im Linux-Kernel bereitgestellten Paketfilter werden durch ein whitelisting erlaubter IP-Adressen so konfiguriert, dass nur ausgewählter Datenverkehr in den dahinter angesiedelten ALG weitergeleitet wird. Damit soll sichergestellt werden, dass die Datenanalyse durch eine Überlastung keine fehlerhafte Auswertungen vornimmt. Der ALG nimmt die Daten vom Socket entgegen und analysiert die Daten in mehreren Stufen. Abschließend werden die validen Daten in das Zieldatennetz gesendet.

4 Als **Systemarchitektur** des Application Level Gateways wird ein Pipeline Konzept [21, S.138] aus drei Hauptprozessen (Server, Analyse und Client) vorgesehen, die nach einem Nachrichten basierten Modell [4, S.181 ff] organisiert sind. Der Server wird als Multilevelprozess implementiert um bei erhöhten Client Anfragen nicht zu einem Datenverlust im Socket-Speicher zu führen. Der unidirektionale Datenaustausch zwischen Server- und Analyseprozess, sowie Analyse- und Clientprozess, wird durch Message-Queues, die nach dem FIFO-Prinzip arbeiten, sichergestellt.

4.5. Aufbau

Aus den Entwurfsentscheidungen lässt sich der in Abbildung 4.7 dargestellte Aufbau bestimmen. Die Simulation (blaue Kästen), umfasst einen herkömmlichen Industrie PC auf dem NMEA Studio installiert ist. Die Daten aus der Simulation werden per USB an den seriellen RS422 Schnittstellenwandler weitergegeben. Für die drei definierten Sensortypen wird jeweils ein serielles-Interface auf dem MagicPlex aufgelegt. Der MagicPlex ist mit seinem Ethernet-Interface an das Sensordatennetz angeschlossen, an dem sich weitere IEC 61162-450 Geräte befinden können. Der Sicherheitsserver, bestehend aus dem Raspberry Pi Model 3 b+ mit zweiten externen Ethernet-Interface, ist mit seinem internen Interface am Sensordatennetz verbunden. Das externe Interface ist mit dem Schiffsdatennetz und einem Zielsystem, ebenfalls aus einem Raspberry Pi bestehend, angeschlossen.

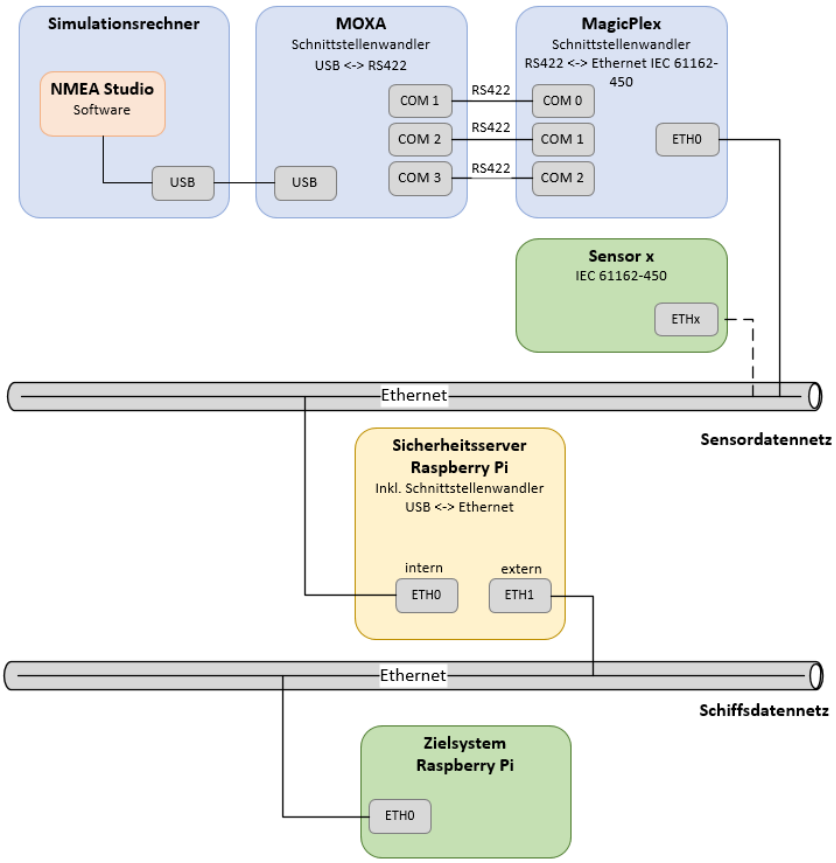


Abbildung 4.7.: Schematische Darstellung und Aufbau der im System verwendeten Komponenten

5. Konzept

Das Konzept beschreibt in einer abstrakten Betrachtungsweise die technische Umsetzung der Entwurfsentscheidungen. Die Bearbeitung wird anlehnend an die von arc42 [27] entwickelte Softwareentwicklungs-Methode beschrieben. Mit der Kontextabgrenzung wird das System nach außen mit allen Schnittstellen beschrieben und definiert somit die Kommunikationspartner. Die Beschreibung teilt sich in einen fachlichen- und technischen Kontext.

5.1. Fachlicher Kontext

In der fachlichen Darstellung werden alle globalen Kommunikationspartner beschrieben die mit dem Sicherheitsserver interagieren.

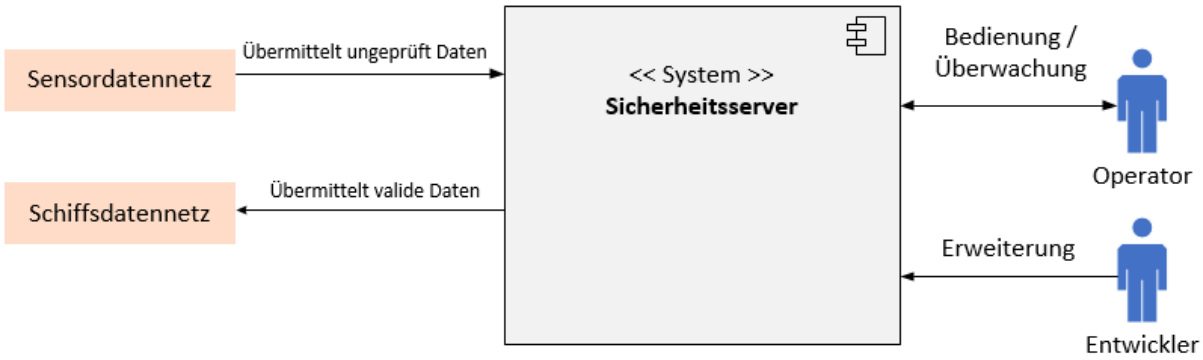


Abbildung 5.1.: Visualisierung des fachlichen Kontext - Sicherheitsserver

Sensordatennetz
Aus dem Sensordatennetz werden Navigationsdaten, die in Abschnitt 2.2 vorgestellt wurden, an den Sicherheitsserver übermittelt und dort nach spezifizierten Kriterien analysiert und bewertet.

Schiffsdatennetz
Die analysierten und als valide bewerteten Navigationsdaten gelangen vom Sicherheitsserver in das Schiffsdatennetz.

Operator
Als Operator wird die Person bezeichnet, die im laufendem Einsatz das System bedient. Der autark arbeitende Sicherheitsserver meldet dem Operator Alarm- und Warnmeldungen, die über ein Monitoring-Modul detaillierter visualisiert werden.

Entwickler
Der Entwickler stellt hier einen Softwareentwickler dar, der die Funktionalität des Sicherheitsservers erweitert oder anpasst.

5.2. Technischer Kontext

Die technische Übersicht stellt alle physikalischen Schnittstellen mit dem verwendeten Protokoll und Kommunikationsparameter dar.

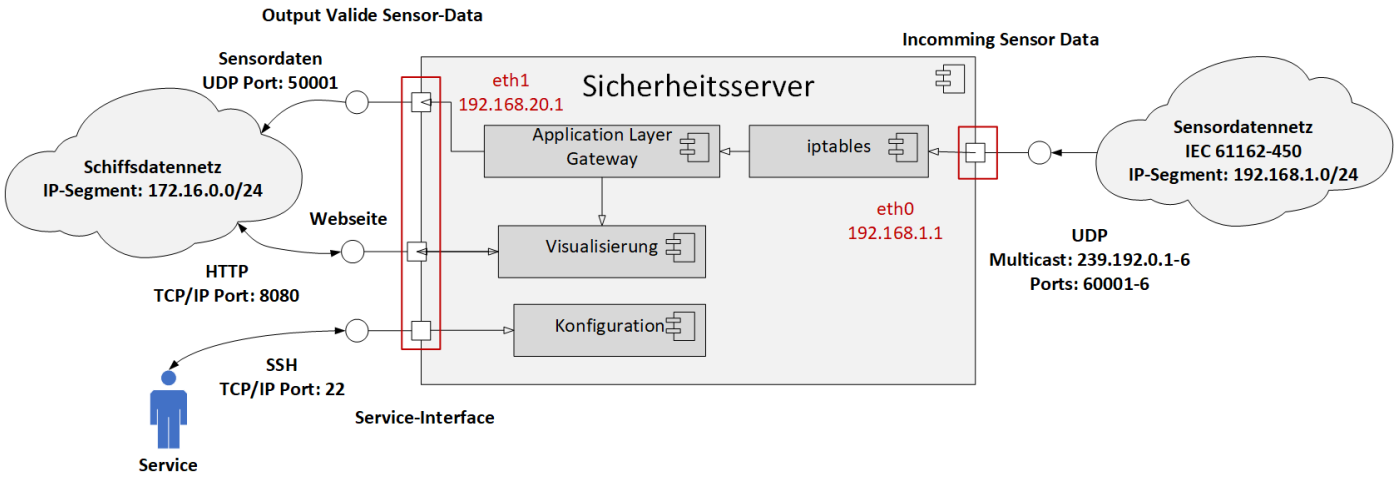


Abbildung 5.2.: Visualisierung des technischen Kontext - Sicherheitsserver

Schiffsdatennetz
Das Schiffsdatennetz ist in dem IP-Segment 172.16.0.0/24 definiert und kommuniziert mit dem Sicherheitsserver auf einer festen IP-Adresse 172.16.20.1 am Interface

eth1. Die Navigationsdaten werden vom Sicherheitsserver zum Zielsystem per UDP/IP-Protokoll an dem Port 50001 übertragen. Die Daten sind nach dem im Abschnitt 2.2.2 beschriebenen Format IEC 61162-450 kodiert. Das Monitoring-Modul wird als Webserver per HTTP-Protokoll angesprochen. Hierfür dient das TCP/IP Übertragungsprotokoll.

Sensordatennetz
Die Sensoren sind in einem frei wählbaren IP-Segment zu definieren. Gewählt wurde in diesem System das IP-Segment 192.168.1.0/24. Der Sicherheitsserver ist auf seinem Interface eth0 mit einer festen IP-Adresse 192.168.1.1 mit diesem Netz verbunden. Die Kommunikation wird mit dem IP-Multicast der in Abschnitt 2.1.5 beschriebenen Methode umgesetzt. Es existieren unterschiedliche Multicastgruppen, denen spezielle Ports zugewiesen sind, siehe Abschnitt 2.2.2. Die verwendeten Kommunikationsparameter sind: IP-Multicast 239.192.0.4 und Port 60004. Die kodierten Navigationsdaten nach IEC 61162-450 werden per Multicast mit dem Transportprotokoll UDP übertragen.

Service
Die Service-Schnittstelle ist an den Interface eth1 gebunden und lässt eine Administrierung des Systems zu. Die Verbindung wird mit dem SSH-Protokoll aufgebaut. Die Kommunikation läuft über TCP/IP auf dem Port 22. Der Zugang dient der Übertragung von Konfigurationsdateien oder der Wartung bei der Inbetriebnahme.

5.3. Bausteinsicht

Dieser Abschnitt beschreibt die Zerlegung des Sicherheitsservers in Module. Die einzelnen Module werden später als Subsysteme bezeichnet.

5.3.1. Abstraktionsebene 1

Der Sicherheitsserver zerfällt in viert Subsysteme mit weiteren internen Schnittstellen und Ablagedateien. Die Pfeile stellen die Abhängigkeit der Module untereinander dar. An den Seiten des Systems sind die externen Schnittstellen, die sogenannten Netzwerkinterfaces, mit dem zuvor in Abschnitt 5.2 beschriebenen Kontextabgrenzungen.

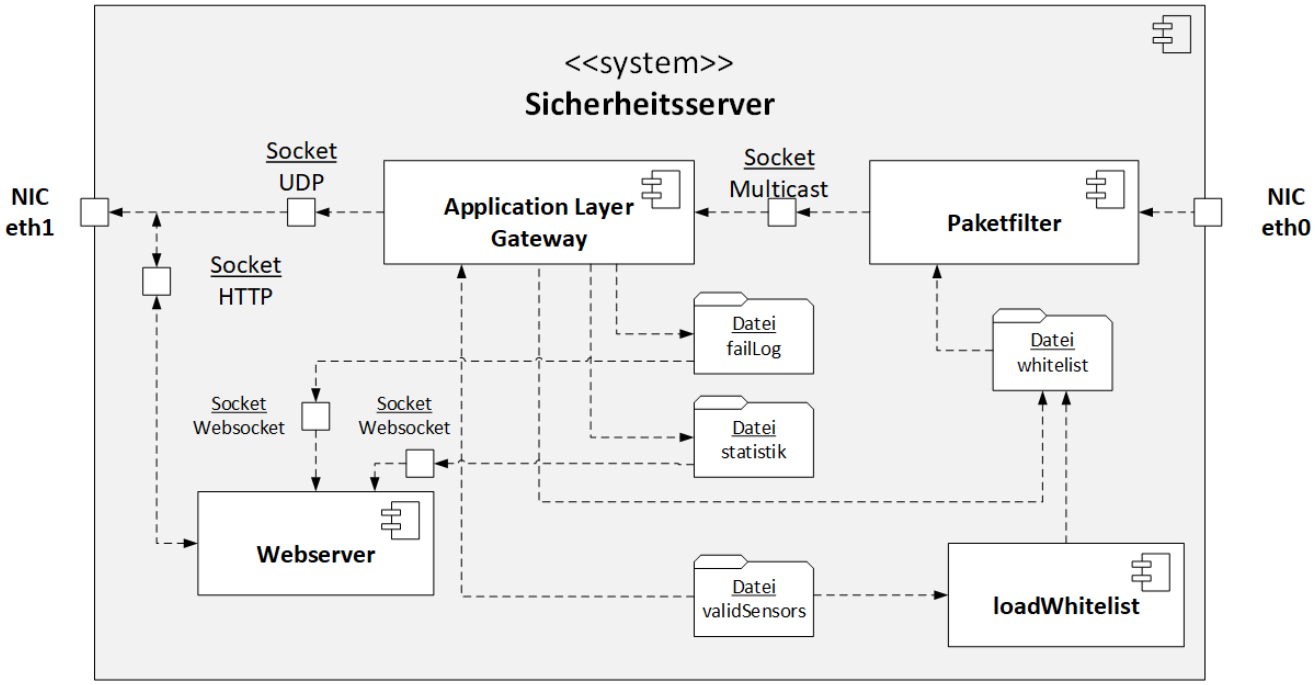


Abbildung 5.3.: Systemansicht des Sicherheitsserver - Abstraktionsebene 1

Paketfilter

Das Subsystem „Paketfilter“ symbolisiert ein Bash-Skript das eine Konfiguration der Eingangsregelkette (Input-Chain) auf dem Interface eth0, in dem „iptables“-Modul, vornimmt. Die Konfiguration entspricht den Prinzipien des „whitelistings“. Die gleichnamige Textdatei beinhaltet alle erlaubten IP-Adressen die den Paketfilter auf bestimmten Ports passieren dürfen. Beim Start des Systems wird die Datei geladen und die entsprechenden Regeln angelegt, die alle dort aufgeführten IP-Adressen akzeptieren. Die Implementierung wird im Kapitel 6 erläutert. Nach erfolgter Filterung werden die Datenpakete den zugewiesenen Sockets übergeben.

Application Layer Gateway

Der nachgeschaltete Application Layer Gateway ist ein C++ Programm, das die Daten von dem als Multicast konfigurierten Socket abruft, analysiert und die Daten auf Korrektheit und Konsistenz prüft. Auf die Analyse wird in Kapitel 6 näher eingegangen. Fehlerfreie Datensätze sendet der ALG über einen UDP-Socket an das Netzwerkinterface. Mit der Textdatei „validSensors“ wird das Modul beim Systemstart initialisiert. Die drei Textdateien „failLog, statistik und whitelist“ dienen als Logdatei, in die während der Laufzeit Datensätze abgelegt werden.

Webserver
Der Webserver dient als Visualisierungsplattform. Erreichbar ist er über den HTTP Socket. Die Visualisierung ist eine Html-Webseite mit einer Java-Skript Implementierung. Mittels Websockets werden aktuelle Datensätze auf der Webseite dynamisch dargestellt. Die Datengrundlagen werden aus den Logdateien „statistik und failLog“ geliefert.

loadWhitelist
Das Bash-Skript „loadWhitelist“ separiert beim Systemstart die IP-Adressen aus der Konfigurationsdatei „validSensors“ heraus und schreibt diese in die „whitelist“ Datei.

Dateien
Es werden vier Textdateien in diesem System benötigt.
Die **whitelist** Datei dient als Container für IP-Adressen, die für das Paketfilter-Modul benötigt werden. Geladen wird die Datei von den Modulen loadWhitelist und dem ALG.
Die **validSensors** Datei beinhaltet alle registrierten Sensoren. Jeder Sensor wird mit der IP, MAC-Adresse, Taker-ID und dem Sentence-Formatter spezifiziert.
In der **failLog** Datei werden alle Alarme während der Laufzeit abgelegt und für eine spätere Analyse aufbewahrt.
Die **statistik** Datei wird ebenfalls während der Laufzeit mit Daten zyklisch beschrieben. Der ALG schreibt in einem festen Intervall statistische Datenwerte in die Datei.

5.3.2. Abstraktionsebene 2

Das Subsystem Application Layer Gateway wird als größtes Modul für eine weitere Ebene aufgelöst. Das Subsystem lässt sich in drei weitere Module (Server-, Analyse- und Client-Prozess) unterteilen, die in C++ umgesetzt werden. Der Aufbau entspricht dem Pipline-Prinzip, bei dem ein Prozess die Daten an einen weiter verarbeitenden Prozess weitergibt. Als Synchronisation zwischen den Prozessen werden Message-Queues eingesetzt, die einen Datenspeicher besitzen und den Datenfluss strukturieren.

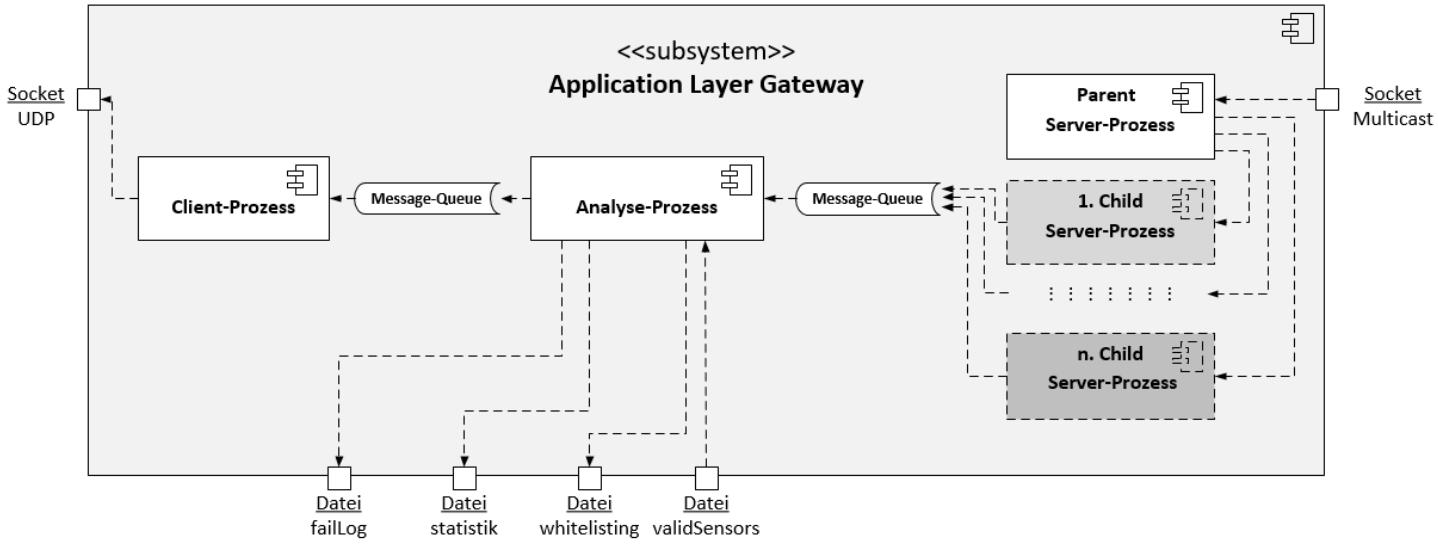


Abbildung 5.4.: Subsystemansicht des Application Layer Gateway - Abstraktionsebene 2

Server Prozess
Der Server ist als Multi-Prozess-Server strukturiert. Eingehende Datenframes lassen den Hauptprozess (Parent) einen neuen Prozess (Child) erzeugen. Der Parent-Prozess befindet sich weiter im Zustand „listen“ , bedeutet er wartet auf neue Datenframes. Der neue Child-Prozess besitzt den aktuellen Datenframe und kann parallel zum Parent-Prozess die Daten weiter verarbeiten. Nach diesem Prinzip können große Client-Anfragen bearbeitet werden ohne zu einem Datenverlust im Socket-Buffer zu führen. Die Child-Prozesse übergeben ihre Datensätze der Message-Queue und werden geschlossen.

Analyse Prozess
Im Analyse Prozess werden die Daten aus der Message-Queue gelesen und nach der Sensor-Fusion-Methode (Abschnitt 5.3.3) analysiert und bewertet. Weiter kommuniziert der Prozess mit den vier Textdateien (failLog, statistik, whitelisting, validSensors) um einen Datenaustausch mit dem Gesamtsystem herzustellen. Die fehlerfreien Datensätze übergibt der Prozess der nachfolgenden Message-Queue.

Client Prozess
Der Client Prozess ließt die Daten aus der Message-Queue und sendet die Daten an den Socket als UDP Paket.

5.3.3. Sensor Fusion

Das Konzept der Sensor Fusion beruht auf einer Analyse von kombinierten Daten, die von unterschiedlichen Quellen bereit gestellt werden. Ein geeignetes Verfahren stellt die Koppelnavigation [17], auch bekannt unter dem Namen „dead-reckoning“ dar. Dabei handelt es sich um eine Methode bei der die zurückgelegte Strecke von einem festen Startpunkt bestimmt wird. Berechnet wird ausgehend von der Startposition unter Berücksichtigung der Geschwindigkeit, dem Kurs und der Zeitdauer die zurückgelegte Strecke und die daraus resultierende Position. Mit den aus Abschnitt 2.2.1 vorgestellten Sensor-Typen werden die benötigten Daten abgedeckt um die geforderten Werte zu bestimmen. Die Positionsbestimmung auf der Erdkugel wird über die Längen- und Breitengrade (Latitude und Longitude) in Winkel-Grad angegeben. Die Verbindung zwischen zwei Punkten auf einer Kugeloberfläche ist mit einem Fehler behaftet, der sich mit der Entfernung skaliert. Mit diesem Wissen wird die Entfernungsberechnung mit dem in der Formel 5.1 dargestellte Algorithmus von Haversine [23] durchgeführt. Die Berechnungsvorschrift beschreibt die Großkreisdistanz zwischen zwei Punkten auf der Erdoberfläche. Der Algorithmus berücksichtigt dabei die Krümmung der Oberfläche und ermöglicht so die exakte Bestimmung der Distanz. Die GPS-Positionsdaten (Abschnitt 2.2.1) sind in Winkel Grad mit einer Aufteilung in Grad-Minuten angegeben und müssen für die Berechnung in Dezimalgrad und anschließend in Bogenmaß umgerechnet werden. Die Abbildung 5.5 beschreibt das Vorgehen der Methode. In dem Berechnungsintervall wird zu diskreten Zeitpunkten eine neue Position bestimmt, die am Ende des Intervalls mit dem Sensorwert verglichen wird. Die für diese Berechnung benötigten Formeln werden anschließend vorgestellt.

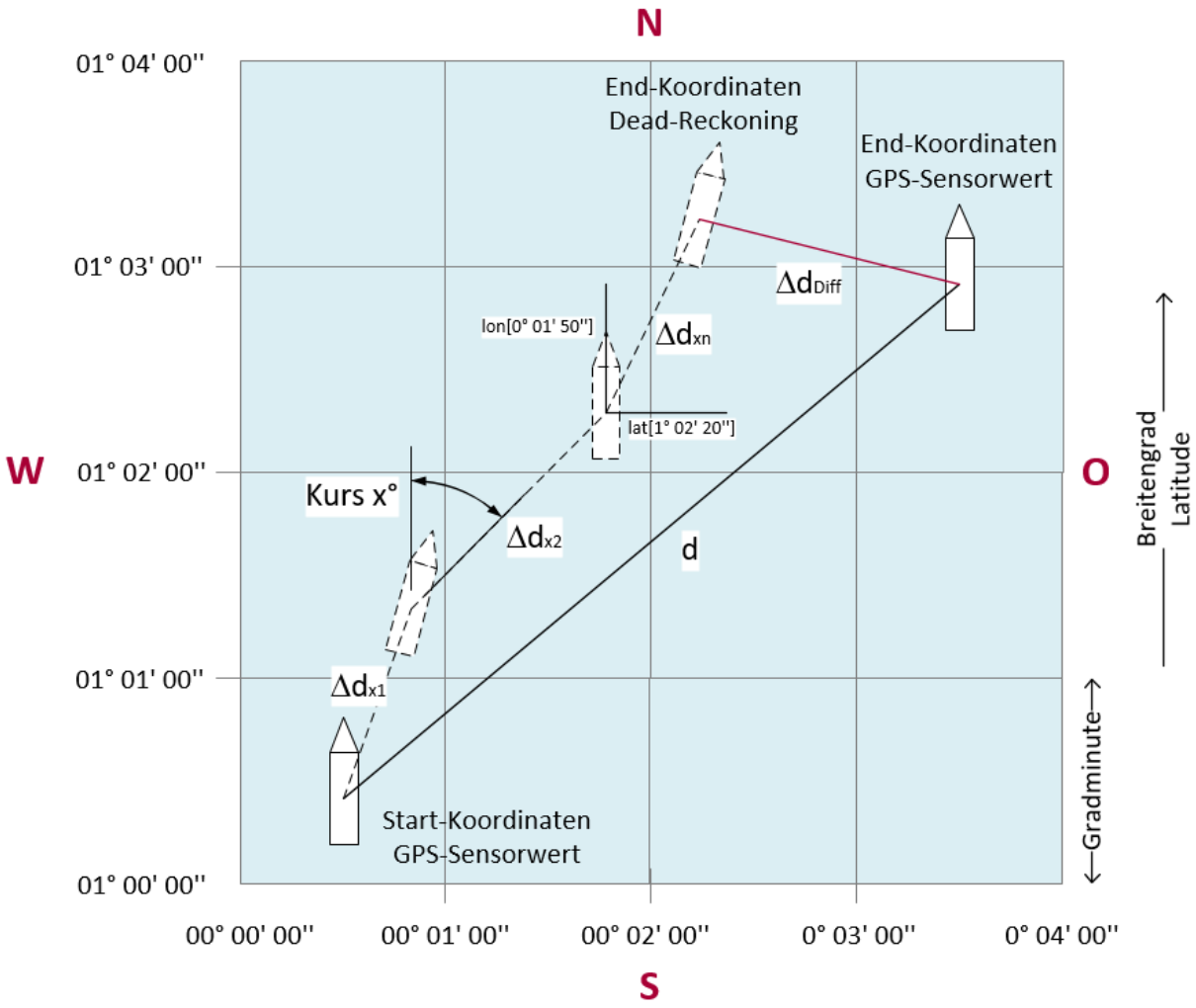


Abbildung 5.5.: Visualisierung der Sensor Fusion - dead reckoning

Die Distanz d zwischen zwei GPS-Positionen lässt sich mit der Formel 5.1 nach Haversine bestimmen. Die Formel berechnet die kürzeste Entfernung über der Erdoberfläche.

$$\begin{aligned} a &= sin^2(\Delta\varphi/2) + cos(\varphi_1) \cdot cos(\varphi_2) \cot sin(\Delta\lambda/2) \\ c &= 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)}) \\ d &= R \cdot c \end{aligned} \tag{5.1}$$

Die resultierende GPS-Position (φ_2/λ_2) von einem bekannten Startpunkt (φ_1/λ_1) mit

einer zurückgelegten Distanz d und einem festen Kurs θ, wird mit der Formel 5.2 nach Haversine berechnet.

$$\begin{aligned} \delta &= d/R \\ \varphi_2 &= \varphi_1 + \delta \cdot cos(\theta) \\ \Delta\psi &= ln(tan(\pi/4 + \varphi_2/2)/tan(\pi/4 + \varphi_1/2)) \\ q &= \Delta\varphi/\Delta\psi \\ \Delta\lambda &= \delta \cdot sin(\theta/q) \\ \lambda_2 &= \lambda_1 + \Delta\lambda \end{aligned} \tag{5.2}$$

Formelzeichen	Beschreibung	Einheit
φ	Breitengrad (Latitude)	Radiant
λ	Längengrad (Longitude)	Radiant
θ	Kurs	Radiant
R	Erdradius	m
d	Distanz	m

Tabelle 5.1.: Haversine Formelzeichen - Bedeutung und Einheit

6. Realisierung

Die Realisierung beschreibt die Umsetzung zur Erreichung der unter Kapitel 3 festgelegten Anforderungen. In den anschließenden Abschnitten wird die hierarchische Struktur der Komponenten-Ansicht, die im Konzept Abschnitt 5.3 beschrieben wurde, als Grundlage zur Beschreibung der Realisierung verwendet. Die Beschreibung beginnt bei der Projektstruktur und der Initialisierung des Gesamtsystems. Daran anknüpfend wird detaillierter in die Struktur der einzelnen Komponenten eingegangen. Um die komplexe Struktur und Kommunikationspfade der einzelnen Komponenten untereinander zu verdeutlichen dient das Laufzeitdiagramm (Abbildung C.3). In dieser Abstraktionsansicht sind die möglichen Programmverzweigungen des Datenstroms vereinfacht dargestellt.
Als Visualisierung werden geeignete UML-Diagramme ausgewählt um die Software-Lösungen zu strukturieren. Eine detaillierte Aufstellung der kompletten Projektdateien ist dem Abschnitt 6.1 zu entnehmen.

