Bachelor Thesis

Umsetzung eines Routing-Algorithmus für die smarte Roadside Unit

Vorgelegt am: 06.09.2018

Von: **Dawid Kulesza**
Max-Schäller-Straße 4
09122 Chemnitz

Studiengang: Technische Informatik

Studienrichtung: Daten- und Kommunikationstechnik

Seminargruppe: TI15

Matrikelnummer: 402225

Praxispartner: MUGLER AG
Hofer Straße 2-4
09353 Oberlungwitz
Tel.: 03723 747 189

Gutachter: Dipl.-Ing. Sven Lorenz

Prof. Torsten Lehnguth

(Staatliche Studienakademie Glauchau)

Themenblatt Bachelorthesis

Studiengang Technische Informatik

Student:	**Dawid Kulesza**
Matrikelnummer:	**4002225**
Seminargruppe:	**4TI15-1**

Thema der Bachelorthesis

Umsetzung eines Routing-Algorithmus für die smarte Roadside Unit

Gutachter/ Betreuer:	Dipl.-Ing. Sven Lorenz
Gutachter (Studienakademie):	Prof. Torsten Lehnguth

Ausgabe des Themas:	**28.05.2018**
Abgabe der Arbeit an den SG am:	**20.08.2018, bis 14:00:00**

Prof. Ingolf Tiator
Vorsitzender des Prüfungsausschusses
Technik

I. Inhaltsverzeichnis

I. Inhaltsverzeichnis I
II. Abbildungsverzeichnis IV
III. Formelverzeichnis IV
IV. Tabellenverzeichnis IV
V. Abkürzungsverzeichnis V
1. Einleitung 1
1.1. Hintergrund 1
1.2. Aufgabenstellung 2
1.2.1. Automatische Anmeldung neuen RSUs im Netzwerk 3
1.2.2. Berücksichtigung bestimmten Kriterien für Paketweiterleitung 3
1.2.3. Weiterleitung der Datenpakete in entsprechende Richtung 3
1.3. Gliederung der Arbeit 3
2. Theoretische Grundlagen 5
2.1. Routing 5
2.2. Aufteilung der Routingprotokolle 6
2.2.1. IGP und EGP 6
2.2.2. Statisch und dynamisch 6
2.2.3. Distanzvektor und Link-State 7
2.3. Dijkstra-Algorithmus 8
2.4. Beschreibung wichtigster Routingprotokolle 9
2.4.1. RIP 9
2.4.2. OSPF 11
2.5. VPN 13
2.5.1. Typen von VPN-Netzen 14
2.5.2. Authentifizierungsarten 14
2.5.3. OpenVPN 15

2.6. Geographische Koordinaten 17
3. Roadside Unit 18
3.1. Hardware 18
3.2. Software 20
3.3. Kommunikation 21
3.4. Aufbau des Demonstrationsnetzwerkes 22
4. Spezifikation 23
4.1. Erste Analysen 23
4.1.1. Geografisches Routing 23
4.1.2. Erste Idee: eigenständiges Programm 23
4.1.3. Entscheidung: Einsatz von fertiger Routingsoftware 23
4.2. Auswahl des Routingprotokolls 24
4.3. Die Routingsoftware 24
4.3.1. Quagga 24
4.3.2. ExaBGP 25
4.3.3. Bird 25
4.3.4. Entscheidung 25
4.4. Koordinaten als Router-ID 26
4.5. Eigenes Programm für geografisch basiertes Routing 26
4.5.1. C++ 11 27
4.5.2. Benutzte Bibliotheken 28
4.5.3. Lightweight Communications and Marshalling 30
5. Aufgabenumsetzung 32
5.1. Routingdaemon bird 32
5.1.1. Installation 32
5.1.2. Konfiguration 32
5.2. Geografisch basiertes Routing 34
5.2.1. Initialisierungsdatei 35

5.2.2. Spezifikation der Nachrichten 36
5.2.3. Logging 38
5.2.4. Der Server - Verwaltung der einkommenden Verbindungen 40
6. Dokumentation 43
7. Test 44
7.1. Unit Test 44
7.2. Systemtest 44
8. Ausblick 45
8.1. Verwendung von mehreren Threads 45
8.2. Verbindungsweg in Abhängigkeit von Anforderungen 45
8.3. Fahrzeuge als Router 47
9. Zusammenfassung 48
Literaturverzeichnis 49
Anhangsverzeichnis 51
Ehrenwörtliche Erklärung 52

II. Abbildungsverzeichnis

Abbildung 1: Unterschied zwischen IGP und EGP 6
Abbildung 2: Beispielgraph mit Entfernungen 8
Abbildung 3: Beispiel OSPF-Netzwerk 12
Abbildung 4: Logo des OpenVPN-Projektes 15
Abbildung 5: Aufteilung der Erde in geografische Koordinaten 17
Abbildung 6: RSU: Gerätesicht 18
Abbildung 7: RSU: Systemsicht 20
Abbildung 8: RSU: Funktionssicht 21
Abbildung 9: Testnetzwerk mit zwei RSUs 22
Abbildung 10: Quagga: Softwareaufbau 24
Abbildung 11: Logo des Projektes Bird 25
Abbildung 12: Logo des Projektes Boost 29
Abbildung 13: OSPF-Konfiguration im Bird 33
Abbildung 14: Logo des Projektes Doxygen 43
Abbildung 15: Aufteilung der Nachrichten nach Anforderungen 46

III. Formelverzeichnis

Formel 1: Haversine-Formel 37

IV. Tabellenverzeichnis

Tabelle 1: Dijkstra-Algorithmus, Berechnungschritte zum Beispielgraph 9
Tabelle 2: RIP Zuordnung 9
Tabelle 3: RIPv1 und RIPv2: Unterschiede 11
Tabelle 4: OSPF Zuordnung 11
Tabelle 5: OSPF-Metriken für populäre Verbindungstypen 13
Tabelle 6: Kommunikationssteuerung: Hardware 19
Tabelle 7: Zusammenfassung benutzten Boost-Bibliotheken 30
Tabelle 8 Aufbau der Initialisierungsdatei 36
Tabelle 9: Aufbau der LCM-Nachricht 36

V. Abkürzungsverzeichnis

ABR ..... *Area Border Router*
AEBS ..... *Advanced Emergency Braking System*
AG ..... *Aktiengesellschaft*
ARM ..... *Advanced RISC Machine*
ARPANNET ..... *Advanced Research Projects Agency Network*
AS ..... *autonomes System*
ASBR ..... *Autonomous System Border Router*
BGP ..... *Border Gateway Protocol*
BR ..... *Border Router*
BSD ..... *Berkeley Software Distribution*
CIDR ..... *Classless Inter-Domain Routing*
DAB ..... *Digital Autio Broadcasting*
DSL ..... *Digital Subscriber Line*
EGP ..... *Exterior Gateway ProtocolSiehe*
FAQ ..... *Frequently Asked Questions*
GNU ..... *General Public License*
GPL ..... *General Public License*
GPS ..... Global Positioning System
ID ..... *Identification*
IDE ..... *Integrated Development Environment*
IGP ..... *Interior Gateway Protocol*
iOS ..... *iPhone Operating System*
IP ..... *Internet Protocol*
IPSec ..... *Internet Protocol Security*
IR ..... *interner Router*
IVS ..... *Intelligente Verkehrssysteme*
JSON ..... *JavaScript Object Notation*
KOM ..... *Kommunikationstechnologien*
LCM ..... *4.5.3. Lightweight Communications and Marshalling*
LKW ..... *Lastkraftwagen*
LSA ..... *Link-State Advertisement*
LTE ..... *Long Term Evolution*
MD5 ..... *Message-Digest Algorithm 5*
NIC ..... *Network Information Center*
OSI ..... *Open Systems Interconnection Model*
OSPF ..... *Open Shortest Path First*
RIP ..... *Routing Information Protocol*
RSA ..... *Rivest-Sharim-Adleman*
RSU ..... *Roadside Unit*
SNMP ..... *Simple Network Management Protocol*
SSLv3 ..... *Secure Socket Layer version 3*
STL ..... *Standard Template Library*
TAP ..... *Terminal Access Point*
TCP ..... *Transmission Control Protocol*
TDD ..... *Test Driven Development*
TLSv1 ..... *Transport Layer Security version 1*
TUN ..... *Tunnel*
UDP ..... *User Datagram Protocol*
VLSM ..... *Variable-Length Subnet Mask*

VPN ..... *Virtual Private Network*
WLAN ..... *Wireless Local Area Network*
XML ..... *Extensible Markup Language*

1. Einleitung

1.1. Hintergrund

Moderne Technik unterstützt die Menschheit an immer mehr Stellen. Neue Entwicklungen erleichtern das Leben oder ermöglichen sogar es zu verlängern. Ein Beispiel dazu: Medizinroboter, die verdeckte Krankheiten früh erkennen oder bei ärztlichen Behandlungen Hilfe leisten. Ein bisschen entspannender Fall: das Angeln. Mit speziellen Detektoren, sogenannten Echoloten, lassen sich die Aufenthaltsorte der Fischschwärme bestimmen. Es bleibt natürlich die Frage offen, ob das Angeln auf diese Art und Weise noch als Angeln bezeichnet werden kann.

Auch im Bereich Transport und Kommunikation finden zurzeit revolutionäre Änderungen statt. Die Autos werden immer „schlauer“. Seit 1. November 2016 müssen alle neuen LKWs über acht Tonnen mit einem Advanced Emergency Braking System (AEBS) ausgestattet werden. Dieser Assistent soll vor dem Fahrzeug stehende Autos automatisch erkennen und wenn nötig, eine Notbremsung durchführen, die einen Unfall entweder verhindert oder wenigstens die Aufprallgeschwindigkeit reduziert.

So wurde also die Erkennung von Anderen durch ein Fahrzeug zum Standard. Neue Autos können aber deutlich mehr. Es werden nicht nur andere Fahrzeuge erkannt, sondern auch Spurlinien, Fußgänger oder Straßenzeichen. Anschließend werden diese Daten ausgewertet und aufgrund davon kann das Auto den Fahrer entsprechend warnen oder sich sogar selbst führen. Die Rede ist hier von autonomen Fahrzeugen. Die Vorreiterfirma auf dem Markt der autonomen Fahrzeuge ist Tesla. Deren geschäftsführendes Vorstandsmitglied Elon Musk sieht die Zukunft sehr perspektivisch. Seiner Ansicht nach wird die Rolle der traditionellen Autos in zwanzig Jahren der Rolle von heutigen Pferden entsprechen[1]. Es ist ein Transportmittel und Leute benutzen es, vorrangig aber als ein Hobby.

Wenn diese Vision ausgefüllt werden soll, darf man sich nicht auf seinen Lorbeeren ausruhen und man muss weitere Entwicklungen vorantreiben. Und es geht hier nicht nur um die Verbesserung der Algorithmen sondern auch um ganz neue Ideen, wie man das Auto noch bewusster macht. Eine davon ist die Kommunikation zwischen

[1] https://www.businessinsider.com/elon-musk-owning-a-car-in-20-years-like-owning-a-horse-2015-11?IR=T

den Fahrzeugen, aber auch zwischen den Fahrzeugen und sich nicht bewegenden Verkehrsteilnehmer wie Lichtsignalanlagen oder Roadside Unit (RSU). Somit, wenn durch ein Auto eine Gefahr, wie zum Beispiel Glatteis, erkannt wird, kann das allen benachbarten Fahrzeugen gemeldet werden und der Fahrer oder das Auto selbst kann die Geschwindigkeit an die äußere Bedingungen anpassen. Ein weiteres Beispiel: ein Rettungswagen im Einsatz kann andere Fahrzeuge informieren, dass er sich nähert bevor er überhaupt zu sehen ist und so kann für ihn schneller freie Durchfahrt gemacht werden. Schnellere Durchfahrt spart Sekunden, die wiederum die Überlebungschancen der Bedürftigen erhöhen können. Die Technik kann das Leben auch so retten. Wenn alle Autos an grüner Ampelphase autonom ohne Verzug starten, können nicht nur die Nerven des Fahrers erspart werden, aber auch die gesamte Durchlassfähigkeit aller Ampelkreuzungen kann verbessert werden.

Um die Weiterentwicklungen in diesem Bereich zu unterstützen, hat die Bundesregierung eine Reihe von Forschungsprojekten etabliert. An mehreren davon nimmt die Firma MUGLER AG teil. Die Aufgabe, die im Rahmen dieser Bachelorarbeit beschrieben wird findet im Rahmen des Projektes IVS-KOM statt.

1.2. Aufgabenstellung

Das Forschungsprojekt IVS-KOM beschäftigt sich mit Erstellung eines Referenzsystems zur IVS Kommunikation mit heterogenen Technologien wie WLAN-11p, Mobilfunk oder DAB+.

Im Rahmen dieses Projektes sollte die Firma MUGLER AG Lösungen für folgende Probleme erstellen:

- Spezifikation und Implementierung einer hybriden Kommunikationssteuerung
- Spezifikation einer smarten Roadside Unit
- Spezifikation und Implementierung eines Gerätemanagements
- Spezifikation und Implementierung eines lokalen Cloud Service
- Spezifikation und Implementierung eines Routingalgorithmus

Die Aufgabe, die in dieser Bachelorarbeit vorgestellt wird, sollte der Erfüllung des fünften Punktes dienen. Verallgemeinert sollte die Weiterleitung der ausgewählten Datenpakete zwischen den Router in Abhängigkeit von dem Nachrichtentyp in entsprechende Richtung möglich sein. Zusätzlich sollte immer die günstigste

Verbindung zwischen den Roadside Units bevorzugt werden. Für die Aufgabe wurden im Voraus folgende Anforderungen gestellt.

1.2.1. Automatische Anmeldung neuen RSUs im Netzwerk

Eine, neu in das gesamte Netzwerk angeschlossene, RSU soll durch andere RSUs automatisch erkannt werden. Das heißt, eine Weiterleitung der Datenpakete an und über diese RSU sollte möglich sein. Dasselbe gilt in zweite Richtung: Eine neue RSU sollte die Netzwerktopologie selbst herausfinden, so dass keine zusätzliche Konfigurationen nötig sind.

1.2.2. Berücksichtigung bestimmten Kriterien für Paketweiterleitung

Die Datenpakete können entweder über einen zentralen Server zwischen den einzelnen RSUs oder über mehrere RSUs, ohne Vermittlung durch diesen Server, ausgetauscht werden. Die Verbindung über den Server wird über das LTE-Mobilfunknetz aufgebaut, deswegen gilt sie als unzuverlässig und ist mit zusätzlichen Kosten verbunden. Das ist die Erklärung, warum falls möglich, immer die Vermittlung der Datenpakete über RSUs stattfinden soll. Die zu erstellende Lösung sollte diese Kriterien berücksichtigen.

1.2.3. Weiterleitung der Datenpakete in entsprechende Richtung

Als dritte Anforderung gilt die gezielte Weiterleitung der ausgewählten Pakete in eine Richtung. Beispielweise, biegt ein Auto an einer Kreuzung rechts, so sollten ausgewählte Nachrichten, die dieses Fahrzeug betreffen, in rechte Seite geleitet werden. Nur dieser Fall sollte in der ersten Lösung betrachtet werden, wobei eine Erweiterung an weitere Fälle möglich sein sollte.

1.3. Gliederung der Arbeit

Im Anschluss an diesen Abschnitt werden die wichtigsten technischen Grundlagen erläutert, die in weiteren Teilen dieser Bachelorarbeit erhebliche Rolle spielen werden. Darunter fällt unter anderem ein Einblick in das Thema VPN, Routing sowie Routingprotokolle.

Danach folgt die Beschreibung der Roadside Unit, wo die Lösung der Aufgabe implementiert werden soll. Es wird auf die Hardware, das System sowie die Funktionalitäten eingegangen. Zusätzlich dazu kommt hier die Vorstellung des

firmeneigenen Demonstrationsnetzwerkes, wo die abschließenden Tests durchgeführt werden.

Im Kapitel Vier wird die Planung und Konzeption dargestellt. Und danach kommt die eigentliche Umsetzung der Aufgabe folgend mit Test, wo die Testszenarien und die Ergebnisse beschrieben werden.

Abschließend erfolgt eine Zusammenfassung, wo ein Fazit über die gesamte Aufgabe gezogen wird, sowie ein Ausblick auf zukünftige Verbesserungen und potenzielle Anpassungen durchgeführt wird.

In gesamter Arbeit gilt, dass wegen dem Umfang der Kommentare, diese bei Quelltextausschnitten gelöscht werden. Dasselbe gilt für unwichtige Quelltextteile die bei der Demonstration der ausgewählten Beispiele keine Rolle spielen.

2. Theoretische Grundlagen

Zum weiteren Verständnis dieser Bachelorarbeit müssen einige Begriffe im Voraus erklärt werden.

2.1. Routing

Als Routing wird die Streckenfestlegung für Datenpakete bezeichnet, so dass diese möglich optimal beim Empfänger ankommen. Für die Streckenbewertung (Metrik) können unterschiedliche Kriterien verwendet werden wie beispielweise Kosten, Latenz, Zuverlässigkeit, Auslastung oder Bandbreite. Die Geräte, die sich mit dem Paketrouting beschäftigen, nennt man Router. In Router kommen Routingprotokolle zum Einsatz, die zum Austausch der Informationen über ansprechbare Netze benutzt werden. Verallgemeinert tun diese Routingprotokolle zwei auf den ersten Blick einfache Sachen:

- sagen der Welt, wer die Nachbaren sind
- sagen Nachbaren, wie die Welt aussieht

Die Datenpakete sind gekennzeichnet durch die Sender- und Empfängeradresse. Beim IP-Routing wird das IP-Protocol benutzt. In diesem Fall wird die Zieladresse mit allen vorhandenen Zieladressen, die sich in der sogenannten Routingstabelle befinden, verglichen.

Das IP-Protokol ermöglicht hierarchische Adressierung, die die Gruppierung der Adressen in Subnetze ermöglicht. Die Subnetzzugehörigkeit wird durch die binäre Und-Verknüpfung der Zieladresse und der Netzmaske berechnet. Damit wird die Not zur Speicherung jedes einzelnen Gerätes vermieden.

Es gibt zwei Arten der Zuordnung zu Subnetzen:

- klassenbezogene (VLSM)
- klassenlose (CIDR)

Die klassenbezogene Zuordnung nutzt die ersten Bits der IP-Adresse zur Festlegung der Klassezugehörigkeit, die wiederum die Netzmaske bestimmt. CIDR weist jeder IP-Adresse die Netzmaske selber zu, die eindeutig das Subnetz festlegt.

Heutzutage wird zumeist CIDR eingesetzt. Die wichtigsten Gründe dafür sind:

- größere Flexibilität
- größere Anzahl der Subnetze
- Vereinfachung der Routingstabelle
- größere Netzwerkeffizienz

2.2. Aufteilung der Routingprotokolle

Die Routingprotokolle können aufgrund von unterschiedlichen Kriterien aufgeteilt werden:

2.2.1. IGP und EGP

Ein autonomes System ist ein Netzwerk oder eine Netzwerkansammlung, das durch einen allgemeinen Administrator verwaltet wird. Ein autonomes System besteht aus mehrerenu Router, die allerdings für externe Netze als ein Gerät betrachtet werden, das heißt ein Router an der Grenze der autonomen Systeme verbirgt alle anderen internen Router.

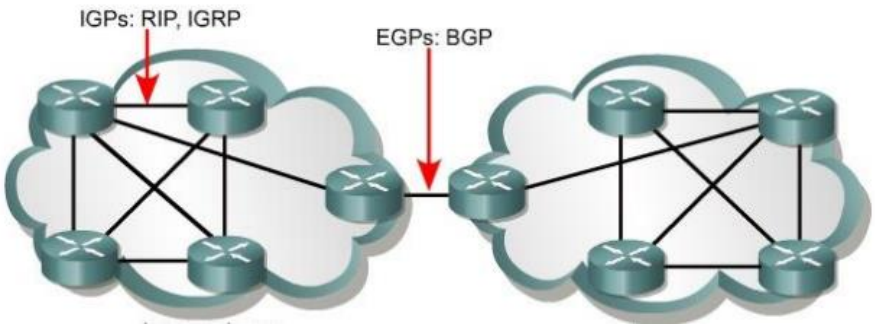


Abbildung 1: Unterschied zwischen IGP und EGP aus [2]

Wenn das Routing innerhalb eines autonomen Systems stattfindet, wird eine IGP (Interior Gateways Protocols)-Lösung eingesetzt. Findet das Routing zwischen autonomen Systemen kommen EGP (Exterior Gateway Protocols) zum Einsatz.

2.2.2. Statisch und dynamisch

Werden statische Protokolle eingesetzt, so muss der Administrator selber alle Änderungen per Hand aktualisieren, die allerdings danach automatisch mithilfe von Routingprotokollen propagiert werden.

Dynamische Protokolle ermöglichen zusätzlich zu den, per Hand eingetragenen Pfaden, die automatische Aktualisierung der Routingstabelle bei Netzwerkänderungen.

2.2.3. Distanzvektor und Link-State

Diese zusätzliche Aufteilung ist nur bei internen Routingprotokollen möglich.