6.1. Projektstruktur

Die gesamte Projektstruktur auf dem Raspberry-Pi ist der Abbildung 6.1 zu entnehmen. Der strukturierte Programmcode inklusive Makefile ist in Abbildung C.2 dargestellt. Der komplette Programmcode ist dem Anhang D angefügt.

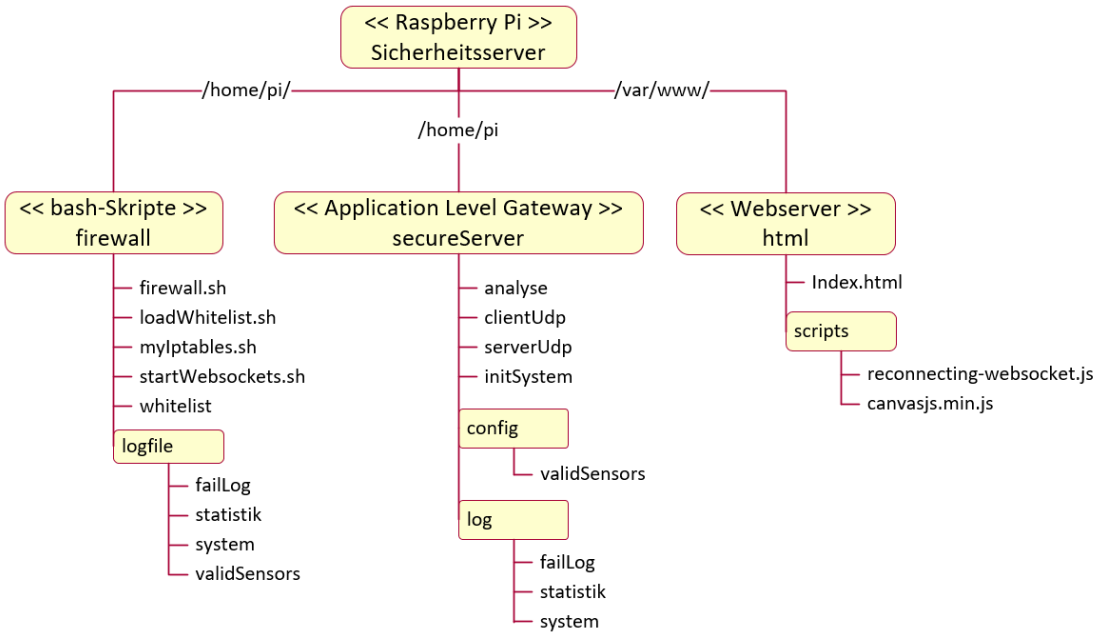


Abbildung 6.1.: Projektstruktur auf dem Raspberry-Pi

6.2. Initialisierung des Systems

Der Initialisierungsvorgang, der in Abbildung 6.2 dargestellt ist, wird nach jedem Systemneustart durchführt. Die Reihenfolge der fünf ineinander verzahnten Skripte gewährleistet einen definierten Systemstart.
Mit dem initialisieren des Betriebssystems wird das Skript „rc.local“ gestartet und führt das Bash-Skript „firewall.sh“ aus. Hierbei handelt es sich um das Initialisierungsskript des Sicherheitsservers, das in fünf Schritte unterteilt ist.

1. Direkt am Start wird das **loadWhitelist.sh** Skript ausgeführt. Dabei werden die IP-Adressen der registrierten Sensoren aus der validSensor-Datei ausgelesen und in eine separate Datei „whitelist“ geschrieben.
2. In der zweiten Instanz startet das Skript **myIptables.sh**. Hier werden die Eingangs-Regeln für das Ethernet-Interface eth0 unter Einbeziehung der zuvor separierten IP-Adressen aus der whitlist-Datei geschrieben.
3. Eine Kopie der **Log-Dateien** wird erzeugt und abgespeichert. Anschließend werden alle Dateien geleert.

4. Die **Websockets**, die als Kommunikationsschnittstellen zwischen den Log-Dateien und dem Webserver dienen, werden erstellt.

5. Der Application Level Gateway wird durch einen Prozess (**initSystem**) definiert gestartet. In der Abbildung 6.2 ist vereinfacht symbolisiert, dass der initSystem-Prozess je eine Abbildung erzeugt um den Programmcode des zu initialisierenden Prozesses zu ersetzen.

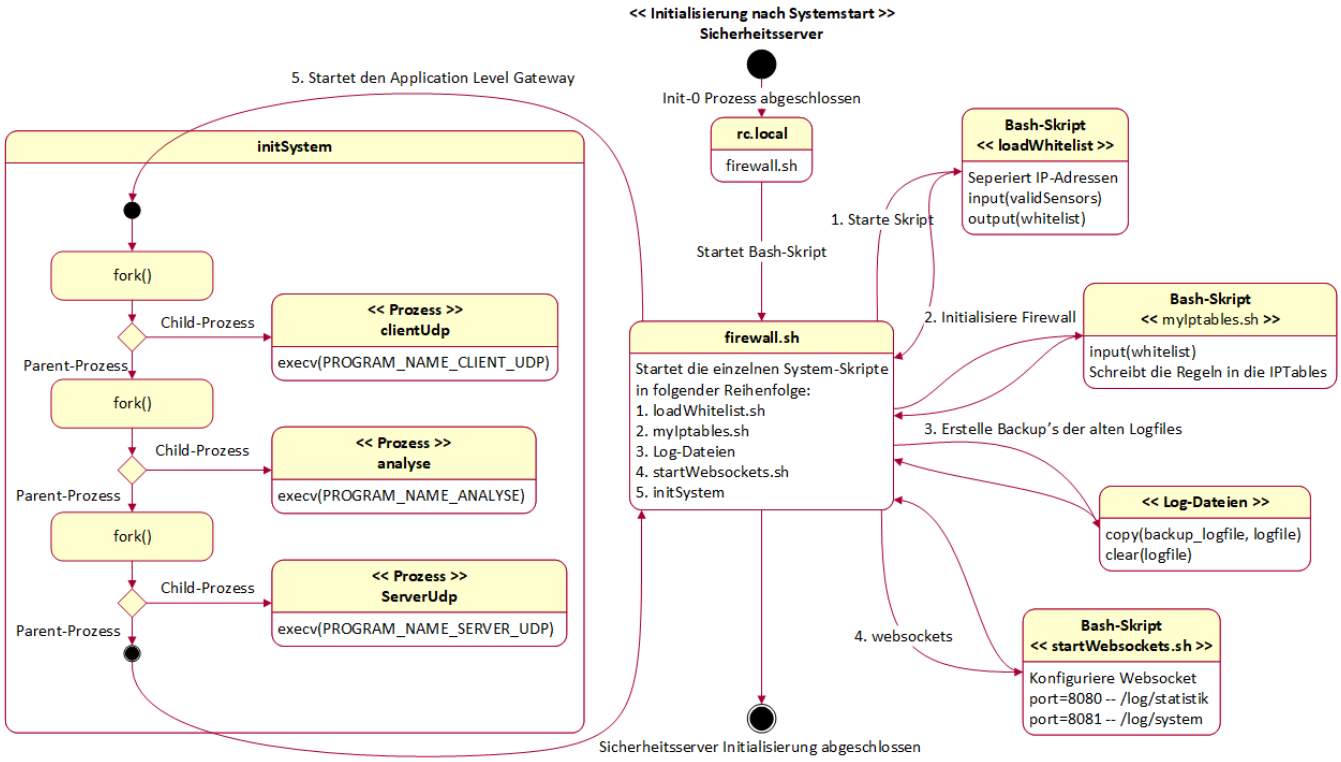


Abbildung 6.2.: Initialisierung des Gesamtsystems

6.3. Paketfilter

Der Paketfilter auf dem externen Interface eth0 (Abbildung 5.3) ist so konfiguriert, dass er jegliche Kommunikation von außerhalb des Sensornetzwerkes blockiert. Es wird nur eine Kommunikation aus dem Netzsegment 192.168.1.0/24 mit den Zieladressen und Port-Nummern aus Tabelle 2.7 erlaubt. Die Konfiguration der „iptables“ wird mit dem Bash-Skript myIptables.sh zum Systemstart vorgenommen.

6.4. Prozess-Aufbau des ALG

Die Kernkomponente der Paketfilteranalyse bildet der ALG. In den nachfolgenden Abschnitten wird jeweils die Hauptroutine der drei Kernprozesse Server-, Analyse- und Client-Prozess dokumentiert.

6.4.1. Server-Prozess

Das Aktivitätsdiagramm 6.3 zeigt den Ablauf des Server-Prozesses. Die Grundfunktionalität ist in die drei Bereiche Initialisierung, Multi-Server Implementierung und Hauptroutine unterteilt.

Initialisierung

Im Initialisierungsschritt werden die Kommunikationsschnittstellen aufgebaut. Bei der Interprozesskommunikation mit dem Analyse-Prozess verbindet sich der Server-Prozess als Client mit der Message-Queue. Die zweite Schnittstelle bildet der Netzwerk-Socket. Dieser ist als Multicast-Socket mit dem Transportprotokoll UDP parametriert.

Multi-Server Implementierung

Die nächste Grundfunktionalität bildet die Implementierung eines Multiprozess-Servers. Nachdem neue Daten in dem Socket-Buffer gespeichert wurden teilt sich der Prozess auf. Der Parent-Prozess ist als Endlosschleife programmiert und wartet auf neue Daten vom Socket. Der neu generierte „Child-Prozess“ ist eine Abbildung des Parent-Prozesses mit den aktuellen Daten vom Socket-Speicher. Der Child-Prozess trennt sich von der Socket-Verbindung und kann unabhängig von dem Parent-Prozess die Eingangsdaten weiter verarbeiten. Die damit bestehende Problematik entstehender Zombie-Prozesse ist mit dem entsprechenden Signal-Handler abgefangen. Weitere Informationen dazu befinden sich im Programm-Code im Anhang D. Die damit erreichten Funktionalitäten erfüllen die geforderten Anforderungen F8 und F9, siehe Tabelle 3.4.

Hauptroutine

Die Hauptroutine sammelt Meta-Daten der Client-Verbindung und analysiert den Syntax der eingehenden Daten, die in den Anforderungen Tabelle 3.1(G2-G3) und Tabelle 3.4(F1-F2) gefordert sind. Das Ergebnis wird mit den Meta-Daten an den Analyse-Prozess übertragen.
Im Detail beinhalten die Meta-Daten die folgenden Parameter: IP, MAC, PID und Timestamp. Die IP-Adresse der Client-Verbindung wird aus dem bereitgestellten Objekt des Sockets angerufen. Die MAC-Adresse wird aus dem ARP-Cache ausgelesen, der eine Zuordnung zwischen IP und MAC aufweist. Anschließend wird eine Syntax-Analyse der Daten auf die zwei Objekte sensorData der Klasse NmeaLwe und sensorHeader der Klasse NmeaSentence aufgeteilt. Die Klasse NmeaLwe überprüft die Daten nach dem IEC 61162-450 Format und verifiziert die dort berechnete Checksumme. Die Klasse NmeaSentence analysiert die Daten nach dem ICE 61162-1 Format und verifiziert ebenfalls die dort berechnete Checksumme. Das resultierende Ergebnis wird in einem Datenstring inklusive Meta-Daten gespeichert und dem Analyse-Prozess über die Message-Queue übertragen.

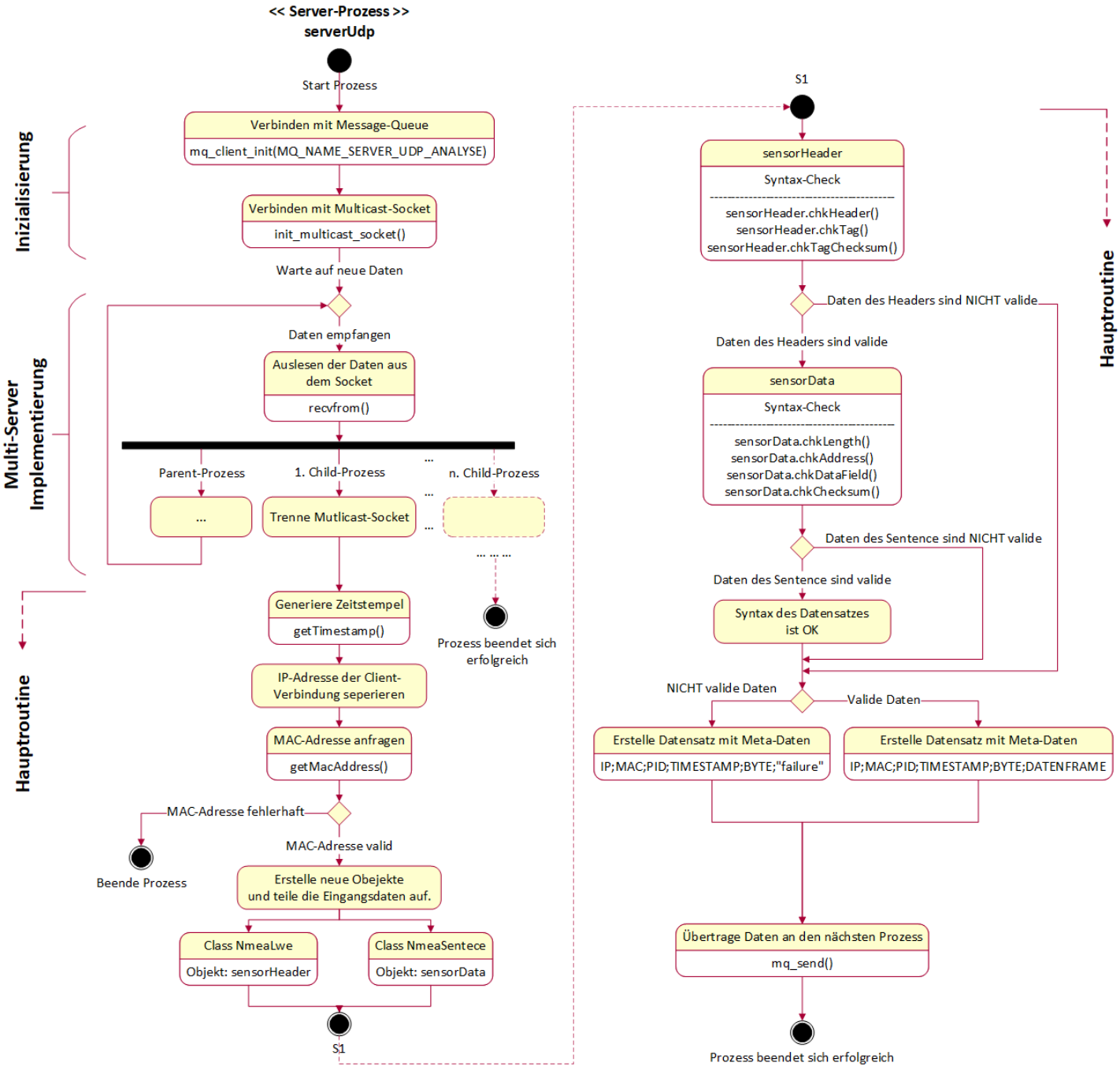


Abbildung 6.3.: Aktivitätsdiagramm der Hauptroutine - Server-Prozess

6.4.2. Analyse-Prozess

Der Analyse-Prozess analysiert die einzelnen Datenframes. Im Detail wird aus dem Datenprotokoll-Header die Information Sequenznummer und Timestamp ausgewertet, die sich aus den Anforderungen in Tabelle 3.4 (F3-F4) ergeben. Die Sensorinformationen werden für sich selbst nach ihren physikalischen Parametern bewertet und im Querverbund in der sogenannten Sensor-Fusion miteinander verglichen. Mit dieser Methode werden die restlichen Anforderungen aus Tabelle 3.1 (G4) und 3.4 (F11) erfüllt. Der dazugehörige Programmcode ist dem Anhang D zu entnehmen. Die Komplexität des Prozesses wird in den zwei Aktivitätsdiagrammen 6.4 und 6.6 gegliedert.

Initialisierung

Die Initialisierung stellt die Schnittstellen zu den Peripherie-Prozessen, dem Timer Modul und den benötigten Objekten zur Verarbeitung bereit. Das Objekt „mySensors“ ist als Vektor definiert und symbolisiert im System enthaltene Sensoren. Dazu werden die spezifizierten Sensoren aus der Konfigurationsdatei „config“ geladen und als neues Element in das Vektorobjekt gespeichert. Diese variable Implementierung der Sensorobjekte gewährleistet eine modulare Erweiterung von Sensoren auf Datenbasis der Konfigurationsdatei. Anschließend wird jedes Sensorobjekt geladen und zugeordnet.

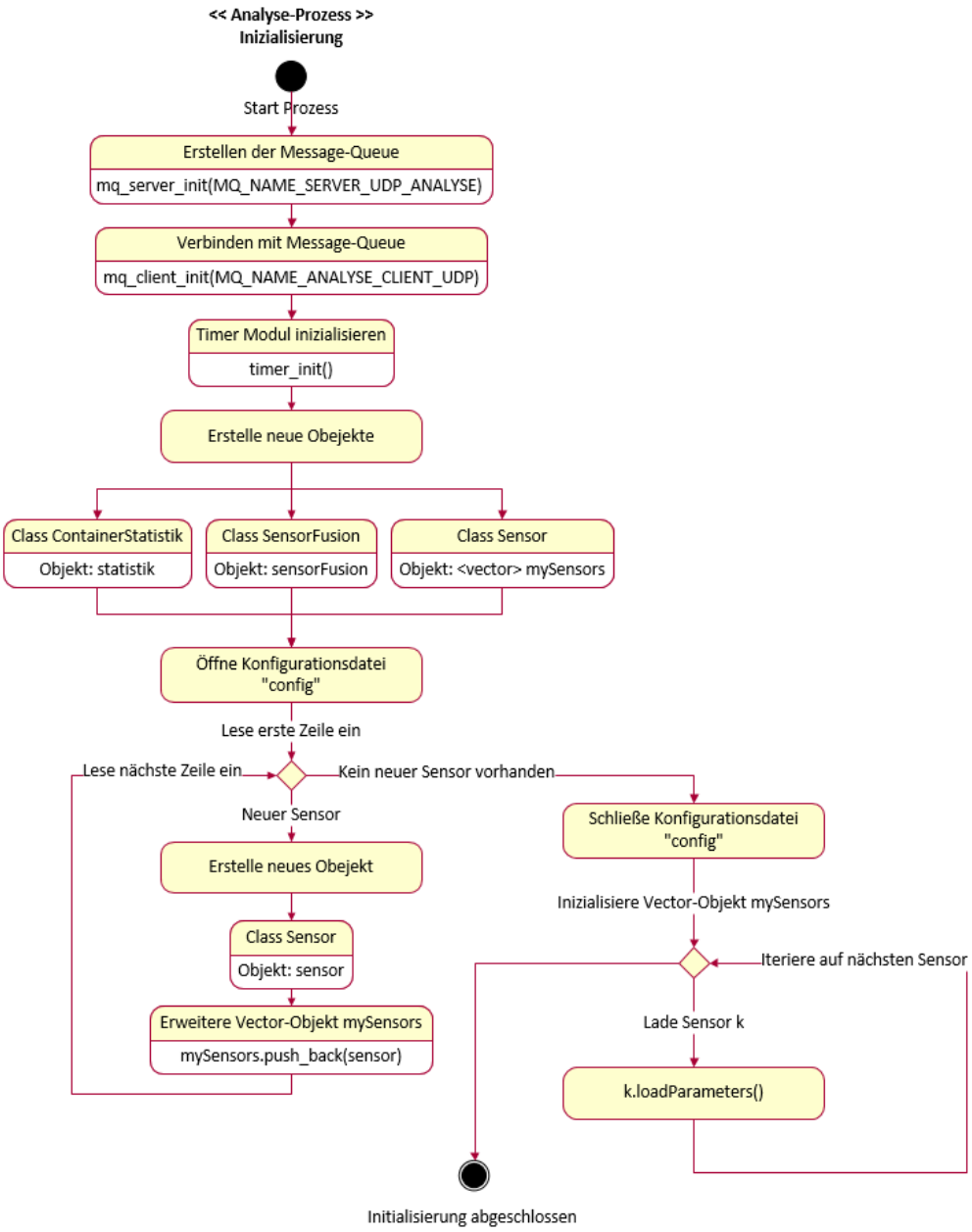


Abbildung 6.4.: Aktivitätsdiagramm der Initialisierungsphase - Analyse-Prozess

Hauptroutine

Die Hauptroutine kann abstrakt in die 6 Segmente **Datenaufbereitung, Sensorüberprüfung, Header-Informationen, Sensorwertebereich, Sensor Fusion und Archivierung** aufgeteilt werden.

In der **Datenaufbereitung** wird der eingehende Datensatz in einem Objekt „incommingSensorData“ der Klasse ContainerInputData gespeichert. Ausgehend von der Syntax-Analyse, die im Server-Prozess durchgeführt wurde, werden die fehlerfreien Daten zur Weiterverarbeitung aufbereitet. Die mit einem Fehler gekennzeichneten Daten werden in einer Log-Datei abgelegt, bewertet und verworfen.

Die anschließende **Sensor Überprüfung** iteriert über das Vektorobjekt „mySensors“ und versucht eine Übereinstimmung mit den Meta-Daten (Abbildung 6.5) zu erzielen. Bei einer Übereinstimmung wird die Referenz auf das Objekt im Vektor für einen späteren Zeitpunkt gespeichert, anderenfalls wird die Analyse abgebrochen, bewertet und die Daten verworfen.

Nachdem eine Zuordnung zwischen Sensor und den Daten vorgenommen wurde, werden die Header-Informationen im Segment **Header** ausgewertet. Die Bewertung der Header-Informationen beschränkt sich in dieser Arbeit auf die Sequenz-Nummer und den Timestamp-Parameter, beschrieben in Abschnitt 2.2.2. Mit der Sequenz-Nummer wird die Reihenfolge der eintreffenden Pakete zwischen den Objekten „incommingSensorData“ (aktueller Timestamp) und „mySensors“ (vorheriger Timestamp) analysiert. Der Timestamp-Parameter wird unter anderem zur Analyse der Zeitdifferenz zwischen den Datenframes genutzt. Ein Vergleich zwischen der Zeitdifferenz im Header-Timestamp und der Zeitdifferenz der eintreffenden Daten im Server werden ausgewertet und in den Objekten abgespeichert. Abweichungen in der Analyse aktivieren ein Warnsignal. Die Datensätze werden dennoch weiter verarbeitet.

In der nächsten Stufe **Sensorwertebereich** werden die Daten auf ihre physikalischen Größen überprüft und dem Objekt „sensorFusion“ der Klasse SensorFusion übermittelt. Datensätze, die das Kriterium nicht einhalten, werden bewertet und verworfen.

Der **Sensor Fusion**-Block führt eine Berechnung der drei Sensorwerte, siehe Anforderungstabelle 3.1(G4), über einen definierten Zeitbereich durch und vergleicht das berechnete Ergebnis mit einem Messwert. Eine detailliertere Beschreibung ist dem Abschnitt 6.4.3 zu entnehmen.

In dem letzten Segment, der **Archivierung**, wird der valide Datensatz der „Message-Queue“ übergeben. In einem periodischen Zeitintervall von 60 Sekunden wird vom Timer-Modul ein Signal-Handler aufgerufen, der die Archivierung des „statistik“ Objektes durchführt. Die aktuellen Werte des Objektes werden in die Log-Datei „statistik“ geschrieben.

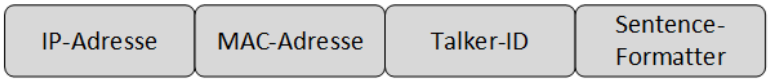


Abbildung 6.5.: Sensor Meta-Daten

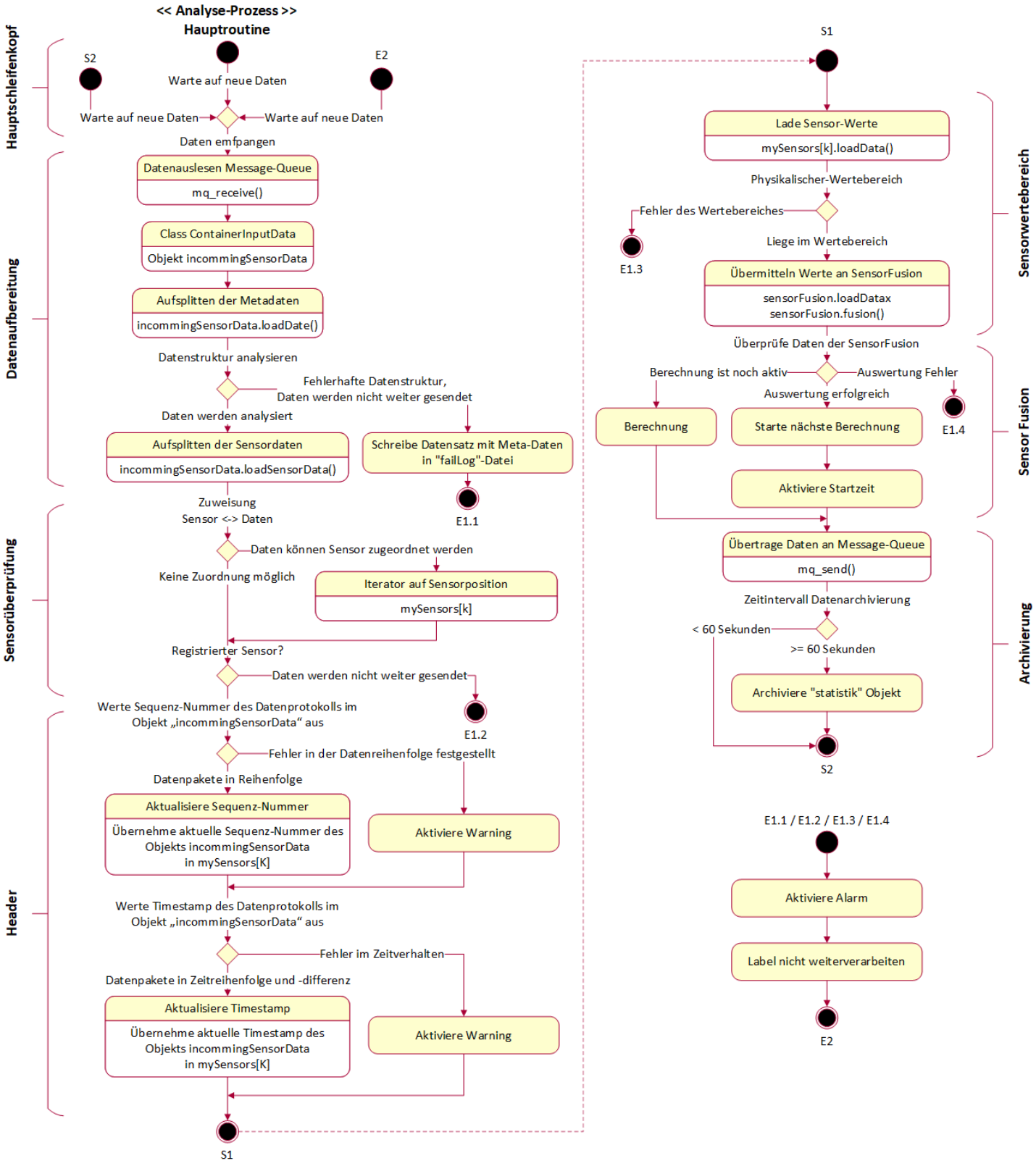


Abbildung 6.6.: Aktivitätsdiagramm der Hauptroutine - Analyse-Prozess

6.4.3. Sensor Fusion

Die Realisierung der Konzeptbeschreibung (siehe Abschnitt 5.3.3) wird in dem Aktivitätsgraphen Abbildung 6.7 schematisch dargestellt. Die Analyse wiederholt sich periodisch in einem definierten Zeitintervall. Nach jedem Zyklus wird die Berechnung durch neue GPS Positionskoordinaten aktiviert. Die Startbedingungen werden als Ausgangssituation gespeichert. Innerhalb des Zeitintervalls wird mit Änderung der Geschwindigkeit eine neue Referenzposition aus der zurückgelegten Distanz und dem aktuellen Kurs bestimmt. Die Auswertung startet mit der Überschreitung des Zeitintervalls. Hierbei wird die Differenz des IST- mit dem SOLL-Wert verglichen und bewertet.