In Distanzvektorlösungen wird die Entfernung, Richtung und der Vektor jeder Verbindung (Link) bestimmt. Die Router, die diese Protokolle benutzen, versenden periodisch an alle benachbarten Router alle seine Einträge aus der Routingtabelle. Dieser Vorgang findet auch dann statt, wenn keine Änderungen stattgefunden haben. Nachdem ein Router eine solche Information bekommt, überprüft er ob Änderungen vorhanden sind und aktualisiert gegebenenfalls seine eigene Routingtabelle. Informationen, welche Netzwerke erreichbar sind, basieren auf Daten von benachbarten Routern, deswegen ist der Aufbau der Netzwerktopologie zeitaufwendig. Zusätzlich eignet sich diese Lösung nicht bei sehr großen Netzwerken, wo es mehrere Wege zu einem Ziel gibt, weil in diesem Fall Schleifen entstehen können: ein Router bekommt zuerst eine Information, dass ein Netzwerk nicht verfügbar ist und leitet diese Nachricht weiter, von langsamerem Nachbar bekommt er eine verspätete Nachricht, dass dieses Netzwerk wieder erreichbar ist und leitet diese an den schnelleren Nachbar weiter. Auf der anderen Seite sind dadurch Protokolle aus dieser Familie einfach zu konfigurieren und gehen sparsam mit Ressourcen um. Das ergibt wiederum billigere Produkte.

Eine andere Vorgehensweise, die oben erwähne Schwächen umgeht, wird bei Link-State Protokollen eingesetzt. Bei dieser Methode werden die Netzwerkänderungen sehr schnell verbreitet, weil Updates automatisch durch die Router versendet werden, wenn ein Netzwerk ausfällt bzw. wenn ein neues hinzugefügt wird. Es werden nicht die gesamten Routingtabellen übertragen, sondern lediglich die Informationen über angeschlossene Netzwerke. Jeder Router besitzt eine komplexe Topologie-Datenbank mit Informationen über entfernte Router. Diese Datenbank wird aufgrund von empfangenen LSA-Updates (Link-State Advertisement) aufgebaut. Das bringt aber einen wichtigen Nachteil mit sich: im anfänglichen Stadium, wenn ein neuer Router an das Netzwerk angeschlossen wird, wird er mit LSA-Paketen überflutet, was mit hoher Auslastung verbunden ist. Zur Berechnung des optimalen Pfades kommt in Link-State Protokollen zumeist der Dijkstra-Algorithmus (siehe Kapitel 2.3).

2.3. Dijkstra-Algorithmus

Der Dijkstra Algorithmus dient der Bestimmung des kürzesten Pfades von einem Startknoten s zu allen anderen in einem gerichteten Graph.

In diesem Algorithmus wird die Menge aller Knoten Q, für die noch keine Berechnung des kürzesten Pfades stattgefunden hat, und der Vektor D[i] der Entfernungen von Knoten s zum i gespeichert. Am Anfang besteht die Menge Q aus allen Knoten und der Vektor D entspricht der ersten Matrixzeile der Kantengewichte A.

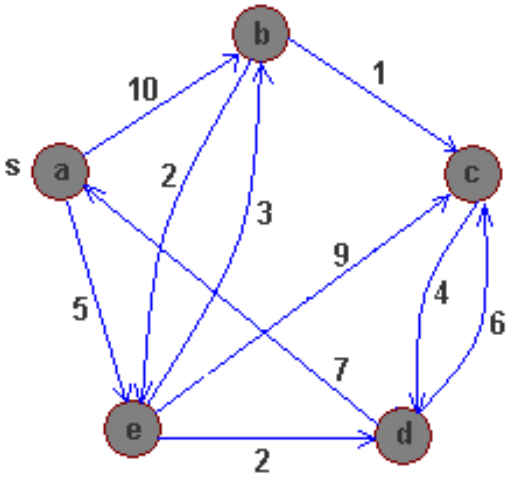
Der Algorithmusablauf erfolgt in drei Schritten:

1. Solange die Menge Q nicht leer ist, führe folgende Schritte auf:
2. Nimm aus der Menge Q die Kante v mit kleinstem Wert D[v] und lösche sie aus der Menge Q
3. Für jeden Folgeknoten v des Knotens i führe den Vergleich D[i]>D[v]+A[v, a]. Das heißt, überprüfe, ob aktuelle Schätzung der Entfernung zum Knoten i größer als die Schätzung der Entfernung zum Knoten v plus Gewicht der Kanten (v, i) ist.

   Ist das der Fall, aktualisiere die Schätzung D[i] durch die Zuweisung von rechter Seite des Vergleichs (also kleineren Wert).

Wie aus der obigen Beschreibung zu sehen ist, wählt dieser Algorithmus immer den „leichtesten“ Knoten, deswegen zählt dieser Algorithmus zu sogenannten Greedy-Algorithmen. Man muss beachten, dass die Greedy-Algorithmen nicht immer das optimalste Ergebnis liefern.

Zur Veranschaulichung der Vorgehensweise wurde folgender Beispielgraph erstellt:



	a	b	c	d	e
a	0	10	∞	∞	5
b	∞	0	1	∞	2
c	∞	∞	0	4	∞
d	7	∞	6	0	∞
e	∞	3	9	2	0

Abbildung 2: Beispielgraph mit Entfernungen aus [2]

Die erfolgte Berechnungsschritte stellt Tabelle 1: Dijkstra-Algorithmus, Berechnungschritte zum Beispielgraph dar:

Q	D(a)	D(b)	D(c)	D(d)	D(e)
{b,c,d,e}	**0**	10	*	*	5
{b,c,d}	**0**	8	14	7	**5**
{b,c}	**0**	8	13	**7**	**5**
{c}	**0**	**8**	9	**7**	**5**
{}	**0**	**8**	**9**	**7**	**5**

Tabelle 1: Dijkstra-Algorithmus, Berechnungschritte zum Beispielgraph

Es gibt mehrere Varianten dieser Methode. Die einfachste benutzt Vektoren zur Aufbewahrung der Knoten aus der Menge Q. Andere Möglichkeit wäre zum Beipiel eine Vorrangwarteschlange oder ein Fibonacci-Heap.

Die einfachste Variante besitzt die Komplexität $O(n^2)$. Durch den Einsatz von Fibonacci-Heap ist eine Verringerung der Komplexität auf O(n log n) möglich.

Ein anderer Algorithmus, der bei manchen Routingprotokollen verwendet wird, heißt Bellman-Ford-Algoritmus. Die Vorgehensweise ist gleich wie beim Dijkstra-Algorithmus, es wird aber auch mit negativen Kantengewichten gearbeitet.

2.4. Beschreibung wichtigster Routingprotokolle

2.4.1. RIP

2.4.1.1. Zuordnung

Intern	Extern
Link-State	Distanz-Vektor
Statisch	Dynamisch

Tabelle 2: RIP Zuordnung

2.4.1.2. Geschichte

Die erste Version dieses Protokolls wurde im Jahr 1982 veröffentlicht, aber manche Implementierungen wurden schon im Jahr 1969 in ARPANNET-Netzwerken verwendet. Somit zählt dieses Protokoll zu den ältesten Routingprotokollen. Die erste Normierung dieses Protokolls fand im Jahr 1988 in RFC 1058 statt. Als Antwort auf wachsende Anfordernisse gibt es mittlerweile eine offizielle zweite Version dieses

Protokoll, die sich allerdings immer noch nicht zum Einsatz in großen Netzwerken eignet.

2.4.1.3. Funktionsweise

Das Routing Information Protocol benutzt zur Entfernungsberechnung zu anderen Netzwerken den Bellman-Ford-Algorithmus.

Das Protokoll RIP sendet seine Netzwerkinformationen (Routingtabelle) in regelmäßigen Abständen (typischerweise 30 Sekunden). Jedes solcher Pakete besteht aus Details zu benachbarten Netzwerken: IP-Adressen und Entfernung zu diesem Netzwerk, sogenannter Metrik.

Die Metrik wird als die Anzahl der Geräte bezeichnet, die auf dem Weg zum Ziel ein Paket überläuft (Hop-Count). Und so direkt angeschlossene Netzwerke haben die Metrik 1 Netzwerke, die durch einen anderen Router erreichbar sind werden mit Metrik 2 verbreitet und Netzwerke, die über zwei Router zu erreichen sind, besitzen die Metrik 3 und so weiter. Wenn also ein Router eine Aktualisierung der Routingtabelle empfängt, addiert er eins zur Metrik und als Adresse des nächsten Sprunges speichert er die IP-Adresse des Senders.

Das Zählen der benötigten Sprünge zum Ziel führt nicht in jedem Fall zum optimalen Ergebnis. Ein Beispiel: Pfad mit Metrik 3, der über drei Ethernet-Netze geht, kann deutlich schneller sein als Pfad mit der Metrik 2, der über eine langsame Luftverbindung geht.

Das Protokoll RIP, dank der Einschränkung der maximalen Sprunganzahl, vermeidet die endlose Verbreitung der Informationen über nicht erreichbare Netzwerke. Maximale Sprungzahl beträgt 15. Wenn ein Router eine Aktualisierung empfängt, und wenn nach der Erhöhung der Metrik um 1, die Überschreitung der Sprunggrenze von 15 auftritt, dann wird dieses Netzwerk als unerreichbar gekennzeichnet und ihre Einträge werden aus der Routingtabelle entfernt. Es findet dann keine Weiterleitung der Informationen über dieses Netzwerk.

2.4.1.4. Unterschiede zwischen RIPv1 und RIPv2

RIPv1	RIPv2
Keine Unterstützung für klassenlosen Routing	Klassenlose Routing wird unterstützt
Keine Sicherheitsmechanismen; keine Authentifizierung	Authentifizierung wird unterstützt. Einsatz von MD5 (128 Bit)
Benutzt die Broadcastadresse 255.255.255.255	Aktualisierungen werden als Multicast 224.0.0.9 gesendet. Dadurch Weiterleitung der Nachrichten nur durch RIPv2-fähige Geräte

Tabelle 3: RIPv1 und RIPv2: Unterschiede

2.4.2. OSPF

Intern	Extern
Link-State	Distanz-Vektor
Statisch	Dynamisch

Tabelle 4: OSPF Zuordnung

2.4.2.1. Geschichte

Die Entwicklungsarbeiten an OSPF-Protokoll begannen im Jahr 1987 in der Organisation IETF. Im Jahr 1989 wurde die erste Version des Protokolls in RFC 1131 veröffentlicht. Zwei Jahren danach erschien die zweite Version mit vielen technischen Verbesserungen. Im Jahr 1998 wurde sie in RFC 2323 standarisiert. Seit 1999 gibt es OSPFv3, wo zusätzlich IPv6 unterstützt wird.

2.4.2.2. Funktionsweise

Sehr verallgemeinert lässt sich das OSPF-Protokoll in 4 Schritte zusammenfassen:

1. Während der Initialisierung oder bei Netzwerkänderungen erstellt der Router ein LSA-Paket (Link State Advertisement), das Informationen zu allen Verbindungen des Routers besitzt. Anschließend versendet der Router dieses Paket.
2. Jeder Router, der so ein Paket empfängt, nutzt es zur Aktualisierung der Informationen in seiner Topologie-Datenbank und leitet dieses Paket an alle angeschlossenen Router weiter (dieser Vorgang wird auch als flooding bezeichnet).

3. Aufgrund der Topologie-Datenbank wird auf dem Router ein Graph der kürzesten Pfade in allen vorhandenen Richtungen gebaut. Zu diesem Zweck wird der Dijkstra-Algorithmus verwendet. Mithilfe von diesem Graph wird die lokale Routingstabelle ausgefüllt.
4. Falls keine Änderungen in der Routerumgebung vorhanden sind, sendet der Router regelmäßig Hello-Nachrichten. Damit wird signalisiert, dass die Verbindung noch korrekt funktioniert.

Damit man die Routeranzahl, an die LSA-Pakete versendet werden, begrenzt, werden die Router in autonome Systeme (AS) gruppiert, die wiederum in getrennte Areas aufteilt werden. Der Basisbereich ist das Backbone 0.0.0.0, zu dem einzelne Areas X.X.X.X zugeordnet werden. Die Router aus dem Backbone-Bereich kennen die gesamte Netzwerktopologie und werden als Border Router (BR) bezeichnet. Arearouter kennen nur die interne Topologie ihrer Area. Dieser Sachverhalt wird in der Abbildung 3: Beispiel OSPF-Netzwerk veranschaulicht.

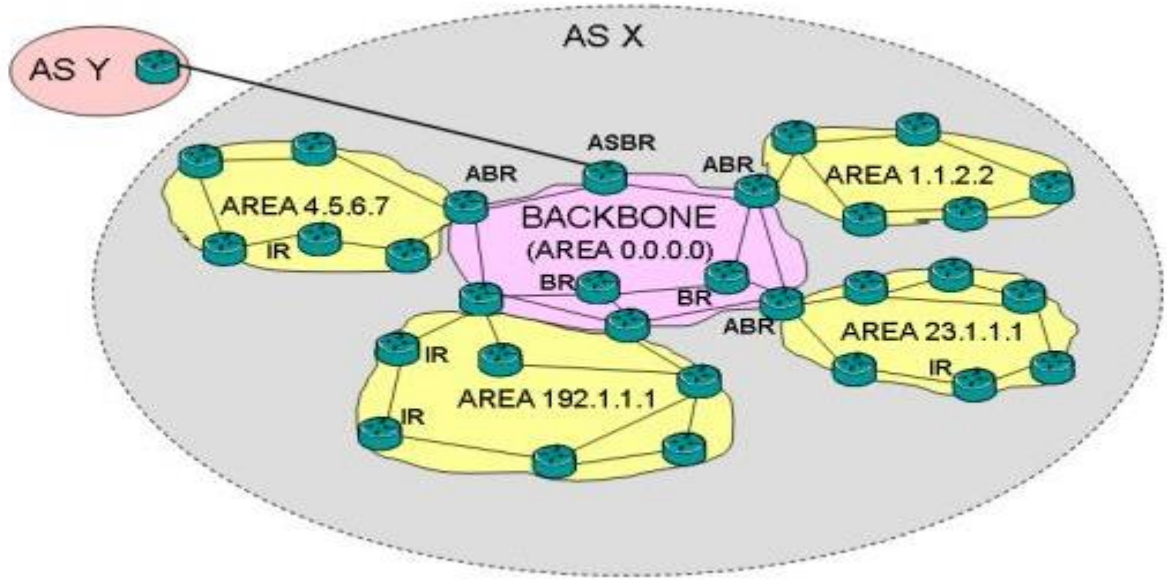


Abbildung 3: Beispiel OSPF-Netzwerk aus [2]

Es gibt unterschiedliche Routertypen in OSPF-Netzen. Router, deren Schnittstellen zu einer und derselben Zone gehören, werden als interne Router (IR) benannt. Wenn diese Router zu einer Zone gehören, bedeutet das, dass sie die gleiche Topologie-Datenbank haben wie andere Router aus dieser Zone. Wenn ein Router mehr als nur einer Zone zugehört, trägt er den Name Area Border Router (ABR). ABR ist verantwortlich für die Distribution der Streckenveröffentlichungen über seine Schnittstellen zu entsprechenden Netzen. Ein Router, der mehrere Protokolle bedient

oder als Gateway zu anderen externen Router dient, wird als ASBR (Autonomous System Border Router) bezeichnet. Es gibt Implementierungen von OSPF, die die gleichzeitige Bedienung von mehreren Pfaden (Equal Cost Multi Path) unterstützen.

Jeder OSPF-Router besitzt eine, wie schon der Name besagt, eindeutige ID. Sie wird zur Unterscheidung der Geräte verwendet und ist in der Routingtabelle zu sehen. Wird eine gleiche ID gleichzeitig zwei oder mehreren Geräten zugewiesen, werden ausgewählte Netzwerke nicht erreichbar.

Es gibt drei übliche Regeln, wie die Router-ID im OSPF-Protokoll gesetzt wird:

1. Manuelle Vergabe durch den Administrator
2. Automatische Wahl der höchsten IP-Adresse an einer Loopback-Schnittstelle
3. Automatische Wahl der höchsten IP-Adresse an einer nicht Loopback-Schnittstelle

2.4.2.3. Der Kostenbegriff - die Metrik

Das OSPF weist jeder seinen Schnittstellen einen Kostenwert aus dem Bereich 1…65535 zu. Dieser Wert ist umgekehrt proportional zur Bandbreite an gegebener Verbindung. Die Formel die ihn beschreibt lautet 10^8/Bandbreite, wobei die Bandbreite in Bits pro Sekunde angegeben werden muss.

Schnittstellentyp	Bandbreite	Metrik
Fast Ethernet	100 Mb/s und mehr	1
Ethernet	10 Mb/s	10
E1	2 Mb/s	48
T1	1,544 Mb/s	64
128kbps	128 b/s	781
56kbps	56 kb/s	1785

Tabelle 5: OSPF-Metriken für populäre Verbindungstypen

Im OSPF werden die Metriken für einzelne Verbindungen und nicht für ganze Strecken verbreitet (im Gegensatz zu RIP). Deswegen gibt es keinen maximalen Wert für die Kosten einer Strecke.

2.5. VPN

Das Virtual Private Network kann man sich als ein Tunnel vorstellen, der ein privates lokales Netzwerk im Rahmen eines öffentlichen Netzwerkes (wie beispielweise

Internet) erstellt. Die Bezeichnung virtuell bedeutet, dass das Netzwerk lediglich als eine logische Struktur funktioniert. Es werden keine physikalischen Verbindungskomponenten, wie Kabel oder Antennen, neu angelegt. Stattdessen wird auf bereits vorhandene Verbindungen und bereits existierendes Netzwerkes, zugegriffen. Trotz solchen Mechanismus, können die Benutzer die VPN-Verbindung gleich wie eine ganz normale, physikalische Verbindung nutzen. Durch die Möglichkeit des Einsatzes von Verschlüsselung und Authentifizierung bietet ein VPN-Tunnel zusätzliche Sicherheit während der Datenübertragung

2.5.1. Typen von VPN-Netzen

Es gibt viele Arten von VPN-Netzwerken, die sich unter anderem mit der Transmissionsrealisierung oder Sicherheitsmechanismen unterscheiden. Die wichtigste Unterteilung sieht folgendermaßen aus:

1. basierend auf dem IPSec-Protokoll
    a. site-to-site Netzwerke:
       sichere Verbindung zwischen zwei oder mehreren Netzwerken. Die Tunnellenden werden durch dedizierte Geräte, wie beispielweise spezielle VPN-Router, abgeschlossen. Es wird keine zusätzliche Konfiguration auf den Endgeräten (Computer, aber auch Handys, Tablets) benötigt.
    b. Remote-access oder client-to-site:
       Verbindung zwischen einzelnen Geräten und Netzwerken. Hier wird eine Installation spezieller VPN-Software benötigt.
2. basierend auf dem SSL-Protokoll

   zumeist vom Typ remote-access, aber es wird hier keine Installation von Zusatzsoftware benötigt. Es kann wie eine Art Proxy-Applikation betrachtet werden. Für die Endnutzer ist nur eine Anmeldung erforderlich. Geeignet vor allem für mobile Endgeräte.

2.5.2. Authentifizierungsarten

Bevor der VPN-Tunnel aufgebaut wird, müssen beide Seiten der Verbindung sich gegenseitig authentifizieren, um sicherzustellen, dass das Gerät an anderem Ende in Wirklichkeit demjenigen entspricht, wofür es gehalten wird.

Bei VPN-Lösungen wird es zwischen drei Authentifizierungsmethoden unterschieden:

- statisches Password (pre-shared key):
  Während der Gerätekonfiguration trägt man das Password direkt in eine Konfigurationsdatei. Diese Vorgehensweise ist nicht empfehlenswert, weil das Passwort in fremde Hände gelangen kann. Zusätzlich kann es zu Schwierigkeiten kommen, wenn eine Verbindung zwischen mehreren Geräten gewünscht wird.
- öffentliche Schlüssel RSA:
  Auf jedem Gerät werden zwei Schlüssel erstellt: ein privater und ein öffentlicher. Anschließend werden die öffentlichen Schlüssel mit allen Verbindungsteilnehmer ausgetauscht. Dieser Vorgang wird durch Administratoren durchgeführt, deswegen ist es bei vielen Geräten zeitaufwendig und fehleranfällig (es sind N*(N-1 Austausche notwendig). Außerdem bei Änderung des Schlüssels auf einem Gerät, ist eine Aktualisierung auf allen Geräten nötig.
- digitales Zertifikat:
  Hier wird die Authentisierung von Dritten, zumeist anerkannten Organisationen, bestätigt. Diese Authentifizierungsart gilt als am sichersten. Es wird mit der Verbindung mit einem Schlüssel eingesetzt. Falls ein Schlüssel entdeckt wird und ein Austausch erforderlich sein wird, kann dieser Vorgang automatisch stattfinden.

2.5.3. OpenVPN

Das Softwarepaket OpenVPN dient zur Erstellung von VPN-Verbindungen entweder auf zweiter oder dritter Schicht des OSI-Systemmodels. Zur Tunnelverschlüsselung nutzt OpenVPN die Bibliothek OpenSSL und Protokolle SSLv3/TLSv1. Im Gegensatz zu typischen VPN-Programmen nutzt diese Software das Protokoll IPSec, obwohl site-to-site Verbindungen auch unterstützt werden, nicht. Dieses Paket wurde für zahlreiche Betriebssysteme erstellt. Erste Version erschien im Jahr 2002. OpenVPN ist mit der GPL-Lizenz geschützt – es kann also gebührenfrei verwendet werden

Abbildung 4: Logo des OpenVPN-Projektes aus [6]

2.5.3.1. Unterschied zwischen Bridging und Routing

Wie bereits erwähnt kann die Software OpenVPN entweder auf zweiter Schicht im Bridge-Modus oder auf dritter Schicht des OSI-Systemmodels im Router-Modus eingesetzt werden. Für Bridging wird eine virtuelle TAP-Schnittstelle im System erzeugt und für Routing TUN.

Bridging (TAP) Vorteile:

- Verhält sich wie ein realer Netzwerkadapter
- Kann jedes Transportprotokoll und nicht ausschließlich IP verwenden
- Arbeitet auf zweiter Schicht des OSI-Referenzmodells, somit werden in diesem Modus ganze Ethernet Frames übertragen
- Ermöglich die Erstellung von realen Bridges

Bridging (TAP) Nachteile:

- Verursacht deutlich mehr Broadcast-Overhead im Vergleich zu TUN
- Deutlicher Performanceverlust (TCP über TCP)
- Kann nicht mit Android- oder iOS-Geräten verwendet werden

Routing (TUN) Vorteile:

- Kleinerer Verkehr-Overhead, Pakete werden immer an ein konkretes Ziel geschickt

Routing (TUN) Nachteile:

- Broadcastpakete werden in Unicastpakete verpackt, kein traditionelles Broadcast möglich
- Benutzt ausschließlich IPv4 als Transportprotokoll
- Kann nicht als Bridge eingesetzt werden
- Schwierigkeiten beim Zugriff auf Geräte in entfernten Netzwerken hinter einem VPN-Client (siehe 5.1.2.5).

2.6. Geographische Koordinaten

Die geographischen Koordinaten bestimmt man durch geographische Breite und Länge. Sie werden in Graden, Minuten und Sekunden gemessen. Ein Grad(°) entspricht 60 Minuten(') und eine Minute 60 Sekunden("). Die Längengrade werden durch Meridiane nach Westen und Osten bis jeweils 180° gezählt. Die Breitengrade werden dafür von dem längsten Breitengrad (Äquator) von 0° bis 90° in Süd und Norden gezählt. Der Nullpunkt ist der Schnittpunkt des Nullmeridians mit dem Äquator.

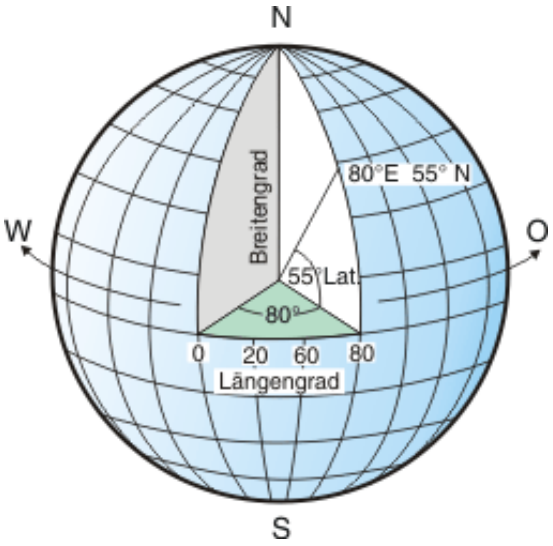


Abbildung 5: Aufteilung der Erde in geografische Koordinaten

3. Roadside Unit

Die Lösung der Aufgabe, die im Rahmen dieser Bachelorarbeit erarbeitet wird, sollte auf hauseigenen Roadside Units der Firma MUGLER AG implementiert werden. In diesem Kapitel werden die wichtigsten Funktionalitäten. Hardware-, sowie Softwarekomponenten vorgestellt. Zusätzlich dazu wird das Beispielnetzwerk mit zwei RSUs beschrieben.

3.1. Hardware

Die RSU des Projektes IVS-KOM soll, um ein breites Anwendungsspektrum und die Kommunikation über verschiedene Kommunikationskanäle ermöglichen und über eine Vielzahl von Kommunikationsschnittstellen verfügen. Leider lassen sich nicht alle Kommunikationseinheiten direkt an die RSU anschließen, deswegen ist ein Switch nötig, was die gesamte Aufgabe kompliziert, da sich eine RSU für die Außerwelt immer hinter dem Switch im lokalen Netzwerk befindet. Die Abbildung 6: RSU: Gerätesicht zeigt die Zusammensetzung wichtigster Komponenten einer Roadside Unit.

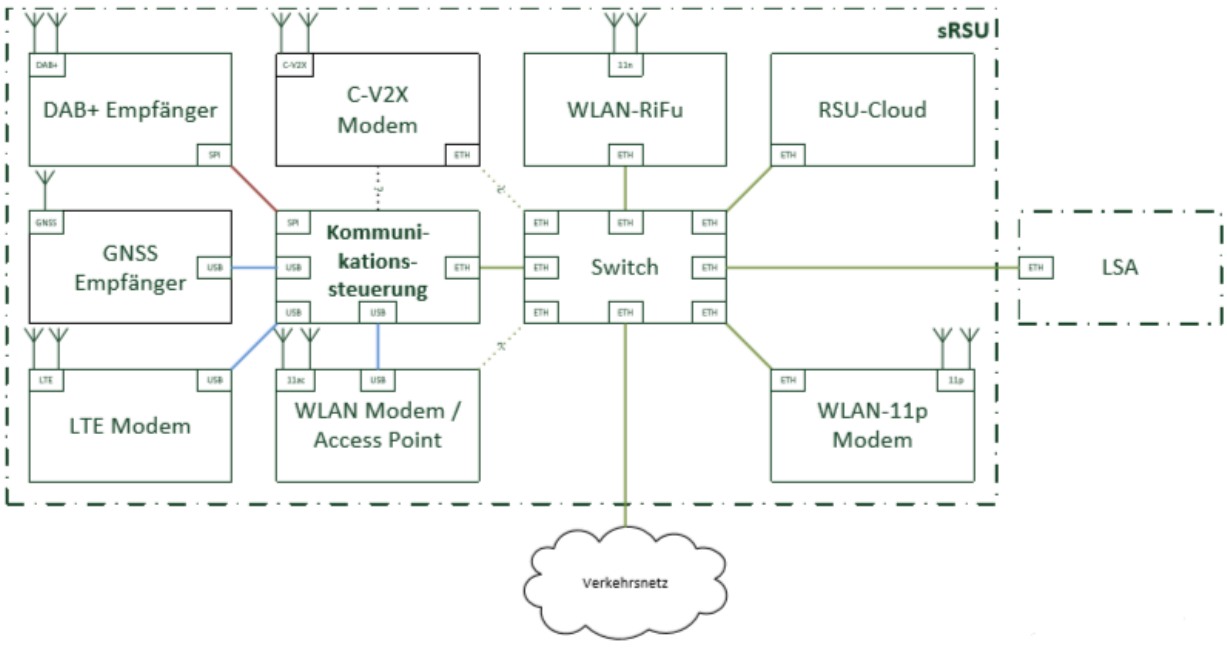


Abbildung 6: RSU: Gerätesicht aus [17]

Das wichtigste Element einer RSU ist die Kommunikationssteuerung, wo das Gerätemanagement stattfindet und über den gesamten Nachrichtenaustauch mit der Außenwelt entschieden wird. Trotz hohen Rechenaufwands, die für diese

Komponente vorgesehen ist, kommt ein ARM-Prozessor zum Einsatz, der für geringeren Energieverbrauch im Vergleich zu traditionellen X86-Prozessoren sorgt. Das bringt allerdings einen Nachteil mit sich: für die ARM-Architektur muss für C++-Programme ein Cross-Compiler verwendet werden, deren Einrichtung mit zusätzlicher Arbeit verbunden ist. Ein weiterer Nachteil: das für diesen Prozessor entwickelte Betriebssystem unterstützt nur eine begrenzte Anzahl der Linux-Standardprogramme. Manche Softwarepakete lassen sich zusätzlich mithilfe von Cross-Compiler erstellen. Dafür muss aber die Voraussetzung erfüllt sein, dass ein entsprechender Algorithmus zur Paketerstellung für den Cross-Compiler vorhanden ist, was nur in ausgewählten Fällen ausgefüllt ist.

Die Tabelle 6: Kommunikationssteuerung: Hardware zeigt die wichtigsten Hardwarekomponenten des Herzes einer RSU, der Kommunikationssteuerung.

Prozessor	Quad Core ARM A9, 1Ghz
RAM-Speicher	2GB 64bit DDR3
Speichermedium	SD-Karte 16 GB

Tabelle 6: Kommunikationssteuerung: Hardware

3.2. Software

In diesem Abschnitt werden nur die wichtigste Softwarepakete, die auf der Kommunikationssteuerung installiert wurde, dargestellt. Die wichtisgsten Komponenten sind in der Abbildung 7: RSU: Systemsicht dargestellt.

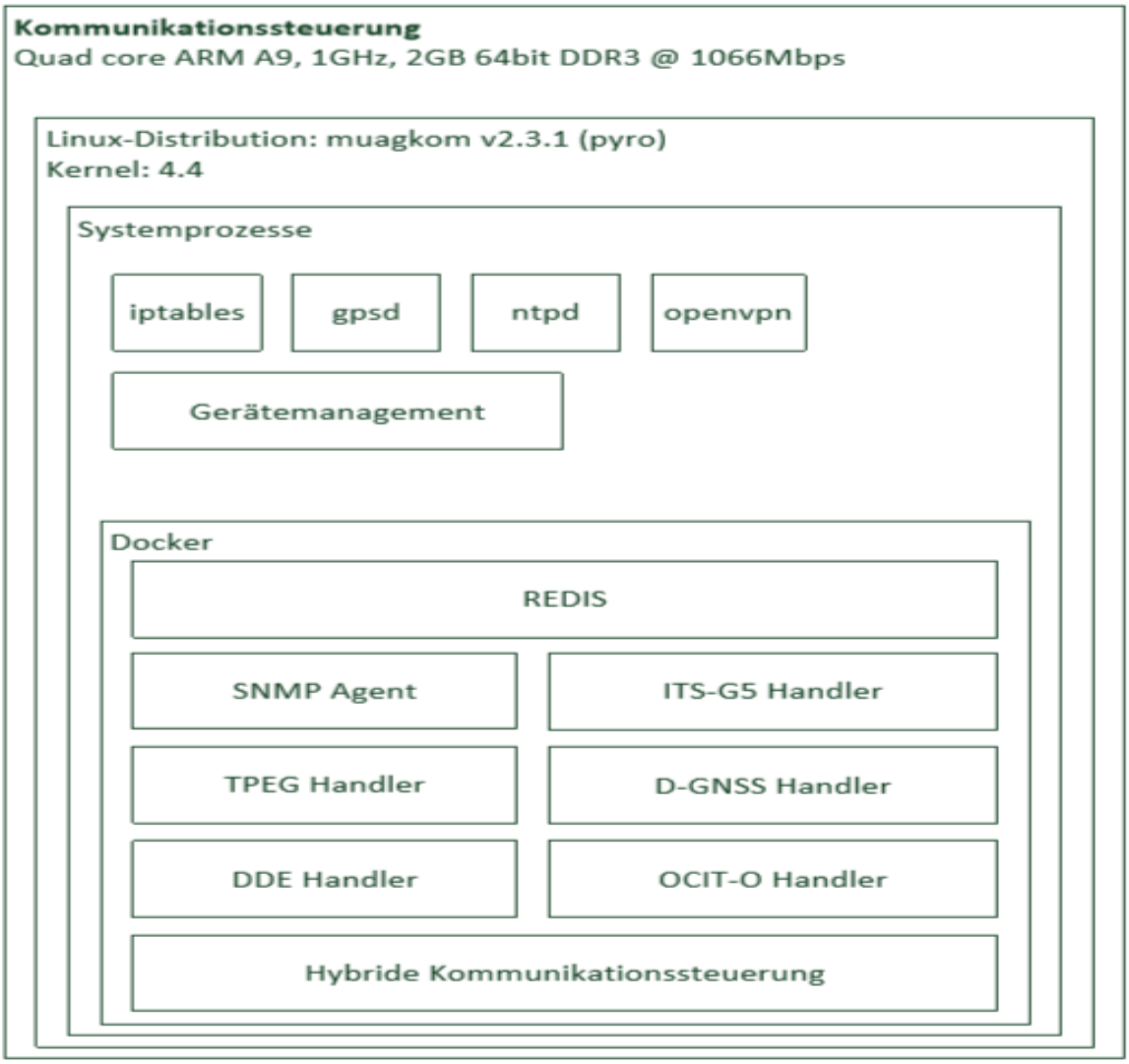


Abbildung 7: RSU: Systemsicht aus [17]

Um korrekte Kommunikation der Hardware mit der Software kümmert sich, eine hauseigene Anpassung der Linux-Distribution, muagkom in der Version 2.3.1. Dieses Betriebssystem wurde speziell für den eingesetzten ARM-Prozessor erstellt.

Interessante Technik stellt die Software Docker dar. Dieses Programm ermöglicht die Erstellung von virtuellen Container, in denen weitere Programme installiert werden. Diese Container sind leicht zwischen unterschiedlichen Linux-Distributionen oder sogar zwischen unterschiedlichen Prozessorarchitekturen übertragbar.

3.3. Kommunikation

Alle Kommunikationswege, über die jede RSU verfügt, sind in der Abbildung 8: RSU: Funktionssicht dargestellt.

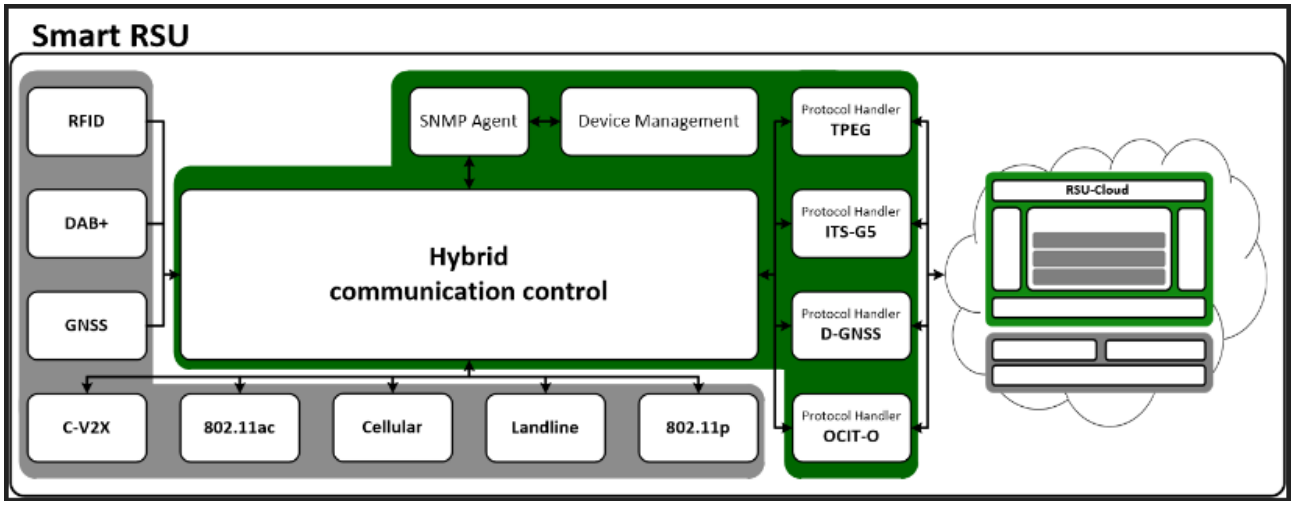


Abbildung 8: RSU: Funktionssicht aus [18]

3.4. Aufbau des Demonstrationsnetzwerkes

Zukünftig sollte die erarbeitete Lösung in komplexen Verkehrsnetzen mit mehreren RSUs eingesetzt werden. Für die Entwicklungsarbeiten wurde ein kleines Beispielnetzwerk mit nur zwei RSUs erstellt. Die Abbildung 9: Testnetzwerk mit zwei RSUs zeigt die bereitgestellte Netzwerkstruktur.

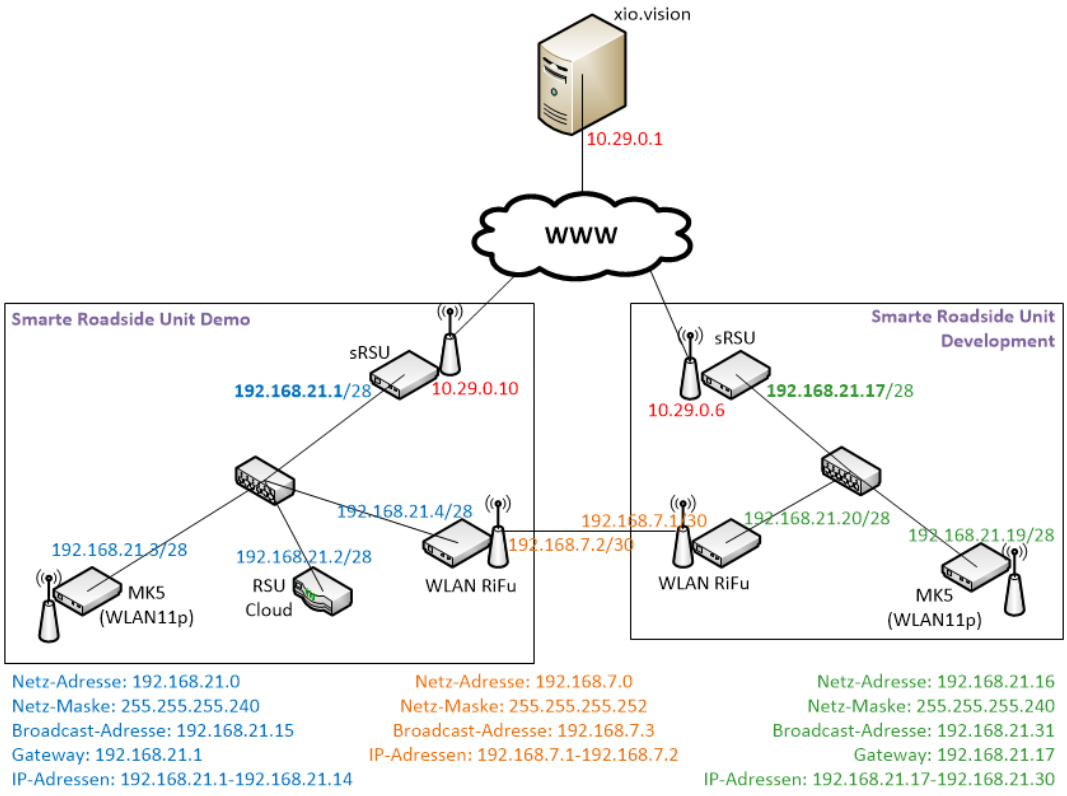


Abbildung 9: Testnetzwerk mit zwei RSUs aus [17]

Der Rechner xio.vision stellt die zentralle hauseigene Entwicklung der MUGLER AG zur Überwachung und Konfiguration von entfernen Geräten mithilfe von SNMP-Protokol dar. Diese zentralle Elemente dient auch als Server für die einzelne OpenVPN-Verbindungen.

4. Spezifikation

Bevor mit der eigentlichen Umsetzung der Aufgabe angefangen wurde, wurde eine ganze Reihe von unterschiedlichen Ideen aufgesammelt. Es fand in einer Gruppe ein Brainstorming-Meeting statt. Die interessantesten Ideen und die daraus resultierenden Entscheidungen werden in diesem Abschnitt beschrieben.

4.1. Erste Analysen

4.1.1. Geografisches Routing

Das geografisch basierte Routing ist nur dann möglich, wenn die eigene Position sowie die Position von benachbarten Geräten einer Roadside Unit bekannt sind. Durch das eingebaute GPS-Modul ist es für die RSU möglich, seine eigenen Koordinaten zu bestimmen. Es wurde die Frage gestellt: woher wird aber eine RSU wissen, wo sich ihre Nachbaren befinden? Daraus folgte die Schlussfolgerung, dass dies nur dann möglich ist, wenn die RSUs seine Position anderen Geräten mitteilen. Jetzt kam die Frage, wie wird das erreicht?

4.1.2. Erste Idee: eigenständiges Programm

Als erste Möglichkeit wurde die Entwicklung von eigener Software gesehen. Es wurde aber im Voraus bekannt, dass die Arbeiten daran sehr zeitaufwendig werden. Zu den größten Schwierigkeiten dabei zählten folgende Punkte:

- Erkennung gestörten Links bei instabilen Mobilfunkverbindungen
- Synchronisation der ankommenden Nachrichten, die über mehrere Wege ankommen können

Der entscheidendste Punkt wurde allerding erst nach gewisser Zeit entdeckt. Bei dieser Lösung müssen die IP-Adressen allen RSU-Netzen manuell in die Routingtabelle eingetragen werden. Bei einer Änderung muss diese Konfiguration durch einen Administrator angepasst werden. Dies widerspricht den Grundanforderungen, dass neue RSUs automatisch in das Netzwerk integriert werden.

4.1.3. Entscheidung: Einsatz von fertiger Routingsoftware

Durch das, im Kapitel 4.1.2 beschriebene Problem, ist es klar geworden, dass eine Bedienung von Routingprotokollen auf jeder RSU notwendig ist. Nur auf diese Art

und Weise ist die automatische Erkennung neuen Verbindungen für eine Roadside Unit möglich. Diese Lösung zeichnete sich auch mit kleiner Komplexität aus. Allerdings muss man bei dieser Lösung einige Kompromisse eingehen. Der wichtigste Nachteil dabei ist das Hingewiesen auf externe Lösungen, die nicht so flexibel anpassbar wären wie hauseigene Lösung.

4.2. Auswahl des Routingprotokolls

Zur Verbindung der einzelnen RSU-Standortnetzwerke kommen die Router der Firma Mikrotik zum Einsatz. Dadurch wurde die Wahl des Routingprotokolls eingeschränkt, da diese Router nur die Protokolle RIP, OSPF, BGP unterstützen.