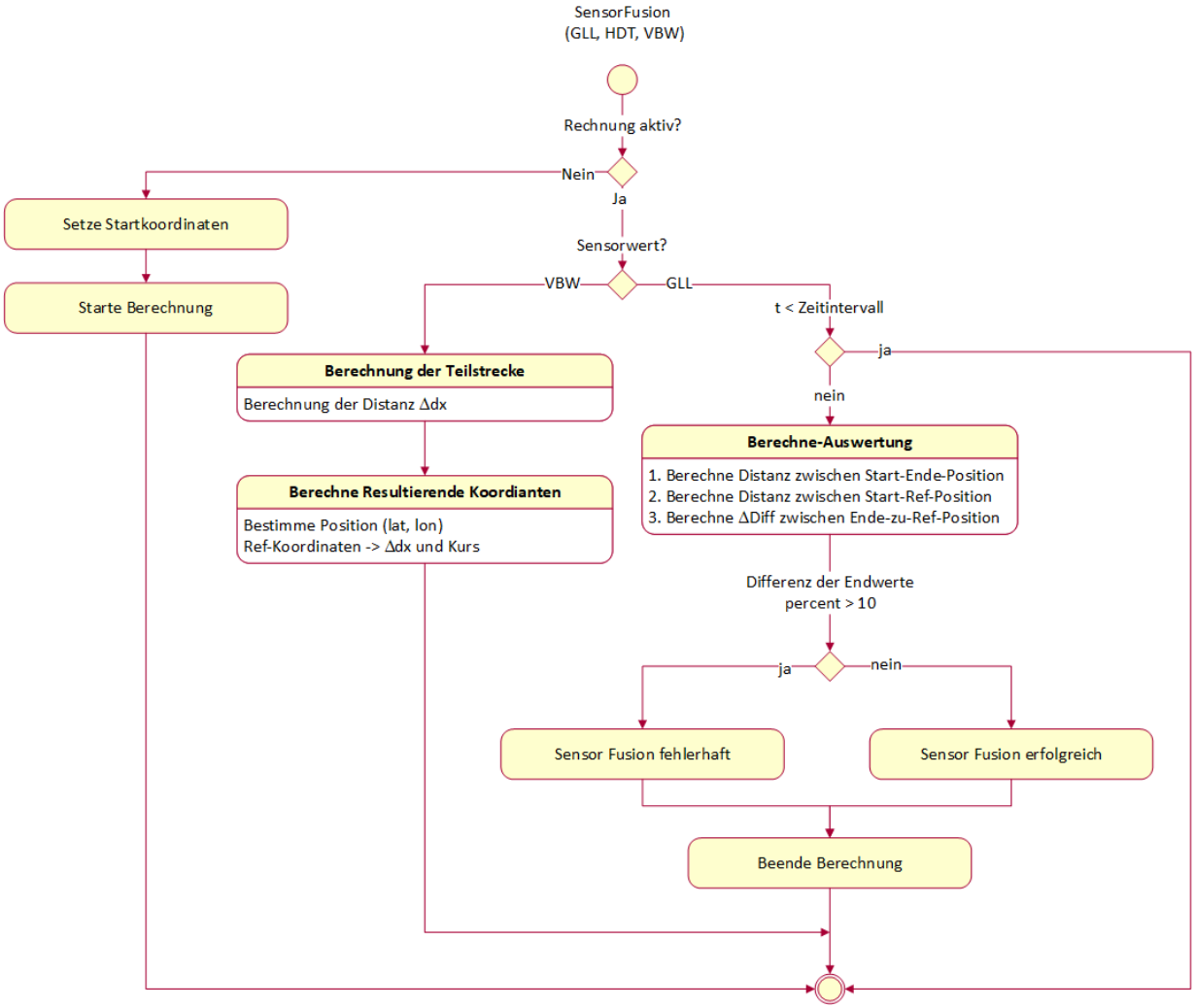


Abbildung 6.7.: Sensor Fusion - Aktivitätsdiagramm

Der Abbildung 6.8 kann beispielhaft die Berechnungsvorschrift der Sensor Fusion entnommen werden.

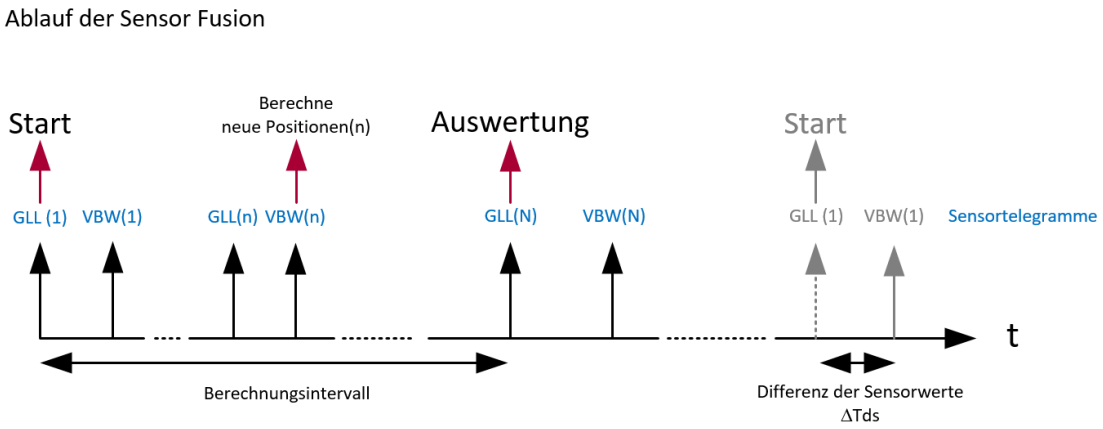


Abbildung 6.8.: Zeitlicher Verlauf der Sensor Fusion

6.4.4. Client-Prozess

Die letzte Komponente des ALG bildet der Client-Prozess. Die Hauptaufgabe besteht darin den Datenfluss vom Analyse-Prozess in das Schiffsdatennetz zu übertragen. Die Schnittstelle Message-Queue fungiert als Server und nimmt die Daten vom Client, dem Analyse-Prozess, entgegen. In der Weiterverarbeitung wird das Datenpaket an einen Socket, der als UDP-Client konfiguriert wurde, übermittelt.

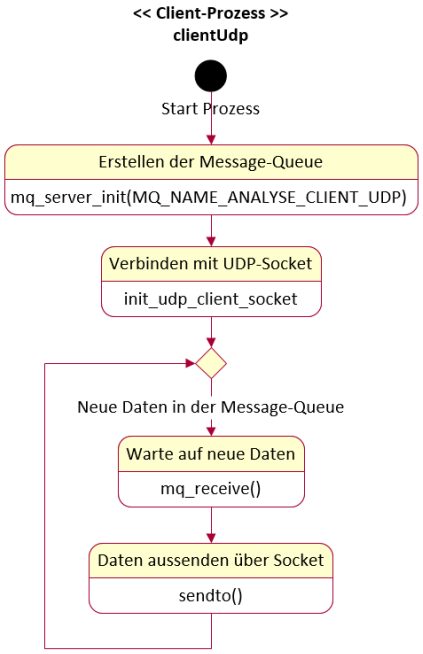


Abbildung 6.9.: Aktivitätsdiagramm der Hauptroutine - Client-Prozess

6.5. Webserver

Der Webserver visualisiert den Datentransfer durch den Sicherheitsserver und gibt Informationen über abweichende Kommunikationsversuche und erfüllt damit die geforderte Anforderung gemäß Tabelle 3.1(G5). Eine dynamische Darstellung der Daten auf dem Webbrowser wird über Websockets realisiert (siehe Abbildung 5.3). Neue Daten vom Application Level Gateway werden so zum Webbrowser des Client übertragen. Die Datengrundlage aus denen die drei Diagramme geniert werden ist die statistik-Log-Datei. Die Homepage ist mit HTML und Java-Skript umgesetzt. Für die optische Darstellung der Daten wurde eine bereits vorhandene Vorlage [7] verwendet.

Die Website (siehe Abbildung 6.10) ist in vier Bereiche aufgeteilt. Der **Kopfbereich** informiert über den Status des Gesamtsystems. Informationen über aktive Warnungen und Alarme sind ebenso aufgeführt wie die Systemzeit.
Das **linke Balkendiagramm** gibt Auskunft über die Datenverteilung der jeweiligen registrierten Sensoren durch den Server.
Mit dem **rechten Balkendiagramm** werden alle fehlerhaften Kommunikationsversuche

sowie detektierten Abweichungen im Application Level Gateway zusammengefasst dargestellt.
Das **Linien Diagramm** im unteren Bereich stellt die Datenübertragung in Byte über den Zeitraum von zwanzig Minuten, für die im System vorhandenen Sensoren, dar.

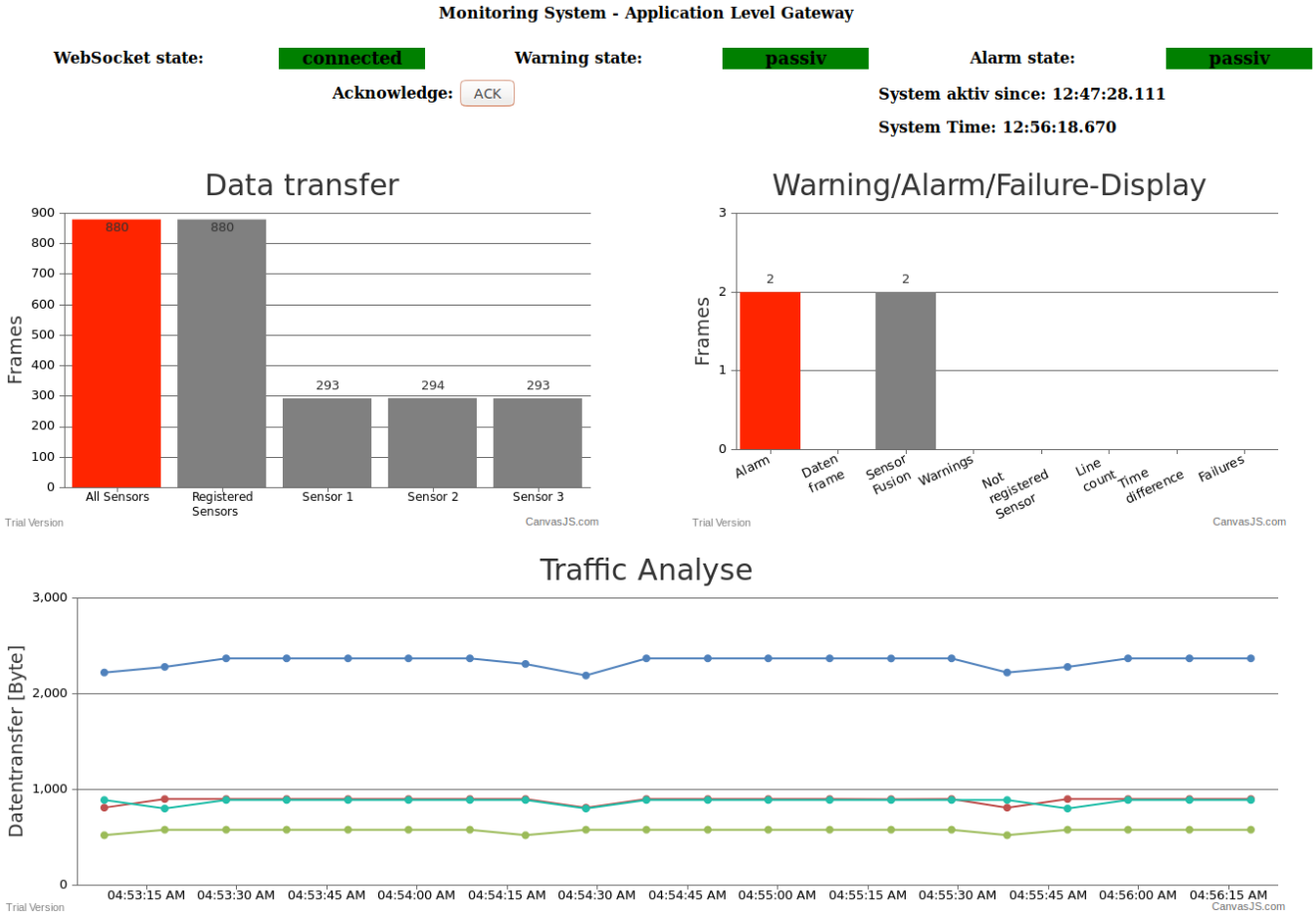


Abbildung 6.10.: Monitoring-System - Sicherheitsserver

7. Verifizierung

Mit der Verifizierung werden die in Kapitel 3 aufgelisteten Anforderungen anhand geeigneter Test Cases überprüft und die dabei auftretenden Abweichungen dokumentiert.

7.1. Test Cases

Die Test Cases leiten sich von den in Abschnitt 3.2.1 aufgestellten Szenarien ab und überprüfen die Realisierung aller Anforderungen in den Tabellen 3.1 und 3.4. Die komplexeren Test Cases werden nachfolgend genauer analysiert. Die gesamten Ergebnisse der Test Cases sind in der Tabelle 7.1 zusammengefasst dargestellt. Die Simulationsdaten und Ergebnisse sind der beigelegten DVD angefügt und können bei Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider eingesehen werden.

Test Case - Schiffssimulation

Mit der Simulationssoftware NMEA-Studio wurden Sensordaten erzeugt, die ein sich bewegendes Schiff simulieren. In diesem Testfall wurden drei Varianten untersucht mit der die Sensor Fusion validiert werden soll. Variante 1 überprüft die korrekte Berechnung der Breitengrade. In Variante 2 werden die Längengrade analysiert und in der Variante 3 werden alle Berechnungen im Algorithmus bewertet.

1 Schiffsobjekt fährt einen Kurs von $0°$-Nord mit einer Geschwindigkeit von 9,7 Knoten

2 Schiffsobjekt fährt einen Kurs von $90°$-Ost mit einer Geschwindigkeit von 9,7 Knoten

3 Schiffsobjekt fährt einen Kurs von $45°$-Nordost mit einer Geschwindigkeit von 9,7 Knoten

Mit den drei Varianten konnte die Implementierung des Algorithmus verifiziert werden. Die Abbildung C.4 zeigt eine Abweichung zwischen dem berechneten und dem tatsächlichen Wert. Die Abweichung lässt sich auf die Differenz zwischen den Sensorwerten zurückführen, die jeweils einmal pro Sekunde übertragen werden (siehe Abbildung 6.8). Da die Berechnungsgrundlage der Zeitstempel im Datenframe ist, kann sich in diesem Testfall eine maximale Abweichung von 4,99 m ergeben.

$$\Delta d_{Diff,max} = 9,7Kn \cdot 1,852km/3600s = 4,99m/s$$

Der untere Subplot zeigt eine Mittlere Abweichung von 3,248 Metern mit einer Schwankung von $\pm 5cm$. Die leichte Abweichung der Ergebnisse zueinander lässt sich über eine Zeitschwankung in den Datentelegrammen je Messreihe erklären.

Die damit erreichte Genauigkeit ist für eine Bewertung auf Plausibilität ausreichend.

Test Case - Zuverlässiger Systemstart

Das System wurde in verschiedenen Variationen (Systemstart, Neustart, Power-Down) neu gestartet. Es konnte gezeigt werden, dass das System stabil startet und die Initialisierung erfolgreich war.

Test Case - Erweiterbarkeit des Systems

Mit der Konfigurationsdatei „config“ und dem variablen Vektorobjekt im Analyse-Prozess ist eine Erweiterung des Systems einfach umsetzbar. Die bekannten Sensortypen sind in Header-Dateien vollständig hinterlegt. Als Test wurde die Konfigurationsdatei mit neuen Sensoren erweitert und von einem zusätzlichen Gerät Daten an den Gateway übermittelt. Die Daten wurden erfolgreich auf Plausibilität geprüft und an das Zielsystem weitergeleitet. Lediglich die Methodik der Sensor-Fusion müsste auf weitere Sensorelemente angepasst und erweitert werden.

Test Case - Datenframe-Analyse

Die Datenframe-Analyse wurde in Unit-Tests durchgeführt. Dazu wurden NMEA-Datenframes in unterschiedlichsten Konstellationen generiert und auf Syntax überprüft. Die folgenden Kriterien wurden dabei in Unit-Tests gemäß Abschnitt 2.2.1 und 2.2.2 validiert.

- Gültige ASCII-Zeichen gemäß Tabelle C.1
- Länge des Datenframes
- Nach IEC 61162-1: Gültige Talker-ID, Sentence-Formatter
- Nach IEC 61162-450: Gültige TAG-Blocks
- Richtige Checksumme

Test Case - Datenprotokoll-Header-Analyse

Aus dem Datenprotokoll-Header wurden die zwei TAG-Blocks Timestamp und Line-Count in die Analyse mit einbezogen. Der Line-Count Parameter dient als Sequenz-Nummer. Als Test wurden Datenpakete mit konstanter oder variierender Sequenz-Nummer übertragen. Die analysierten Ergebnisse zeigten, dass alle Abweichungen detektiert und gespeichert wurden.
Der Timestamp-Parameter wurde nach der gleichen Prozedur wie die Sequenz-Nummer analysiert. Jedoch konnte bei einem dynamischen Test mehrerer Sensoren eine erhöhte Abweichung der Signallaufzeit einiger Datenframes detektiert werden.

$$t_{Signallaufzeit} > 200\ ms$$

Als Fehlerquelle wurde der Datenkonverter identifiziert, der bei erhöhten Datenraten vermutlich interne Verzögerungen hervorruft.

Test Case - Server Implementierung

Der Server-Prozess wurde als separate Einheit getestet. Dazu wurden generierte Datenframes von einem Client ausgesendet und auf dem Server gespeichert. So konnte gezeigt werden, dass die Server Implementierung stabil arbeitet und es zu keinem Datenverlust sowie Zombie-Prozessen führt.

Test Case - Netzwerkanalyse

In der Netzwerkanalyse wurde die Konfiguration der „iptables“ überprüft. Dazu wurde versucht eine Kommunikation zum Server von außerhalb des IP-Segmentes und von nicht registrierten Sensoren innerhalb des IP-Segmentes auf die Ziel-Ports aus Abschnitt

2.2.2 aufzubauen. Die Auswertung der Paketfilter zeigt eine vollständige Abschirmung der eintreffenden Datenpakete.

Test Case - Manipulierte Sensorwerte

Eine Manipulation der Sensorwerte die einem Angriffsszenarium, dem „Man-in-the-Middle“, entsprechen würde, konnte nicht in einen Testfall eingebunden werden. Auf Grund der Komplexität der Simulation müsste dies in einer weiteren Bearbeitung angesetzt werden.

7.2. Review

Mit dem Review wird die Umsetzung der Anforderungen anhand durchgeführten Test-Cases aus Abschniit 7.1 beurteilt und Abweichungen festgehalten. Die Tabelle 7.1 gibt eine Übersicht der erreichten Anforderungen.

Test Cases	ID	Anforderungen	Ergebnisse
Schiffssimulation	G4	Plausible Sensordaten	✓
	F11	Sensor Fusion	✓
Zuverlässiger Systemstart	F10	Initialisierung	✓
Erweiterbarkeit des Systems	G1	Wartungs- und Erweiterbarkeit	✓
	G5	Visualisierung	✓
Datenframe-Analyse	G2	Syntax nach IEC 61162-1	✓
	G3	Syntax nach IEC 61162-450	✓
	F1	Checksumme verifizieren	✓
	F2	Checksumme verifizieren	✓
Datenprotokoll-Header-Analyse	F3	Sequenz-Nummer	✓
	F4	Unix-Timestamp	✗
Server Implementierung	F8	Multiprozess-Server	✓
	F9	Zombie-Prozesse	✓
Netzwerkanalyse	F5	Registrierte Sensoren	✓
	F6	Kommunikationskanäle	✓
	F7	Netzzugriff	✓
Manipulierte Sensorwerte	F12	Man-in-the-Middle	✗

Tabelle 7.1.: Übersicht der implementierten und nicht umgesetzten Anforderungen

Unix-Timestamp

Der Zeitstempel im Datenformat wird richtig ermittelt. Der Datenkonverter der Firma Veinland GmbH liefert nicht konsistente Zeitstempel. Es kam in dem Test zu Verzögerungen von mehr als 200 ms zwischen den Datenpaketen und die Auswertung konnte nicht zuverlässig getestet werden. Als Verifizierung dieser Abweichung müsste dieser Test mit alternativen Quellen wiederholt werden. Im Rahmen dieser Arbeit konnte diese Abweichung nicht bestätigt werden.

Man-in-the-Middle

Der Test-Case für diese Anforderung konnte in dieser Arbeit in dem Zeitrahmen nicht weiter analysiert werden. Die Sensor Fusion deckt einen geringen Teil der geforderten Funktionalität ab, erreicht aber nicht den benötigten Umfang.

8. Fazit

8.1. Zusammenfassung

Ziel der Arbeit war der Entwurf und die Entwicklung eines Sicherheitsservers für Navigationsdaten. Dessen Aufgabe ist, ein spezielles Sensorformat, das seine Daten mittels Multicast-Nachrichten überträgt, zu analysieren und zu bewerten. Zum erreichen dieser Ziele wurden zunächst durch Literaturrecherche die benötigten Grundlagen erarbeitet. Insbesondere der strukturelle Aufbau einer Firewall und die Kommunikationstechnik der Sensorik wurde aus geeigneten Quellen recherchiert.
Auf Basis dieser theoretischen Grundlagen wurden die Anforderungen aus einer Fehlermöglichkeits- und Einflussanalyse abgeleitet. Die Grundlage bilden mögliche Fehler-Szenarien, die zuvor definiert wurden.
Um die Anforderungen erfüllen zu können, sind Systementscheidungen unter Berücksichtigung der Randbedingungen getroffen worden.
Anschließend wurde sich mit der Systemarchitektur beschäftigt und die globale Struktur der Softwarelösung beschrieben. Insbesondere wird die Verknüpfung der unterschiedlichen Komponenten, der Paketfilter und der Application Level Gateway, beschrieben. Durch den Einsatz eines Multi-Prozessservers wird eine Entkoppelung der Datenverarbeitung von der Datenannahme im Socket erreicht. Zudem wird eine weitere Analysemethode zur Detektierung fehlerhaften Daten einzelner Sensoren, die Sensor-Fusion, eingefügt. In dieser Analyse werden Sensordaten unterschiedlicher Quellen miteinander verrechnet um aus den Ergebnissen eine Abschätzung über deren Plausibilität im Gesamtsystem zu erhalten.
Auf Grundlage der System-Konzeptionierung wurde die Realisierung der Software durchgeführt und mit geeigneten UML-Diagrammen dokumentiert. Die Kernkomponente bildet dabei der Application Level Gateway, in dem die komplexeren Funktionen der Sensordatenüberwachung umgesetzt wurden. Um die Komplexität der Komponente und eine Entkopplung der Ressourcen zu erhalten, wurden drei Prozesse in einem nachrichtenbasierten Modell implementiert.
Abschließend wurde die Verifizierung des Softwarekonzepts durchgeführt, um die Einhaltung der spezifizierten Anforderungen zu überprüfen. Im Fokus stand dabei die Über-

prüfung der Sensor-Fusion und die erfolgreiche Syntax-Analyse des Sensordatenframes. Die damit erreichte Funktionalität schließt die wesentlichen Test-Cases mit ein. Die Ergebnisse der Test-Cases konnten mit Ausnahme von zwei Abweichungen erfolgreich abgedeckt werden. Durch eine Abweichung im Testaufbau konnte der Timestamp-Parameter im Sensor-Header nicht vollständig überprüft werden und die Manipulation von Sensorwerten konnte in dieser Arbeit nicht weiter untersucht werden.

8.2. Kritische Betrachtung

Die in Abschnitt 1.2 festgelegte Zielsetzung konnte mit der hier erarbeiteten Lösung eines Sicherheitsservers weitgehend abgedeckt werden.
Als kritische Implementierung erwies sich der Server-Prozess, der stabil und ohne auftreten von Zombie-Prozessen implementiert werden musste. Dabei wurde durch eine parallele Prozessstruktur die Datenverarbeitung, von dem kontinuierlichen einlesen von Daten, entkoppelt.
Des Weiteren ist die modulare Registrierung zusätzlicher Sensoren im ALG mittels Konfigurationsdateien so umgesetzt, dass vom System ein Vektorobjekt erzeugt wurde, welches alle Sensoren beinhaltet und diese verwaltet. Zusätzlich werden die dazugehörigen IP-Adressen dem Paketfilter übermittelt, der die Regelketten zulässiger Kommunikationspartner nach einem Systemstart ergänzt.
Mit der Sensor-Fusion wird die Plausibilität von Daten unterschiedlicher Quellen überprüft. Ein entscheidender Faktor ist die Zeitbasis, die der Berechnung zu Grunde liegt. In einem Netzwerk können Verzögerungen in der Übertragung zu einem abweichenden Zeitpunkt im Zielsystem führen. Aus diesem Grund wurde der Zeitstempel im Daten-Header als Berechnungsgrundlage bestimmt. Die dadurch auftretenden Abweichungen sind dokumentiert. Durch eine weitere Optimierung in der Implementierung würde sich eine systematische Abweichung reduzieren lassen.

8.3. Ausblick

Im Rahmen einer zukünftigen Arbeit könnte der Sicherheitsserver um die folgenden Funktionen erweitert werden:

Die Analysemethode der Sensor-Fusion kann auf weitere Sensoren und Szenarien erweitert werden um eine möglichst gute Abdeckung der verwendeten Sensorik zu erhalten. Des Weiteren könnte der Einsatz des Kalman-Filters analysiert werden um

beispielsweise die Genauigkeit von GPS Signalen zu erhöhen.

Mit den in dieser Arbeit vorgestellten Methoden ist eine 100% sichere Datenübertragung nicht gewährleistet. Theoretisch wäre es denkbar, dass sämtliche Daten manipuliert werden und somit die Datenplausibilität dennoch gewährleistet ist. Eine sichere Kontrolle der Authentizität kann nur über kryptografische Verfahren sichergestellt werden. Dabei könnte die Implementierung eines Public-Privat-Key-Verfahrens untersucht werden, mit dem es möglich ist, die Authentizität über die digitale Signatur sicher zu stellen und die Daten mit einer angemessenen Bitlänge zu verschlüsseln.