Um Daten zwischen zwei RSUs auszutauschen, überläuft das Paket zwei Router, die sich um die Verbindung der beiden RSU-Netzwerke kümmern. (siehe Kapitel 3.4 Aufbau des Demonstrationsnetzwerkes). Somit wird der Hop-Counter sehr schnell überlaufen, wenn das RIP-Protokoll seine Aktualisierungen sendet. Dieser Nachteil sowie verspätete Erkennung unterbrochener Verbindungen entschieden über den Verzicht des Einsatzes von diesem Routingprotokoll.

Das OSPF gilt als bewährt und ist sehr weit verbreitet. Zusätzlich wird jedem Router eine ID zugewiesen, die in dieser Rolle eine spezielle Aufgabe bekommt (siehe Kapitel 5.2). Außerdem bieten die meisten Routingsoftware-Pakete (siehe nächstes Kapitel: 4.3) Unterstützung für dieses Protokoll.

4.3. Die Routingsoftware

Nach der Entscheidung für den Einsatz von fertigen Lösungen zur Bedienung eines Routingprotokolls, wurde die Suche nach entsprechender Software begonnen. Es wurden drei Kandidaten zur Erfüllung dieser Aufgabe gefunden.

4.3.1. Quagga

Das Softwarepaket Quagga ist ist eine Fortsetzung des GNU Zebra Projektes, das eine Implementierung der Routingprotokolle RIP, OSPF und BGP umsetzt. Es wurde nach einer ausgestorbenen

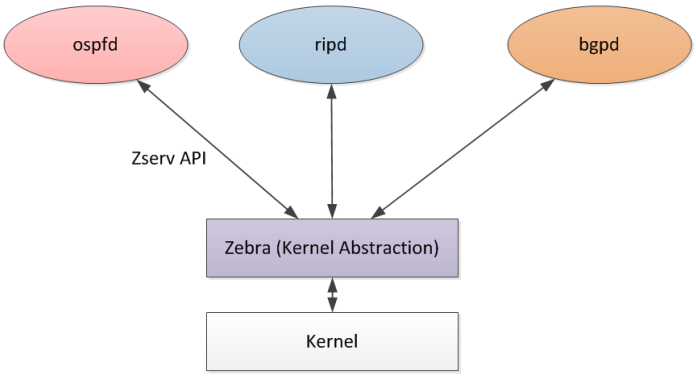


Abbildung 10: Quagga: Softwareaufbau aus [20]

Zebra-Art benannt. Dieser Daemon wird mit GPL-Lizenz geschützt. Es besteht aus einer Hauptanwendung, die die Kommunikation mit dem Systemkern regelt, sowie kleineren Diensten, die für die Bedienung der einzelnen Protokolle sorgt. Die Konfiguration ist CISCO-Geräten ähnlich, findet aber über einen Telnet-Port statt.

4.3.2. ExaBGP

ExaBGP steht unter der BSD-Lizenz bietet allerdings nur die Unterstützung für BGP, deswegen wurde auf die eigentliche Funktionsweise nicht tiefer eingegangen. Es wird mit dem Motto „the BGP swiss army knife of networking“ geworben.

4.3.3. Bird

Ursprünglich entstand Bird als ein Studentenprojekt in Charles Universität in Prag. Mittlerweile wird diese Software durch tschechische Organisation CZ.NIC.Labs betreut. Bird wurde unter der GNU-Lizenz veröffentlicht. Seit 2012 gilt Bird als meistverwendete Lösung durch Internet Server Provider in Europa. Es besitzt sehr umfangreiche Nutzer- sowie Programmiereranleitung.

Abbildung 11: Logo des Projektes Bird aus [12]

Bird bietet eine eigene Skriptsprache zur Filterung der empfangenenen/gesendeten Nachrichtten. So können bestimmte IP-Adresse vollkommen ignoriert werden oder es kann eine Anpassung für bestimmte Datenpakete durchgeführt werden. Zusätzliich bietet diese Software gleichzeitige Verwendung von mehreren Routingstabellen.

4.3.4. Entscheidung

Die Zahlen lügen nicht. Die steigende Popularität der Bird-Software folgt besserer Dokumentation sowie besseren Performance-Ergebnissen im Vergleich zu Quagga.

Zusätzlich muss die Software für den ARM-Prozessor der RSU entsprechend cross-compiliert werden. Dabei gibt es bereits vorgefertigte Rezepte, die zur Übersetzung dieses Paketes benutzt werden können.

Das waren die wichtigsten Gründe, warum Bird zur Bedienung der Routingprotokolle benutzt wird.

4.4. Koordinaten als Router-ID

Wie bereits oben erwähnt, spielt die Router-ID eine spezielle Rolle in Erarbeitung der Lösung für die Aufgabe, die im Rahmen dieser Bachelorarbeit beschrieben wird. Irgendwie müssen die Koordinaten aller anderen RSUs für eine RSU mitgeteilt werden. Da die Router-ID nur eindeutig sein soll und so ist auch die Position jeder RSU, wird die ID zur Speicherung der Koordinaten genutzt. Die ersten zwei Oktette werden für die geografische Länge und die letzten beiden für geografische Breite benutzt. Dabei wird die erste Stelle links für Grad und die zweite für Minuten verwendet.

Da, die Software voraussichtlich nur in Deutschland eingesetzt wird, es wird immer von nordisch–östlichen Koordinaten ausgegangen.

Durch solche Vorgehensweise gibt es keine systematische Regel in der ID-Vergabe, was teilweise dem Begriff ID wiederspricht. Durch das gewonnene Arbeitsersparnis wird jedoch dieser Nachteil akzeptiert.

Zusätzlich wird ein Programm erstellt, was automatisch die Position einer RSU bestimmt, falls GPS-Daten vorhanden sind, und erst danach wird das eigentliche Routingprogramm, Bird, gestartet. Damit wird die automatische Inbetriebnahme einzelnen RSU-Netzwerken in die Gesamtheit allen Netzwerken gesichert.

4.5. Eigenes Programm für geografisch basiertes Routing

Für geografisch basierte Routing muss ein neues Programm erstellt werden, das bestimmte Nachrichten empfängt und an Drittapplikationen weiterleitet. Dasselbe gilt für die zweite Richtung. Das Programm muss die Möglichkeit haben, interne Benachrichtigungen zu verarbeiten, um sie in entsprechende Richtung zu schicken. Für die Vermittlung der Nachrichten zwischen diesem Programm und externen Programmen wird sich unten beschriebenes Lightweight Communications and Marshalling Protocol kümmern.

Obwohl in der ersten Version die Nachrichten jeweils in eine Richtung ohne Entfernungsbegrenzung verschickt werden, ist in der Zukunft mit einer Beschränkung der Ziel-RSUs an diejenige, die eine bestimmte Entfernung nicht überschreiten.

Zusätzlich sollte das Programm seinen Zustand loggen. Das Logniveau der Ablageort, der Logdateien, sowie andere Parameter sollten konfigurierbar sein.

Diese und weitere Einstellungen wird das Programm aus einer Initialisierungsdatei auslesen, deren Aufbau im weiteren Kapitel (5.2.1) beschrieben wird.

Da, diese Anwendung ursprünglich mit mehreren Threads arbeiten soll und auf dem Zielsystem keine Interpreter für Skriptsprachen vorhanden sind, wurde sich auf die Programmiersprache C++11 entschieden. Dieser Dialekt der C++-Programmiersprache ermöglicht den Einsatz von Lambdaausdrücken, die einfache und schnelle Erstellung von anonymen Funktionen, unterstützt. Dies und weitere interessanteste Aspekte neuer Version C++ werden im folgenden Unterkapitel vorgestellt.

4.5.1. C++ 11

Die C++ Programmiersprache wurde im Jahr 1985 durch Bjarne Stroustrup entwickelt. Um mit den aktuellen Fortschritten der sich schnell verändernden Technik Schritt zu halten, wird immer in Abständen von ein paar Jahren eine Standardisierung und Erweiterung dieser Programmiersprache durchgeführt. Die letzte wurde im Dezember 2017 durchgeführt. Die Programmierarbeiten werden im Standard C++ 11 durchgeführt. Die wichtigsten Neuerungen, die dieser Standard mit sich bringt und die in der Erarbeitung der Aufgabenlösung benutzt wurden, werden in weiteren Unterkapiteln beschrieben.

Manche von den unten beschriebenen Änderungen wurden bereits in älteren Versionen von C++ in verschiedenen Bibliotheken implementiert. Die Verwendung von solchen Erweiterungen war aber nicht besonders benutzerfreundlich, deswegen bietet C++ 11 native Unterstützung für diese Funktionalitäten.

4.5.1.1. Auto

C++11 führt den Datentyp Auto ein, der erlaubt eine automatische Typerkennung durch den Compiler während der Übersetzung. Auf diese Art und Weise können vor allem Templates einfacher verwendet werden.

4.5.1.2. Lambda

Als Lambda werden anonyme Funktionen bezeichnet, das heißt Funktionen, die nicht über Ihren Namen angesprochen werden. Diese Funktionen werden am häufigsten als Argument für andere Funktionen verwendet. Es ist besonders vorteilhaft bei der Arbeit mit Templates.

4.5.1.3. Range-based For

C++ erweitert die for-Anweisung, so dass die als foreach-Anweisung (bekannt unter anderem aus Java) arbeitet. Das heißt es muss keine Abschlussbedingung und Änderung der Schleifenparameter durchgeführt werden. Es wird lediglich ein Array übergeben und die Iteration über alle Elemente findet automatisch statt.

4.5.1.4. Der Null-Zeiger: *nullptr*

Der Null-Zeiger dient zur Zuweisung des Null-Wertes für einen Pointer. Bisher wurde für diese Zwecke der Wert NULL verwendet, der sich auf Zahlentypen wie int oder double konvertieren lässt. Der *nullptr* meldet bei solchen Konvertierungen entsprechende Meldungen und dadurch können Fehler vermieden werden.

4.5.1.5. Smart pointers

Smart Pointer gelten für viele Entwickler als wichtigste Feature des C++11-Standards. Es gibt drei Typen von Smart Pointers:

- Unique
- Shared
- Weak

Der erste dient zur Erzeugung von exklusiven Pointer. Dadurch kann der Pointer nicht kopiert werden, was zufälligen Missbrauch vermeidet. Shared Pointer sorgen für eine automatische Speicherbereinigung, wenn der letzte Verweis auf das benutzte Objekt gelöscht wird. Der dritte Typ, Weak, wird benutzt, um zyklische Referenzen mit Shared Pointer zu vermeiden.

4.5.1.6. Strongly-typed enums

In älteren Versionen von C++ wurden Enums bereichsübergreifend exportiert. Das heißt, die Enumsschlüssel durften nicht mehr als Funktions-, Variablennamen oder andere Enumsschlüssel verwendet werden. In C++ 11 wurden Enum-Klassen eingeführt, die diesen Nachteil einfach umgehen. Zusätzlich wurde eine Vererbung von einer Enum-Klasse ermöglicht.

4.5.2. Benutzte Bibliotheken

Trotz der großen Anzahl der neuen nützlichen Funktionalitäten, die C++ 11 mit sich bringt, gibt es, wie in den meisten Programmiersprachen, keine native Unterstützung für Lösungen für wiederkehrende Probleme, wie beispielweise entsprechende

Datencontainer (wie verkettete Listen, Vektoren oder Maps) oder Funktionen zum Loggen oder Threadverwaltung. Es gibt allerdings eine ganze Reihe von Bibliotheken, die diese und andere Aufgaben bewältigen können. Durch entsprechende Auswahl dieser Bibliotheken, können Entwicklungsarbeiten deutlich beschleunigt werden, wobei zumeist am Anfang mit mehr Aufwand zu rechnen ist, da eine Einarbeitung notwendig ist. Dabei gilt, je populärer eine Bibliothek ist, desto sicherer derer Einsatz ist, da sie durch mehrere Personen getestet werden konnte. Die meisten Bibliotheken nutzen generische Programmierung mithilfe von Templates. Dadurch lassen sich benutzerdefinierte Datentypen (Klassen und Strukturen) in diesen Bibliotheken verwenden. Ein großer Teil der Libraries wurde auf Betriebssystemunabhängigkeit ausgelegt und somit lassen sich mit ihnen Programme für UNIX sowie Windows erstellen ohne besonderen Konfigurationen.

Als Nachteil gilt, dass der Einsatz von Templates deutlich längere Übersetzungszeiten von Programmen mit sich bringt.

Bei den Programmierarbeiten wurden folgende Bibliotheken zum Einsatz ausgewählt.

4.5.2.1. Standard Template Library

Standard Template Library (STL) bietet Unterstützung für die meistverwendete Datencontainer. Zusätzlich dazu stellt diese Bibliothek den Datentyp *string* bereit, der eine bequemere Arbeit mit Zeichenketten ermöglicht.

Zu anderen interessanten Funktionen, die allerdings keine große Rolle in der Erarbeitung der Lösung der Titelaufgabe gespielt haben zählen: Threads-management, Streamsverarbeitung oder Tuples (Objektpaare).

4.5.2.2. Boost

Boost ist eine Ansammlung von unterschiedlichen Bibliotheken. Was Boost besonders macht, ist der Fakt, dass ständig neue Bibliotheken hinzugefügt werden können. Boost erlaubt den Einsatz von ausgewählten Programmierparadigmen, wie zum Beispiel funktionale Programmierung, wo Funktionen als Objekte behandelt werden. Außerdem bietet Boost Unterstützung für Bildverarbeitung, Nebenläufigkeit und zahlreiche andere Funktionalitäten.

Abbildung 12: Logo des Projektes Boost aus [21]

Die folgende Tabelle Tabelle 7: Zusammenfassung benutzten Boost-Bibliotheken zeigt die, im Projekt verwendete, Boost-Bibliotheken sowie deren Funktionen

Bibliothek	**Funtion**
PropertyTree	• Speicherung der Werte als Verknüpfung von Schlüssel und Wert • Baumartige Struktur (Unterstützung für XML und JSON) • Zum Einlesen und Aufbewahrung der Konfigurationsdatei
Log	• Erstellung der Log-Meldungen aus einem Programm • Unterstützung für unterschiedliche Log-Levels und anderen (eigenen) Parameter, Filters sowie Verwaltung der Logdateien • wird im Kapitel 5.2.3 Logging näher dargestellt
Algorithm:**String**	• Auffinden bestimmten Muster im Text • Wird zum Parsen der Netzwerktopologie verwendet
Asio	• Unterstützung für asynchrone Operationen • Regelmäßige Ausführung bestimmten Funktionen • Asynchrone Netzwerkprogrammierung
Test	• Unit Test einzelnen Klassen • Ermöglicht TDD (Test Driven Development)

Tabelle 7: Zusammenfassung benutzten Boost-Bibliotheken

4.5.3. Lightweight Communications and Marshalling

Das zu erstellende Programm sollte Nachrichten von internen Prozessen auf dem System empfangen können und sie entsprechend an benachbarte RSUs weiterleiten. Eine Übergabe der empfangenen Benachrichtigungen an weitere Programme sollte auch möglich sein (wird aber zurzeit einfach als Parameter beim Programmaufruf übergeben). Es müsste überlegt werden, wie der Aufbau einer solchen Nachricht gestaltet wird. Es wird davon ausgegangen, dass zukünftig auch neue Nachrichtentypen benutzt werden können. Um diese Funktionalitäten zu ermöglichen, wurde als Lösung die, bereits in der Firma bekannte, Lightweight Communications and Marshalling Bibliothek gewählt.

Das Hauptziel von LCM ist die Vereinfachung der Entwicklung von Systemen, die untereinander Nachrichten austauschen. Die Nachrichten in dieser Bibliothek werden

system- und programmierspracheunabhängig deklariert und anschließend für eine ausgewählte Programmiersprache übersetzt. Die Übersetzung findet automatisch nach der Wahl des Zielsystems. Der Nachrichtenaustausch findet über Channels, die eindeutige Namen besitzen, statt. Typischerweise wird ein Kanal für einen Nachrichtentyp und zwei verbundene Programme verwendet. Alle Daten, die über einen Channel gesendet werden, können geloggt werden – somit ist das schnelle Debugging bei Problemen möglich.

5. Aufgabenumsetzung

5.1. Routingdaemon bird

Wie bereits im Kapitel 4.3.4 erwähnt, zur Bedienung des OSPF-Routingprotokolls kommt die Software Bird zum Einsatz. Die Inbetriebnahme von diesem Programm hat den geschätzten Aufwand deutlich überschritten, deswegen war eine Verlängerung der Bearbeitungszeit dieser Bachelorthesis notwendig.

Mithilfe von Bird wurden zwei von drei Grundanforderungen aus Kapitel 1.2 Aufgabenstellung erfüllt.

5.1.1. Installation

Zur Installation wurde ein fertiges cross-übersetztes Softwarepaket zur Verfügung gestellt. Die Installation wurde mit dem Paketverwaltungsprogramm dnf durchgeführt und hat keine Schwierigkeiten dargestellt.

Allerdings es gab mehrere Hürden bevor die eigentliche Konfiguration dieser Software durchgeführt wurde. Zum einen hat sich die LTE-Internetverbindung als sehr unzuverlässig rausgestellt, so dass eine Änderung der Netzwerktopologie (siehe Kapitel 3.4 Aufbau des Demonstrationsnetzwerkes) nötig war. Deswegen wurde zu Testzwecken für den VPN-Tunnelaufbau ein Ethernet-Anschluss verwendet. Zum anderen ist während des Betriebs ein Switch ausgefallen, was schwer zu entdecken war.

Außerdem mussten Konfigurationen vorgenommen werden, die die Bedienung des OSPF-Protokolls auf den Router zwischen beiden RSUs ermöglichten, was auch mit viel Aufwand verbunden war.

5.1.2. Konfiguration

5.1.2.1. Filterung der Netzwerke

Standardmäßig leitet Bird, die mithilfe von Routingprotokollen empfangenen, Routen an System-Routingtabellen weiter. Dasselbe gilt für die zweite Richtung: jede Systemroute wird nach außen verschickt. Das hat dazu geführt, dass nachdem Bird gestartet wurde, wurde die Kommunikation mit der RSU unterbrochen. Der Grund: Bird hat automatisch den Standard-Gateway von benachbarten Geräten übernommen. Durch den Ansatz von Filter wurde die Anpassung gemacht, dass nur die Netzwerke, die über die Ethernet- bzw. die Tunnel-Schnittstelle von OpenVPN

erreichbar sind, exportiert von System-Routingstabelle und importiert von benachbarten Router werden. Zusätzlich werden keine Änderungen an Standard-Gateway von Bird unternommen.

5.1.2.2. Allgemeine OSPF-Einstellungen

Eine der wichtigsten Einstellungen im OSPF-Protokoll ist die Vergabe der Area-ID. In diesem Projekt arbeiten die RSU-Router im Area 0.0.0.0.

```
protocol ospf rsuOSPF {
        debug all;
        import filter importTapEthNetwokNoGateway;
        export filter exportTapEth;
        area 0.0.0.0 {
                interface "eth0" {
                        cost 5;
                        type bcast;
                        hello 10;
                        transmit delay 5;
                        wait 10;
                        dead 40;
                        stub no;
                };
                interface "tap0" {
                        cost 100;
                        type bcast;
                        stub no;
                        hello 10;
                        transmit delay 5;
                        wait 10;
                        dead 40;
                };
        };
}
```

Abbildung 13: OSPF-Konfiguration im Bird

Eine weitere wichtige Einstellung, die der Erfüllung einer der drei Grundanforderungen dient, ist die Setzung der entsprechenden Kosten für bestimmte Netzwerkschnittstellen. Es wurde entschieden, dass die Ethernet-Schnittstelle den Wert 5 haben wird, und die OpenVPN-Tunnel Schnittstelle den Wert 100. Damit werden die billigere direkte Ethernetverbindungen immer bevorzugt.

Andere Parameter, die gesetzt wurden, können Sie der Abbildung 13: OSPF-Konfiguration im Bird entnehmen.

5.1.2.3. Aktivierung der Weiterleitung zwischen den Schnittstellen

Um Angriffe von außen zu verhindern wurde auf jeder RSU sowie auf dem zentralen xio.vision-Server die Firewallsoftware Iptables installiert. Standardmäßig deaktiviert diese Software die Weiterleitung der Datenpakete zwischen unterschiedlichen Netzwerksschnittstellen, was in diesem Projekt allerdings nicht gewünscht ist. Um diese Standardeinstellung zu überschreiben werden bei jedem Systemstart folgende Befehle ausgeführt:

```
iptables -A FORWARD -i tap0 -o eth0  -j ACCEPT
iptables -A FORWARD -i eth0 -o tap0  -j ACCEPT
```

Diese Änderung wird nicht auf dem Server vorgenommen, da dort hinter der Ethernet-Schnittstelle sich keine RSUs befinden.

5.1.2.4. Aktivierung des Routing im Linux-Kernel

Zusätzlich zu Aktivierung der Weiterleitung zwischen den Schnittstellen (siehe 5.1.2.3) muss auch das Routing im Kernel aktiviert werden. Dies geschieht mit folgendem Befehl:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Im Gegensatz zu 5.1.2.3 muss diese Einstellung auch auf dem Server aktiviert werden.