Der Sicherheitsserver ist noch in keinem System aktiv integriert. Die Anbindung an ein solches Zielsystem mit einer aktiven bidirektionalen Steuerung könnte untersucht und erweitert werden. Die im System vorhandenen Alarm und Warnmeldungen können abgerufen und mit einem weiteren Prozess verknüpft werden.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] ALBANNA, Z.: *IANA Guidelines for IPv4 Multicast Address Assignments*. August 2001. – URL https://tools.ietf.org/html/rfc3171. – Eingesehen am 13.08.2018, 18:00 Uhr

[2] AYUSO, Pablo N.: *Netfilter-Iptables Project*. – URL https://www.netfilter.org/. – Eingesehen am 06.10.2018, 11:00Uhr

[3] BAUN, Christian: *Betriebssysteme kompakt* -. Berlin Heidelberg New York : Springer-Verlag, 2017. – ISBN 978-3-662-53143-3

[4] BENGEL, Günther ; BAUN, Christian ; KUNZE, Marcel ; STUCKY, Karl-Uwe: *Masterkurs Parallele und Verteilte Systeme - Grundlagen und Programmierung von Multicore-Prozessoren, Multiprozessoren, Cluster, Grid und Cloud*. Berlin Heidelberg New York : Springer-Verlag, 2015. – ISBN 978-3-834-82151-5

[5] COTTON, M.: *Special-Purpose IP Address Registries*. April 2013. – URL https://tools.ietf.org/html/rfc6890. – Eingesehen am 13.08.2018, 11:00 Uhr

[6] EHSES, Erich ; KÖHLER, Lutz ; RIEMER, Petra ; STENZEL, Horst ; VICTOR, Frank: *Systemprogrammierung in UNIX / Linux - Grundlegende Betriebssystemkonzepte und praxisorientierte Anwendungen*. Berlin Heidelberg New York : Springer-Verlag, 2011. – ISBN 978-3-834-88277-6

[7] FENOPIX, INC.: *JavaScript Charts*. – URL https://canvasjs.com/. – Eingesehen am 29.09.2018, 18:00 Uhr

[8] FOUNDATION, BeagleBoard.org: *BeagleBoard X15*. – URL http://beagleboard.org. – Eingesehen am 10.09.2018, 10:00 Uhr

[9] FOUNDATION, Raspberry P.: *Raspberr Pi 3 B+*. – URL https://www.raspberrypi.org. – Eingesehen am 18.09.2018, 12:00 Uhr

[10] GMBH, VEINLAND: *8NMEAto8-E - MagicPlex 8 GIC LWE - NMEA Matrix 8-fach/Ethernet*. – URL https://www.veinland.net/maritim/hardwaresolution/nmeakonverter/8nmeato8-e-magicplex-8-gic-lwe.html. – Eingesehen am 21.09.2018, 19:30 Uhr

[11] GMBH ads-tec: *Instruction manual IT Infrastructure IRF2000 series - standard*. – URL https://www.ads-tec.de/. – Eingesehen am 10.09.2018, 15:00 Uhr

[12] GOWARD, DANA: *Mass GPS Spoofing Attack in Black Sea?* Juli 2017. – URL https://www.maritime-executive.com/editorials/mass-gps-spoofing-attack-in-black-sea. – Eingesehen am 07.10.2018, 19:45Uhr

[13] IEC: Maritime navigation and radiocommunication equipment and systems – Digital interfaces – Part 450: Multiple talkers and multiple listeners – Ethernet interconnection / International Electrotechnical Commission. Geneva, Switzerland, 2017 (IEC 61162-450 Entwurf). – IEC

[14] ITWISSEN.INFO: *Konsistenz von Daten*. – URL https://www.itwissen.info/Konsistenz-consistance.html. – Eingesehen am 06.10.2018, 12:00Uhr

[15] ITWISSEN.INFO: *Plausibilität von Daten*. – URL https://www.itwissen.info/Plausibilitaet-plausibility.html. – Eingesehen am 06.10.2018, 12:00Uhr

[16] KAPPES, Martin: *Netzwerk- und Datensicherheit - Eine praktische Einführung*. Berlin Heidelberg New York : Springer-Verlag, 2013. – ISBN 978-3-834-88612-5

[17] KLUSSMANN, Niels ; MALIK, Arnim: *Lexikon der Luftfahrt* -. Berlin Heidelberg New York : Springer-Verlag, 2011. – ISBN 978-3-642-22500-0

[18] MANDL, Peter: *TCP und UDP Internals - Protokolle und Programmierung*. Berlin Heidelberg New York : Springer-Verlag, 2018. – ISBN 978-3-658-20149-4

[19] NICOLAI, Birger: *Hacker nehmen Frachtschiffe ins Visier*. Februar 2017. – URL https://www.welt.de/print/die_welt/wirtschaft/article161832758/Hacker-nehmen-Frachtschiffe-ins-Visier.html. – Eingesehen am 07.10.2018, 19:45Uhr

[20] NMEA: *NMEA 0183 Standard*. 2018. – URL https://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp. – Eingesehen am 03.10.2018, 18:00Uhr

[21] RAUBER, Thomas ; RÜNGER, Gudula: *Parallele Programmierung* -. Berlin Heidelberg New York : Springer-Verlag, 2012. – ISBN 978-3-642-13604-7

[22] ROHWEDDER, Lars H.: *IPv4 Multicast to Mac Address German*. September 2007. – URL https://de.wikipedia.org/wiki/Datei:IPv4_Multicast_to_Mac_Address_German.svg. – Eingesehen am 13.08.2018, 17:00 Uhr

[23] R.W. SINNOTT: *Virtues of the haversine*. Sky and Telescope, August 1984

[24] SAILSOFT.NL: *Nmea Studio*. – URL http://www.sailsoft.nl/index.html. – Eingesehen am 09.01.2018, 12:00 Uhr

[25] SAMES, Dr. Pierre C.: *Unmanned ships on the horizon*. – URL https://www.dnvgl.com/article/unmanned-ships-on-the-horizon-94273. – Eingesehen am 07.10.2018, 19:45Uhr

[26] SCHNABEL, Patrick: *Netzwerktechnik-Fibel - Grundlagen, Übertragungssysteme, TCP/IP, Dienste, Sicherheit*. Elektronik-Kompendium, 2016. – ISBN 978-3-981-53024-7

[27] STARKE, Gernot: *arc42 - Entwicklung und Konstruktion effektiver Softwarearchitekturen*. – URL https://www.arc42.de/. – Eingesehen am 10.09.2018, 10:00 Uhr

[28] TP-LINK TECHNOLOGIES CO., LTD: *USB zu Ethernet Konverter*. – URL https://www.tp-link.com. – Eingesehen am 10.09.2018, 10:00 Uhr

[29] WERDICH, Martin: *FMEA - Einführung und Moderation - Durch systematische Entwicklung zur übersichtlichen Risikominimierung (inkl. Methoden im Umfeld)*. Berlin Heidelberg New York : Springer-Verlag, 2013. – ISBN 978-3-834-82217-8

[30] WIKIPEDIA.ORG: *IPv4 Header*. – URL https://de.wikipedia.org/wiki/IP-Paket. – Eingesehen am 10.08.2018, 13:30Uhr

[31] WIKIPEDIA.ORG: *UDP-Datagramm*. – URL https://de.wikipedia.org/wiki/User_Datagram_Protocol. – Eingesehen am 10.08.2018, 13:30Uhr

[32] WIKIPEDIA.ORG: *Ethernet Paket*. 25.02.2009. – URL https://de.wikipedia.org/wiki/Ethernet. – Eingesehen am 10.08.2018, 13:30Uhr

A. Abkürzungsverzeichnis

NMEA National Marine Electronics Association

IEC International Electrotechnical Commission

LAN Local Area Network

UDP User Datagram Protocol

OSI Open System Interconnection

ISO International Organization for Standardization

IP Internet Protocol

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

ALGs Application-Level-Gateways

FMEA Failure Mode and Effects Analysis

SPS Speicher Programmierbare Steuerung

UML Unified Modeling Language

B. Aufgabenstellung

B.1. Aufgabenstellung der Bachelorarbeit

Wärtsilä SAM Electronics GmbH

Bachelorarbeit Nils Parche

Sicherheitsserver für Navigationsinformationen in schiffsinternen Datennetzen.

Motivation

Die Firma Wärtsila SAM Electronics GmbH entwickelt Lösungen für den maritimen Markt im Bereich ANC (Automation, Navigation und Communication). In diesem System kommunizieren Sensoren mit Endgeräten unter Anderem über das NMEA-Telegramm, beschrieben im Standard IEC-61162. Diese Kommunikation ist in den meisten Anwendungen über die Seriellen-Schnittstelle implementiert. In einer neuen Fassung der IEC-61162-450 wird ein Sensornetzwerk mit Ethernet Kommunikation beschrieben, welches eine Kommunikation zwischen mehrerer Sendern und Empfängern erlaubt. Um die zukünftige Sicherheit des Netzwerkverkehrs zwischen Sensor- und ANC-Netz zu erhöhen soll ein Proxy-Server als Gateway zwischen diesen Netzwerken eingerichtet werden.

Ziele

Herr Parche erhält die Aufgabe einen Proxy-Server zur Überwachung der NMEA-Datenkommunikation zwischen zwei Netzwerksegmenten einzurichten. Anstrebenswerte Ziele werden dabei in folgenden Aspekten gesehen:

- Blockierung von nicht zugelassener Kommunikation
- Paketanalyse von NMEA-Datensätzen nach IEC-61162 / zunächst reduzierte Implementierung für bestimmte „Talker-Identifier“.
- Robuste Implementierung der Paketanalyse
- Monitoring der Netzwerkkommunikation

Aufgabenstellung

Herr Parche soll in seiner Bachelorarbeit die folgenden Arbeitspakete umsetzen:

1. **Einarbeitung und Analyse der Rahmenbedingungen und Lösungsmöglichkeiten**
    - Einarbeitung in die Zielstellung und Aufgabe der Bachelorarbeit durch Recherche und Literatur
    - Erfassen der Anforderungen und technischen Daten sowie Aufwandsabschätzung
    - Analyse der verfügbaren Komponenten
    - Recherche von alternativen zur vorgeschlagenen Hardware
    - Planung des zeitlichen Rahmens

Wärtsilä SAM Electronics GmbH
Behringstraße 120
22763 Hamburg
info.sam.de@wartsila.com
www.sam-electronics.de
www.wartsila.com
Telefon: 040-8825-0
Fax: 040-8825-4000

Sitz der Gesellschaft: Hamburg
Handelsregister
Hamburg HRB 89408
Ust-IdNr. DE813811359
Steuernummer 42/756/00690

Geschäftsführung:
Carsten Sippel, Maik Stövhase

Aufsichtsrat:
Stephan Kuhn (Vorsitzender)

BNP Paribas S.A. Niederlassung Deutschland
IBAN: DE28 3701 0600 4220 4700 19
BIC: BNPADEFFXXX **(nur Euro Zahlungen)**
Nordea Bank Finland Plc
IBAN: FI98 1660 5200 0334 60
BIC: NDEAFIHHXXX **(nur USD Zahlungen)**

Wärtsilä SAM Electronics GmbH

2. Grundlagen erarbeiten

- **Firewall**
  - Einarbeitung durch Recherche und Fachliteratur
  - Darstellung der Aufgabe und Funktionsweise
- **Proxy-Server**
  - Einarbeitung durch Recherche und Literatur
  - Darstellung der Aufgaben und Funktionsweise
- **Programmiersprachen / Toolchain**
  - Analysieren und abwägen der zu verwendenden Programmiersprache und Toolchain
- **Datenprotokoll NMEA 61162**
  - Einarbeitung durch Recherche und Literatur
  - Analyse des Aufbaus eines Datentelegramms
  - Darstellung eines Sensor-Netzwerkes nach 61162-450
  - Datenfluss im Netzwerk, Kommunikationswege

3. Fehlerszenarium und -analyse

- Auflistung möglicher Szenarien
- Bewertung nach FMEA – Methode
- Auswahl von Maßnahmen
- Auswählen der Komponenten: Firewall, Proxy-Server

4. Praktischer Laboraufbau

- Auswahl der geeigneten Hardware: Sensorik, Proxy-Server, Endknoten
- Beschreibung der Zielhardware, verwendetes Betriebssystem
- Verknüpfung der Komponenten und Integration zu einem System

5. Entwickeln der Hauptanwendung

- Strukturierung der Anwendung
- Erstellung des Programmcodes

6. Entwickeln einer Visualisierung

- Monitoring des Netzwerkverkehrs

7. Erprobung des Gesamtsystems

- Fehlerbehebung und Optimierung
- Analyse der Performance, Latenzzeiten und Systemauslastung

8. Auswertung und Bewertung

- Darstellung und Dokumentation der Lösungsvarianten
- Diskussion und Bewertung unter Berücksichtigung der Ergebnisse
- Aufbau

Wärtsilä SAM Electronics GmbH

Dokumentation

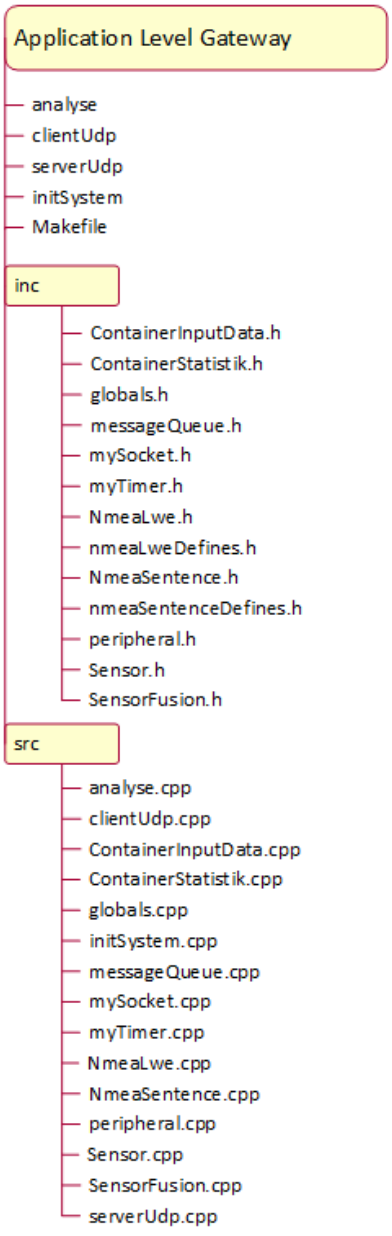
Die Fachliteratur, die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Dabei sind Grundlagen, welche für die vorgesehene Anwendung wichtig sind, näher zu betrachten.
Die Funktionsweise der Software ist gut nachvollziehbar zu dokumentieren und mit Programmablaufplänen zu ergänzen.
Die gesetzten Rahmenbedingungen, die Grundkonzeption, auftretende Probleme und wesentliche Folgerungen sollen beschrieben werden. Die Messergebnisse sind in repräsentativen Umfang zu erfassen. Sie sind auszuwerten und als Diagramme zusammenfassend darzustellen.

C. Diagramme, Tabellen und Abbildungen

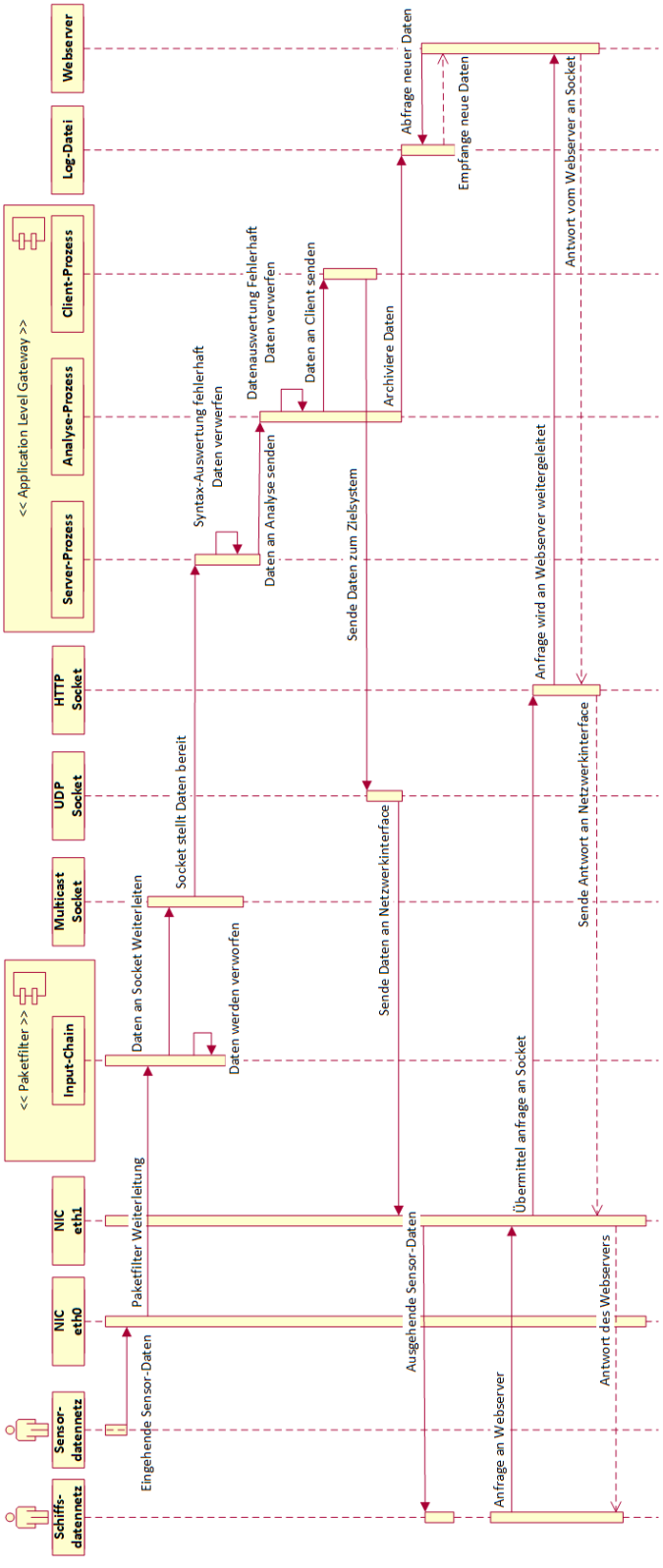
C.1. Gültige ASCII-Zeichen [13]

ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC
Space	20	32	@	40	64	`	60	96
Reserved	21	33	A	41	65	a	61	97
"	22	34	B	42	66	b	62	98
#	23	35	C	43	67	c	63	99
Reserved	24	36	D	44	68	d	64	100
%	25	37	E	45	69	e	65	101
&	26	38	F	46	70	f	66	102
'	27	39	G	47	71	g	67	103
(	28	40	H	48	72	h	68	104
)	29	41	I	49	73	i	69	105
Reserved	2A	42	J	4A	74	j	6A	106
+	2B	43	K	4B	75	k	6B	107
Reserved	2C	44	L	4C	76	l	6C	108
–	2D	45	M	4D	77	m	6D	109
.	2E	46	N	4E	78	n	6E	110
/	2F	47	O	4F	79	o	6F	111
0	30	48	P	50	80	p	70	112
1	31	49	Q	51	81	q	71	113
2	32	50	R	52	82	r	72	114
3	33	51	S	53	83	s	73	115
4	34	52	T	54	84	t	74	116
5	35	53	U	55	85	u	75	117
6	36	54	V	56	86	v	76	118
7	37	55	W	57	87	w	77	119
8	38	56	X	58	88	x	78	120
9	39	57	Y	59	89	y	79	121
:	3A	58	Z	5A	90	z	7A	122
;	3B	59	[	5B	91	{	7B	123
<	3C	60	Reserved	5C	92	|	7C	124
=	3D	61	]	5D	93	}	7D	125
>	3E	62	Reserved	5E	94	Reserved	7E	126
?	3F	63	_	5F	95	Reserved	7F	127

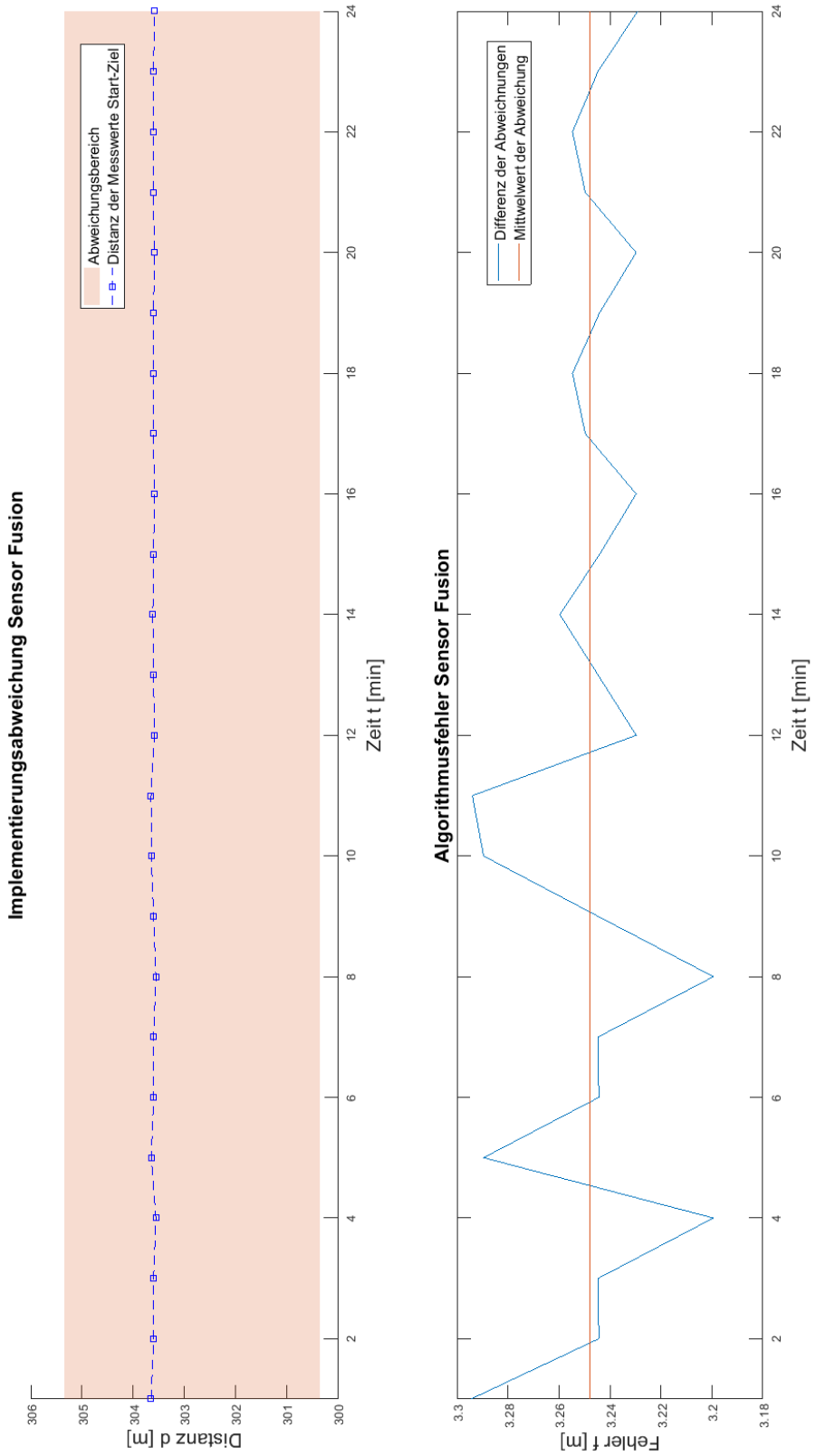
C.2. Projektverzeichnis-Struktur Application Level Gateway



C.3. Laufzeitdiagramm des Sicherheitsservers für eingehende Sensorkommunikation und Abfrage des Webservers



C.4. Genauigkeit der Sensor Fusion bei der Auflösung von 60-Werten pro Minute und einer Geschwindigkeit von 4,99 m/s



D. Quellcode

Listings

D.1. Sicherheitsserver

D.1.1. Makefile

```
#/*****************************************************************************
# * Description: Makefile
# * Date-created: 16/07/18
# * Date-Updated: 02/10/2018
# * Author:     Nils Parche
# *****************************************************************************/

# declare Variable
CC=g++ # select Compiler

CFLAGS=-Wall -lrt -lpthread -Wno-psabi # FLAGS

HEADER_PATH=inc/
SOURCE_PATH=src/

MyLib=$(SOURCE_PATH)peripheral.cpp $(SOURCE_PATH)messageQueue.cpp $(SOURCE_PATH)mySocket.cpp $(SOURCE_PATH)NmeaLwe.
      cpp $(SOURCE_PATH)NmeaSentence.cpp $(SOURCE_PATH)myTimer.cpp $(SOURCE_PATH)globalAnalyse.cpp $(SOURCE_PATH)
      globals.cpp $(SOURCE_PATH)SensorFusion.cpp
MyObj=peripheral.o messageQueue.o mySocket.o NmeaLwe.o NmeaSentence.o Sensor.o ContainerInputData.o
      ContainerStatistik.o myTimer.o globalAnalyse.o globals.o SensorFusion.o

all: initSystem clientUdp serverUdp analyse clean

initSystem: $(SOURCE_PATH)initSystem.cpp $(MyObj)
      $(CC) $(MyObj) $(SOURCE_PATH)initSystem.cpp $(CFLAGS) -o initSystem
clientUdp: $(SOURCE_PATH)clientUdp.cpp $(MyObj)
      $(CC) $(MyObj) $(SOURCE_PATH)clientUdp.cpp $(CFLAGS) -o clientUdp
serverUdp: $(SOURCE_PATH)serverUdp.cpp $(MyObj)
      $(CC) $(MyObj) $(SOURCE_PATH)serverUdp.cpp $(CFLAGS) -o serverUdp
analyse: $(SOURCE_PATH)analyse.cpp $(MyObj)
      $(CC) $(MyObj) $(SOURCE_PATH)analyse.cpp $(CFLAGS) -o analyse
clean:
      rm *.o

peripheral.o: $(SOURCE_PATH)peripheral.cpp
      $(CC) -c $(SOURCE_PATH)peripheral.cpp -o peripheral.o
messageQueue.o: $(SOURCE_PATH)messageQueue.cpp
      $(CC) -c $(SOURCE_PATH)messageQueue.cpp -o messageQueue.o
mySocket.o: $(SOURCE_PATH)mySocket.cpp
      $(CC) -c $(SOURCE_PATH)mySocket.cpp -o mySocket.o
NmeaLwe.o: $(SOURCE_PATH)NmeaLwe.cpp
      $(CC) -c $(SOURCE_PATH)NmeaLwe.cpp -o NmeaLwe.o
NmeaSentence.o: $(SOURCE_PATH)NmeaSentence.cpp
      $(CC) -c $(SOURCE_PATH)NmeaSentence.cpp -o NmeaSentence.o
Sensor.o: $(SOURCE_PATH)Sensor.cpp
      $(CC) -c $(SOURCE_PATH)Sensor.cpp -o Sensor.o
ContainerInputData.o: $(SOURCE_PATH)ContainerInputData.cpp
      $(CC) -c $(SOURCE_PATH)ContainerInputData.cpp -o ContainerInputData.o
ContainerStatistik.o: $(SOURCE_PATH)ContainerStatistik.cpp
      $(CC) -c $(SOURCE_PATH)ContainerStatistik.cpp -o ContainerStatistik.o
myTimer.o: $(SOURCE_PATH)myTimer.cpp
      $(CC) -c $(SOURCE_PATH)myTimer.cpp -o myTimer.o
globalAnalyse.o: $(SOURCE_PATH)globalAnalyse.cpp
      $(CC) -c $(SOURCE_PATH)globalAnalyse.cpp -o globalAnalyse.o
globals.o: $(SOURCE_PATH)globals.cpp
      $(CC) -c $(SOURCE_PATH)globals.cpp -o globals.o
SensorFusion.o: $(SOURCE_PATH)SensorFusion.cpp
      $(CC) -c $(SOURCE_PATH)SensorFusion.cpp -o SensorFusion.o
```

D.1.2. ContainerInputData.h

```
/*****************************************************************************
 * Description: ContainerInputData.h
 * Date-created: 24/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#ifndef ANALYSEDATA_CONTAINERINPUTDATA_H
#define ANALYSEDATA_CONTAINERINPUTDATA_H

#include <string>
#include <vector>

using namespace std;

class ContainerInputData {

public:
   // constructor
   ContainerInputData(string data);

   // methodes
   string getRawData();
   string getSegmentedData();
   string getSensorData();
   double getHeaderTimespamp();
   unsigned int getHeaderLineCount();
   string getSentenceTalkerID();
   string getSentenceFormatter();
   string getDataPayload();
   string getIPAddress();
   string getMacAddress();
   int getDataBytes();
   double getTimestampIncommingDate();
   void loadSensorData();
   void split(vector<string> &cont, string &str, char delim);
   void loadDate();

private:

   string mInputData = "0";
   vector <string> mHeaderVector;
   vector <string> mSentenceVector;
   vector <string> mDataVector;
   string mIpAddress = "0";
   string mMacAddress = "0";
   string mPid = "0";
   double mTimestampIncommingData = 0;
   string mSensorData = "0";
   string mSentenceTalkerId;
   string mSentenceFormatter;
   unsigned int mHeaderLineCount = 0; // n:
   double mHeaderTimetamp = 0; // c:
   string mDataPayload;
   int mDataBytes = 0;
};

#endif //ANALYSEDATA_CONTAINERINPUTDATA_H
```

D.1.3. ContainerStatistik.h

```
/*******************************************************************************
 * Description: ContainerStatistik.h
 * Date-created: 25/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 ******************************************************************************/

#ifndef ANALYSEDATA_CONTAINERSTATISTIK_H
#define ANALYSEDATA_CONTAINERSTATISTIK_H

class ContainerStatistik {

public:
   // constructor
   ContainerStatistik();

   // methodes
   void incAllData();
   void incFailureDataFrame();
   void incRegisteredSensorData();
   void incFailureDataLineCount();
   void incFailureTimeDiff();
   void incNotRigsteredSensorData();
   void incValidData();
   void incValidSensor1();
   void incValidSensor2();
   void incValidSensor3();
   void incWarnings();
   void incAlarms();
   void setWarning();
   void setAlarm();
   void clearAlarm();
   void clearWarning();
   void setSensor1DataSize(int bytes);
   void setSensor2DataSize(int bytes);
   void setSensor3DataSize(int bytes);
   void clearDataSize();
   long long incFailureSensorFusion();

   long long getAllData();
   long long getFailureIncommingData();
   long long getRegisteredSensorData();
   long long getFailureDataLineCount();
   long long getFailureTimeDiff();
   long long getNotRigsteredSensorData();
   long long getValidData();
   long long getValidSensor1();
   long long getValidSensor2();
   long long getValidSensor3();
   long long getFailureAll();
   bool getWarningsActive();
   bool getAlarmsActive();
   long long getWarningsAll();
   long long getAlarmsAll();
   int getSensor1DataSize();
   int getSensor2DataSize();
   int getSensor3DataSize();
   long long getFailureSensorFusion();

private:

   // variables
   long long mAllData = 0;       // All incomming Data
   long long mFailureIncommingData = 0; // failure incomming Data
   long long mValidIncommingData = 0; // valide incomming Data
   long long mRegisteredSensorData = 0; // sensor data from registered sensor detected
   long long mFailureDataLineCount = 0; // incomming sensordata not in correct order
   long long mFailureTimeDiff = 0; // data to high time diff
   long long mNotRigsteredSensorData = 0; // sensor data from not registered sensor detected
   long long mValidData = 0;     // All valid Data
   long long mValidSensor1 = 0;  // valid data sensor1
   long long mValidSensor2 = 0;  // valid data sensor2
   long long mValidSensor3 = 0;  // valid data sensor3
   long long mFailureAll = 0;    // sum of all failures
   long long mWarningsAll = 0;   // sum of all warnings detected
   bool mWarningsActive = false; // active Warnings
   long long mAlarmsAll = 0;     // sum of all alarms detected
   bool mAlarmsActive = false;   // active alarms
   long long mTransferDataAll = 0; // Datenuebertragen aller Sensoren
   long long mTransferDataSensor1 = 0; // Datentransfer Sensor 1
```

```
81      long long mTransferDataSensor2 = 0; // Datentransfer Sensor 2
82      long long mTransferDataSensor3 = 0; // Datentransfer Sensor 3
83      long long mFailureSensorFusion = 0; // Sensorfusion Indikator
84
85  };
86
87  #endif //ANALYSEDATA_CONTAINERSTATISTIK_H
```

D.1.4. globals.h

```
1   /*******************************************************************************
2    * Description: globals.h
3    * Date-created: 01/07/18
4    * Date-Updated: 02/10/2018
5    * Author:     Nils Parche
6    *******************************************************************************/
7
8   #ifndef GET_STARTED_C_GLOBALS_H
9   #define GET_STARTED_C_GLOBALS_H
10
11  #define M_PI 3.141592
12
13  #define UDP_BUFFER_SIZE    512
14  #define ETH_INTERFACE_SENSOR "eth0"  // RP-NEW eth0 // RP enxb827ebbd3ad1 // LAPTOP enp0s25
15
16  // port number for socket
17  #define PORTNUM_MULTICAST_MISC 60001
18  #define PORTNUM_MULTICAST_TGTD 60002
19  #define PORTNUM_MULTICAST_SATD 60003
20  #define PORTNUM_MULTICAST_NAVD 60004
21  #define PORTNUM_MULTICAST_VDRD 60005
22  #define PORTNUM_MULTICAST_RCOM 60006
23  #define PORTNUM_MULTICAST_TIME 60007
24  #define PORTNUM_INTERN     50001
25
26  // multicast groups
27  #define MULTICAST_GROUP_MISC "239.192.0.1"
28  #define MULTICAST_GROUP_TGTD "239.192.0.2"
29  #define MULTICAST_GROUP_SATD "239.192.0.3"
30  #define MULTICAST_GROUP_NAVD (char *) "239.192.0.4"
31  #define MULTICAST_GROUP_VDRD "239.192.0.5"
32  #define MULTICAST_GROUP_RCOM "239.192.0.6"
33  #define MULTICAST_GROUP_TIME "239.192.0.7"
34
35  // desination source
36  #define IP_DESTINATION_INTERN "172.16.1.66"
37  #define IP_INTERFACE_EXTERN "192.168.1.1"
38
39  // program names
40  #define PROGRAM_NAME_ANALYSE (char *) "/home/pi/secureServer/analyse" // "./analyse"
41  #define PROGRAM_NAME_CLIENT_UDP (char *) "/home/pi/secureServer/clientUdp" // "./clientUdp"
42  #define PROGRAM_NAME_SERVER_UDP (char *) "/home/pi/secureServer/serverUdp" // "./serverUdp"
43
44  // file name
45  #define FILE_VALID_SENSORS "/home/pi/secureServer/config/validSensors" // file-name validSensors, direct pfad
46  #define FILE_FAIL_LOG "/home/pi/secureServer/log/failLog" // file-name failLog, direct pfad
47  #define FILE_STATSTIK "/home/pi/secureServer/log/statistik" // file-name statistik, direct pfad
48  #define FILE_SYSTEM "/home/pi/secureServer/log/system" // file-name statistik, direct pfad
49
50  // message queue names & parameters
51  #define MQ_NAME_SERVER_UDP_ANALYSE (char *) "/mqServerUdpAnalyse"
52  #define MQ_NAME_ANALYSE_CLIENT_UDP (char *) "/mqAnalyseClientUdp"
53  #define MAX_SIZE_QUEUE 1024
54  // message queue rights format: user, group, others
55  #define MODE_MQ_SA (S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)
56
57  // globale variables
58  extern bool volatile minuteFlag;
59
60  // Debug-Mode
61  //#define DEBUG
62
63  #endif //GET_STARTED_C_GLOBALS_H
```

D.1.5. messageQueue.h

```
/*******************************************************************************
 * Description: messageQueue.h
 * Date-created: 20/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 ******************************************************************************/

#ifndef SECURESERVER_MESSAGE_QUEUE_H
#define SECURESERVER_MESSAGE_QUEUE_H

mqd_t mq_server_init(char* mq_name);
mqd_t mq_client_init(char* mq_name);

#endif //SECURESERVER_MESSAGE_QUEUE_H
```

D.1.6. mySocket.h

```
//
// Created by nils on 22.07.18.
//

#ifndef SECURESERVER_SOCKET_INIT_H
#define SECURESERVER_SOCKET_INIT_H

void init_multicast_socket(int *pMySocket, struct sockaddr_in *pSockaddr, char *pMulticastGroup, int port);
void init_udp_server_socket(int *socket, struct sockaddr_in *sockaddr, int port, char *ip_addr);
void init_udp_client_socket(int *socket, struct sockaddr_in *sockaddr, int port, char *ip_addr);

#endif //SECURESERVER_SOCKET_INIT_H
```

D.1.7. myTimer.h

```
/*******************************************************************************
 * Description: myTimer.h
 * Date-created: 29/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 ******************************************************************************/

#ifndef SECURESERVER_MYTIMER_H
#define SECURESERVER_MYTIMER_H

int timer_init(void);

#endif //SECURESERVER_MYTIMER_H
```

D.1.8. NmeaLwe.h

```
/*****************************************************************************
 * Description: NmeaLwe.h
 * Date-created: 29/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <iostream>
#include <string>
#include <vector>

#ifndef NMEA_PARSER_NMEALWE_H
#define NMEA_PARSER_NMEALWE_H

using namespace std;

class NmeaLwe {

public:
   // variables
   string nmeaData="0";

   // constructor
   NmeaLwe(string nmeaData);

   // methoden
   bool chkHeader();
   bool chkTag();
   bool isValidLwe();
   bool isTagBlockFormCorrect(string tagBlockId, string tagBlockField);
   bool chkTagChecksum();
   string getNmeaHeader();
   string getNmeaTagBlock();
   string getNmeaTagChecksum();
   string getNmeaLwe();

private:
   // variables
   bool mNmeaLwe = false;
   bool mNmeaHeader = false;
   bool mNmeaTagBlock = false;
   bool mNmeaTagChecksum = false;
   bool mNmeaValidLwe = false;
   string mNmeaLweStr="0";
   string mNmeaHeaderStr="0";
   string mnmeaTagBlockStr = "0";
   vector <string> mNmeaTagBlockVectorStr;
   string mNmeaTagChecksumStr="0";
};

#endif //NMEA_PARSER_NMEALWE_H
```

D.1.9. nmeaLweDefines.h

```
/*****************************************************************************
 * Description: nmeaLweDefines.h
 * Date-created: 29/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <iostream>

#include <vector>

#ifndef NMEA_PARSER_NMEALWEDEFINES_H
#define NMEA_PARSER_NMEALWEDEFINES_H

#define NMEALWE_HEADERLENGTH 6

#define NMEALWE_TAGBLOCKMAXLENGTH 80 // 61162-450 Annex B
#define NMEALWE_TAGBLOCKMINLENGTH 4 // mindestens 1 Tag + ein Buchstabe und Zahl
#define NMEALWE_CHECKSUMLENGTH 2 // checksum max 2

//vector <string> nmeaLweHeaderMaster[5] = {
const string nmeaLweHeaderMaster[5] = {
      "UdPbC",
      "RaUdP",
      "RrUdP",
      "RpUdP",
      "NkPgN"
};

const string nmeaLweTabBlockMaster[6] = {
      "d:", // destination
      "s:", // source
      "g:", // group
      "t:", // text
      "n:", // line-count
      "c:"  // time

};

#endif //NMEA_PARSER_NMEALWEDEFINES_H
```

D.1.10. NmeaSentence.h

```
/*****************************************************************************
 * Description: NmeaSentence.h
 * Date-created: 20/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <iostream>
#include <string>
#include <vector>

#ifndef NMEA_PARSER_NMEASENTENCE_H
#define NMEA_PARSER_NMEASENTENCE_H

using namespace std;

class NmeaSentence{

public:
   // variables
   string inputData;

   // constructor
   NmeaSentence(string inputData);

   // methoden
   bool chkLength();
   int getLength();
   bool chkAddress();
   string getAddress();
   bool chkDataField();
   string getDataField();
   bool isDataFieldCorrect(string dataField);
   bool chkChecksum();
   string getChecksum();
   string getSentence();
   bool isValidSentence();

private:
   // variables
   int mNmeaSentenceLengthCount = 0;
   string mNmeaSentenceAddressStr = "0";
   string mNmeaSentenceDataStr = "0";
   vector <string> mNmeaSentenceDataVector;
   string mNmeaSentenceChecksumStr = "0";

   bool mNmeaSentenceLength = false;
   bool mNmeaSentenceAddress = false;
   bool mNmeaSentenceChecksum = false;
   bool mNmeaSentenceData = false;
   bool mNmeaValidSentence = false;

};

#endif //NMEA_PARSER_NMEASENTENCE_H
```

D.1.11. nmeaSentenceDefines.h

```
/*****************************************************************************
 * Description: nmeaSentenceDefines.h
 * Date-created: 20/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <string>

#ifndef NMEA_PARSER_NMEASENTENCEDEFINES_H
#define NMEA_PARSER_NMEASENTENCEDEFINES_H


#define NMEASENTENCE_MAXLENGTH 82+6
#define NMEASENTENCE_CHECKSUMLENGTH 2

const string nmeaTalkterIdentifier[57] = {
      "AG",
      "AP",
      "AI",
      "BI",
      "BN",
      "CD",
      "CR",
      "CS",
      "CT",
      "CV",
      "CX",
      "DF",
      "DU",
      "EC",
      "EI",
      "EP",
      "ER",
      "FD",
      "FE",
      "FR",
      "FS",
      "GA",
      "GP",
      "GL",
      "GN",
      "HC",
      "HE",
      "HF",
      "HN",
      "HD",
      "HS",
      "II",
      "IN",
      "LC",
      "NL",
      "P",
      "RA",
      "RC",
      "SD",
      "SG",
      "SN",
      "SS",
      "TI",
      "UP",
      "U#",
      "VD",
      "VM",
      "VW",
      "VR",
      "WD",
      "WL",
      "YX",
      "ZA",
      "ZC",
      "ZQ",
      "ZV",
      "WI"
};

const string nmeaSentenceFormatter[108] = {
      "Q",
      "AAM",
      "ABK",
```

```
81          "ABM",
82          "ACA",
83          "ACK",
84          "ACS",
85          "AIR",
86          "AKD",
87          "ALA",
88          "ALR",
89          "APB",
90          "BBM",
91          "BEC",
92          "BOD",
93          "BWC",
94          "BWR",
95          "BWW",
96          "CBR",
97          "CUR",
98          "DBT",
99          "DDC",
100         "DOR",
101         "DPT",
102         "DSC",
103         "DSE",
104         "DTM",
105         "ETL",
106         "EVE",
107         "FIR",
108         "FSI",
109         "GBS",
110         "GEN",
111         "GFA",
112         "GGA",
113         "GLL",
114         "GNS",
115         "GRS",
116         "GSA",
117         "GST",
118         "GSV",
119         "HBT",
120         "HDG",
121         "HDT",
122         "HMR",
123         "HMS",
124         "HSC",
125         "HSS",
126         "HTC",
127         "HTD",
128         "LR1",
129         "LR2",
130         "LR3",
131         "LRF",
132         "LRI",
133         "MEB",
134         "MSK",
135         "MSS",
136         "MTW",
137         "MWD",
138         "MWV",
139         "NAK",
140         "NRM",
141         "NRX",
142         "OSD",
143         "POS",
144         "PRC",
145         "RMA",
146         "RMB",
147         "RMC",
148         "ROR",
149         "ROT",
150         "RPM",
151         "RSA",
152         "RSD",
153         "RTE",
154         "SFI",
155         "SSD",
156         "STN",
157         "THS",
158         "TLB",
159         "TLL",
160         "TRC",
161         "TRD",
162         "TTD",
163         "TTM",
```

```
164         "TUT",
165         "TXT",
166         "UID",
167         "VBW",
168         "VDR",
169         "VER",
170         "VHW",
171         "VLW",
172         "VPW",
173         "VSD",
174         "VTG",
175         "WAT",
176         "WCV",
177         "WNC",
178         "WPL",
179         "XDR",
180         "XTE",
181         "XTR",
182         "ZDA",
183         "ZDL",
184         "ZFO",
185         "ZTG"
186 };
187
188 #endif //NMEA_PARSER_NMEASENTENCEDEFINES_H
```

D.1.12. peripheral.h

```
1  /*******************************************************************************
2   * Description: peripheral.h
3   * Date-created: 01/07/18
4   * Date-Updated: 02/10/2018
5   * Author:     Nils Parche
6   *******************************************************************************/
7
8  #include <vector>
9  #include <string>
10
11 using namespace std;
12
13 #ifndef GET_STARTED_C_PERIPHERAL_H
14 #define GET_STARTED_C_PERIPHERAL_H
15
16 string getTimestamp();
17 int getMacAddress(string &ipAddress, string &macAddress);
18 void proc_exit(int signr);
19 void error(const char *msg);
20 void timerThread(union sigval arg);
21
22 #endif //GET_STARTED_C_PERIPHERAL_H
```

D.1.13. Sensor.h

```
/*****************************************************************************
 * Description: Sensor.h
 * Date-created: 23/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#ifndef ANALYSEDATA_SENSOR_H
#define ANALYSEDATA_SENSOR_H

#include <string>

using namespace std;

class Sensor {

public:
   // variables
   string parameterdata = "0";
   string timestampDataFrame = "0";
   string timestampIncommingData = "0";

   // constructor
   Sensor();
   Sensor(string data);

   // methoden
   bool chkSensorParameter(string ip, string mac, string talkerId, string formatter);
   string getIPAddress();
   string getMacAddress();
   string getRawParameter();
   string getParameter();
   double getTimestampincommingDate();
   unsigned int getLineCount();
   double getTimestampHeader();
   double getTimeDiffHeader();
   double getTimeDiffInData();
   double getGllLat();
   double getGllLon();
   double getHdtDegree();
   double getVBWLonWaterSpeed();
   double getVBWTransWaterSpeed();
   string getDataPayload();
   string getFormatter();
   int getDataSize();
   void setSensorInputData(string inputData);
   void setLineCount(unsigned int lineCount);
   void setTimestampHeader(double timestamp);
   void setTimestampIncommingData(double timestamp);
   void setTimeDiffHeader(double diff);
   void setTimeDiffInData(double diff);
   void setDataPayload(string payload);
   void setDataSize(int bytes);
   bool chkDiff();
   void loadParameters();
   void loadData();

private:
      // variables
   string mIPAddress = "0";
   string mMacAddress = "0";
   string mTalkerId = "0";
   string mFormatter = "0";
   unsigned int mLineCount = 0;
   double mTimestampHeader = 0;
   double mTimestampIncommingData = 0;
   double mTimeDiffHeader = 0;
   double mTimeDiffInData = 0;
   double msignalDelay = 0;
   string mDataPayload = "0";
   int mDataSize = 0;

   // GPGGL Source GP: GPS - Global Position Sensor
   //       GLL
   double mGpGllLat = 0.0;
   double mGpGllLon = 0.0;
   double mGpGllTime = 0.0;

   // HEHDT Source HE: Heading Sensor - gyro-north seeking sensor
   double mHeHdtDegree = 0.0;
```

```
81
82      // VD    Source VD: Velocity sensor - general
83      //       VBW
84      double mLonWaterSpeed = 0.0; // knots
85      double mTransWaterSpeed = 0.0; // knots
86      double mLonGroundSpeed = 0.0; // knots
87      double mTransGroundSpeed = 0.0; // knots
88  };
89
90  #endif //ANALYSEDATA_SENSOR_H
```

D.1.14. SensorFusion.h

```
/*****************************************************************************
 * Description: SensorFusion.h
 * Date-created: 30/08/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#ifndef SECURESERVER_SENSORFUSION_H
#define SECURESERVER_SENSORFUSION_H

#include <string>

using namespace std;

class SensorFusion {

public:
   // constructor
   SensorFusion();

   // methoden
   void loadDataGll(double time, double lat, double lon);
   void loadDataVBW(double time, double speedLon, double speedTrans);
   void loadDataHDT(double time, double degree);
   bool checkGllParameters();
   bool checkVbwParameters();
   bool checkHdtParameters();
   void fusion();
   bool isFusionValid();
      double gllToDezimal(double gllDegree);
   void getGpsPosition(double latS, double lonS, double *latD, double *lonD, double distance, double bearing);
   double toRadians(double degree);
   double distanceGps(double lat1, double lon1, double lat2, double lon2);
   double timeDiffGps(double time1, double time2);
   double knotsToMeters(double knots);
   double metersToKnots(double meters);

private:
   // variables
   // GPS
   double mLatStore = 0.0;
   double mLonStore = 0.0;
   double mLatNew = 0.0;
   double mLonNew = 0.0;
   double mGllTimeNew = 0.0;
   double mGllTimeStore = 0.0;
   // Speed
   double mLonWaterSpeedStore = 0.0; // meter
   double mTransWaterSpeedStore = 0.0; // meter
   //double mLonGroundSpeedStore = 0.0; // meter
   //double mTransGroundSpeedStore = 0.0; // meter
   double mLonWaterSpeedNew = 0.0; // meter
   double mTransWaterSpeedNew = 0.0; // meter
   double mVbwTimeNew = 0.0;
   double mVbwTimeStore = 0.0;
   //double mLonGroundSpeedNew = 0.0; // meter
   //double mTransGroundSpeedNew = 0.0; // meter
   double mLatCalc = 0.0;
   double mLonCalc = 0.0;
   // Degree
   double mHdtTimeNew = 0;
   double mHdtTimeStore = 0;
   double mBearingStore = 0.0; // grad
   double mBearingNew = 0.0; // grad
   bool mGllValid = false;
   bool mHdtValid = false;
   bool mVbwValid = false;
   bool mFusionValid = false;
   bool mFusionAktiv = false;
   double mDistanceCalc = 0.0;

   double mTimeWaterSpeedDiff = 0.0;
   double mDistanceCalcTotoal = 0.0;
};

#endif //SECURESERVER_SENSORFUSION_H
```

D.1.15. analyse.cpp

```
/*****************************************************************************
 * Description: analyse.cpp
 * Date-created: 16/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <mqueue.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <regex>
#include <sstream>
#include <signal.h>

/*********************************************
 * Eigene Include-Dateien
 *********************************************/

#include "../inc/messageQueue.h"
#include "../inc/peripheral.h"
#include "../inc/globals.h"
#include "../inc/Sensor.h"
#include "../inc/ContainerInputData.h"
#include "../inc/ContainerStatistik.h"
#include "../inc/myTimer.h"
#include "../inc/SensorFusion.h"



int main(int argc, char *argv[]) {

   /*********************************************
    * VARIABLEN
    *********************************************/
   bool validSensorData = false;
   bool validSensorParameter = false;
   int position;
   vector <Sensor> :: iterator k;

   char bufMsg[MAX_SIZE_QUEUE];
   int bytes_read;

   /*********************************************
    * Create Message-Queue Server
    * Kommunikationskanal:
    * UPD_SERVER -> ANALYSE
    *********************************************/
   mqd_t mq_sa;
   mq_sa = mq_server_init(MQ_NAME_SERVER_UDP_ANALYSE);

   /*********************************************
    * Create Message-Queue Client
    * Kommunikationskanal:
    * ANALYSE -> CLIENT_UDP
    *********************************************/
   mqd_t mq_ac;

   // delay time, spend time to create mq in process client_upd
   sleep(1);
   mq_ac = mq_client_init(MQ_NAME_ANALYSE_CLIENT_UDP);

   /*********************************************
    * Create Timer
    * Timer-Element zur Steuerung der Datenablage
    *********************************************/
   timer_init();

   string systemStartTimestamp = getTimestamp();
   ofstream file_o;

   file_o.open(FILE_SYSTEM, ofstream::in | ofstream::app);

   if(!file_o.is_open()) { cout << "failure - open file: " << FILE_SYSTEM << endl; error("failure open file: system
         "); exit(EXIT_FAILURE); }
```

```
80
81      file_o << systemStartTimestamp << ";" << endl;
82
83      file_o.close();
84
85      /***********************************************
86       * Statistik Objekt
87       * Speichert alle relevanen Daten
88       ***********************************************/
89      ContainerStatistik statistik;
90
91      /***********************************************
92       * SensorFusion Objekt
93       * Validiert Sensordaten
94       ***********************************************/
95      SensorFusion sensorFusion;
96
97      /***********************************************
98       * Vector Sensor Objekt "mySensors"
99       * Enthaelt alle Sensoren aus der config Datei
100      ***********************************************/
101     vector<Sensor> mySensors;
102     int sensorNumber = 0;
103
104     string line;
105     ifstream file;
106
107     file.open(FILE_VALID_SENSORS);
108
109     if(!file.is_open()) { cout << "failure - open file: " << FILE_VALID_SENSORS << endl; error("failure open file:
            validSensors"); exit(EXIT_FAILURE); }
110
111     while(getline(file, line)) {
112        Sensor sensor(line);
113        mySensors.push_back(sensor);
114     }
115
116     file.close();
117
118     /***********************************************
119      * Init der Sensoren aus der Config Datei
120      ***********************************************/
121     for (k=mySensors.begin(); k!=mySensors.end(); k++) {
122        k->loadParameters();
123     }
124
125     int loop=1;
126
127     /***********************************************
128      * while-loop
129      * Hauptroutine - analyse Prozess
130      ***********************************************/
131     while( loop ) {
132
133        sensorNumber = 0;
134        validSensorData = true;
135        validSensorParameter = true;
136
137        // clear input memory
138        memset(bufMsg, 0, sizeof(bufMsg));
139
140        // read data from the message-queue
141        bytes_read = mq_receive(mq_sa, bufMsg, MAX_SIZE_QUEUE, NULL);
142
143        // increment all incomming data
144        statistik.incAllData();
145
146        // create object ContainerInputData
147        ContainerInputData incommingSensorData(bufMsg);
148
149        /***********************************************
150         * Aufsplitten der Eingangsdaten
151         * IP | MAC | PID | TIMESTAMP_SOCKET | REC-BYTES | SENSOR-DATA
152         ***********************************************/
153        incommingSensorData.loadDate();
154        cout << "incommingSensorData.getDataBytes(): " << incommingSensorData.getDataBytes() << endl;
155
156        /***********************************************
157         * Abfrage des Dateninhalts
158         * getSensorData() == failure
159         * true => failure - failLog
160         * false => Datenverarbeitung
161         ***********************************************/
```

```
162         if (incommingSensorData.getSensorData() == "failure") {
163
164            validSensorData = false;
165
166            ofstream file;
167            file.open(FILE_FAIL_LOG, ofstream::in | ofstream::app);
168            if(!file.is_open()) { cout << "failure - open file: " << FILE_FAIL_LOG << endl; error("failure open file:
                    failLog"); exit(EXIT_FAILURE); }
169            file << bufMsg << endl;
170            file.close();
171
172            /*
173             * ALARM daten nicht versenden!!
174             * setzen der Alarmvalues
175             */
176            statistik.incFailureDataFrame();
177            statistik.incAlarms();
178            statistik.setAlarm();
179
180
181            /*
182             * Bei bestimmten wiederholungsfällen sperren der IP "Black-List"
183             */
184
185            /*********************************************
186             * Abfrage des Dateninhalts
187             * getSensorData() == failure
188             * false => Datenverarbeitung
189             *********************************************/
190         } else {
191
192            /*********************************************
193             * Aufsplitten der SENSOR-DATA
194             * HEADER-TIMESTAMP | HEADER-LINECOUNT | TALKER-ID | SENTENCE-FORMATTER | SENTENCE-PAYLOAD
195             *********************************************/
196            incommingSensorData.loadSensorData();
197
198            /*********************************************
199             * Suche nach registrierten Sensoren
200             * IP | MAC | TALKER-ID | SENTENCE-FORMATTER
201             *********************************************/
202            for(k=mySensors.begin(); k!=mySensors.end(); k++) {
203
204               position = distance(mySensors.begin(), k);
205
206               if( k->chkSensorParameter(incommingSensorData.getIPAddress(), incommingSensorData.getMacAddress(),
                        incommingSensorData.getSentenceTalkerID(), incommingSensorData.getSentenceFormatter()) ){
207                  sensorNumber = position+1;
208                  statistik.incRegisteredSensorData();
209               }
210
211            }
212
213            /*********************************************
214             * Uebereinstimmenden Sensor gefunden
215             *********************************************/
216            if (sensorNumber > 0) {
217               sensorNumber--; // correction
218
219               /*********************************************
220                * LINE-COUNT
221                * Ueberpruefen des Parameters
222                * 1. Datenwert nach der Initialisierung
223                *********************************************/
224               if (mySensors[sensorNumber].getLineCount() == 0) {
225                  mySensors[sensorNumber].setLineCount(incommingSensorData.getHeaderLineCount());
226
227                  /*********************************************
228                   * LINE-COUNT
229                   * Vorlaufende Abfrage der Datenreihenfolge
230                   * Datenwerte sind in der Reihenfolge
231                   *********************************************/
232               } else if(( incommingSensorData.getHeaderLineCount() -1 ) == mySensors[sensorNumber].getLineCount()) {
233                  // set first line count value
234                  mySensors[sensorNumber].setLineCount(incommingSensorData.getHeaderLineCount());
235
236                  /*********************************************
237                   * LINE-COUNT
238                   * Datenwerte NICHT in numerischen Reihenfolge
239                   *********************************************/
240               } else {
241                  /*
242                   * Warning
```

```
243                 * Daten dennoch versenden
244                 */
245                statistik.setWarning();
246                statistik.incWarnings();
247                statistik.incFailureDataLineCount();
248
249                // set new line count value
250                mySensors[sensorNumber].setLineCount(incommingSensorData.getHeaderLineCount());
251             }
252
253             /*********************************************
254              * TIMESTAMP
255              * Ueberpruefen des Parameters
256              * 1. Datenwert nach der Initialisierung
257              *********************************************/
258             if( mySensors[sensorNumber].getTimestampincommingDate() == 0 ) {
259                mySensors[sensorNumber].setTimestampIncommingData(incommingSensorData.getTimestampIncommingDate());
260                mySensors[sensorNumber].setTimestampHeader(incommingSensorData.getHeaderTimespamp());
261
262                /*********************************************
263                 * TIMESTAMP
264                 * Datenwerte im laufdendem Prozess
265                 *********************************************/
266             } else {
267
268                // calc time difference between values
269                double timeDiffI = incommingSensorData.getTimestampIncommingDate() - mySensors[sensorNumber].
                        getTimestampincommingDate();
270                double timeDiffH = incommingSensorData.getHeaderTimespamp() - mySensors[sensorNumber].
                       getTimestampHeader();
271
272                // set time difference
273                mySensors[sensorNumber].setTimeDiffInData(timeDiffI);
274                mySensors[sensorNumber].setTimeDiffHeader(timeDiffH);
275
276                /*********************************************
277                 * TIMESTAMP
278                 * Check der Zeitdifferenz zwischen:
279                 * Timestamp Sensordaten | Timestamp Socket
280                 * Differenz ist im Wertebereich
281                 *********************************************/
282                if ( mySensors[sensorNumber].chkDiff() ) {
283
284                   /*********************************************
285                    * TIMESTAMP
286                    * Check der Zeitdifferenz zwischen:
287                    * Timestamp Sensordaten | Timestamp Socket
288                    * Differenz ist NICHT im Wertebereich
289                    *********************************************/
290                } else {
291                      /*
292                       * Warning .........
293                       */
294                      statistik.incWarnings();
295                      statistik.setWarning();
296                      statistik.incFailureTimeDiff();
297                }
298                // new data ist old data
299                mySensors[sensorNumber].setTimestampIncommingData(incommingSensorData.getTimestampIncommingDate());
300                mySensors[sensorNumber].setTimestampHeader(incommingSensorData.getHeaderTimespamp());
301             }
302
303             /*********************************************
304              * Sensor-Parameter
305              * Values fuer SensorFusionsBlock vorbereiten
306              *********************************************/
307
308             mySensors[sensorNumber].setDataPayload(incommingSensorData.getDataPayload());
309             mySensors[sensorNumber].loadData();
310             mySensors[sensorNumber].setDataSize(incommingSensorData.getDataBytes());
311
312             /*********************************************
313              * SensorFusion
314              * Auswertung der Daten nach der Sensorfusion
315              *********************************************/
316
317             /*********************************************
318              * SensorFusion
319              * Physikalische Eigenschaften checken
320              *********************************************/
321
322             if (mySensors[sensorNumber].getFormatter() == "GLL") {
323                cout << "SensorFusion-GLL: " << endl;
```

```
324                printf("Lat: %f\n", mySensors[sensorNumber].getGllLat());
325                printf("Lon: %f\n", mySensors[sensorNumber].getGllLon());
326
327                sensorFusion.loadDataGll(
328                      mySensors[sensorNumber].getTimestampHeader(),
329                      mySensors[sensorNumber].getGllLat(),
330                      mySensors[sensorNumber].getGllLon()
331                      );
332                if ( sensorFusion.checkGllParameters() == false ) {
333                   /*
334                    * Sensorwerte nicht ok -> verwerfen
335                    * ALARM
336                    * notvalid
337                    */
338                   validSensorParameter = false;
339                }
340             } else if (mySensors[sensorNumber].getFormatter() == "HDT") {
341                sensorFusion.loadDataHDT(
342                      mySensors[sensorNumber].getTimestampHeader(),
343                      mySensors[sensorNumber].getHdtDegree()
344                      );
345                if ( sensorFusion.checkHdtParameters() == false ) {
346                   /*
347                    * Sensorwerte nicht ok -> verwerfen
348                    * ALARM
349                    * notvalid
350                    */
351                   validSensorParameter = false;
352                }
353             } else if (mySensors[sensorNumber].getFormatter() == "VBW") {
354                sensorFusion.loadDataVBW(
355                      mySensors[sensorNumber].getTimestampHeader(),
356                      mySensors[sensorNumber].getVBWLonWaterSpeed(),
357                      mySensors[sensorNumber].getVBWTransWaterSpeed()
358                      );
359                if ( sensorFusion.checkVbwParameters() == false ) {
360                   /*
361                    * Sensorwerte nicht ok -> verwerfen
362                    * ALARM
363                    * notvalid
364                    */
365                   validSensorParameter = false;
366                }
367             }
368
369             /*********************************************
370              * SensorFusion
371              * Berechnung Werte
372              *********************************************/
373
374             if (validSensorParameter) {
375                sensorFusion.fusion();
376
377                if (sensorFusion.isFusionValid() == false) {
378                   statistik.incFailureSensorFusion();
379                   statistik.incAlarms();
380                   statistik.setAlarm();
381                   //validSensorData = false;
382                   /*
383                    * Sensorfusion valid
384                    */
385                }
386             }
387
388             /*********************************************
389              * KEIN Uebereinstimmenden Sensor gefunden
390              *********************************************/
391          } else {
392
393             // alarm
394             statistik.incNotRigsteredSensorData();
395             statistik.incAlarms();
396             statistik.setAlarm();
397          }
398       }
399
400       /*********************************************
401        * Datenausgabe an Message-Queue
402        * Nach Auswertung von validSensorData
403        * datenausgabe an Kommunikationskanal -> Client_UDP
404        *********************************************/
405
406       if (validSensorData) {
```

```
407
408            if (mySensors[sensorNumber].getFormatter() == "GLL") {
409               cout << "GLL++" << endl;
410               statistik.incValidSensor1();
411               statistik.setSensor1DataSize(mySensors[sensorNumber].getDataSize());
412
413            } else if (mySensors[sensorNumber].getFormatter() == "HDT") {
414               statistik.incValidSensor2();
415               statistik.setSensor2DataSize(mySensors[sensorNumber].getDataSize());
416               cout << "HDT++" << endl;
417            } else if (mySensors[sensorNumber].getFormatter() == "VBW") {
418               cout << "VWB++" << endl;
419               statistik.incValidSensor3();
420               statistik.setSensor3DataSize(mySensors[sensorNumber].getDataSize());
421            }
422
423            statistik.incValidData();
424
425            mq_send(mq_ac, bufMsg, bytes_read, 0);
426         }
427
428         /*********************************************
429          * Datenausgabe an Message-Queue
430          * Nach Auswertung von validSensorData
431          * datenausgabe an Kommunikationskanal -> Client_UDP
432          *********************************************/
433         if (minuteFlag) {
434
435            ofstream file;
436            file.open(FILE_STATSTIK, ofstream::in | ofstream::app);
437            if(!file.is_open()) { cout << "failure - open file: " << FILE_STATSTIK << endl; error("failure open file:
                   statistik"); exit(EXIT_FAILURE); }
438
439            /* schreibe datenwerte
440             * Zeitstempel | Statistikwerte (IncommingData / Failure welcher art auch immer)
441             */
442
443            string dataLogTimestamp = getTimestamp();
444
445            stringstream statistikLog;
446
447            // value 1
448            statistikLog << dataLogTimestamp << ";";
449            // value 2
450            statistikLog << statistik.getValidData() << ";";
451            // value 3
452            statistikLog << statistik.getValidSensor1() << ";";
453            // value 4
454            statistikLog << statistik.getValidSensor2() << ";";
455            // value 5
456            statistikLog << statistik.getValidSensor3() << ";";
457            // value 6
458            statistikLog << statistik.getAlarmsAll() << ";";
459            // value 7
460            statistikLog << statistik.getWarningsAll() << ";";
461            // value 8
462            statistikLog << statistik.getNotRigsteredSensorData() << ";";
463            // value 9
464            statistikLog << statistik.getFailureAll() << ";";
465            // value 10
466            statistikLog << statistik.getAlarmsActive() << ";";
467            // value 11
468            statistikLog << statistik.getWarningsActive() << ";";
469            // value 12
470            statistikLog << (statistik.getSensor1DataSize() + statistik.getSensor2DataSize() + statistik.
                   getSensor3DataSize()) << ";";
471            // value 13
472            statistikLog << statistik.getSensor1DataSize() << ";";
473            // value 14
474            statistikLog << statistik.getSensor2DataSize() << ";";
475            // value 15
476            statistikLog << statistik.getSensor3DataSize() << ";";
477            // value 16
478            statistikLog << statistik.getFailureIncommingData() << ";";
479            // value 17
480            statistikLog << statistik.getRegisteredSensorData() << ";";
481            // value 18
482            statistikLog << statistik.getFailureDataLineCount() << ";";
483            // value 19
484            statistikLog << statistik.getFailureTimeDiff() << ";";
485            // value 20
486            statistikLog << statistik.getFailureSensorFusion() << ";";
487
```

```
488             statistikLog << endl;
489
490             file << statistikLog.rdbuf();
491
492             file.close();
493
494             statistik.clearWarning();
495             statistik.clearAlarm();
496             statistik.clearDataSize();
497
498             minuteFlag = false;
499          }
500       }
501
502       // unmount message qoueue
503       mq_unlink(MQ_NAME_SERVER_UDP_ANALYSE);
504       mq_close(mq_sa);
505       mq_close(mq_ac);
506
507       return EXIT_SUCCESS;
508    }
```