5.1.2.5. Umstellung des Routingmodus auf Bridgemodus

Während der Inbetriebnahme von Bird hat zu einem Zeitpunkt die Kommunikation von RSUs zum Server funktioniert, andersherum konnte aber keine Verbindung aufgebaut werden. Es wurden mehrere Ideen ausprobiert und erst nach mehreren Arbeitstagen wurde die Ursache gefunden. Es wurde auch die Frage[2], was falsch sein kann, an Bird-Mailinglist[3] gestellt. Die Antwort auf diese Frage wurde mittlerweile zu Bird-FAQ hinzugefügt.

Bevor die Frage beantwortet wurde, wurde die Ursache für dieses Missverhalten bei der MUGLER AG entdeckt. Verantwortlich war die Software OpenVPN, die im Routingmodus interne Mechanismen nutzt, um auf Netzwerke, die über weitere VPN-Clients erreichbar sind, zuzugreifen. Dabei werden die System-Routingtabellen nicht berücksichtigt, obwohl sie durch Bird richtig aktualisiert werden. Die externen Netzwerke müssen in der Server-Konfiguration mit dem Parameter iroute hinzugefügt werden. Eine dynamische Änderung ohne Neustart von OpenVPN ist aber nicht möglich, deswegen kam diese Lösung nicht in Frage.

Glücklicherweise schaffte die Umstellung des Betriebsmodus von OpenVPN von Routing auf Bridge Abhilfe. Dabei wurden die Nachteile aus Kapitel 2.5.3.1 Unterschied zwischen Bridging und Routing akzeptiert.

Das obige Problem stellte die größte Schwierigkeit bei der Aufgabenumsetzung dar.

5.2. Geografisch basiertes Routing

Wie bereits erwähnt für geografisch basiertes Routing wird sich ein zu diesem Zweck erstelltes C++-Programm kümmern. Um die Übersicht zu behalten, wird in weiteren

[2] http://trubka.network.cz/pipermail/bird-users/2018-August/012609.html
[3] https://gitlab.labs.nic.cz/labs/bird/wikis/FAQ#dynamic-routing-issues-with-openvpn

Abschnitten dieser Arbeit der Name „Vermittler“ für dieses Programm verwendet. Die interessantesten Lösungen, die dabei benutzt wurden, sind in diesem Kapitel beschrieben.

5.2.1. Initialisierungsdatei

Der Vermittler startet mit einer Initialisierungsdatei durch die das Verhalten des Programms gesteuert werden kann. In nachfolgender Tabelle 8 Aufbau der Initialisierungsdatei werden die wichtigsten Werte aus dieser Datei kurz besprochen.

Block	Parameter	Bedeutung
main	port	Port für ankommende und ausgehende Nachrichten für Vermittler
	lon	Geografische Länge der RSU
	lat	Geografische Breite der RSU
	delimiter	Abschluss-Trennzeichen für ankommende Nachrichten
logFile	logFile	Ablageort der Logdatei
	severity	Level der Lognachrichten
lcm	channel	Name des Channels für LCM-Nachrichten
	interval	Abstand zwischen Abfragen für neue LCM-Nachrichten, in Sekunden
monitoring	interval	Abstand zwischen Abfragen für neue Nachrichten von externen RSUs, in Sekunden
	command	Befehl, der mit der empfangenen Nachricht von externen RSUs, ausgeführt wird
parsing	regularRouter	Festlegung, wie die RouterID der „normalen“ Router beginnt. Diese Router werden nicht als RSUs behandelt; es werden keine Nachrichten an diese Router gesendet.
	rsuMetric	Festlegung, welche Metrik das „eigene“ Netzwerk eines Routers besitzt. Zusammen mit dem Parameter „network“ dient zur Bestimmung der IP-Adresse einer RSU
	network	Festlegung, der Netzwerksanteil einer IP-Adresse von

		einer RSU. Eindeutige Bestimmung für gesamte Netzwerk

Tabelle 8 Aufbau der Initialisierungsdatei

Der Block *parsing* dient zum Herausfinden der ansprechbaren RSUs durch einen Vermittler aus der Ausgabe von Befehl *birdc show ospf topology*, mit dem die aktuelle Netzwerktopologie angezeigt wird. Eine Beispielausgabe können Sie dem Anhang 3. : Konsolenausgabe: birdc ospf network topology entnehmen.

5.2.2. Spezifikation der Nachrichten

Interne Prozesse können an Vermittler Nachrichten übergeben. Diese Nachrichten bestehen aus drei Parametern, die in folgender Tabelle 9: Aufbau der LCM-Nachricht beschrieben werden.

Parameter	**Typ**	**Bedeutung**
Direction	int8_t	Gewünschte Richtung, an die eine Nachricht gesendet werden soll
Range	int32_t	Maximale Entfernung zur benachbarten RSU
Content	string	Die eigentliche Nachricht

Tabelle 9: Aufbau der LCM-Nachricht

Der Vermittler wertet diese Nachrichten aus und anschließend sendet den gewünschten Inhalt an ausgewählte Nachbarn, falls es die in entsprechender Richtung und Entfernung gibt

Aus den drei Parametern aus der Tabelle 9: Aufbau der LCM-Nachricht wurde folgende LCM-Struktur definiert:

```
package rsu_lcm;
struct message
{
      int16_t direction;
      int64_t distance;
      string content;
}
```

Anschließend wurde mit dem Befehl:

```
lcm-gen –x package.lcm
```

entsprechende C++ Headerdatei erstellt.

5.2.2.1. Bestimmung der Senderichtung

Zur Speicherung der Richtung wurde folgende Enum im Vermittler erstellt:

```
enum class Direction
{
	NORTH = 1 << 0,//!< NORTH
	EAST = 1 << 1, //!< EAST
	SOUTH = 1 << 2,//!< SOUTH
	WEST = 1 << 3  //!< WEST
};
```

Durch die binäre Verschiebung bekommen die Richtungen entsprechend die Werte 1,2,4 und 8. Es wurde der binäre Operator Und für diesen Enum überladen:

```
inline Direction operator&(Direction a, Direction b)
{
	return static_cast<Direction>(static_cast<int>(a) &
	static_cast<int>(b));
}
```

Und somit lassen sich mehrere Richtungen für eine Nachricht gleichzeitig setzen:

```
bool Rsus::isRightDirection(const s_rsu& remoteRsu, Direction direction)
{
	if( (direction&Direction::NORTH)==Direction::NORTH)
	{
		if(remoteRsu.lat > this->itself.lat)
			return true;
	}
	//...
```

Zum Beispiel der Wert 3 für den Parameter Direction enspricht den Richtungen Norden und Osten. In der Funktion *isRightDirection*, wird für jede RSU entschieden, ob sie sich in gewünschter Richtung befindet.

5.2.2.2. Bestimmung der Entfernung zwischen zwei RSUs

Zur Bestimmung der Entfernung zwischen zwei geographischen Koordinaten wird die Haversine-Formel verwendet, die wie folgt definiert ist:

$$d = 2r\arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)}\right)$$
$$= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Formel 1: Haversine-Formel

Die Umsetzung dieser Formel findet in der Funktion *Rsus::distanceEarth* statt. Anschließend wird verglichen, ob die Entfernung zwischen den beiden RSUs kleiner ist, als die gewünschte maximale Entfernung, die in der LCM-Nachricht festgelegt

wurde. Ist das der Fall und entspricht die Richtung zu dieser RSU der gewünschten Richtung, so wird die Nachricht an diese RSU durch den Vermittler gesendet.

5.2.3. Logging

Wie bereits erwähnt für Loggingzwecke wurde die Bibliothek Boost Log benutzt. Diese Library ermöglicht umfangreiche Konfiguration der Logeinträge, die im folgenden Unterkapitel näher dargestellt wird.

Die, im Rahmen dieser Bachelorthesis erstellte Erweiterung der Log-Klasse wird zukünftig auch in weiteren Projekten verwendet.

5.2.3.1. Funktionsweise

Log-Einträge bestehen aus Attributen, die entweder global oder in bestimmten Threads oder Klassen gesetzt werden können. Den Attributen folgt die eigentliche Meldung. Ein Beispiel solchen Log-Eintrags sehen Sie unten:

```
[1] [] [] [2018-08-23 09:11:50] [debug] :    Server started on port: 56333
<Server.cpp:14>
[2] [] [] [2018-08-23 09:11:50] [trace] :    New Connection created
<Connection.cpp:15>
[3] [] [] [2018-08-23 09:11:50] [trace] :    Listening for new connections
<Server.cpp:38>
[4] [] [] [2018-08-23 09:11:50] [warning] :    Interval timeout greater
than interval for lcm handling <LcmHandler.cpp:29>
[5] [] [] [2018-08-23 09:11:50] [debug] :    Birdc topology changed
<RsuDetector.cpp:33>
[6] [] [] [2018-08-23 09:11:50] [debug] :    Found itself
<BirdcParser.cpp:55>
[7] [] [] [2018-08-23 09:11:50] [trace] :    12.92.50.8
<BirdcParser.cpp:56>
```

In zweiten und dritten eckigen Klammern sollte entsprechend *ThreadID* und *ThreadName* stehen. Aufrung davon, dass der Vermittler nur einen Thread nutzt, wurden diese Attribute nicht gesetzt und deswegen sind diese Felder leer.

Die benutzerdefinierte Attribute werden durch den Aufruf der Funktion *add_global_attribute* definiert:

```
boost::log::core::get()->add_global_attribute("File",
      boost::log::attributes::mutable_constant<std::string>(""));
```

Zusätzlich dazu können Default-Attribute (Thread-ID, Zeitstempel, Zeilennummer) mit einem Funktionsaufruf hinzugefügt werden:

```
boost::log::add_common_attributes();
```

Die Aktivierung der Attribute bedeutet allerdings nicht, dass sie automatisch in den Log-Meldungen erscheinen. Dazu ist die Definition eines Formatters nötig, der den Operator % überlädt mit den eigentlichen Attribut-Objekten:

```
auto fmtTimeStamp = boost::log::expressions::format_date_time<
      boost::posix_time::ptime>("TimeStamp", "%Y-%m-%d %H:%M:%S");

boost::log::formatter logFmtFile = boost::log::expressions::format(
      "[%1%] [%2%] [%3%] [%4%] [%5%] :    %6% <%7%:%8%>") % lineId
      % threadName % threadId % fmtTimeStamp % fmtSeverity
      % boost::log::expressions::smessage % filenameLog % lineNo;
```

Abschließend muss noch definiert werden, wo die Log-Dateien geschrieben werden:

```
auto fsSink = boost::log::add_file_log(boost::log::keywords::file_name =
      filename, boost::log::keywords::auto_flush = true,
      boost::log::keywords::rotation_size = 10 * 1024 * 1024,
      boost::log::keywords::min_free_space = 30 * 1024 * 1024,
      boost::log::keywords::open_mode = std::ios_base::app);
fsSink->set_formatter(logFmtFile);
```

Im obigen Beispiel sehen Sie zusätzlich die Aktivierung der Rotation und Kompression der Logdateien.

In der Log-Klasse wurden zwei Ziele der Logmeldungen definiert: eine Datei und der Terminal, in dem das Programm gestartet wird. Da im Terminal der Attribut *LineID* unnötig ist, wurde für dieses Ziel ein zusätzlicher Formatter definiert.

Zusätzlich dazu, Boost Log unterstützt Verwendung von Filter, die Aussortierung bestimmten Nachrichten ermöglicht:

```
boost::log::core::get()->set_filter(
      boost::log::trivial::severity >= severityLevel);
```

Eine sehr interessante Möglichkeit stellt die Verwendung von Scopes zur Verfügung, die die eine Anzeige der Aufrufreihenfolge von Funktionen ermöglicht. Besonders nützlich kann das bei der Entwicklung von Programmen für Embedded Systems sein, wo kein Debugging möglich ist. Da der Vermittler auch für traditionelle x86-Prozesore übersetzt wird, ist das Debugging direkt aus der IDE Eclipse während der Entwicklung möglich und deswegen wurde auf Scopes in dieser Klasse verzichtet.

5.2.3.2. Interessante Ansätze

Außer der Klassedefinition wurde ein paar Konstrukte erstellt, die die Verwendung der Log-Klasse deutlich erleichtern. Unten sind die interessantesten Tricks.

Zunächst wurde eine map mit Logging-Level definiert:

```
static std::map<std::string, boost::log::trivial::severity_level>
severiyLevel={
            {"trace", boost::log::trivial::trace},
            {"debug", boost::log::trivial::debug},
            {"info", boost::log::trivial::info},
            {"warning", boost::log::trivial::warning},
            {"error", boost::log::trivial::error},
            {"fatal", boost::log::trivial::fatal}
};
```

Dadurch lässt sich die Log-Klasse einfach aus den Werten aus der Initialisierungsdatei erstellen:

```
MyLogger::init(severiyLevel[pt.get<std::string>("log.severity")],
            pt.get<std::string>("log.logFile"));
```

Außerdem wurde eine Template erstellt, die das Hinzufügen von beliebigen Attributen zu Log-Nachrichten ermöglicht:

```
template<typename ValueType>
ValueType set_get_attrib(const char* name, ValueType value) {
   auto attr =
boost::log::attribute_cast<boost::log::attributes::mutable_constant<ValueTy
pe>>(boost::log::core::get()->get_global_attributes()[name]);
   attr.set(value);
   return attr.get();
}
```

Und anschließend wurde eigene Makro zur Erstellung der Log-Einträge mit zusätzlichen Informationen über Quelle der Log-Nachricht erstellt:

```
#define MY_LOG(sev) \
   BOOST_LOG_STREAM_WITH_PARAMS( \
               ::boost::log::trivial::logger::get(),\
         (set_get_attrib("File", MyLogger::pathToFilename(__FILE__))) \
         (set_get_attrib("Line", __LINE__)) \
         (::boost::log::keywords::severity = (boost::log::trivial::sev)) \
   )
```

5.2.4. Der Server - Verwaltung der einkommenden Verbindungen

Für einkommende Verbindungen mit Nachrichten von entfernten RSUs sorgt die Klasse Server, die die Boost Asio Bibliothek nutzt. Wie bereits im Kapitel 4.5.2.2 erwähnt, dient sie zur Erstellung von asynchronen Operationen. Asynchron heißt, ereignisgesteuert: es wird etwas gemacht, wenn etwas passiert, wobei das Ereignis in der Zukunft stattfinden kann. Durch die Verwendung dieser Bibliothek wird nicht gewartet, bis ein Ereignis geschieht, sondern es werden im Hintergrund weitere Operationen durchgeführt.

Programme, die zur asynchronen Datenverarbeitung Boost.Asio einsetzen, erstellen immer wenigstens ein Objekt der *Klasse io_service. io_service* stellt die Verknüpfung

zur Betriebssystemschnittstellen zur Verfügung. Über diesen *io_service* wird auf unterschiedliche betriebssystem-abhängige und unabhängige I/O-Objekte zugegriffen. Dazu zählen unter anderem serielle Schnittstellen, Dateien aber auch TCP- und UDP-Ports. Im Vermittler wird auf zwei Typen von I/O-Objekten zugegriffen: zum einen auf *steady_timer*, der Ausführung von Funktionen in regulären Zeitspannen ermöglicht, und zum anderen eben auf TCP-Ports zur Empfangen und Versenden der Nachrichten.

Da die Funktionen, die asynchron ausgeführt werden, zumeist zeitverzögert sind, kann es sein, dass bei klassischer Übergabe der Objekte für diese Funktion als Referenz, diese Objekte zum Zeitpunkt der Funktionsausführung nicht mehr existieren. Das kann zum Speicherzugriffsfehler führen, was das gesamte Programm abstürzen kann.

Abhilfe schaffen bereits im Kapitel 4.5.1.5 erwähnte smarte Pointers. Jede ankommende Verbindung wird getrennt behandelt und dazu wurde die Klasse Connection erstellt, deren Deklaration folgendermaßen aussieht:

```
class Connection
: public std::enable_shared_from_this<Connection>{

public:
	typedef std::shared_ptr<Connection> ConnectionSharedPointer;
	void start();
	static ConnectionSharedPointer create(boost::asio::io_service&
io_service, const std::string& delimiter);
	tcp::socket &socket();
	virtual ~Connection();
private:
	void handle_read(const boost::system::error_code error, std::size_t
bytes_transferred);
	Connection(boost::asio::io_service& io_service, const std::string&
delimiter);
	tcp::socket socket_;
	std::string message;
	boost::asio::streambuf buffer;
	const std::string& delimiter;
};
```

Besonders wichtig hier: der Konstruktor wurde als *private* definiert. Die Objekterstellung dieser Klasse findet über die statische Funktion *create* statt. Der Grund dafür ist, dass auf die Funktionen aus dieser Klasse über einen smarten Pointer zugegriffen wird, der sich automatisch löscht, wenn die letzte Referenz zu ihm gelöscht wird. Die Referenz zu diesem Pointer wird immer als Händler für Boost.Asio Ereignisse definiert. Ein Beispiel, die gleichzeitig die Erzeugung von neuen Connection-Objekten darstellt:

```
void Server::start_accept() {
      Connection::ConnectionSharedPointer new_connection =
Connection::create(
      acceptor_.get_io_service(), this->delimiter);
      acceptor_.async_accept(new_connection->socket(),
            boost::bind(&Server::handle_accept, this, new_connection,
            boost::asio::placeholders::error));
}
```

Weitere Übergabe von diesem Pointer für asynchrone Operationen findet dann innerhalb der Klasse Connection statt:

```
void Connection::start() {
      boost::asio::async_read_until(socket_, buffer, delimiter,
            boost::bind(&Connection::handle_read, shared_from_this(),
            boost::asio::placeholders::error,
            boost::asio::placeholders::bytes_transferred));
}
```

Die *bind*-Funktion erwartet als zweiten Parameter einen Pointer zum Objekt auf dem die Funktion aus erstem Parameter aufgerufen wird. In diesem Fall wird das ein smarter Pointer, der durch die Funktion *shared_from_this* erstellt wird. Dies ist möglich, durch die Vererbung der Klasse *enable_shared_from_this* durch die Klasse Connection, was Sie aus obiger Header-definition entnehmen können.

Gleiche Vorgehensweise wurde bei der Erstellung von ausgehenden Verbindungen verwendet.

6. Dokumentation

Die Dokumentation des Programmes erfolgte mithilfe der Software Doxygen. Dieses Programm erstellt automatisch aus Kommentaren des Quelltextes Webseiten, die einzelne Komponenten farbig und dadurch auch übersichtlicher darstellen. Zusätzlich sind alle Abhängigkeiten aufgeführt und können über Links aufgerufen werden.

Abbildung 14: Logo des Projektes Doxygen aus [19]

Von Anfang an wurde der C++ Quellcode entsprechend der Vorgaben von Doxygen formatiert, so dass die Erstellung der Webseiten einwandfrei funktioniert hat. Genauere Beschreibung einzelnen Klassen war allerdings nach der Entwicklung noch zusätzlich nötig.

Als Anhang 1. : Dokumentation mit Doxygen finden Sie ein paar Seiten auf diese Art und Weise erstellter Dokumentation.

Die Installation und Konfiguration der Software Bird wurde auf einer firmeneigenen Wiki beschrieben. Dabei wurden alle nötigen Systemänderungen erwähnt, inklusive der Umstellung der OpenVPN-Software auf Bridging-Modus. Im Anhang 2. : Dokumentation in firmeneigener Wiki wurde ein Screenshot dieser Wiki-Seite beigefügt.

7. Test

Die Tests dienen dem Ziel während der Produktentwicklung gestellte Anforderungen an das Produkt zu überprüfen und ein fehlerfreies Produkt am Markt zu etablieren. Die Testergebnisse stellen eine Kenngröße der Qualitätsmessung dar.

Bei der Testdurchführung müssen folgende Grundsätze beachtet werden:

- Es ist nicht möglich eine Software vollkommen zu testen.
- Ein Test beweist nicht, dass eine Software fehlerfrei ist.
- Je mehr Fehler ein Test gefunden hat, desto größere ist die Wahrscheinlichkeit, dass weitere Fehler vorhanden sind.
- Je später ein Fehler während des Entwicklungszyklus eines Produktes festgestellt wird, desto größere Kosten verursacht die Behebung.

7.1. Unit Test

Für den Quelltext von Vermittler wurden mithilfe von der Bibliothek Boost Test mehrere Unit Tests geschrieben. Teilweise wurde auch Test Driven Development eingesetzt, d.h. zuerst wurden die Tests geschrieben und erst danach der eigentliche Code, der die Voraussetzungen zur Bestehung der Testfälle erfüllte.

7.2. Systemtest

Während des Systemtests wurde das gesamte integrierte System auf korrekte Funktion überprüft. Für Systemtests gilt, dass die Testumgebung möglichst gleich, wie die zukünftige Produktivumgebung beim Kunden sein soll. Leider gibt es zurzeit nur zwei einsatzfähige RSUs bei der MUGLER AG und alle Tests wurden nur mit diesen beiden Geräten durchgeführt.

Die Systemtests wurden mit der Black-Box Methode gemacht. Das heißt: es wurde ohne Berücksichtigung des internen Aufbaus der Softwareelemente die korrekte Arbeit getestet.

Die Ergebnisse von diesen Tests zeigten die Erfüllung der vordefinierten Voraussetzungen. Es wurden keine Fehler entdeckt. Die, im Rahmen dieser Bachelorarbeit erstellte Software, kann in den regelmäßigen Betrieb überführt werden.

8. Ausblick

Bereits während der Entwicklungsarbeiten an Vermittler wurde eine Reihe von möglichen Verbesserungen dieser Software aufgestellt. In folgenden Unterkapiteln werden die wichtigsten Ideen dargestellt.