D.1.16. clientUdp.cpp

```
/*****************************************************************************
 * Description: clientUdp.cpp
 * Date-created: 16/07/18
 * Date-Updated: 27/08/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <mqueue.h>
#include <arpa/inet.h>
#include <cstring>
#include <iostream>
#include <string>

#include "../inc/messageQueue.h"
#include "../inc/peripheral.h"
#include "../inc/globals.h"
#include "../inc/mySocket.h"

using namespace std;

int main(int argc, char *argv[]) {

   int i;
   char bufMsg[MAX_SIZE_QUEUE];

      /*********************************************
       * create message queue server
       * IPC: analyse -> client_udp
       *********************************************/

   mqd_t mq_sa;
   mq_sa = mq_server_init(MQ_NAME_ANALYSE_CLIENT_UDP);

      /*********************************************
       * Create UDP-client socket
       *********************************************/

   int clientSocket;
   struct sockaddr_in client_addr;
   int portNum = PORTNUM_INTERN;
   char ipClientChar[16] = IP_DESTINATION_INTERN;

   char outputBufferSocket[UDP_BUFFER_SIZE];
   int sendBytes;

   init_udp_client_socket(&clientSocket, &client_addr, portNum, ipClientChar);

   strcpy(outputBufferSocket, "TEST Message\n");
   string test = "Test Message 440f0!";
   int len = strlen(outputBufferSocket);

   while(1) {

      // clear buffer
      memset(bufMsg, 0, sizeof(bufMsg));
      mq_receive(mq_sa, bufMsg, MAX_SIZE_QUEUE, 0);

      strcpy(outputBufferSocket ,bufMsg);
      sendBytes = sendto(clientSocket, outputBufferSocket, strlen(outputBufferSocket), 0, (struct sockaddr *)&
            client_addr, sizeof(client_addr));
   }

   mq_close(mq_sa);
   mq_unlink(MQ_NAME_ANALYSE_CLIENT_UDP);

   return EXIT_SUCCESS;
}
```

D.1.17. ContainerInputData.cpp

```
/*****************************************************************************
 * Description: ContainerInputData.cpp
 * Date-created: 29/07/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <sstream>
#include <iostream>
#include "../inc/ContainerInputData.h"

// construktor
ContainerInputData::ContainerInputData(string data) {
   mInputData = data;
}

string ContainerInputData::getRawData() {

   return mInputData;
}

void ContainerInputData::split(vector<string> &cont, string &str, char delim) {

   string token;
   stringstream ss;
   ss.str(str);

   while(getline(ss, token, delim)) {
      cont.push_back(token);
   }

}

void ContainerInputData::loadDate() {

   split(mDataVector, mInputData, ';');

   if (mDataVector.size() == 6 ) {
      mIpAddress = mDataVector[0];
      mMacAddress = mDataVector[1];
      mPid = mDataVector[2];
      mTimestampIncommingData = stod(mDataVector[3]);
      mDataBytes = stod(mDataVector[4]);
      mSensorData = mDataVector[5];
   }

}

int ContainerInputData::getDataBytes() {
   return mDataBytes;
}


string ContainerInputData::getSegmentedData() {
   string segmentedData;
   segmentedData = mIpAddress + mMacAddress + mPid + to_string(mTimestampIncommingData) + mSensorData;
   return segmentedData;
}

string ContainerInputData::getSensorData() {
   return mSensorData;
}

void ContainerInputData::loadSensorData() {

   int headerStart;
   int headerEnd;
   int sentenceStart;
   int sentenceEnd;
   string headerFrame;
   string sentenceFrame;
   string tag;
   int end;
   int i;

   /*
    * zerlegen des datenstreams in header und sentence informationen
    */

   headerStart = (mSensorData.find_first_of(0x5C) + 1);
```

```
81     headerEnd = mSensorData.find_first_of(0x2A);
82     sentenceStart = (mSensorData.find_last_of(0x24) + 1);
83     sentenceEnd = mSensorData.find_last_of(0x2A);
84
85     headerFrame = mSensorData.substr(headerStart, headerEnd-headerStart);
86     sentenceFrame = mSensorData.substr(sentenceStart, sentenceEnd-sentenceStart);
87
88     /*
89      * read header-info
90      */
91
92     split(mHeaderVector, headerFrame, ',');
93
94     for (i=0; i<mHeaderVector.size(); i++) {
95        tag = mHeaderVector[i].substr(0,1);
96        end = mHeaderVector[i].length();
97        if (tag == "n") {
98           this->mHeaderLineCount = stoi(mHeaderVector[i].substr(2,end));
99        } else if (tag == "c") {
100          this->mHeaderTimetamp = stod(mHeaderVector[i].substr(2,end));
101          mHeaderTimetamp = mHeaderTimetamp/1000;
102          //printf("loadSensorData - mHeaderTimestamp: %lf\n", mHeaderTimetamp);
103       }
104    }
105
106    /*
107     * read sentence
108     */
109
110    split(mSentenceVector, sentenceFrame, ',');
111
112    /*
113     * feste position der Talker-ID und des Sentence Formatter
114     */
115    this->mSentenceTalkerId = mSentenceVector[0].substr(0,2);
116    this->mSentenceFormatter = mSentenceVector[0].substr(2,3);
117
118    mDataPayload = sentenceFrame.substr((sentenceFrame.find_first_of(',')+1),sentenceFrame.length()-(sentenceFrame.
           find_first_of(',')+1));
119
120 }
121
122 unsigned int ContainerInputData::getHeaderLineCount() {
123    return mHeaderLineCount;
124 }
125
126 double ContainerInputData::getHeaderTimespamp() {
127    return mHeaderTimetamp;
128 }
129
130 string ContainerInputData::getSentenceTalkerID() {
131    return mSentenceTalkerId;
132 }
133
134 string ContainerInputData::getSentenceFormatter() {
135    return mSentenceFormatter;
136 }
137
138 string ContainerInputData::getDataPayload() {
139    return mDataPayload;
140 }
141
142 string ContainerInputData::getIPAddress() {
143    return mIpAddress;
144 }
145
146 string ContainerInputData::getMacAddress() {
147    return mMacAddress;
148 }
149
150 double ContainerInputData::getTimestampIncommingDate() {
151    return mTimestampIncommingData;
152 }
```

D.1.18. ContainerStatistik.cpp

```
/*******************************************************************************
 * Description: ContainerStatistik.cpp
 * Date-created: 25/08/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 ******************************************************************************/

#include "../inc/ContainerStatistik.h"

// constructor
ContainerStatistik::ContainerStatistik() {

};

long long ContainerStatistik::getAllData() {
   return mAllData;
}

long long ContainerStatistik::getFailureIncommingData() {
   return mFailureIncommingData;
}

long long ContainerStatistik::getRegisteredSensorData() {
   return mRegisteredSensorData;
}

long long ContainerStatistik::getFailureDataLineCount() {
   return mFailureDataLineCount;
}

long long ContainerStatistik::getFailureTimeDiff() {
   return mFailureTimeDiff;
}

long long ContainerStatistik::getNotRigsteredSensorData() {
   return mNotRigsteredSensorData;
}

long long ContainerStatistik::getValidData() {
   return mValidData;
}

long long ContainerStatistik::getValidSensor1() {
   return mValidSensor1;
}

long long ContainerStatistik::getValidSensor2() {
   return mValidSensor2;
}

long long ContainerStatistik::getValidSensor3() {
   return mValidSensor3;
}


void ContainerStatistik::incAllData() {
   mAllData++;
}

void ContainerStatistik::incFailureDataFrame() {
   mFailureIncommingData++;
}

void ContainerStatistik::incRegisteredSensorData() {
   mRegisteredSensorData++;
}

void ContainerStatistik::incFailureDataLineCount() {
   mFailureDataLineCount++;
}

void ContainerStatistik::incFailureTimeDiff() {
   mFailureTimeDiff++;
}

void ContainerStatistik::incNotRigsteredSensorData() {
   mNotRigsteredSensorData++;
}

void ContainerStatistik::incValidData() {
```

```
81      mValidData++;
82  }
83
84  void ContainerStatistik::incValidSensor1() {
85      mValidSensor1++;
86  }
87
88  void ContainerStatistik::incValidSensor2() {
89      mValidSensor2++;
90  }
91
92  void ContainerStatistik::incValidSensor3() {
93      mValidSensor3++;
94  }
95
96  void ContainerStatistik::incAlarms() {
97      mAlarmsAll++;
98  }
99
100 void ContainerStatistik::incWarnings() {
101     mWarningsAll++;
102 }
103
104 void ContainerStatistik::setAlarm() {
105     mAlarmsActive = true;
106 }
107
108 void ContainerStatistik::setWarning() {
109     mWarningsActive = true;
110 }
111
112 void ContainerStatistik::clearAlarm() {
113     mAlarmsActive = false;
114 }
115
116 void ContainerStatistik::clearWarning() {
117     mWarningsActive = false;
118 }
119
120
121 long long ContainerStatistik::getFailureAll() {
122     mFailureAll = mFailureTimeDiff+mFailureDataLineCount+mFailureIncommingData;
123     return mFailureAll;
124 }
125
126 bool ContainerStatistik::getWarningsActive() {
127     return mWarningsActive;
128 }
129
130 long long ContainerStatistik::getWarningsAll() {
131     return mWarningsAll;
132 }
133
134 bool ContainerStatistik::getAlarmsActive() {
135     return mAlarmsActive;
136 }
137
138 long long ContainerStatistik::getAlarmsAll() {
139     return mAlarmsAll;
140 }
141
142 void ContainerStatistik::setSensor1DataSize(int bytes) {
143     mTransferDataSensor1 += bytes;
144 }
145
146 void ContainerStatistik::setSensor2DataSize(int bytes) {
147     mTransferDataSensor2 += bytes;
148 }
149
150 void ContainerStatistik::setSensor3DataSize(int bytes) {
151     mTransferDataSensor3 += bytes;
152 }
153
154 int ContainerStatistik::getSensor1DataSize() {
155     return mTransferDataSensor1;
156 }
157
158 int ContainerStatistik::getSensor2DataSize() {
159     return mTransferDataSensor2;
160 }
161
162 int ContainerStatistik::getSensor3DataSize() {
163     return mTransferDataSensor3;
```

```
164 }
165
166 long long ContainerStatistik::incFailureSensorFusion() {
167    mFailureSensorFusion++;
168 }
169
170 long long ContainerStatistik::getFailureSensorFusion() {
171    return mFailureSensorFusion;
172 }
173
174 void ContainerStatistik::clearDataSize() {
175    mTransferDataSensor1 = 0;
176    mTransferDataSensor2 = 0;
177    mTransferDataSensor3 = 0;
178 }
```

D.1.19. globals.cpp

```
1  /*****************************************************************************
2   * Description: globals.cpp
3   * Date-created: 29/08/18
4   * Date-Updated: 02/09/2018
5   * Author:     Nils Parche
6   *****************************************************************************/
7
8  #include "../inc/globals.h"
9
10 bool volatile minuteFlag = false;
```

D.1.20. initSystem.cpp

```
/*****************************************************************************
 * Description: initSytem.cpp
 * Date-created: 16/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

/*
 * System includes
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

/*
 * Project includes
 */

#include "../inc/peripheral.h"
#include "../inc/globals.h"

int main(void) {

   /*
    * System - Variables
    */

   int ret;
   pid_t pid;
   char errorMsg[256];

   // signal handler exiting child process
   signal(SIGCHLD, proc_exit);

      /*
    * load - ClientUdp
    */

   pid = fork();

   switch (pid) {
      case -1: /* error by using fork() */
         sprintf(errorMsg, "failure using fork %s", PROGRAM_NAME_CLIENT_UDP);
         error(errorMsg);
         break;
      case 0:  /* child process */
      {
         /* create command-line parameter
          * param 1: program name
                   * param 2: ....
          * param n: must be NULL
          */
         char *args[] = {PROGRAM_NAME_CLIENT_UDP, NULL};
         ret = execv(args[0], args);
         if ( ret == -1) {
            sprintf(errorMsg, "failure using execv %s", PROGRAM_NAME_CLIENT_UDP);
            error(errorMsg);
         }
         /* this section will never reached */
      }
         break;
      default: /* parent process */
         break;
   }

      /*
    * load - analyse
    */

   pid = fork();

   switch (pid) {
      case -1: /* error by using fork() */
         sprintf(errorMsg, "failure using fork %s", PROGRAM_NAME_ANALYSE);
         error(errorMsg);
         break;
      case 0:  /* child process */
      {
```

```
81          /* create command-line parameter
82           * param 1: program name
83           * param 2: ....
84           * param n: must be NULL
85           */
86          char *args[] = {PROGRAM_NAME_ANALYSE, NULL};
87          ret = execv(args[0], args);
88          if ( ret == -1) {
89             sprintf(errorMsg, "failure using execv %s", PROGRAM_NAME_ANALYSE);
90             error(errorMsg);
91          }
92          /* this section will never reached */
93       }
94          break;
95       default: /* parent process */
96          break;
97    }
98
99       /*
100    * load - serverUdp
101    */
102
103   pid = fork();
104
105   switch (pid) {
106      case -1: /* error by using fork() */
107         sprintf(errorMsg, "failure using fork %s", PROGRAM_NAME_SERVER_UDP);
108         error(errorMsg);
109         break;
110      case 0:  /* child process */
111      {
112         /* create command-line parameter
113          * param 1: program name
114          * param 2: ....
115          * param n: must be NULL
116          */
117         char *args[] = {PROGRAM_NAME_SERVER_UDP, NULL};
118         ret = execv(args[0], args);
119         if ( ret == -1) {
120            sprintf(errorMsg, "failure using execv %s", PROGRAM_NAME_SERVER_UDP);
121            error(errorMsg);
122         }
123         /* this section will never reached */
124      }
125         break;
126      default: /* parent process */
127         break;
128   }
129
130   for(;;);
131
132   return EXIT_SUCCESS;
133}
```

D.1.21. messageQueue.cpp

```
/*****************************************************************************
 * Description: messageQueue.cpp
 * Date-created: 20/07/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <mqueue.h>
#include <string.h>

#include "../inc/peripheral.h"
#include "../inc/globals.h"
#include "../inc/messageQueue.h"

/*
 * message queue
 * create message queue - server or client
 * directory: /dev/mqueue
 */

mqd_t mq_server_init(char* mq_name) {

   char errorMsg[256];

   mqd_t mq; /* wird per call by reference der funktion uebergeben */
   mq_attr mq_sa_attr;

   /*
    * message queue
    * create message queue - server
    * directory: /dev/mqueue
    */

   /* initialize the queue attributes */
   // maximum message size
   mq_sa_attr.mq_msgsize = MAX_SIZE_QUEUE;
   // maximum number of messages, max is 10
   mq_sa_attr.mq_maxmsg = 10;
   // message queue flags
   mq_sa_attr.mq_flags = 0;
   // number of messages currently in queue
   mq_sa_attr.mq_curmsgs = 0;

   /* create the message queue */
   mq = mq_open(mq_name, O_CREAT | O_RDONLY, MODE_MQ_SA, &mq_sa_attr);
   if (mq == -1) { sprintf(errorMsg, "failure creating mq_open %s", mq_name); error(errorMsg); }

   return mq;
}

mqd_t mq_client_init(char* mq_name) {

   char errorMsg[256];

   mqd_t mq;

   mq = mq_open(mq_name, O_WRONLY);
   if (mq == -1) { sprintf(errorMsg, "failure opening mq_open %s", mq_name); error(errorMsg); }

   return mq;
}
```

D.1.22. mySocket.cpp

```cpp
/*****************************************************************************
 * Description: mySocket.cpp
 * Date-created: 22/07/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 ****************************************************************************/

#include <sys/socket.h>
#include <arpa/inet.h>
#include <iostream>
#include <cstring>
#include <unistd.h>

#include "../inc/globals.h"
#include "../inc/mySocket.h"
#include "../inc/peripheral.h"

using namespace std;

void init_multicast_socket(int *pMySocket, struct sockaddr_in *pSockaddr, char *pMulticastGroup, int port) {

   struct ip_mreq group;
   int optval = 1;
   int res = 0;
   // set socket format ipv4
   (*pSockaddr).sin_family = AF_INET;
   // set socket port
   (*pSockaddr).sin_port = htons(port);
   // ip addr is accepted
   (*pSockaddr).sin_addr.s_addr = htons(INADDR_ANY);

   // create socket - ipv4, UDP
   *pMySocket = socket(AF_INET, SOCK_DGRAM, 0);
   if (*pMySocket < 0 ) { error("can't create sockets:"); close(*pMySocket); exit(EXIT_FAILURE); };

   /*
    * 1. parameter socket
    * 2. parameter verarbeitungslevel SOL_SOCKET = all sockets
    * 3. parameter options SO_REUSADDR = use local address
    * 4. parameter
    */
   res = setsockopt(*pMySocket, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));
   if (res < 0) { error("failure - SO_REUSEADDR: "); close(*pMySocket); exit(EXIT_FAILURE); };

   // binding to socket
   // bind the ip address and port to a socket
   res = bind(*pMySocket, (struct sockaddr *)&(*pSockaddr), sizeof((*pSockaddr)));
   if (res < 0) { error("can't bind socket: "); close(*pMySocket); exit(EXIT_FAILURE); };

   /*
    * join multicast group xxx.xxx.xxx.xxx on the local xxx.xxx.xxx.xxx interface
    * IP_ADD_MEMBERSHIP option must called for each local interface for receiveing multicast datagrams
    */

   // local interface
   group.imr_interface.s_addr = inet_addr(IP_INTERFACE_EXTERN);

   // multicast group
   group.imr_multiaddr.s_addr = inet_addr(pMulticastGroup);

   // IPPROTO_IP = level
   // IP_ADD_MEMBERSHIP = join multicast group
   res = setsockopt(*pMySocket, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *) &group, sizeof(group));
   if (res < 0) { error("failure - adding multicast group: "); close(*pMySocket); exit(EXIT_FAILURE); };
}

/*
 * function parameters:
 * 1. socket-descriptor -> call by reference
 * socket-descriptor to communication application <-> network-interface
 * 2. struct sockaddr_in -> call by reference
 * config format -> ipv4, ip-addr
 * 3. port number -> call by value
 * port number to bind application to a port
 * 4. ip_addr -> call by value
 * to select a interface ethx
 * 0: ALL INTERFACES
 * value: select Interfaces & network
 */
void init_udp_server_socket(int *mySocket, struct sockaddr_in *sockaddr, int port, char *ip_addr) {
```

```
81
82     int res;
83
84     // set socket format ipv4
85     (*sockaddr).sin_family = AF_INET;
86     // set socket port
87     (*sockaddr).sin_port = htons(port);
88     // ip addr is accepted
89     if (ip_addr == NULL) {
90        (*sockaddr).sin_addr.s_addr = htons(INADDR_ANY);
91     } else {
92        inet_pton(AF_INET, ip_addr, &(*sockaddr).sin_addr.s_addr);
93     }
94
95     // create socket - ipv4, UDP
96     *mySocket = socket(AF_INET, SOCK_DGRAM, 0);
97     if (*mySocket < 0 ) { error("can't create sockets:"); exit(EXIT_FAILURE); };
98
99     // htons -> host to networt host-byte-order return network-byte-order
100    // ntons -> network to host network-byte-order return host-byte-order
101
102    // binding to socket
103    // bind the ip address and port to a socket
104
105    res = bind(*mySocket, (struct sockaddr *)&(*sockaddr), sizeof((*sockaddr)));
106    if (res < 0) { error("can't bind socket: "); exit(EXIT_FAILURE); };
107
108 #if defined(DEBUG)
109    cout << "socket established: " << port << endl;
110 #endif
111 }
112
113 /*
114  * function parameters:
115  * 1. socket-descriptor -> call by reference
116  * socket-descriptor to communication application <-> network-interface
117  * 2. struct sockaddr_in -> call by reference
118  * config format -> ipv4, ip-addr
119  * 3. port number -> call by value
120  * port number to bind application to a port
121  * 4. ip_addr -> call by value
122  * to select a interface ethx
123  * >> destination addr
124  */
125 void init_udp_client_socket(int *mySocket, struct sockaddr_in *sockaddr, int port, char *ip_addr) {
126
127    int res;
128
129    // set socket format ipv4
130    (*sockaddr).sin_family = AF_INET;
131    // set socket port
132    (*sockaddr).sin_port = htons(port);
133    // ip addr is accepted
134    (*sockaddr).sin_addr.s_addr = htons(INADDR_ANY);
135
136    // create socket - ipv4, UDP
137    *mySocket = socket(AF_INET, SOCK_DGRAM, 0);
138    if (*mySocket < 0 ) { error("can't create sockets:"); exit(EXIT_FAILURE); };
139
140    // htons -> host to networt host-byte-order return network-byte-order
141    // ntons -> network to host network-byte-order return host-byte-order
142
143    // binding to socket
144    // bind the ip address and port to a socket
145
146    res = bind(*mySocket, (struct sockaddr *)&(*sockaddr), sizeof((*sockaddr)));
147    if (res < 0) { error("can't bind socket: "); exit(EXIT_FAILURE); };
148
149    // change ip-addr to destination-addr
150    inet_pton(AF_INET, ip_addr, &(*sockaddr).sin_addr.s_addr);
151
152 #if defined(DEBUG)
153    cout << "socket established: " << port << endl;
154 #endif
155 }
```

D.1.23. myTimer.cpp

```
/*****************************************************************************
 * Description: myTimer.cpp
 * Date-created: 29/08/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <signal.h>
#include <pthread.h>

#include "../inc/globals.h"
#include "../inc/peripheral.h"


int timer_init(void) {

   int i , evid;
   timer_t timerID;
   struct itimerspec timerSettings;
   struct sigevent timerEvent;
   printf ("timerSettings program will start with a delay of 1s...\n");

   // sigevent setting for thread
   timerEvent.sigev_notify = SIGEV_THREAD;
   timerEvent.sigev_value.sival_int = evid;
   timerEvent.sigev_notify_attributes = NULL;
   timerEvent.sigev_notify_function = timerThread;
   /*
    * timerSettings create
    * time base | sigevent struct | timer_id
    */
   if (timer_create(CLOCK_REALTIME, &timerEvent, &timerID) == -1) {
      printf ("Unable to attach timerSettings: %s\n", strerror(errno));
      return EXIT_FAILURE;
   }

   /*
    * timer parameter settings
    * it_value = inital parameter
    * it_interval = timer intervall
    * tv_sec = seconds
    * tv_nsec = nanoseconds
    */
   timerSettings.it_value.tv_sec = 10L;
   timerSettings.it_value.tv_nsec = 0L;
   timerSettings.it_interval.tv_sec = 10L;
   timerSettings.it_interval.tv_nsec = 0L;

   /*
    * timerSettings settime
    * timer_id | flags | timerSettings struct elapsed time | timerSettings struct remaining time
    */
   timer_settime(timerID, 0, &timerSettings, NULL);

}
```