8.1. Verwendung von mehreren Threads

Durch die Verwendung von multiplen Threads kann die gesamte Performance des Programms verbessert werden. Mögliche Aufteilung des Programms in Threads wäre jeweils ein Thread für die Verwaltung der eingehenden Verbindungen, ein Thread für die Verwaltung der ausgehenden Verbindungen, ein Thread für den Empfang von LCM-Nachrichten und schließlich einer zur Überwachung der Änderungen der Netzwerkstruktur.

Zu diesem Zweck können native Threads der Programmiersprache C++ benutzt werden oder bereits vorhandene Klasse der Bibliotheken STL oder Boost, die die Verwendung und Synchronisation von Threads deutlich erleichtern.

Ein anderer Ansatz wäre die Verwendung von mehreren *io_services* für unterschiedliche Aufgaben. Diese Möglichkeit wurde aber bisher nicht tiefer eingegangen.

8.2. Verbindungsweg in Abhängigkeit von Anforderungen

Diese Funktion wurde bereits während der Spezifikation in Betracht gezogen. Es ist möglich für die Verbindung zwischen zwei RSUs mehrere Wege zu nutzen. Der einfachste Fall wurde in Kapitel 3.4 dargestellt. Dabei können zwei RSUs entweder direkt über Richtfunk oder über den OpenVPN-Server kommunizieren.

Nach der Inbetriebnahme von mehreren RSUs kann es sein, dass eine RSU eine andere über mehrere andere RSUs ansprechen kann. In diesem Fall kann solche Verbindung als instabil betrachtet werden. Dasselbe kann auch für die Richtfunkverbindung gelten, wenn die beiden Router weiter voneinander entfernt sind bzw. wenn auf dem Verbindungsweg Störungen auftreten wie zum Beispiel fahrende LKWs.

Deswegen könnte man in Abhängigkeit von Nachrichtentyp eine bestimmte Verbindung bevorzugen. Sollte ein bestimmter Nachrichtentyp sehr häufig

vorkommen und dabei nicht so relevant sein, kann man den Weg über mehrere RSUs bevorzugen und damit Kosten, die durch die Nutzung von OpenVPN-Tunnels entstehen, vermeiden. Wobei, was sich bereits während der Entwicklungsarbeiten rausgestellt hat, der VPN-Tunnel über LTE-Mobilfunknetz kann auch nicht als zuverlässig betrachtet werden.

Nach erster Überlegung können vier Kriterien zukünftig für bestimmte Nachrichtentypen berücksichtigt werden:

- Kosten
- Latenz
- Zuverlässigkeit
- Bandbreite

Es wurde auch eine kurze Liste möglicher Use Cases erstellt, die sich in Abhängigkeit von Anforderungen an Zuverlässigkeit und Latenz auf einem Diagramm darstellen lassen. Siehe folgende Abbildung 15: Aufteilung der Nachrichten nach Anforderungen.

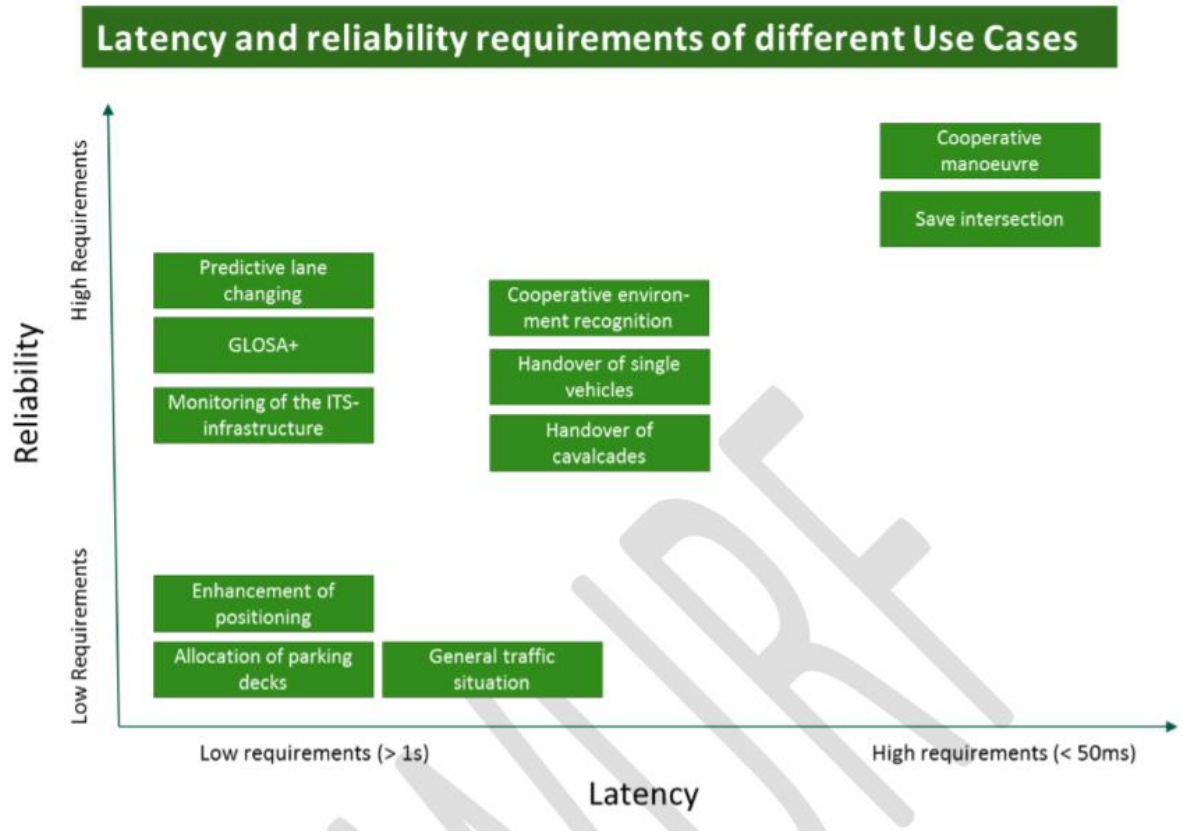


Abbildung 15: Aufteilung der Nachrichten nach Anforderungen aus [16]

Diese Use Cases kann man als weiteren Parameter in der LCM-Nachricht definieren und in Abhängigkeit davon die ausgehende Schnittstelle entsprechend wählen.

Beim Ausprobieren der Einrichtung von mehreren Gateways (z.B. VPN-Tunnelverbindung und Ethernet) hat sich herausgestellt, dass Linux standardmäßig diese Option nicht unterstützt. Dazu sind zusätzliche Programme notwendig, die Verwaltung von mehreren System-Routingtabellen ermöglichen. Die ständige Anpassung dieser Routingtabellen bei Änderungen der Netzwerkstruktur, die durch Bird propagiert werden, ist mit sehr großem Aufwand verbunden, deswegen wurde zuerst auf diese Idee verzichtet.

8.3. Fahrzeuge als Router

Eine weitere Idee wäre die Weiterleitung der Nachrichten zwischen zwei RSUs über Fahrzeuge, die sich in der Reichweite beider RSUs befinden. Dazu kann man die Technologie WLAN802.11p, die Kommunikation zwischen Fahrzeugen ermöglicht benutzen.

Wenn Informationen über Fahrzeuge und derer Fahrrichtung auf RSUs vorhanden sind, kann man mit entsprechenden Algorithmen ermitteln, wann die Kommunikation zwischen zwei RSUs über ein Auto möglich ist.

Aufgrund davon, dass der WLAN802.11p-Standard und selbst der Algorithmus zur Bestimmung, ob Kommunikation über Fahrzeuge möglich ist, sehr kompliziert ist und dadurch die Entwicklungsarbeiten an einer solchen Lösung sehr aufwendig sein können, wurde diese Möglichkeit zuerst ausgeschlossen.

9. Zusammenfassung

Trotz der zeitlichen Verzögerung wegen der Probleme mit der Konfiguration von Softwarepaketen Bird und OpenVPN, wurde ein funktioneller Prototyp entwickelt, die alle gestellten Voraussetzungen ausfüllt.

Zukünftig muss die, in Rahmen dieser Bachelor Thesis, erarbeitete Aufgabe, in größeren Netzwerken und in realen Bedingungen getestet werden.

Des Weiteren wurde bereits während der Entwicklungsarbeit ein paar Ideen entwickelt, die umgesetzt werden können. Die Entscheidung, ob das Programm Vermittler weiterentwickelt wird, ist noch nicht gefallen. Als nächster Schritt wird ein Format für die übermittelte Nachrichten erstellt.

Literaturverzeichnis

1. **waldner.** *OpenVPN and iroute.* [Online] 5. November 2009. [Zitat vom: 2018. August 24.] https://backreference.org/2009/11/15/openvpn-and-iroute/.

2. **Krysiak, Karol.** *Sieci komputerowe. Kompedium. Wydanie II.* Gliwice : Helion, 2005. 9788373619951.

3. **Schulte, Wolfgang.** *Handbuch der Routing-Protokolle.* Berlin : VDE, 2016. 9783800740666.

4. **Tom.** Using BIRD to route over OpenVPN tunnels. [Online] 9. Oktober 2014. [Zitat vom: 15. August 2018.] http://www.blissfulidiot.com/2013/10/using-bird-to-route-over-openvpn-tunnels.html.

5. **Klapmuetz.** OpenVPN Bridge. [Online] 24. Juli 2005. [Zitat vom: 15. August 2018.] https://wiki.archlinux.org/index.php/OpenVPN_Bridge.

6. **OpenVPN Inc.** OpenVPN HOWTO. [Online] [Zitat vom: 1. August 2018.] https://openvpn.net/index.php/open-source/documentation/howto.html.

7. OpenVPN Routing. [Online] 7. Juli 2010. [Zitat vom: 13. August 2018.] https://openmaniak.com/openvpn_routing.php.

8. **Huang Albert, Edwin Olson, David Moore.** *LCM: Lightweight Communications and Marshalling.* 2010.

9. **Cadie, Thompson.** Elon Musk: In less than 20 years, owning a car will be like owning a horse. [Online] 4. November 2015. [Zitat vom: 10. Juli 2018.] https://www.businessinsider.com/elon-musk-owning-a-car-in-20-years-like-owning-a-horse-2015-11?IR=T.

10. AW: Strange behavior, cannot reach 4th hop. [Online] 13. August 2018. http://trubka.network.cz/pipermail/bird-users/2018-August/012609.html.

11. **Ondrej, Zajicek.** BIRD FAQ. [Online] [Zitat vom: 18. August 2018.] https://gitlab.labs.nic.cz/labs/bird/wikis/FAQ#dynamic-routing-issues-with-openvpn.

12. **Filip Ondrej, Machek Pavel, Mares Martin, Matejka Jan, Zajicek Ondrej.** BIRD User's Guide (version 2.0.2). [Online] [Zitat vom: 16. Juli 2018.] https://bird.network.cz/?get_doc&f=bird.html&v=20.

13. **Cisco.** [Online] 10. August 2005. [Zitat vom: 6. Juli 2018.] https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html.

14. **Sanders Peter, Singler Johannes.** Kürzeste Wege. [Online] Institut für Theoretische Informatik, Universität Karlsruhe. [Zitat vom: 1. August 2018.] http://algo2.iti.kit.edu/documents/chapter4A.pdf.

15. **Prof. Chopde Nitin, Mr Nichat Mangesh.** Landmark Based Shortest Path Detection by Using A* and Haversine Formula. [Online] 2. April 2013. [Zitat vom: 18. Juli 2018.] http://www.ijircce.com/upload/2013/april/17_V1204030_Landmark_H.pdf.

16. **Projekt IVS-KOM.** *Gesamtsystemspezifikation: sRSU.* [Dokument] 2018.

17. —. *Routing algorithms.* [Dokument] Oberlungwitz : s.n., 2018. 0.2.

18. —. *Komponentenspezifikation: Hybride Kommunikationssteuerung.* [Dokument] 2018.

19. **Doxygen.** Doxygen Manual. [Online] 23. Mai 2018. [Zitat vom: 2018. September 2018.] http://www.doxygen.nl/manual/index.html.

20. **frustschieber.** Dynamisches Routing. [Online] 3. Mai 2017. [Zitat vom: 1. August 2018.] https://wiki.ubuntuusers.de/Dynamisches_Routing/.

21. **Boost.** Boost 1.68.0 Library Documentation. [Online] [Zitat vom: 16. Juli 2018.] https://www.boost.org/doc/libs/1_68_0/.

Anhangsverzeichnis

Anhang 1. : Dokumentation mit Doxygen I

Anhang 2. : Dokumentation in firmeneigener Wiki IV

Anhang 3. : Konsolenausgabe: birdc ospf network topology VI

Anhang 4. : Vermittler: Quelltextausschnitte VII

Anhang 5. : Unit Test: Ergebnisse XI

Ehrenwörtliche Erklärung

„Ich erkläre hiermit ehrenwörtlich“,

1. dass ich meine **Bachelor Thesis** mit dem Thema
„Umsetzung eines Routing-Algorithmus für die smarte Roadside Unit“
Ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

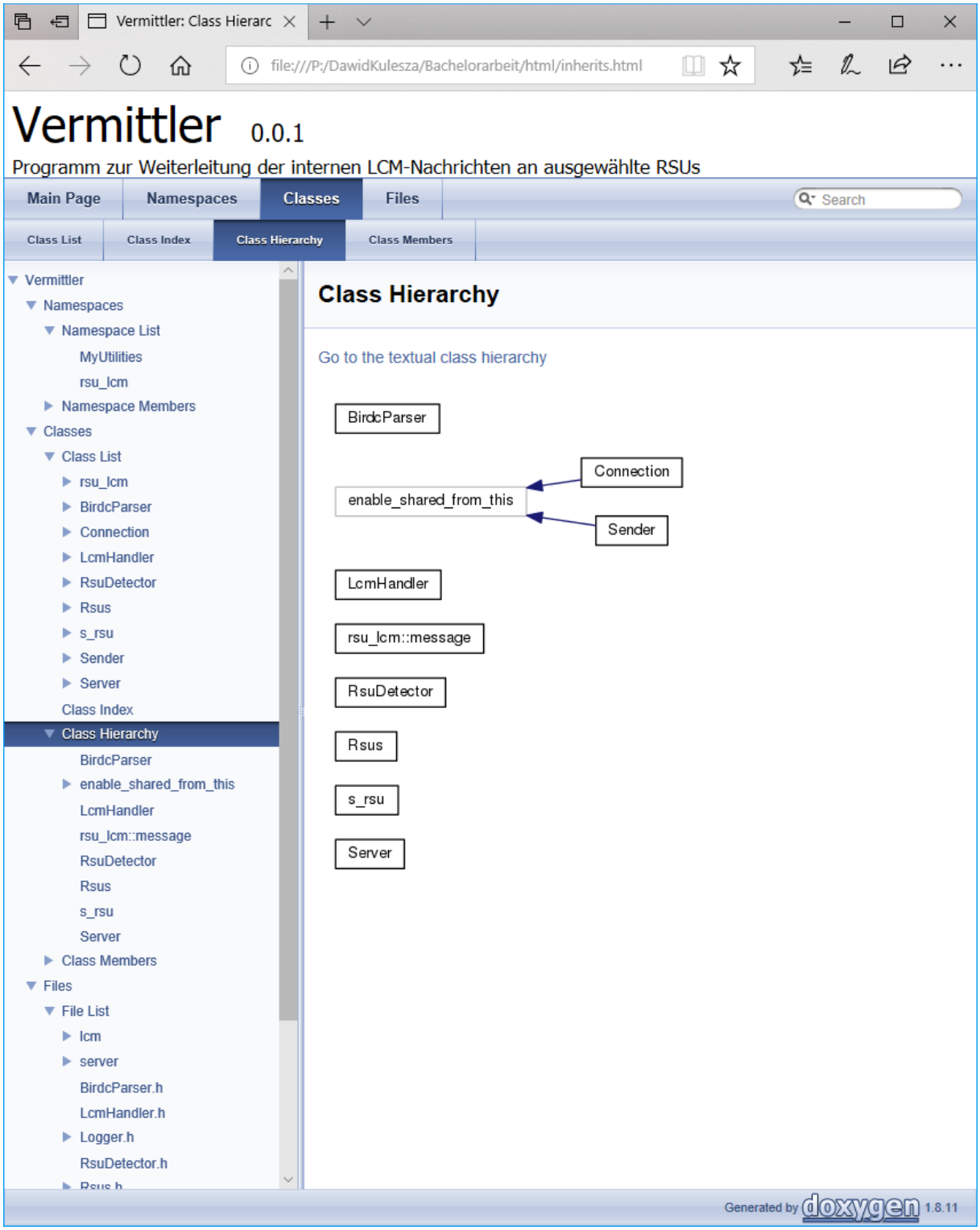
3. dass ich meine **Bachelor Thesis** bei keiner anderen Prüfung vorgelegt habe.

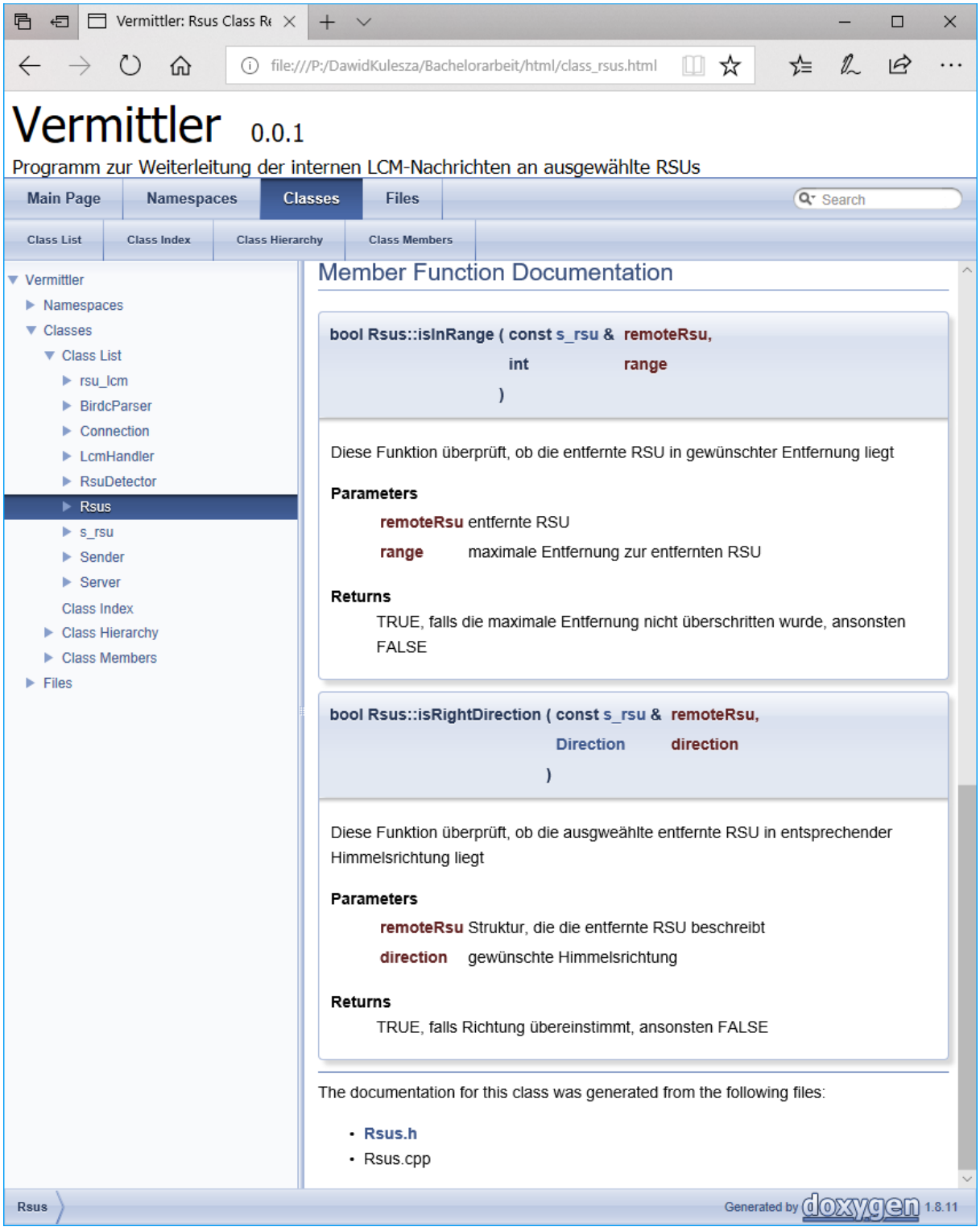
Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