D.1.24. NmeaLwe.cpp

```cpp
/*****************************************************************************
 * Description: NmeaSentence.cpp
 * Date-created: 29/07/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <iostream>
#include <string>
#include <sstream>
#include <cstring>

#include "../inc/NmeaLwe.h"
#include "../inc/nmeaLweDefines.h"

using namespace std;

// consructor
NmeaLwe::NmeaLwe(string nmeaData) {
   this->nmeaData = nmeaData;
}

// methoden

/*
 * check the header
 * the header is hard defined
 * 3 lvl-parser
 * 1. end of header
 * 2. length of header
 * 3. is header defined
 */
bool NmeaLwe::chkHeader() {
   bool validHeader = false;
   string tempHeader = "0";
   int i = 0;
   int startPosition = 0;
   int endPosition = 0;
   int readNmeaLweHeaderLength = 0;

   // 1. check where the header ends
   endPosition = nmeaData.find_first_of(0x5C);
   if (endPosition < 0) { cout << "failure - endPosition: " << endPosition << endl; return false; }

   // 2. check the headerlength
   readNmeaLweHeaderLength = endPosition - startPosition;
   if (!(readNmeaLweHeaderLength == NMEALWE_HEADERLENGTH)) { cout << "failure - headerlength: " <<
         readNmeaLweHeaderLength << endl; return false; }

   tempHeader = nmeaData.substr(0, endPosition-1);

   // 3. check if header is defined
   for (i=0; i< nmeaLweHeaderMaster->size(); i++) {
      if (nmeaLweHeaderMaster[i].compare(tempHeader) == 0) {
         validHeader = true;
         this->mNmeaHeader = true;
         this->mNmeaHeaderStr = nmeaLweHeaderMaster[i];
      }
   }

   if (!validHeader) { cout << "failure - invalid header: " << tempHeader << endl; }

   return validHeader;
}

/*
 * check the tag-blocks
 * 6 dif. tag-block possible max size 80 character
 * 1. g: group    - alphanumeric    e.g. 1-2-44 range 1-99
 * 2. s: source   - alphanumeric max. 15 e.g. HI0001
 * 3. d: destination - alphanumeric max. 15 e.g. SI0005
 * 4. n: line count -               e.g. positiv integer range 1-999
 * 5. t: text     - free text       e.g. free text
 * 6. c: time ??? - ????            e.g. ????
 *
 * check:
 * 1. start- endposition
 * 2. length
 *
 */
```

```
80  bool NmeaLwe::chkTag() {
81     int i;
82     bool validTag = false;
83     int startPosition;
84     int endPosition;
85     int readNmeaLweTagLength = 0;
86     char deliminer = ',';
87     string tempTagBlock;
88     vector <string> vectorTempTagBlock;
89     string tagBlock;
90     string tagBlockId;
91     string tagBlockField;
92     int tagBlockIdSet;
93
94     // 1. check start- endposition
95     // tag block begins with char '\' and ends with char '*'
96     startPosition = nmeaData.find_first_of(0x5C);
97     endPosition = nmeaData.find_first_of('*');
98     if ( (startPosition < 0) or (endPosition < 0) ) { cout << "failure - starPosition: " << startPosition << " or
           endPosition: " << endPosition << endl; return false; }
99
100    // 2. check length
101    readNmeaLweTagLength = endPosition - startPosition;
102    if ( (readNmeaLweTagLength > NMEALWE_TAGBLOCKMAXLENGTH) or (readNmeaLweTagLength < NMEALWE_TAGBLOCKMINLENGTH) )
           { cout << "failure - length: " << readNmeaLweTagLength << endl; return false; }
103
104    // start+1 -> without char '\'
105    tempTagBlock = nmeaData.substr((startPosition+1), endPosition-(startPosition+1));
106
107    // tag block contains
108    stringstream stringstreamTagBlock(tempTagBlock);
109
110    while( getline(stringstreamTagBlock, tagBlock, deliminer) ) {
111       tagBlockIdSet=0;
112
113       /*
114        * detection and parse
115        * min. 1x tagBlock
116        */
117
118       tagBlockId = tagBlock.substr(0, 2);
119       tagBlockField = tagBlock.substr(2, tagBlock.size());
120
121       // check if tagblockID is defined
122       int stringMasterSize = sizeof(nmeaLweTabBlockMaster)/ sizeof(nmeaLweTabBlockMaster[0]);
123       for (i=0; i < stringMasterSize; i++) {
124          if (nmeaLweTabBlockMaster[i].compare(tagBlockId) == 0) {
125             tagBlockIdSet = 1;
126          }
127       }
128
129       if ( tagBlockIdSet ) {
130          // check tagblock field
131                  // if tagblock id & field is okay
132          if (isTagBlockFormCorrect(tagBlockId, tagBlockField)) {
133             vectorTempTagBlock.push_back(tagBlock);
134          }
135
136       } else {
137          cout << "failure - tag block id: " << tagBlockId << endl;
138          return false;
139       }
140    }
141
142    // tag block entry > 0
143    if ( vectorTempTagBlock.size() == 0 ) { cout << "failure - no tagblock detectet: " << vectorTempTagBlock.size()
           << endl; return false; }
144
145    mNmeaTagBlockVectorStr = vectorTempTagBlock;
146    validTag = true;
147    mNmeaLwe = true;
148
149    return validTag;
150 }
151
152 bool NmeaLwe::isTagBlockFormCorrect(string tagBlockId, string tagBlockField) {
153
154    bool validTagBlockForm = false;
155
156    int i;
157    int checkNumeric;
158    int checkAlphabetic;
159    int checkSpecial;
```

```
160     int maxSize;
161     /*
162      * 1 = alphanumeric
163      * 2 = digits
164      * 3 = special form 1-2-3
165      */
166     int testUnit;
167
168     /*
169      * tagBlockId -> maxSize
170      */
171     if (tagBlockId.compare("d:")) {
172        maxSize = 15;
173        testUnit = 1;
174        checkNumeric = 1;
175        checkSpecial = 1;
176     } else if (tagBlockId.compare("s:")) {
177        maxSize = 15;
178        testUnit = 1;
179        checkNumeric = 1;
180        checkSpecial = 1;
181     } else if (tagBlockId.compare("g:")) {
182        maxSize = 11;
183        testUnit = 3;
184        checkNumeric = 1;
185        checkAlphabetic = 1;
186     } else if (tagBlockId.compare("t:")) { // max char 80 = 2 + 78
187        maxSize = 78;
188        testUnit = 1;
189        checkNumeric = 1;
190        checkSpecial = 1;
191     } else if (tagBlockId.compare("n:")) { // uint = 10 digits
192        maxSize = 10;
193        testUnit = 2;
194        checkAlphabetic = 1;
195        checkSpecial = 1;
196     } else { cout << "failure - tagBlockId: " << tagBlockId << endl; return false; }
197
198     // check max length
199     if ( tagBlockField.size() > maxSize ) { cout << "failure - tagBlockFlied too long : " << tagBlockField.size() <<
           endl; return false; }
200
201     for(i=0; i<tagBlockField.size(); i++){
202
203        switch(testUnit) {
204           case 1:
205              checkAlphabetic = isalnum(tagBlockField[i]);
206              break;
207           case 2:
208              checkNumeric = isdigit(tagBlockField[i]);
209              break;
210           case 3:
211              checkSpecial = isdigit(tagBlockField[i]) | strcmp(&tagBlockField[i], "-");
212              break;
213        }
214
215        if ( (checkNumeric | checkAlphabetic | checkSpecial) == 0 ) { cout << "failure - tagBlockFlied: " <<
              tagBlockField << ", position: " << (i+1) << endl; return false; }
216     }
217
218     validTagBlockForm = true;
219
220     return validTagBlockForm;
221 }
222
223
224 bool NmeaLwe::chkTagChecksum() {
225     bool validChecksum = false;
226     char code = 0x00;
227     string message;
228     string checksumData;
229     char checksumCalc[3];
230     int startPosition;
231
232     message = getNmeaTagBlock();
233
234     for (char &stringElement : message) {
235        code ^= stringElement;
236     }
237
238     if (code < 0x10) {
239        sprintf(checksumCalc, "0%X",code);
240     } else {
```

```
241         sprintf(checksumCalc, "%X",code);
242      }
243
244         /*********************************************
245          * checksum vom inputstream selektieren und vergleichen -> fertig
246          *********************************************/
247      startPosition = nmeaData.find_first_of('*');
248      checksumData = nmeaData.substr((startPosition+1), NMEALWE_CHECKSUMLENGTH);
249
250      if (checksumCalc == checksumData) {
251         validChecksum = true;
252         mNmeaTagChecksumStr = checksumCalc;
253         mNmeaTagChecksum = true;
254      } else {
255         cout << "failure - checksum verify checksumData " << checksumData << " checksumCalc: " << checksumCalc <<
                endl; return false;
256      }
257
258      return validChecksum;
259   }
260
261   bool NmeaLwe::isValidLwe() {
262      mNmeaValidLwe = mNmeaHeader and mNmeaLwe and mNmeaTagChecksum;
263      return mNmeaValidLwe;
264   }
265
266   string NmeaLwe::getNmeaHeader() {
267      return mNmeaHeaderStr;
268   }
269
270   string NmeaLwe::getNmeaTagBlock() {
271      string nmeaTagBlockStr;
272
273      vector <string> ::iterator i;
274
275      for(i = mNmeaTagBlockVectorStr.begin(); i != mNmeaTagBlockVectorStr.end(); i++) {
276         if (!(i == mNmeaTagBlockVectorStr.begin())) {
277            nmeaTagBlockStr += ",";
278         }
279         nmeaTagBlockStr += *i;
280      }
281
282      mnmeaTagBlockStr = nmeaTagBlockStr;
283
284      return nmeaTagBlockStr;
285   }
286
287   string NmeaLwe::getNmeaTagChecksum() {
288      return mNmeaTagChecksumStr;
289   }
290
291   string NmeaLwe::getNmeaLwe() {
292      string nmeaLwe;
293      nmeaLwe = mNmeaHeaderStr + char (0x5C) + mnmeaTagBlockStr + char (0x2A) + mNmeaTagChecksumStr;
294
295      return nmeaLwe;
296   }
```

D.1.25. NmeaSentence.cpp

```
/*****************************************************************************
 * Description: NmeaSentence.cpp
 * Date-created: 20/08/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 ****************************************************************************/

#include <sstream>
#include "../inc/NmeaSentence.h"
#include "../inc/nmeaSentenceDefines.h"

// consructor
NmeaSentence::NmeaSentence(string inputData) {
   this->inputData = inputData;
}

bool NmeaSentence::chkLength() {
   int length = inputData.end() - inputData.begin();

   if ( (length > NMEASENTENCE_MAXLENGTH) or ( length == 0 ) ) { cout << "failure - sentenceLength: " << length <<
         endl; return false; }

   mNmeaSentenceLengthCount = length;
   mNmeaSentenceLength = true;

   return true;
}

int NmeaSentence::getLength() {
   return mNmeaSentenceLengthCount;
}

bool NmeaSentence::chkAddress() {

   bool validAddress = false;
   int endPositionAddress = 0;
   int i = 0;
   int checkAlphabetic = 0;
   string sentenceAddress = "0";
   int addressRange = 0;
   string talkerIdentifier = "0";
   string sentenceFormatter = "0";
   int talkerIdSet = 0;
   int sentenceFormatterSet = 0;
   int startPositionAddress = 1;

   if(inputData[0] == 0x24) { // start Sentence

      endPositionAddress = inputData.find_first_of(0x2C);
      addressRange = endPositionAddress - startPositionAddress;

      if (addressRange == 5) {
         sentenceAddress = inputData.substr(startPositionAddress, endPositionAddress-1);

         for(i=0 ; i < addressRange; i++) {
            checkAlphabetic = isalpha(sentenceAddress[i]);
            if (!checkAlphabetic) { cout << "failure - sentenceAddress: " << sentenceAddress << endl; return false;
                  }
         }

         talkerIdentifier = sentenceAddress.substr(0,2);
         sentenceFormatter = sentenceAddress.substr(2,4);

         int stringTalkerIdentifierSize = sizeof(nmeaTalkterIdentifier)/ sizeof(nmeaTalkterIdentifier[0]);
         int stringSentenceFormatterSize = sizeof(nmeaSentenceFormatter)/ sizeof(nmeaSentenceFormatter[0]);

         for(i=0; i<stringTalkerIdentifierSize; i++) {
            if(nmeaTalkterIdentifier[i].compare(talkerIdentifier) == 0) {
               talkerIdSet = 1;
            }
         }

         for(i=0; i<stringSentenceFormatterSize; i++) {
            if(nmeaSentenceFormatter[i].compare(sentenceFormatter) == 0) {
               sentenceFormatterSet = 1;
            }
         }

         if ( (talkerIdSet == 0) || (sentenceFormatterSet == 0)) {cout << "failure - talkerID or SentenceFormatter:
               " << sentenceAddress << endl; return false;}
```

```
78
79          mNmeaSentenceAddressStr = sentenceAddress;
80          validAddress = true;
81          mNmeaSentenceAddress = true;
82       }
83    }
84
85    return validAddress;
86 }
87
88 string NmeaSentence::getAddress() {
89    return mNmeaSentenceAddressStr;
90 }
91
92 bool NmeaSentence::chkDataField() {
93    char deliminer = ',';
94    string dataField = "0";
95    bool validDataField = false;
96    string tempDataField;
97    int startPosition;
98    int endPosition;
99    int dataFieldCount = 0;
100   int dataFieldSet = 0;
101   vector <string> dataFieldVector;
102
103   startPosition = inputData.find_first_of(",");
104   endPosition = inputData.find_first_of("*");
105
106   tempDataField = inputData.substr(startPosition+1, endPosition-startPosition-1);
107   stringstream stringstreamDataField(tempDataField);
108
109
110   // selektiere die einzelnen Datenfelder
111   while( getline(stringstreamDataField, dataField, deliminer) ) {
112      dataFieldCount++;
113      dataFieldSet = 0;
114
115      if (isDataFieldCorrect(dataField)) {
116         dataFieldVector.push_back(dataField);
117         dataFieldSet = 1;
118      }
119
120      if (dataFieldSet) {
121         validDataField = true;
122         // weitere Prüfung
123      } else {
124         cout << "failure - data field: " << dataField << endl;
125         return false;
126      }
127
128   }
129
130   if (validDataField) {
131      mNmeaSentenceData = true;
132   }
133
134   mNmeaSentenceDataVector = dataFieldVector;
135
136   return validDataField;
137 }
138
139 string NmeaSentence::getDataField() {
140    string nmeaDataField;
141
142    vector <string> ::iterator i;
143
144    for(i = mNmeaSentenceDataVector.begin(); i != mNmeaSentenceDataVector.end(); i++) {
145       if (!(i == mNmeaSentenceDataVector.begin())) {
146          nmeaDataField += ",";
147       }
148       nmeaDataField += *i;
149    }
150
151    mNmeaSentenceDataStr = nmeaDataField;
152
153    return mNmeaSentenceDataStr;
154 }
155
156 bool NmeaSentence::isDataFieldCorrect(string dataField) {
157    bool validDataField = false;
158    bool alnum = false;
159    bool dot = false;
160    bool result = false;
```

```
161     int i = 0;
162
163     if (dataField.length() == 0) {
164        validDataField = true;
165     } else {
166        for (i = 0; i < dataField.length(); i++) {
167           bool alnum = false;
168           bool dot = false;
169           bool result = false;
170
171           alnum = isalnum(dataField[i]);
172           dot = dataField[i] == '.';
173
174           result = alnum | dot;
175
176           if (result) {
177              validDataField = true;
178           } else {
179              validDataField = false;
180              return validDataField;
181           }
182
183        }
184     }
185
186     return validDataField;
187  }
188
189  bool NmeaSentence::chkChecksum() {
190     int startPosition = 0;
191     bool validChecksum = false;
192     string message = "0";
193     char code = 0x00;
194     char checksumCalc[3];
195     string checksumData = "0";
196
197     message = getAddress() + "," + getDataField();
198
199     for (char &stringElement : message) {
200        code ^= stringElement;
201     }
202
203     if (code < 0x10) {
204        sprintf(checksumCalc, "0%X",code);
205     } else {
206        sprintf(checksumCalc, "%X",code);
207     }
208
209     startPosition = inputData.find_first_of('*');
210     checksumData = inputData.substr((startPosition+1), NMEASENTENCE_CHECKSUMLENGTH);
211
212     if (checksumCalc == checksumData) {
213        validChecksum = true;
214        mNmeaSentenceChecksumStr = checksumCalc;
215        mNmeaSentenceChecksum = true;
216     } else {
217        cout << "failure - checksum verify checksumData " << checksumData << " checksumCalc: " << checksumCalc <<
               endl; return false;
218     }
219
220     return validChecksum;
221  }
222
223  string NmeaSentence::getChecksum() {
224     return mNmeaSentenceChecksumStr;
225  }
226
227  bool NmeaSentence::isValidSentence() {
228     mNmeaValidSentence = mNmeaSentenceLength and mNmeaSentenceAddress and mNmeaSentenceData and
           mNmeaSentenceChecksum;
229     return mNmeaValidSentence;
230  }
231
232  string NmeaSentence::getSentence() {
233     string nmeaSentence;
234     nmeaSentence = "$" + mNmeaSentenceAddressStr + "," + mNmeaSentenceDataStr + "*" + mNmeaSentenceChecksumStr + "<
           CR><LF>";
235     return nmeaSentence;
236  }
```

D.1.26. peripheral.cpp

```
/*****************************************************************************
 * Description: peripheral.cpp
 * Date-created: 16/07/18
 * Date-Updated: 02/09/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <fstream>
#include <iostream>
#include <string>
#include <regex>
#include <sstream>

#include "../inc/globals.h"

#include <sys/wait.h>
#include <sys/time.h>

using namespace std;

string getTimestamp() {

   struct timeval timestamp;
   stringstream timestampStr;
   gettimeofday(&timestamp, 0);
   timestampStr << timestamp.tv_sec << "." << timestamp.tv_usec;

   return timestampStr.str();
}

/*********************************************
 * Programm zur Ermittelung der MAC-Adresse zu einer IP-Adresse
 * Als Datenquelle wird der Cache des ARP-Befehls ausgewertet
 *********************************************/
int getMacAddress(string &ipAddress, string &macAddress) {

   vector <string> :: iterator i;
   int position=0;
   int result = EXIT_FAILURE;

   vector <string> ipBuffer;
   vector <string> macBuffer;
   vector <string> interfaceBuffer;

   fstream f;
   string zeile;

   string ip, tmp, mac, interface;

      /*********************************************
       * START: Daten aufbereiten
       *********************************************/
   f.open("/proc/net/arp", ios::in);

   if (!f.eof()) {
      getline(f, zeile);

      while (getline(f, zeile)) {

         // Datenstrom aus einer Zeichenkette
         stringstream zeilenbuffer(zeile);

         zeilenbuffer >> ip >> tmp >> tmp >> mac >> tmp >> interface;

         ipBuffer.push_back(ip);
         macBuffer.push_back(mac);
         interfaceBuffer.push_back(interface);
         i++;

      }
   }

      /*********************************************
       * START: IP-Adresse MAC-Adresse zuweisen
       *********************************************/
   for (i=ipBuffer.begin() ; i!=ipBuffer.end(); i++ ) {

      // Position des Iterator im Schleifendurchlauf
      position = distance(ipBuffer.begin(), i);

```

```
81          if ( (*i == ipAddress) && (interfaceBuffer[position] == ETH_INTERFACE_SENSOR) ) {
82
83             macAddress = macBuffer[position];
84
85          }
86       }
87
88          /*********************************************
89           * Check auf gleiche Vektorlänge ip zu mac
90           *********************************************/
91       if (ipBuffer.size() == macBuffer.size()) {
92          result = EXIT_SUCCESS;
93       } else {
94          result = EXIT_FAILURE;
95       }
96
97       // Stream wird geschlossen
98       f.close();
99
100      return result;
101   }
102
103   void proc_exit(int signr) {
104      pid_t pid;
105      int status;
106
107      pid = waitpid(-1, &status, WNOHANG);
108
109      while (pid > 0) {
110         if (!(WIFEXITED(status))) { printf("Child wurde nicht normal beendet!\n"); }
111         return;
112      }
113   }
114
115   /*********************************************
116    * Funktion zur Fehlerausgabe!
117    *********************************************/
118
119   void error(const char *msg) {
120      perror(msg);
121      exit(EXIT_FAILURE);
122   }
123
124   void timerThread(union sigval arg) {
125      minuteFlag = true;
126   }
```

D.1.27. Sensor.cpp

```
/*****************************************************************************
 * Description: Sensor.cpp
 * Date-created: 16/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <sstream>
#include <vector>
#include <iostream>
#include <cmath>

#include "../inc/Sensor.h"

// constructor

Sensor::Sensor() {}

Sensor::Sensor(string data) {
   this->parameterdata = data;
}

// methodes
string Sensor::getRawParameter() {
   return parameterdata;
}

string Sensor::getParameter() {
   string parameter;
   parameter = "IP-Address:" + mIPAddress + ";" + "MAC-Address:" + mMacAddress + ";" + "Talker-ID:" + mTalkerId + "
         ;" + "Sentence-Formatter:" + mFormatter;
   return parameter;
}

string Sensor::getFormatter() {
   return mFormatter;
}

void Sensor::loadParameters() {

   stringstream ss(parameterdata);
   vector <string> cont;
   string token;

   while(getline(ss, token, ';')) {
      cont.push_back(token);
   }

   if (cont.size() == 4) {
      mIPAddress = cont[0];
      mMacAddress = cont[1];
      mTalkerId = cont[2];
      mFormatter = cont[3];
   }

}

void Sensor::setLineCount(unsigned int lineCount) {
   this->mLineCount = lineCount;
}

void Sensor::setSensorInputData(string inputData) {

}

string Sensor::getIPAddress() {
   return mIPAddress;
}

string Sensor::getMacAddress() {
   return mMacAddress;
}

bool Sensor::chkSensorParameter(string ip, string mac, string talkerId, string formatter) {
   bool valid = false;

   if (ip.compare(mIPAddress) == 0) {
      if (mac.compare(mMacAddress) == 0) {
         if (talkerId.compare(mTalkerId) == 0) {
            if (formatter.compare(mFormatter) == 0) {
```

```
80                valid = true;
81                cout << "MATCH" << endl;
82             }
83          }
84       }
85    }
86
87    return valid;
88 }
89
90 unsigned int Sensor::getLineCount() {
91    return mLineCount;
92 }
93
94 double Sensor::getTimeDiffHeader() {
95    return mTimeDiffHeader;
96 }
97
98 double Sensor::getTimeDiffInData() {
99    return mTimeDiffInData;
100 }
101
102 double Sensor::getTimestampHeader() {
103    return mTimestampHeader;
104 }
105
106 double Sensor::getTimestampincommingDate() {
107    return mTimestampIncommingData;
108 }
109
110 void Sensor::setTimestampHeader(double timestamp) {
111    mTimestampHeader = timestamp;
112 }
113
114 void Sensor::setTimestampIncommingData(double timestamp) {
115    mTimestampIncommingData = timestamp;
116 }
117
118 void Sensor::setTimeDiffInData(double diff) {
119    mTimeDiffInData = diff;
120 }
121
122 void Sensor::setTimeDiffHeader(double diff) {
123    mTimeDiffHeader = diff;
124 }
125
126 bool Sensor::chkDiff() {
127    bool valid = false;
128    double diff=0;
129    double timeHysteresis = 0.2000; // Hysteresis
130
131    if ((mTimeDiffHeader > 0) and (mTimeDiffInData > 0) ) {
132       diff = abs(mTimeDiffHeader - mTimeDiffInData);
133       if (diff < timeHysteresis) {
134          valid = true;
135          msignalDelay = diff;
136       } else {
137                  // Debug
138       }
139    } else {
140             // Debug
141    };
142
143    return valid;
144 }
145
146 void Sensor::setDataPayload(string payload) {
147    mDataPayload = payload;
148 }
149
150 string Sensor::getDataPayload() {
151    return mDataPayload;
152 }
153
154 void Sensor::loadData() {
155
156    string payload = mDataPayload;
157    stringstream ss1, ss2;
158    ss1.str(payload);
159    string value1 = "0";
160    string value2 = "0";
161    string value3 = "0";
162    string value4 = "0";
```

```
163     string temp;
164
165     while(getline(ss1, temp, ',')) {
166        ss2 << temp << " ";
167     }
168
169     if (mFormatter == "GLL") { // GPS [lat | lon]
170
171        ss2 >> value1 >> temp >> value2 >> temp >> value3;
172
173        mGpGllLat = stof(value1);
174        mGpGllLon = stof(value2);
175        mGpGllTime = stof(value3);
176
177     } else if (mFormatter == "HDT") { // Heading [degree]
178
179        ss2 >> value1;
180        mHeHdtDegree = stof(value1);
181
182     } else if (mFormatter == "VBW") { // Speed [knots]
183
184        ss2 >> value1 >> value2 >> temp >> value3 >> value4;
185
186        mLonWaterSpeed = stof(value1);
187        mTransWaterSpeed = stof(value2);
188        mLonGroundSpeed = stof(value3);
189        mTransGroundSpeed = stof(value4);
190     }
191  }
192
193  void Sensor::setDataSize(int bytes) {
194     mDataSize = bytes;
195  }
196
197  int Sensor::getDataSize() {
198     return mDataSize;
199  }
200
201  double Sensor::getGllLat() {
202     return mGpGllLat;
203  }
204  double Sensor::getGllLon() {
205     return mGpGllLon;
206  }
207  double Sensor::getHdtDegree() {
208     return mHeHdtDegree;
209  }
210  double Sensor::getVBWLonWaterSpeed() {
211     return mLonWaterSpeed;
212  }
213  double Sensor::getVBWTransWaterSpeed() {
214     return mTransWaterSpeed;
215  }
```

D.1.28. SensorFusion.cpp

```cpp
/*****************************************************************************
 * Description: SensorFusion.cpp
 * Date-created: 16/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 *****************************************************************************/

#include <iostream>
#include <math.h>
#include <fstream>
#include "../inc/SensorFusion.h"
#include "../inc/globals.h"

using namespace std;

// constructor
SensorFusion::SensorFusion() {
}

void SensorFusion::loadDataGll(double time, double lat, double lon) {
   double tm_lat, tm_lon;

   tm_lat = gllToDezimal(lat);
   mLatNew = toRadians(tm_lat);

   tm_lon = gllToDezimal(lon);
   mLonNew = toRadians(tm_lon);

   mGllTimeNew = time;
}

void SensorFusion::loadDataHDT(double time, double degree) {
   mBearingNew = toRadians(degree);
   mHdtTimeNew = time;
}

void SensorFusion::loadDataVBW(double time, double speedLon, double speedTrans) {
   mLonWaterSpeedNew = knotsToMeters(speedLon);
   mTransWaterSpeedNew = knotsToMeters(speedTrans);
   mVbwTimeNew = time;
}

bool SensorFusion::checkGllParameters() {
   bool valid = false;
   mGllValid = false;

   if ( (mLatNew/100) <= 90.0 ) { // Skallierung
      if ( (mLonNew/100) <= 180.0 ) { // Skallierung
         valid = true;
         mGllValid = true;
      }
   }

   return valid;
}

bool SensorFusion::checkVbwParameters() {
   bool valid = false;
   mVbwValid = false;

   if ( mLonWaterSpeedNew < 15.0) {
      if (mTransWaterSpeedNew < 2.5) {
         valid = true;
         mVbwValid = true;
      }
   }
   return valid;
}

bool SensorFusion::checkHdtParameters() {
   bool valid = false;
   mHdtValid = false;

   if ( mBearingNew < 360.0 ) {
      valid = true;
      mHdtValid = true;
   }

   return valid;
}
```

```
81
82  void SensorFusion::fusion() {
83
84      double calcTime = 60.0;
85      double gpsDistance = 0.0;
86      double gpsTimeDiff = 0.0;
87      double gpsAverageSpeed = 0.0;
88      double latCalcNew = 0.0;
89      double lonCalcNew = 0.0;
90      double calcDistance = 0.0;
91      double calcDistanceDiff = 0.0;
92      double percent = 0.0;
93
94      mFusionValid = true;
95
96      /*********************************************
97       * Sensorwerte sind Valid
98       *********************************************/
99      if (mVbwValid and mHdtValid and mGllValid) {
100
101         /*********************************************
102          * Berechnung ist aktiv
103          *********************************************/
104         if (mFusionAktiv) {
105
106            if (mVbwTimeStore != mVbwTimeNew) {
107
108               /*********************************************
109                * Berechnung der Entfernung
110                *********************************************/
111               mDistanceCalc = mLonWaterSpeedStore * (mVbwTimeNew - mVbwTimeStore);
112
113               mDistanceCalcTotoal += mDistanceCalc;
114               mTimeWaterSpeedDiff += (mVbwTimeNew - mVbwTimeStore);
115
116               /*********************************************
117                * Berechnung der Position
118                *********************************************/
119               getGpsPosition(mLatCalc, mLonCalc, &latCalcNew, &lonCalcNew, mDistanceCalc, mBearingStore);
120
121               mLatCalc = latCalcNew;
122               mLonCalc = lonCalcNew;
123
124               // speichern der neuen Werte
125               mLonWaterSpeedStore = mLonWaterSpeedNew;
126               mVbwTimeStore = mVbwTimeNew;
127               mBearingStore = mBearingNew;
128            }
129
130            /*********************************************
131             * Berechnung innerhalb des Zeitintervalls
132             *********************************************/
133            if ( (mGllTimeNew - mGllTimeStore ) < calcTime) {
134
135               /*********************************************
136                * Bestimmen des Ergebnisses
137                *********************************************/
138            } else {
139
140               /*********************************************
141                * Berechnung der Parameter aus zwei GPS Messwerten
142                * 1. Entfernung [m]
143                * 2. Zeitdifferenz [s]
144                * 3. Gemittelte Geschwindigkeit [m/s]
145                *********************************************/
146               gpsDistance = distanceGps(mLatStore, mLonStore, mLatNew, mLonNew);
147               gpsTimeDiff = timeDiffGps(mGllTimeStore, mGllTimeNew);
148               gpsAverageSpeed = gpsDistance / gpsTimeDiff;
149
150               /*********************************************
151               * Berechnung der Parameter aus start GPS Position
152               * 2. Entfernung [m]
153               * 3. Gemittelte Geschwindigkeit [m/s]
154               *********************************************/
155               calcDistance = distanceGps(mLatStore, mLonStore, mLatCalc, mLonCalc);
156               calcDistanceDiff = distanceGps(mLatNew, mLonNew, mLatCalc, mLonCalc);
157
158               /*********************************************
159                * Vergleich Ergebnisse - Messwerte zu Berechnung
160                *********************************************/
161               ofstream file;
162
163               file.open("/home/pi/secureServer/auswertung/sensorfusion", ofstream::in | ofstream::app);
```

```
164             if(!file.is_open()) { cout << "failure - open file: AUSWERTUNG" << endl ; exit(EXIT_FAILURE); }
165             file << mLatStore << ";" << mLonStore << ";" << mLatCalc << ";" << mLonCalc << ";" << mLatCalc << ";"
                      << mLonCalc << ";" << gpsDistance << ";" << calcDistance << ";" << calcDistanceDiff << ";" <<
                      mDistanceCalcTotoal << ";" << gpsTimeDiff << ";" << mTimeWaterSpeedDiff << ";" << endl;
166             file.close();
167
168             percent = (( calcDistance * calcDistanceDiff ) / 100 );
169
170             if ( percent > 10 ) {
171                printf("Prozentuale Abweichung: %lf\n", percent);
172                mFusionValid = false;
173             } else {
174                mFusionValid = true;
175             }
176
177             mDistanceCalc = 0.0;
178             mTimeWaterSpeedDiff = 0.0;
179             mDistanceCalcTotoal = 0.0;
180             mFusionAktiv = false;
181          }
182
183          /*********************************************
184           * Berechnung ist nicht aktiv
185           * 1. setzen der Vergleichwerte
186           * 2. setze status -> starte Berechnung
187           *********************************************/
188       } else {
189          // set GPS data
190          mLatStore = mLatNew;
191          mLonStore = mLonNew;
192          mGllTimeStore = mGllTimeNew;
193
194          mLatCalc = mLatNew;
195          mLonCalc = mLonNew;
196
197          // set other sensor values
198          mLonWaterSpeedStore = mLonWaterSpeedNew;
199          mVbwTimeStore = mVbwTimeNew;
200          mBearingStore = mBearingNew;
201          mHdtTimeStore = mHdtTimeNew;
202
203          // set calc
204          mFusionAktiv = true;
205       }
206
207       /*********************************************
208        * Fehlerhafter Sensorwert
209        * Ruecksetzen der Berechnung
210        *********************************************/
211    } else {
212
213       mFusionAktiv = false;
214       mFusionValid = false;
215    }
216 }
217
218 bool SensorFusion::isFusionValid() {
219    return mFusionValid;
220 }
221
222 void SensorFusion::getGpsPosition(double latS, double lonS, double *latD, double *lonD, double distance, double
       bearing) {
223    double R, theta, Delta, deltaPhi, phi1, phi2, deltaPsi, q, deltaLambda, lambda1, lambda2;
224
225    R = 6371 * 1e3;
226    phi1 = latS;
227    lambda1 = lonS;
228    theta = bearing;
229
230    Delta = distance/R;
231    deltaPhi = Delta * cos(theta);
232    phi2 = phi1 + deltaPhi;
233
234    deltaPsi = log( tan(M_PI/4 + phi2/2) / tan(M_PI/4 + phi1/2) );
235    q = deltaPhi / deltaPsi;
236    deltaLambda = (Delta * sin(theta)) / q;
237    lambda2 = lambda1 + deltaLambda;
238
239    *latD = phi2;
240    *lonD = lambda2;
241 }
242
243 double SensorFusion::toRadians(double degree) {
```

```
244     double radian = 0.0;
245 
246     radian = (( 2*M_PI / 360 ) * degree);
247     return radian;
248 }
249 
250 double SensorFusion::gllToDezimal(double gllDegree) {
251     double dezimalDegree = 0.0;
252     double angleDegree = 0.0;
253     double minute = 0.0;
254     double second = 0.0;
255     double minuteDegree = 0.0;
256     double secondDegree = 0.0;
257 
258     angleDegree = (int) (gllDegree/100);
259     minute = (gllDegree - (angleDegree*100));
260     minuteDegree = minute / 60;
261 
262     dezimalDegree = angleDegree + minuteDegree; //+ secondDegree;
263 
264     return dezimalDegree;
265 }
266 
267 double SensorFusion::knotsToMeters(double knots) {
268     double faktorKnots = (1.852) * (1000.0/3600.0);
269     double meters = knots * faktorKnots;
270     return meters;
271 }
272 
273 double SensorFusion::metersToKnots(double meters) {
274     double faktorMeters = 3600 / 1852;
275     double knots = 0;
276     knots = meters * faktorMeters;
277 
278     return knots;
279 }
280 
281 double SensorFusion::distanceGps(double lat1, double lon1, double lat2, double lon2) {
282     double R; // radius
283     double a;
284     double c;
285     double d; // distance
286 
287     double phi1;
288     double phi2;
289     double deltaLambda;
290     double deltaPhi;
291 
292     R = 6371 * 1e3; // radius der Erde [m]
293 
294     phi1 = lat1;
295     phi2 = lat2;
296 
297     deltaPhi = lat2-lat1;
298     deltaLambda = lon2-lon1;
299 
300     a = sin(deltaPhi/2) * sin(deltaPhi/2) + cos(phi1) * cos(phi2) * sin(deltaLambda/2) * sin(deltaLambda/2);
301     c = 2 * atan2(sqrt(a), sqrt(1-a));
302     d = R * c;
303 
304     return d;
305 }
306 
307 double SensorFusion::timeDiffGps(double time1, double time2) {
308     return time2 - time1;
309 }
```

D.1.29. serverUdp.cpp

```
/*****************************************************************************
 * Description: serverUdp.cpp
 * Date-created: 16/07/18
 * Date-Updated: 02/10/2018
 * Author:     Nils Parche
 ****************************************************************************/

#include <iostream>
#include <netinet/in.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <cstring>
#include <string>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <sys/wait.h>
#include <net/if.h>
#include <vector>
#include <algorithm>
#include <mqueue.h>
#include <sys/time.h>

/*********************************************
 * Eigene Include-Dateien
 ********************************************/
#include "../inc/peripheral.h"
#include "../inc/globals.h"
#include "../inc/messageQueue.h"
#include "../inc/mySocket.h"
#include "../inc/NmeaLwe.h"
#include "../inc/NmeaSentence.h"

using namespace std;

int main(void) {

   /*********************************************
    * VARIABLEN
    ********************************************/
   pid_t pid;
   int pid_status;

   // Variables - Connection
   int serverSocket, clientSocket;

   char dataBufferMQ[MAX_SIZE_QUEUE];

   // Variables - Connection Parameter
   string ipClientStr;
   string macClientStr;
   char ipClientChar[256];

   string timestampIncommingDataStr;

   mqd_t mq_sa;

   // communication - param
   char inputBufferSocket[UDP_BUFFER_SIZE];
   int recBytes;
   int sendNumbers;

   // socket
   struct sockaddr_in server_addr, client_addr;
   socklen_t client_addr_size = sizeof(client_addr);

      /*********************************************
    * open message queue client
    ********************************************/
   // delay time, spend time to create mq in process analyse
   sleep(1);
   mq_sa = mq_client_init(MQ_NAME_SERVER_UDP_ANALYSE);

      /*********************************************
    * signal handler exiting child process
    ********************************************/
   signal(SIGCHLD, proc_exit);

      /*********************************************
    * create multicast UPD socket
```

```
81      ********************************************/
82    init_multicast_socket(&serverSocket, &server_addr, MULTICAST_GROUP_NAVD, PORTNUM_MULTICAST_NAVD);
83
84    while(true) {
85
86       // clear buffer
87       memset(inputBufferSocket, 0, sizeof(inputBufferSocket));
88
89       recBytes = recvfrom(serverSocket, inputBufferSocket, UDP_BUFFER_SIZE, 0, (struct sockaddr *)&client_addr, &
               client_addr_size);
90
91       pid = fork();
92
93       switch(pid) {
94
95          case -1: /* failure using fork() */
96
97             break;
98          case 0:  /* child process */
99          {
100            // close sockets
101            close(serverSocket);
102
103            int i=0;
104            string inputBufferSocketStr;
105            string nmeaFrameStr;
106            bool validData = false;
107
108                        // create timestamp
109            timestampIncommingDataStr = getTimestamp();
110            char *timestampIncommingDataChar = new char[timestampIncommingDataStr.length()+1];
111            strcpy(timestampIncommingDataChar, timestampIncommingDataStr.c_str());
112
113            // convert IP-addr into char array
114            inet_ntop(AF_INET, &client_addr.sin_addr.s_addr, ipClientChar, sizeof(ipClientChar));
115            // store char into string
116            ipClientStr = ipClientChar;
117
118            // detect MAC-addr for incoming ip connection
119            // 1 = failure, 0 = Successful
120            if (getMacAddress(ipClientStr, macClientStr)) error("failure by detecting MAC-addr");
121            char *macClientChar = new char[macClientStr.length()+1];
122            strcpy(macClientChar, macClientStr.c_str());
123
124                        /*********************************************
125                         * Funktion detekt input Data parse to NMEA Formattern.
126                         * Wenn daten vollständig gecheckt übergebe daten an zweiten Prozess zur Analyse/
                                Archivierung
127                         *********************************************/
128            // pharseDataToNMEAFormatter
129            // convertieren into string
130
131            for(i=0; i<recBytes;i++) {
132               inputBufferSocketStr.push_back(inputBufferSocket[i]);
133            }
134
135                        /*********************************************
136                         * Zerlegen des NMEA-Datentelegramms
137                         *********************************************/
138            int deliminerLweSentence = inputBufferSocketStr.find_last_of(0x5C);
139            string lweStr = inputBufferSocketStr.substr(0, deliminerLweSentence+1);
140            string nmeaSentence = inputBufferSocketStr.substr(deliminerLweSentence+1);
141
142            // class -> parse LWE-Header
143            NmeaLwe sensorHeader(lweStr);
144
145            //class -> parse NMEA-Sentence
146            NmeaSentence sensorData(nmeaSentence);
147
148            sensorHeader.chkHeader();
149            sensorHeader.chkTag();
150            sensorHeader.chkTagChecksum();
151
152            if (sensorHeader.isValidLwe()) {
153
154               sensorData.chkLength();
155               sensorData.chkAddress();
156               sensorData.chkDataField();
157               sensorData.chkChecksum();
158
159               if (sensorData.isValidSentence()) {
160                  // transferDataToValidationProcess
161                  nmeaFrameStr = sensorHeader.getNmeaLwe() + char (0x5C) + sensorData.getSentence();
```

```
162                     validData = true;
163                 } else {
164                     cout << "failure - is not correct format for sentence" << endl;
165                 }
166             } else {
167                 cout << "failure - is not correct format for header" << endl;
168             }
169 
170             string recBytesStr = to_string(recBytes);
171             char *recBytesChar = new char[recBytesStr.length()+1];
172             strcpy(recBytesChar, recBytesStr.c_str());
173 
174             if (validData) {
175                 char *nmeaFrameChar = new char[nmeaFrameStr.length()+1];
176                 strcpy(nmeaFrameChar, nmeaFrameStr.c_str());
177                 sprintf(dataBufferMQ, "%s;%s;%d;%s;%s;%s", ipClientChar, macClientChar, getpid(),
                        timestampIncommingDataChar, recBytesChar, nmeaFrameChar);
178             } else {
179                 sprintf(dataBufferMQ, "%s;%s;%d;%s;%s;%s", ipClientChar, macClientChar, getpid(),
                        timestampIncommingDataChar, recBytesChar, "failure");
180             }
181 
182             mq_send(mq_sa, dataBufferMQ, strlen(dataBufferMQ), 0);
183 
184             exit(EXIT_SUCCESS);
185             break;
186         }
187         default: /* Elternprozess */
188 
189             break;
190     }
191 
192     // send message back
193     sendNumbers = sendto(serverSocket, inputBufferSocket, recBytes, 0, (struct sockaddr *)&client_addr, sizeof(
            client_addr));
194 
195 }
196 
197 // unmount message qoueue
198 mq_unlink(MQ_NAME_SERVER_UDP_ANALYSE);
199 mq_close(mq_sa);
200 
201 // close sockets
202 close(serverSocket);
203 close(clientSocket);
204 
205 // Warte darauf bis alle Kindprozesse geschlossen sind. Damit keine Zombies entstehen.
206 wait(&pid_status);
207 
208 return EXIT_SUCCESS;
209 }
```

D.2. Bash-Skripte

D.2.1. rc.local

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address


_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi


exec 2>/tmp/rc.local.log
exec 1>&2

echo "starte firewall skript"

/home/pi/firewall/firewall.sh

echo "EXIT"

exit 0
```

D.2.2. firewall.sh

```
#!/bin/bash
# startup script myFirewall

echo "Start firewall script..."

echo "clear whitelist"
>/home/pi/firewall/whitelist

echo "Load Whitelist - config file validSensors"
/home/pi/firewall/loadWhitelist.sh

echo "Start myIptables..."
/home/pi/firewall/myIptables.sh

sleep 1

echo "Save old logfiles"
cp /home/pi/secureServer/config/validSensors /home/pi/firewall/logfile/validSensors
cp /home/pi/secureServer/log/failLog /home/pi/firewall/logfile/failLog
cp /home/pi/secureServer/log/system /home/pi/firewall/logfile/system
cp /home/pi/secureServer/log/statistik /home/pi/firewall/logfile/statisik

echo "Clear all logfiles"
>/home/pi/secureServer/log/failLog
>/home/pi/secureServer/log/system
>/home/pi/secureServer/log/statistik

sleep 1

echo "Start websockets for webinterface..."
/home/pi/firewall/startWebsockets.sh

sleep 1

#echo "Start secureServer...."
#/home/pi/secureServer/initSystem
```

D.2.3. loadWhitelist.sh

```
#!/bin/bash

# files
WHITELIST_SRC=/home/pi/secureServer/config/validSensors
WHITELIST_DEST=/home/pi/firewall/whitelist


# IP-Address

IP_STR=()
FIRST_IP=0
IP_SET=0

# READ IP FROM LIST
for IP in `grep -v ^# $WHITELIST_SRC | awk '{split($0,a,";");print a[1];}'`; do

      IP_SET=0
      echo "IPSET Value: $IP_SET"
      echo "IP_SENSOR: $IP"

      if [ $FIRST_IP -eq 0 ]; then
            FIRST_IP=1
            echo "First IP!- SET VALUE FIRST_IP: $FIRST_IP"
            IP_STR+=($IP)
            echo $IP>$WHITELIST_DEST
            #sed -i '1 i\dsfsdf' $WHITELIST_DEST
            IP_SET=$(( $IP_SET + 1 ))
      else

            # CHECK ARRAY IP KNOWN
            for i in ${IP_STR[@]}; do
                  echo "ARRAY ELEMT: $i"
                  if [ $i == $IP ]; then
                        echo "IP known! - nothing happend"
                        IP_SET=$(( $IP_SET + 1 ))
                  fi
            done
      fi

      if [ $IP_SET -eq 0 ]; then
            IP_STR+=($IP)
            #echo $IP>$WHITELIST_DEST
            sed -i '1 i\'"$IP"'' $WHITELIST_DEST
      fi

done
```

D.2.4. myIptables.sh

```
#!/bin/bash

# iptables-command
IPT=/sbin/iptables
# internes interface
INT=eth1
# externes interface
EXT=eth0
#
WHITELIST=/home/pi/firewall/whitelist
#
SENSORNETZ="192.168.1.0/24"

# ALLOWED EXT PORTS
ALLOWED_EXT_PORTS="60001 60002 60003 60004 60005 60006 60007"
# MULTICAST
MULTI_ADDR="239.192.0.1 239.192.0.2 239.192.0.3 239.192.0.4 239.192.0.5 239.192.0.6 239.192.0.7"



echo "Delete iptables: chains and rules..."

# flush iptables
# -X delete all existing chains
$IPT -X
# -F delete all rules
$IPT -F

# CONFIGURE ALLOWED INCOMMING TRAFFIC ON INTERFACHE EXT

# PORTS
for port in $ALLOWED_EXT_PORTS; do
      if [ $port = "60001" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.1 -j ACCEPT
      elif [ $port = "60002" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.2 -j ACCEPT
      elif [ $port = "60003" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.3 -j ACCEPT
      elif [ $port = "60004" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.4 -j ACCEPT
      elif [ $port = "60005" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.5 -j ACCEPT
      elif [ $port = "60006" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.6 -j ACCEPT
      elif [ $port = "60007" ]; then
            echo "Permitted-Ports UDP: $port"
            $IPT -A INPUT -i $EXT -p udp --dport $port -s 192.168.1.0/24 -d 239.192.0.7 -j ACCEPT
      fi

done

# IP-ADDRESS

IP_STR=()
FIRST_IP=0
IP_SET=0

# READ IP FROM LIST
for IP in `grep -v ^# $WHITELIST | awk '{split($0,a,";");print a[1];}'`; do

      IP_SET=0
      echo "IPSET Value: $IP_SET"
      echo "IP_SENSOR: $IP"

      if [ $FIRST_IP -eq 0 ]; then
            FIRST_IP=1
            echo "First IP!- SET VALUE FIRST_IP: $FIRST_IP"

      fi

      # CHECK ARRAY IP KNOWN
      for i in ${IP_STR[@]}; do
            echo "ARRAY ELEMT: $i"
            if [ $i == $IP ]; then
```

```
81                  echo "IP known! - nothing happend"
82                  IP_SET=$(( $IP_SET + 1 ))
83            fi
84      done
85
86      if [ $IP_SET -eq 0 ]; then
87            $IPT -A INPUT -i $EXT -s $IP -d 239.192.0.0/28 -j ACCEPT
88            IP_STR+=($IP)
89      fi
90
91  done
92
93  echo "IP_ARRAY START"
94  for i in ${IP_STR[@]}; do
95        echo $i
96  done
97  echo "IP_ARRAY END"
98
99  # DROP ALL OTHER TRAFFIC
100 $IPT -A INPUT -i $EXT -j DROP
101
102
103
104
105
106
107 echo "IPTABLES aktiv!"
```

D.2.5. startWebsockets.sh

```
1  #!/bin/bash
2
3
4  #sudo websocketd --port=8080 ~/Bachelorarbeit/skripts/count.sh
5  sudo websocketd --port=8080 tail -f /home/pi/secureServer/log/statistik &
6  sudo websocketd --port=8081 tail -f /home/pi/secureServer/log/system &
```

D.3. Webserver

D.3.1. index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>websocketd count example</title>
    <!-- including librarys -->
    <script src="scripts/reconnecting-websocket.js"></script>
    <script src="scripts/canvasjs.min.js"></script>

    <style>
      table {
        border-spacing: 0.5rem;
      }
      td {
        padding: 0.5rem;
        text-align: center;
      }

      #count {
        font: bold 150px arial;
        margin: auto;
        padding: 10px;
        text-align: center;
      }
    </style>

    <script>
      // variables
      var data; //input Buffer

      /*
       * Chart ValidData
       */
      var sensorDataValid = [5];
      var labelSegmentValid = ["All Sensors", "Registered Sensors","Sensor 1", "Sensor 2", "Sensor 3"];
      var sensorColor1 = ["#FF2500", "#808080", "#808080", "#808080", "#808080"]; // ROT, GRAU, ....
      /*
       * Chart Failures
       */
      var sensorchartFailures = [8];
      var labelSegmentFailures = ["Alarm", "Daten frame", "Sensor Fusion", "Warnings", "Not registered Sensor", "Line
            count", "Time difference", "Failures"];
      var sensorColor2 = ["#FF2500", "#808080", "#808080", "#FFFF00", "#808080", "#808080", "#808080", "#808080"]; //
            ROT, GRAU , Geld , Grau, ....
      /*
       * Chart Traffic
       */
      var trafficAll = [];
      var trafficSensor1 = [];
      var trafficSensor2 = [];
      var trafficSensor3 = [];
      var xValTraffic = 0;
      var yValTraffic = 100;
      var dataLength = 20; // number of dataPoints visible at any point

    </script>

<script>


  window.onload = function () {
    // create websocket and reconnect if connection lost
    //var ws = new WebSocket('ws://localhost:8080/');
    var ws = new ReconnectingWebSocket('ws://172.16.20.1:8080/');
    var ws2 = new ReconnectingWebSocket('ws://172.16.20.1:8081/');

    // if socket ist connected
    ws.onopen = function() {
      document.getElementById('connection').style.backgroundColor = 'green';
      document.getElementById('alarmId').style.backgroundColor = 'green';
      document.getElementById('warningId').style.backgroundColor = 'green';
      document.getElementById('WebsocketState').value = "connected";
      document.getElementById('warningState').value = "passiv";
      document.getElementById('alarmState').value = "passiv";

```

```
73      //document.body.style.backgroundColor = '#cfc';
74    };
75
76    // if connection ist closed
77    ws.onclose = function() {
78      document.getElementById('connection').style.backgroundColor = 'red';
79      document.getElementById('WebsocketState').value = "disconneted";
80      //document.body.style.backgroundColor = null;
81    };
82
83    ws2.onopen = function() {
84      document.getElementById('test').style.background = 'green';
85    };
86
87    ws2.onclose = function() {
88      document.getElementById('test').style.background = 'red';
89    };
90
91    ws2.onmessage = function(event) {
92      data2 = event.data.split(';');
93      // built timestamp
94      timestamp2 = new Date(parseFloat(parseFloat(data2[0])*1000)); // expects ms
95
96      //UTC Format +-0h
97      timestamp2.setUTCHours(2);
98      var hours2 = timestamp2.getHours() +8 ; // hinweis anpassen
99      var minutes2 = "0" + timestamp2.getMinutes();
100     var seconds2 = "0" + timestamp2.getSeconds();
101     var milliseconds2 = timestamp2.getMilliseconds();
102     //Build Time String
103     //formatted_time = minutes.substr(-2);
104     formatted_time2 = hours2 + ":" + minutes2.substr(-2) + ":" + seconds2.substr(-2) + "." + milliseconds2;
105
106     document.getElementById('timestampSytem').value = formatted_time2;
107   };
108
109   // if message incomming
110   ws.onmessage = function(event) {
111     // split incomming data
112     data = event.data.split(';');
113     /*
114      * Chart ValidData
115      */
116     sensorDataValid[0] = data[1];
117     sensorDataValid[1] = data[16];
118     sensorDataValid[2] = data[2];
119     sensorDataValid[3] = data[3];
120     sensorDataValid[4] = data[4];
121     /*
122      * Chart Failures
123      */
124     // Alarms
125     sensorchartFailures[0] = data[5];
126     // DataFrame
127     sensorchartFailures[1] = data[15];
128     // Sensor Fusion
129     sensorchartFailures[2] = data[19];
130     // Warnings
131     sensorchartFailures[3] = data[6];
132     // Not registered Sensors
133     sensorchartFailures[4] = data[7];
134     // Line Count
135     sensorchartFailures[5] = data[17];
136     // Time Diff
137     sensorchartFailures[6] = data[18];
138     // Failures
139     sensorchartFailures[7] = data[8];
140     /*
141      * Chart Traffic
142      */
143     // if new data incomming load values into chart
144     updateChart();
145   };
146
147   /*
148    * Chart Valid-Data
149    */
150   var chartValidData = new CanvasJS.Chart("chartContainerValidData", {
151     title: {
152       text: "Data transfer"
153     },
154     axisY: {
155       title: "Frames",
```

```
      suffix: ""
    },
    data: [{
      type: "column",
      yValueFormatString: "#,###",
      indexLabel: "{y}",
      dataPoints: [
                { label: "All Sensors", y: 206 },
                { label: "Registered Sensors", y:200 },
                { label: "Sensor 1", y: 163 },
                { label: "Sensor 2", y: 154 },
                { label: "Sensor 3", y: 176 }
                ]
    }]
  });
  /*
   * Chart Warnings-Alarm-Failures-Data
   *
   */
  var chartFailures = new CanvasJS.Chart("chartContainerFailures", {
    title: {
      text: "Warning/Alarm/Failure-Display"
    },
    axisY: {
      title: "Frames",
      suffix: ""
    },
    data: [{
      type: "column",
      yValueFormatString: "#,###",
      indexLabel: "{y}",
      dataPoints: [
                { label: "Alarms", y: 206 },
                { label: "Data frame", y: 206 },
                { label: "Sensor Fusion", y: 206 },
                { label: "Warnings", y: 163 },
                { label: "Not registered Sensor", y: 154 },
                { label: "Line Count", y: 206 },
                { label: "Time Diff", y: 206 },
                { label: "Failure", y: 176 }
                ]
    }]
  });
  /*
   * Chart Traffic
   */
  var chartTraffic = new CanvasJS.Chart("chartContainerTraffic", {
    animationEnabled: true,
    title :{
      text: "Traffic Analyse"
    },
    axisY: {
      title: "Datentransfer [Byte]",
      includeZero: true
    },
    toolTip: {
      shared: true
    },

    data: [{
      type: "line", // splineArea
      dataPoints: trafficAll
    },
    {
      type: "line", // stackedArea
      dataPoints: trafficSensor1
    },
    {
      type: "line", // line
      dataPoints: trafficSensor2
    },
    {
      type: "line",
      dataPoints: trafficSensor3
    }]
  });

  function updateChart() {

    // built timestamp
    timestamp = new Date(parseFloat(data[0]*1000)); // expects ms
    //UTC Format +-0h
    timestamp.setUTCHours(2);
```

```
239     var hours = timestamp.getHours() +8;
240     var minutes = "0" + timestamp.getMinutes();
241     var seconds = "0" + timestamp.getSeconds();
242     var milliseconds = timestamp.getMilliseconds();
243     //Build Time String
244     //formatted_time = minutes.substr(-2);
245     formatted_time = hours + ":" + minutes.substr(-2) + ":" + seconds.substr(-2) + "." + milliseconds;
246
247     document.getElementById('timestampAktiv').value = formatted_time;
248
249     /*
250      * Chart Valid Data
251      */
252     var dpsValidData = chartValidData.options.data[0].dataPoints;
253
254     for (var i = 0; i < dpsValidData.length; i++) {
255
256       yValValidData = parseInt(sensorDataValid[i]);
257       dpsValidData[i] = {label: labelSegmentValid[i] , y: yValValidData, color: sensorColor1[i]};
258     }
259     chartValidData.options.data[0].dataPoints = dpsValidData;
260     chartValidData.render();
261
262     /*
263      * Chart Warnings/Alarms/Failures
264      */
265     var dpsFailures = chartFailures.options.data[0].dataPoints;
266     for (var j = 0; j < dpsFailures.length; j++) {
267       yValFailures = parseInt(sensorchartFailures[j]);
268       dpsFailures[j] = {label: labelSegmentFailures[j] , y: yValFailures, color: sensorColor2[j]};
269     }
270     chartFailures.options.data[0].dataPoints = dpsFailures;
271     chartFailures.render();
272
273     /*
274      * Chart Traffic
275      */
276     xValTraffic = xValTraffic + 1;
277     yValTraffic = parseInt(data[5]);
278
279     trafficAll.push({
280        x: timestamp,
281        y: parseInt(data[11])
282       });
283
284     trafficSensor1.push({
285        x: timestamp,
286        y: parseInt(data[12])
287       });
288
289     trafficSensor2.push({
290        x: timestamp,
291        y: parseInt(data[13])
292       });
293
294     trafficSensor3.push({
295        x: timestamp,
296        y: parseInt(data[14])
297       });
298
299     if (trafficAll.length > dataLength) {
300       trafficAll.shift();
301       trafficSensor1.shift();
302       trafficSensor2.shift();
303       trafficSensor3.shift();
304     }
305     chartTraffic.render();
306
307     // output element html
308     warningIn = parseInt(data[10]);
309     alarmIn = parseInt(data[9]);
310     // warning
311     if(warningIn > 0) {
312       warningMessage="aktiv";
313       document.getElementById('warningId').style.backgroundColor = 'yellow';
314     }
315     // alarm
316     if(alarmIn > 0) {
317       alarmMessage="aktiv";
318       document.getElementById('alarmId').style.backgroundColor = 'red';
319     }
320     document.getElementById('alarmState').value = alarmMessage;
321     document.getElementById('warningState').value = warningMessage;
```

```
322     }
323   };
324
325 function myFunctionAck(){
326   warningMessage = "passiv";
327   alarmMessage = "passiv";
328   document.getElementById('alarmState').value = alarmMessage;
329   document.getElementById('warningState').value = warningMessage;
330   document.getElementById('alarmId').style.backgroundColor = 'green';
331   document.getElementById('warningId').style.backgroundColor = 'green';
332   //document.body.style.backgroundColor = '#cfc';
333 }
334
335 </script>
336   </head>
337   <body>
338
339     <table style="width:100%">
340       <tr>
341         <th colspan=2>Monitoring System - Application Level Gateway</th>
342       </tr>
343       <tr>
344         <td colspan=2>
345           <table style="width:100%">
346             <tr>
347               <th style="width:34%">
348                 WebSocket state:
349                 <div id="connection" style="width:150px; float:right; font-size:large;">
350                   <output id="WebsocketState" for="volume"></output>
351                 </div>
352               </th>
353               <th style="width:33%">
354                 Warning state:
355                 <div id="warningId" style="width:150px; float:right; font-size:large;">
356                   <output id="warningState" for="volume"></output>
357                 </div>
358               </th>
359               <th style="width:33%">
360                 Alarm state:
361                 <div id="alarmId" style="width:150px; float:right; font-size:large;">
362                   <output id="alarmState" for="volume"></output>
363                 </div>
364               </th>
365             </tr>
366             <tr>
367               <th colspan=2>
368                 Acknowledge:
369                 <button onclick="myFunctionAck()">ACK</button>
370               </th>
371               <th align="left">
372                 System aktiv since:
373                 <!-- <div id="test" style="width:150px; height: 10px; float:right; font-size:large;"> -->
374                 <output id="timestampSytem" for="volume">
375                 </output>
376               </th>
377             </tr>
378             <tr>
379               <th colspan=2></th>
380               <th align="left">
381                 System Time:
382                 <output id="timestampAktiv" for="volume"></output>
383               </th>
384             </tr>
385
386           </table>
387         </td>
388       </tr>
389       <tr>
390         <td style="width:50%"><div id="chartContainerValidData" style="height: 370px; width: 90%; margin: auto;"></
              div></td>
391         <td style="width:50%"><div id="chartContainerFailures" style="height: 370px; width: 90%; margin: auto;"></div
              ></td>
392       </tr>
393       <tr>
394         <td colspan=2><div id="chartContainerTraffic" style="height: 370px; width: 95%; margin: auto;"></div></td>
395       </tr>
396
397     </table>
398
399   </body>
400 </html>
```

E. Inhalt der CD

Die CD enthält sämtliche Quellcodes, sowie Simulationsdaten.

Pfad	Beschreibung
Anhang/	Diverse Anhänge der Thesis
Quellcode/	Erstellter Quellcode
Simulationsdaten/	Simulationswerte und Log-Dateien
NPThesis.pdf	Bachelorthesis in PDF-Form

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 9. Oktober 2018	
Ort, Datum	Unterschrift