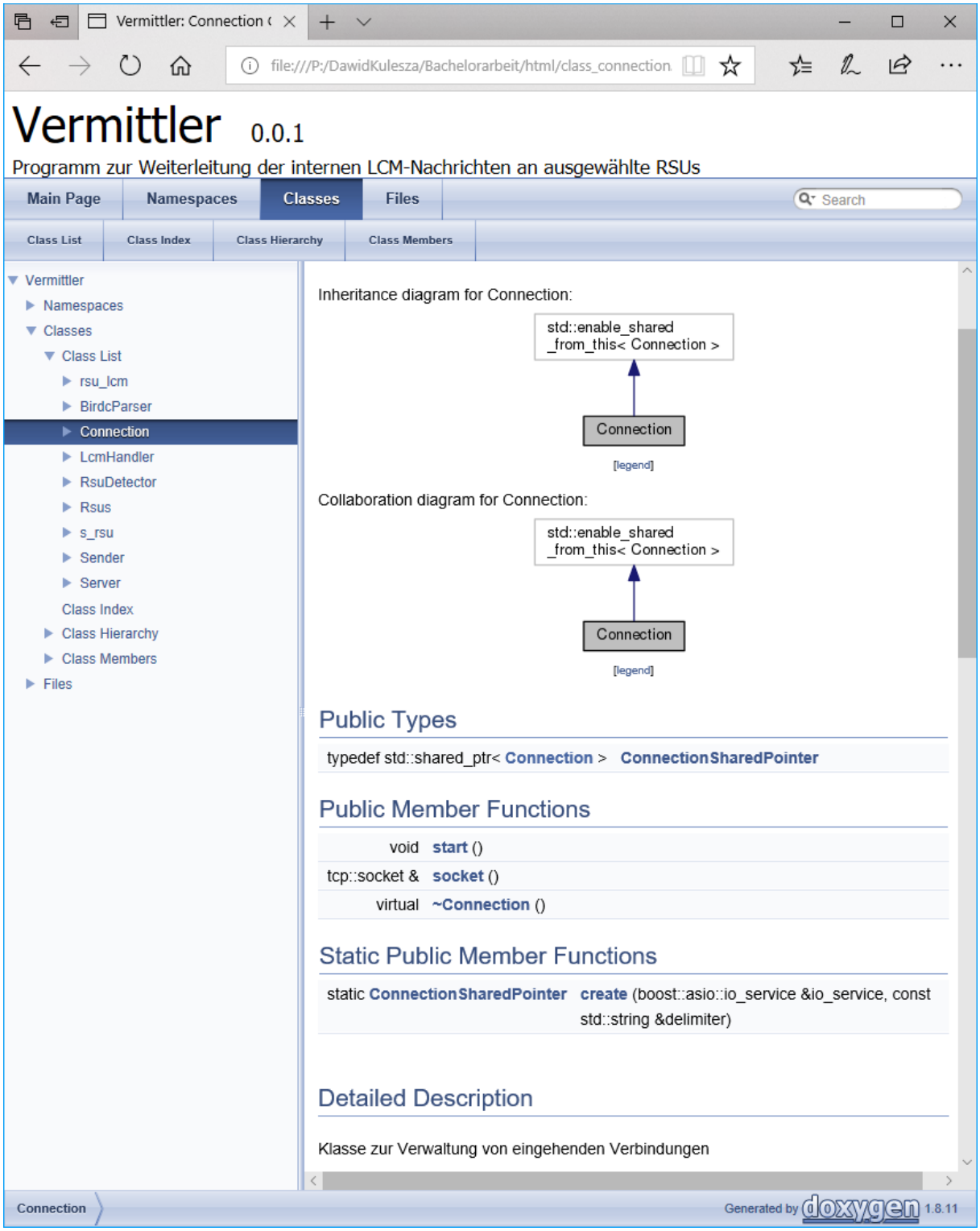
Oberlungwitz, den 06.09.2018 _____

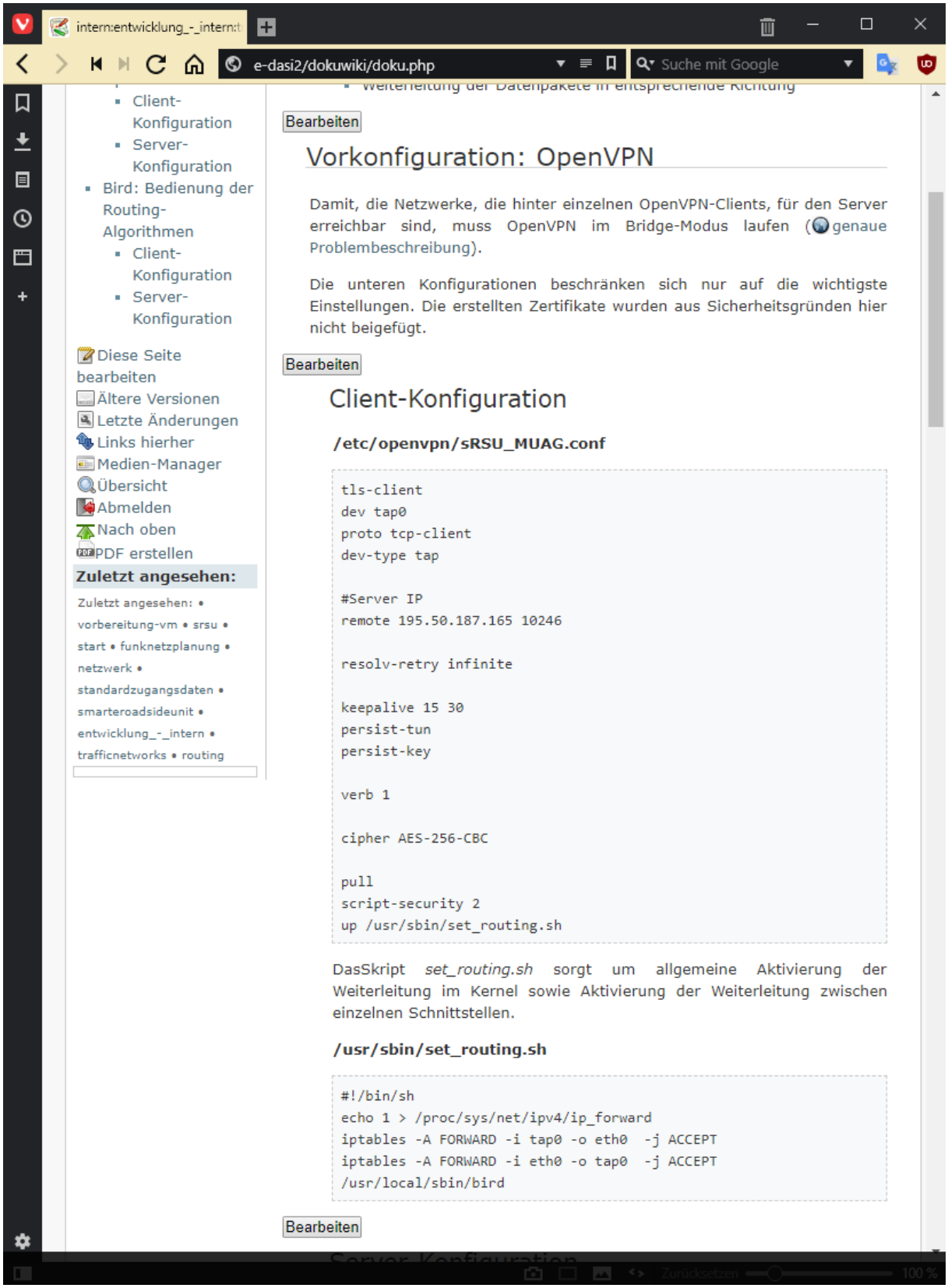
Unterschrift

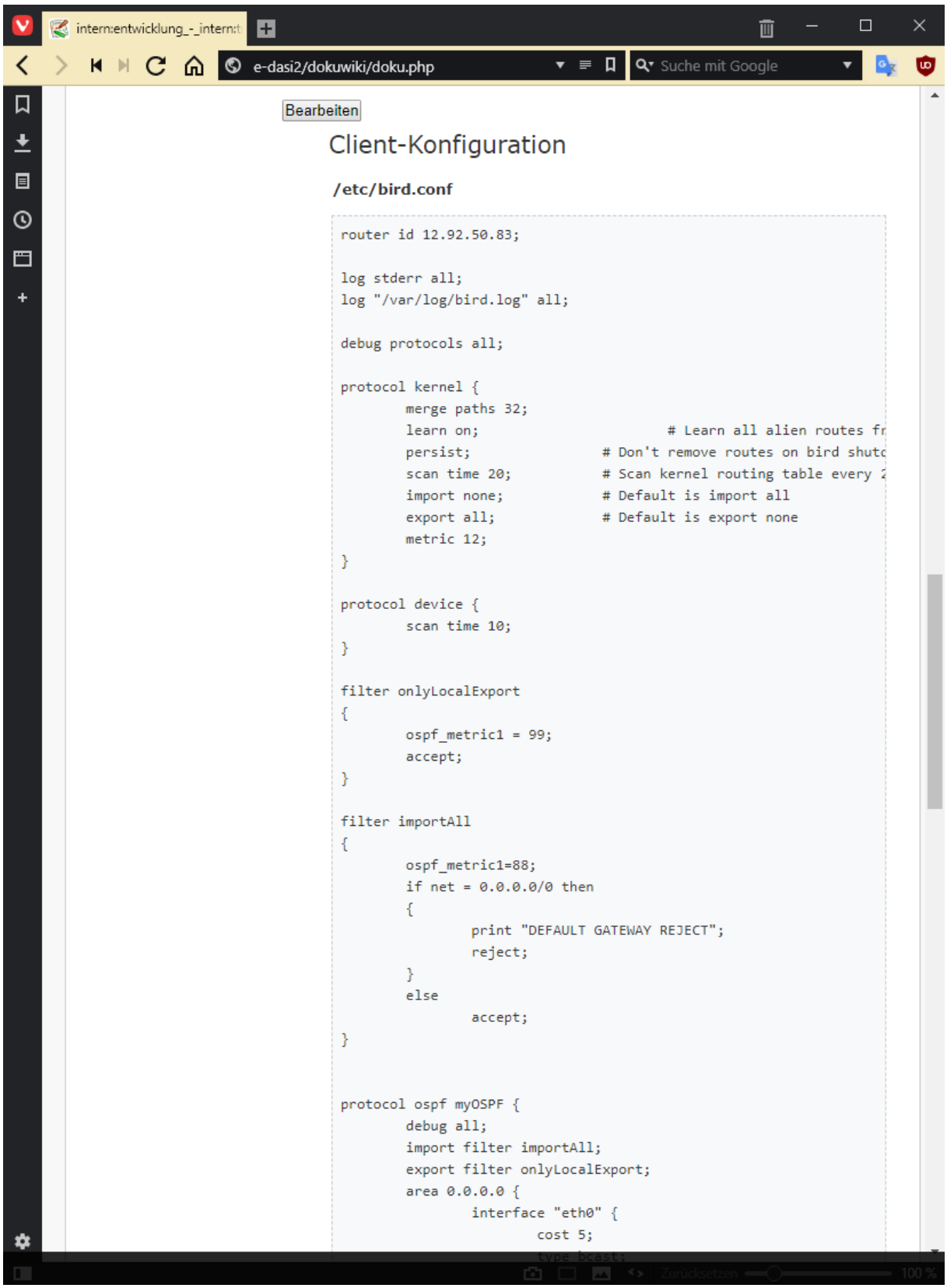
Anhang 1. : *Dokumentation mit Doxygen*



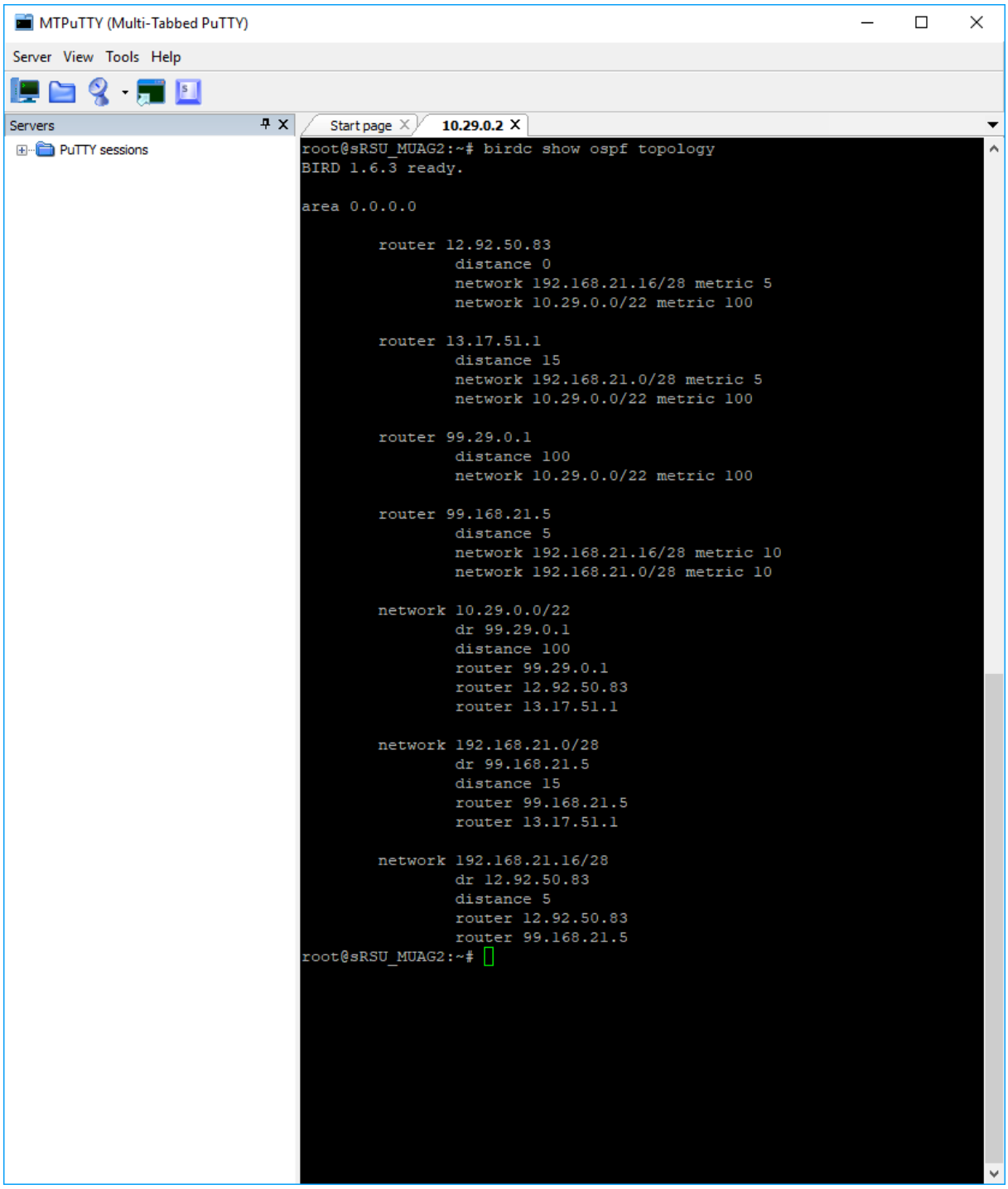

Vermittler: Rsus Class Re
file:///P:/DawidKulesza/Bachelorarbeit/html/class_rsus.html
Vermittler 0.0.1
Programm zur Weiterleitung der internen LCM-Nachrichten an ausgewählte RSUs
Main Page
Namespaces
Classes
Files
Search
Class List
Class Index
Class Hierarchy
Class Members
Vermittler
Namespaces
Classes
Class List
rsu_lcm
BirdcParser
Connection
LcmHandler
RsuDetector
Rsus
s_rsu
Sender
Server
Class Index
Class Hierarchy
Class Members
Files
Member Function Documentation
bool Rsus::isInRange (const s_rsu & remoteRsu,
int range
)
Diese Funktion überprüft, ob die entfernte RSU in gewünschter Entfernung liegt
Parameters
remoteRsu entfernte RSU
range maximale Entfernung zur entfernten RSU
Returns
TRUE, falls die maximale Entfernung nicht überschritten wurde, ansonsten FALSE
bool Rsus::isRightDirection (const s_rsu & remoteRsu,
Direction direction
)
Diese Funktion überprüft, ob die ausgweählte entfernte RSU in entsprechender Himmelsrichtung liegt
Parameters
remoteRsu Struktur, die die entfernte RSU beschreibt
direction gewünschte Himmelsrichtung
Returns
TRUE, falls Richtung übereinstimmt, ansonsten FALSE
The documentation for this class was generated from the following files:
Rsus.h
Rsus.cpp
Rsus
Generated by doxygen 1.8.11


Vermittler: Connection (
file:///P:/DawidKulesza/Bachelorarbeit/html/class_connection
Vermittler 0.0.1
Programm zur Weiterleitung der internen LCM-Nachrichten an ausgewählte RSUs
Main Page
Namespaces
Classes
Files
Search
Class List
Class Index
Class Hierarchy
Class Members
Vermittler
Namespaces
Classes
Class List
rsu_lcm
BirdcParser
Connection
LcmHandler
RsuDetector
Rsus
s_rsu
Sender
Server
Class Index
Class Hierarchy
Class Members
Files
Inheritance diagram for Connection:
std::enable_shared _from_this< Connection >
Connection
[legend]
Collaboration diagram for Connection:
std::enable_shared _from_this< Connection >
Connection
[legend]
Public Types
typedef std::shared_ptr< Connection > ConnectionSharedPointer
Public Member Functions
void start ()
tcp::socket & socket ()
virtual ~Connection ()
Static Public Member Functions
static ConnectionSharedPointer create (boost::asio::io_service &io_service, const std::string &delimiter)
Detailed Description
Klasse zur Verwaltung von eingehenden Verbindungen
Connection
Generated by doxygen 1.8.11


intern:entwicklung_-_intern:t
e-dasi2/dokuwiki/doku.php
Suche mit Google
Client-Konfiguration
Server-Konfiguration
Bird: Bedienung der Routing-Algorithmen
Client-Konfiguration
Server-Konfiguration
Diese Seite bearbeiten
Ältere Versionen
Letzte Änderungen
Links hierher
Medien-Manager
Übersicht
Abmelden
Nach oben
PDF erstellen
Zuletzt angesehen:
Zuletzt angesehen: • vorbereitung-vm • srsu • start • funknetzplanung • netzwerk • standardzugangsdaten • smarteroadsideunit • entwicklung_-_intern • trafficnetworks • routing
Bearbeiten
Vorkonfiguration: OpenVPN
Damit, die Netzwerke, die hinter einzelnen OpenVPN-Clients, für den Server erreichbar sind, muss OpenVPN im Bridge-Modus laufen (genaue Problembeschreibung).
Die unteren Konfigurationen beschränken sich nur auf die wichtigste Einstellungen. Die erstellten Zertifikate wurden aus Sicherheitsgründen hier nicht beigefügt.
Bearbeiten
Client-Konfiguration
/etc/openvpn/sRSU_MUAG.conf
tls-client
dev tap0
proto tcp-client
dev-type tap
#Server IP
remote 195.50.187.165 10246
resolv-retry infinite
keepalive 15 30
persist-tun
persist-key
verb 1
cipher AES-256-CBC
pull
script-security 2
up /usr/sbin/set_routing.sh
DasSkript set_routing.sh sorgt um allgemeine Aktivierung der Weiterleitung im Kernel sowie Aktivierung der Weiterleitung zwischen einzelnen Schnittstellen.
/usr/sbin/set_routing.sh
#!/bin/sh
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A FORWARD -i tap0 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth0 -o tap0 -j ACCEPT
/usr/local/sbin/bird
Bearbeiten


intern:entwicklung_-_intern:t
e-dasi2/dokuwiki/doku.php
Suche mit Google
Bearbeiten
Client-Konfiguration
/etc/bird.conf
router id 12.92.50.83;
log stderr all;
log "/var/log/bird.log" all;
debug protocols all;
protocol kernel {
merge paths 32;
learn on; # Learn all alien routes fr
persist; # Don't remove routes on bird shutc
scan time 20; # Scan kernel routing table every 2
import none; # Default is import all
export all; # Default is export none
metric 12;
}
protocol device {
scan time 10;
}
filter onlyLocalExport
{
ospf_metric1 = 99;
accept;
}
filter importAll
{
ospf_metric1=88;
if net = 0.0.0.0/0 then
{
print "DEFAULT GATEWAY REJECT";
reject;
}
else
accept;
}
protocol ospf myOSPF {
debug all;
import filter importAll;
export filter onlyLocalExport;
area 0.0.0.0 {
interface "eth0" {
cost 5;
Zurücksetzen
100 %

Anhang 3. : *Konsolenausgabe: birdc ospf network topology*



Anhang 4. : *Vermittler: Quelltextausschnitte*

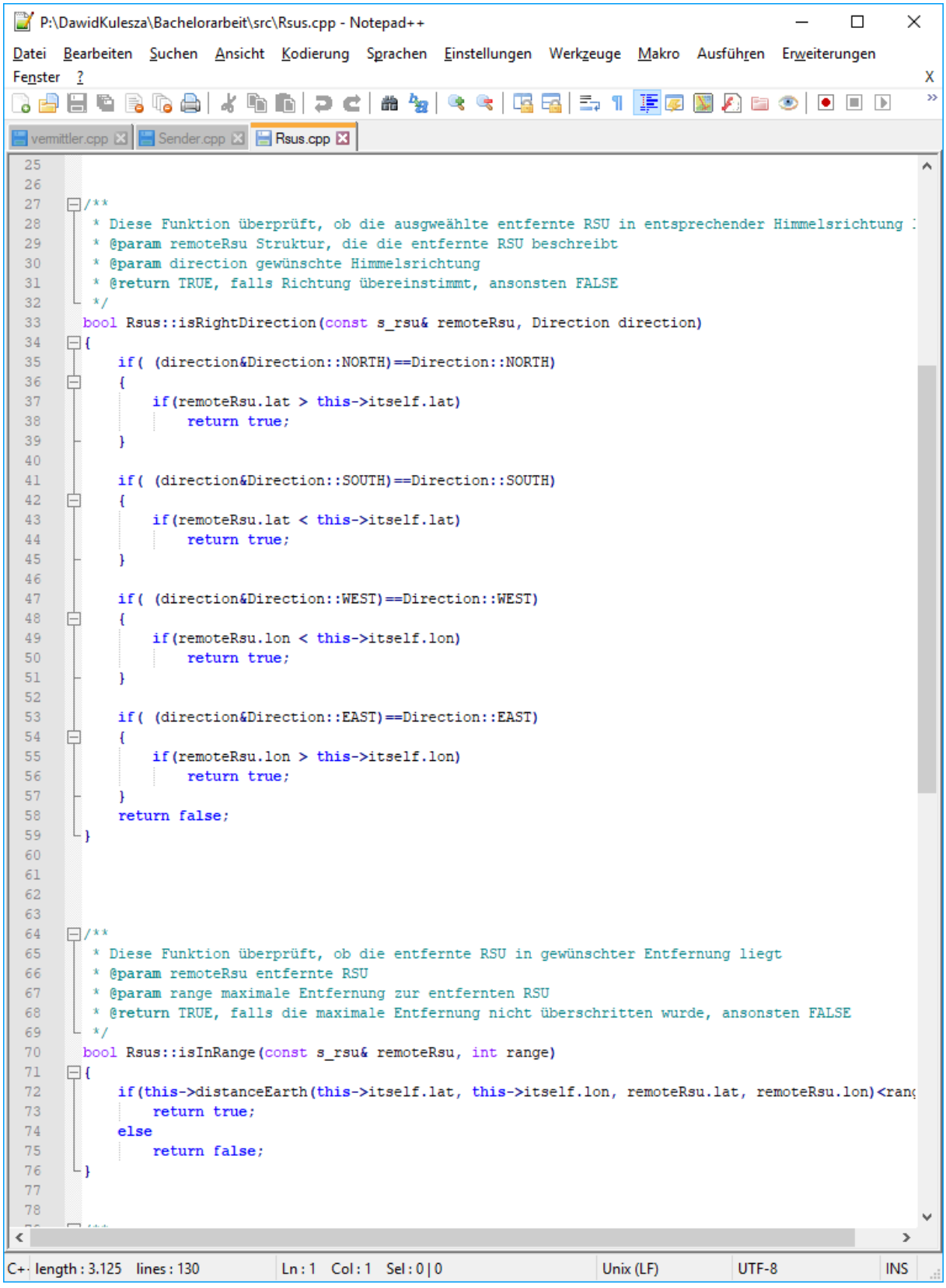
P:\DawidKulesza\Bachelorarbeit\src\vermittler.cpp - Notepad++

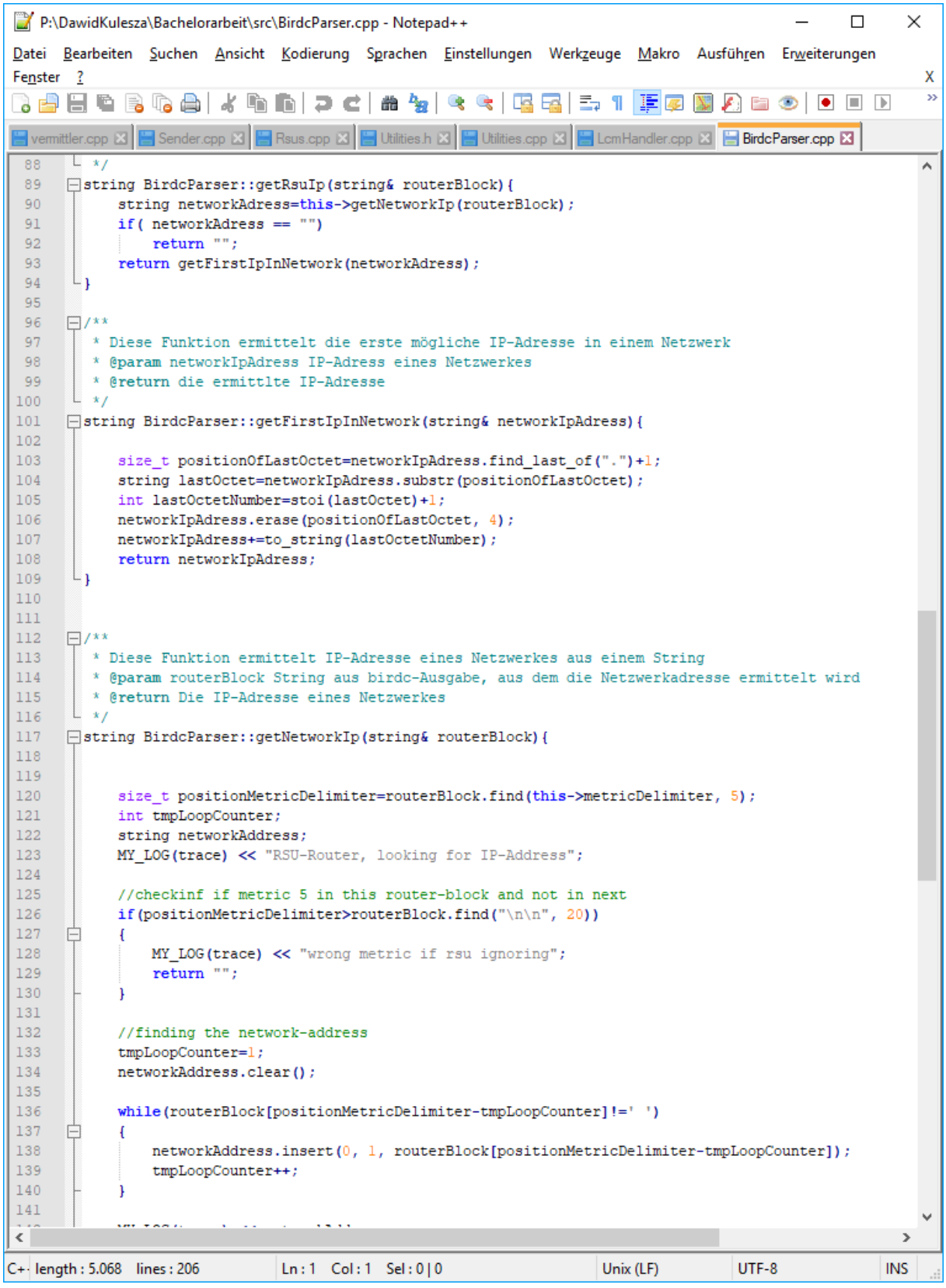
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen
Fenster ?

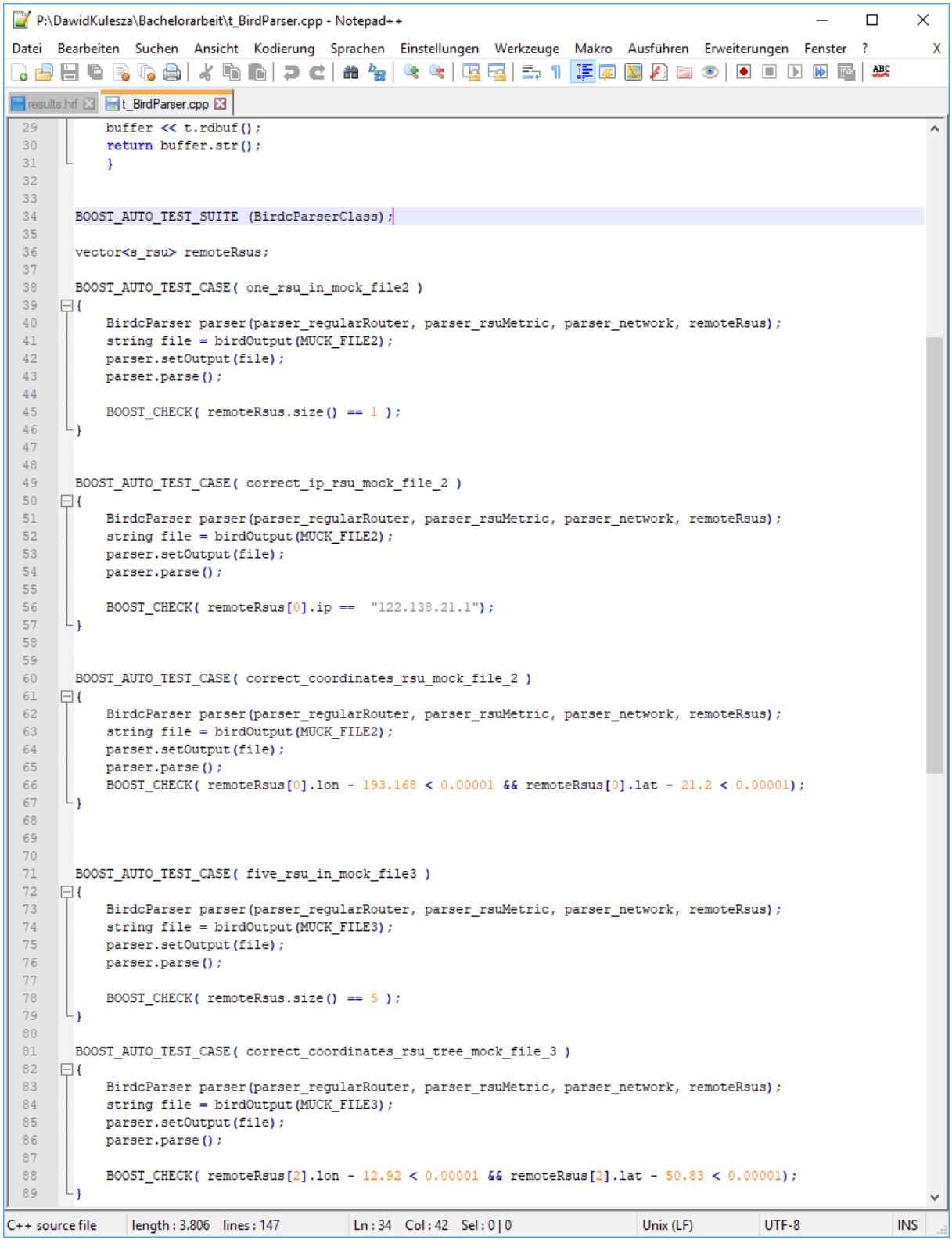
vermittler.cpp

```
// Name        : vermittler.cpp
// Author      : Dawid Kulesza
// Version     :
// Copyright   : (c) MUGLER AG
// Description : Program to route lcm-messages in desired direction between RSUs
//============================================================================


#include <iostream>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/ini_parser.hpp>

#include "server/Server.h"

#include "LcmHandler.h"
#include "RsuDetector.h"
#include "Logger.h"

using namespace std;

/**
 * Einstiegspunkt des Programms, in dieser Funktion wird die Initialisierungsdatei eingelesen
 * und anschließend werden alle Objekte mit diesen Werten initialisiert.
 * \note Diese Funktion wird nur als Startpunkt des Programms für Compiler definiert. Sie wird n
 * @param argc Anzahl der übergegebenen Parameter (wird automatsich bestimmt)
 * @param argv Parameter, benötigt nur einer: Pfad zur Initialisierungsdatei
 * @return immer 0
 */
int main(int argc, char* argv[]) {
    if(argc <2 )
    {
        cout<<"NO INI-FILE"<<endl;
        return -1;
    }
    boost::property_tree::ptree pt;
    boost::property_tree::ini_parser::read_ini(argv[1], pt);

    MyLogger::init(severiyLevel[pt.get<std::string>("log.severity")], pt.get<std::string>("log.l
    //boost::log::core::get()->add_thread_attribute("ThreadName", boost::log::attributes::consta

    boost::asio::io_service io_service;

    Server listener(io_service, pt.get<int>("main.port"), pt.get<string>("main.delimiter")  );

    Rsus allRsus(io_service, pt.get<float>("main.lon"), pt.get<float>("main.lat"), pt.get<int>("

    LcmHandler myLcmHandler(io_service, pt.get<int>("lcm.interval"), pt.get<int>("lcm.intervalTi

    RsuDetector myRsuDetector(io_service, pt, allRsus.remoteRsus);

    io_service.run();

    return 0;
}
```

C++ length : 1.950 lines : 56 | Ln : 19 Col : 1 Sel : 0 | 0 | Unix (LF) | UTF-8 | INS

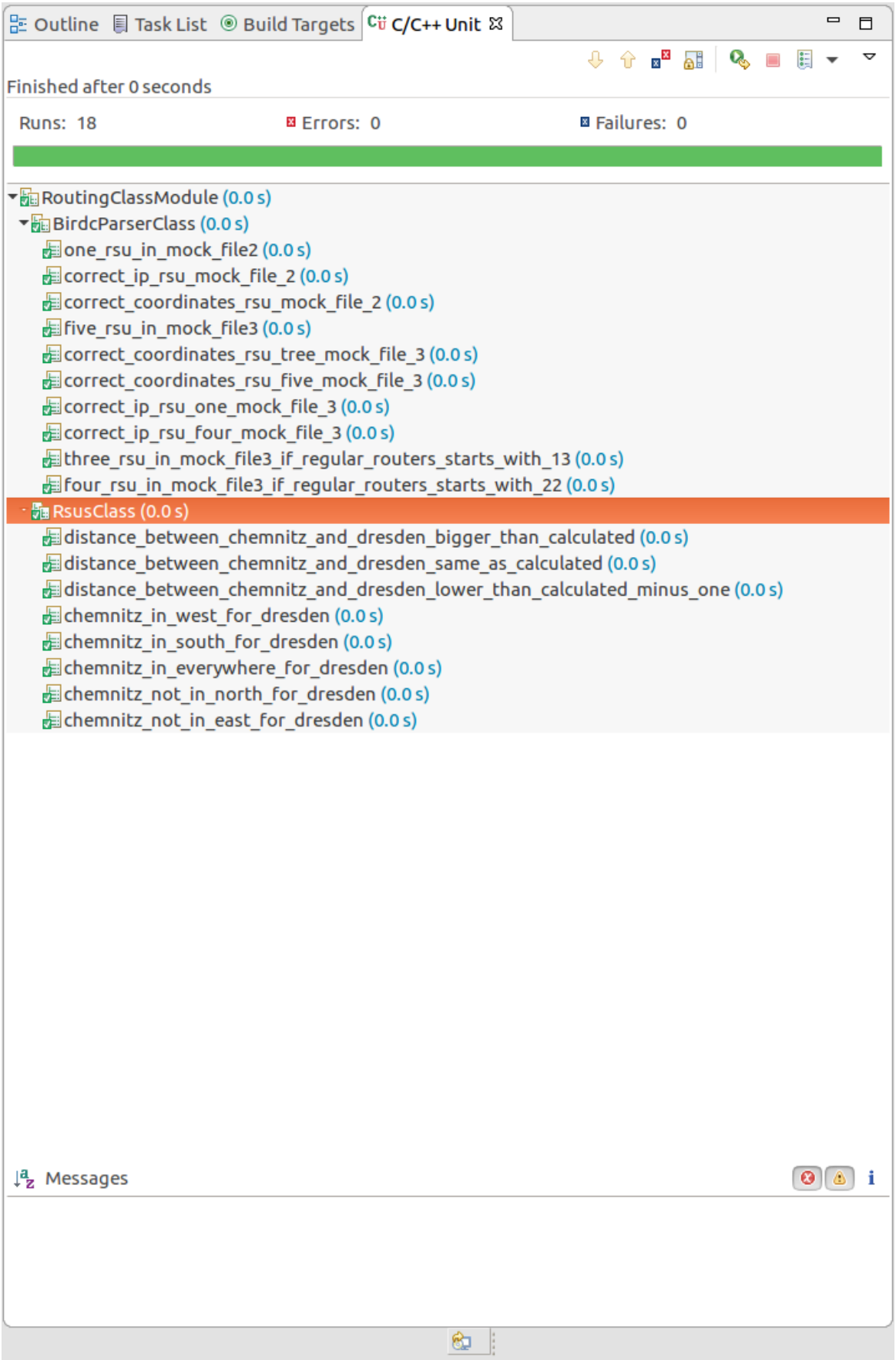

P:\DawidKulesza\Bachelorarbeit\src\Rsus.cpp - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen
Fenster ?
vermittler.cpp
Sender.cpp
Rsus.cpp
25
26
27 /**
28 * Diese Funktion überprüft, ob die ausgwählte entfernte RSU in entsprechender Himmelsrichtung
29 * @param remoteRsu Struktur, die die entfernte RSU beschreibt
30 * @param direction gewünschte Himmelsrichtung
31 * @return TRUE, falls Richtung übereinstimmt, ansonsten FALSE
32 */
33 bool Rsus::isRightDirection(const s_rsu& remoteRsu, Direction direction)
34 {
35 if((direction&Direction::NORTH)==Direction::NORTH)
36 {
37 if(remoteRsu.lat > this->itself.lat)
38 return true;
39 }
40
41 if((direction&Direction::SOUTH)==Direction::SOUTH)
42 {
43 if(remoteRsu.lat < this->itself.lat)
44 return true;
45 }
46
47 if((direction&Direction::WEST)==Direction::WEST)
48 {
49 if(remoteRsu.lon < this->itself.lon)
50 return true;
51 }
52
53 if((direction&Direction::EAST)==Direction::EAST)
54 {
55 if(remoteRsu.lon > this->itself.lon)
56 return true;
57 }
58 return false;
59 }
60
61
62
63
64 /**
65 * Diese Funktion überprüft, ob die entfernte RSU in gewünschter Entfernung liegt
66 * @param remoteRsu entfernte RSU
67 * @param range maximale Entfernung zur entfernten RSU
68 * @return TRUE, falls die maximale Entfernung nicht überschritten wurde, ansonsten FALSE
69 */
70 bool Rsus::isInRange(const s_rsu& remoteRsu, int range)
71 {
72 if(this->distanceEarth(this->itself.lat, this->itself.lon, remoteRsu.lat, remoteRsu.lon)<ran
73 return true;
74 else
75 return false;
76 }
77
78
C++ length : 3.125 lines : 130
Ln : 1 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS


P:\DawidKulesza\Bachelorarbeit\src\BirdcParser.cpp - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen
Fenster ?
vermittler.cpp
Sender.cpp
Rsus.cpp
Utilities.h
Utilities.cpp
LcmHandler.cpp
BirdcParser.cpp
88 */
89 string BirdcParser::getRsuIp(string& routerBlock){
90 string networkAdress=this->getNetworkIp(routerBlock);
91 if(networkAdress == "")
92 return "";
93 return getFirstIpInNetwork(networkAdress);
94 }
95
96 /**
97 * Diese Funktion ermittelt die erste mögliche IP-Adresse in einem Netzwerk
98 * @param networkIpAdress IP-Adress eines Netzwerkes
99 * @return die ermittlte IP-Adresse
100 */
101 string BirdcParser::getFirstIpInNetwork(string& networkIpAdress){
102
103 size_t positionOfLastOctet=networkIpAdress.find_last_of(".")+1;
104 string lastOctet=networkIpAdress.substr(positionOfLastOctet);
105 int lastOctetNumber=stoi(lastOctet)+1;
106 networkIpAdress.erase(positionOfLastOctet, 4);
107 networkIpAdress+=to_string(lastOctetNumber);
108 return networkIpAdress;
109 }
110
111
112 /**
113 * Diese Funktion ermittelt IP-Adresse eines Netzwerkes aus einem String
114 * @param routerBlock String aus birdc-Ausgabe, aus dem die Netzwerkadresse ermittelt wird
115 * @return Die IP-Adresse eines Netzwerkes
116 */
117 string BirdcParser::getNetworkIp(string& routerBlock){
118
119
120 size_t positionMetricDelimiter=routerBlock.find(this->metricDelimiter, 5);
121 int tmpLoopCounter;
122 string networkAddress;
123 MY_LOG(trace) << "RSU-Router, looking for IP-Address";
124
125 //checkinf if metric 5 in this router-block and not in next
126 if(positionMetricDelimiter>routerBlock.find("\n\n", 20))
127 {
128 MY_LOG(trace) << "wrong metric if rsu ignoring";
129 return "";
130 }
131
132 //finding the network-address
133 tmpLoopCounter=1;
134 networkAddress.clear();
135
136 while(routerBlock[positionMetricDelimiter-tmpLoopCounter]!=' ')
137 {
138 networkAddress.insert(0, 1, routerBlock[positionMetricDelimiter-tmpLoopCounter]);
139 tmpLoopCounter++;
140 }
141
C++ length : 5.068 lines : 206
Ln : 1 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS


P:\DawidKulesza\Bachelorarbeit\t_BirdParser.cpp - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ?
results.hrf
t_BirdParser.cpp
buffer << t.rdbuf();
return buffer.str();
}
BOOST_AUTO_TEST_SUITE (BirdcParserClass);
vector<s_rsu> remoteRsus;
BOOST_AUTO_TEST_CASE(one_rsu_in_mock_file2)
{
BirdcParser parser(parser_regularRouter, parser_rsuMetric, parser_network, remoteRsus);
string file = birdOutput(MUCK_FILE2);
parser.setOutput(file);
parser.parse();
BOOST_CHECK(remoteRsus.size() == 1);
}
BOOST_AUTO_TEST_CASE(correct_ip_rsu_mock_file_2)
{
BirdcParser parser(parser_regularRouter, parser_rsuMetric, parser_network, remoteRsus);
string file = birdOutput(MUCK_FILE2);
parser.setOutput(file);
parser.parse();
BOOST_CHECK(remoteRsus[0].ip == "122.138.21.1");
}
BOOST_AUTO_TEST_CASE(correct_coordinates_rsu_mock_file_2)
{
BirdcParser parser(parser_regularRouter, parser_rsuMetric, parser_network, remoteRsus);
string file = birdOutput(MUCK_FILE2);
parser.setOutput(file);
parser.parse();
BOOST_CHECK(remoteRsus[0].lon - 193.168 < 0.00001 && remoteRsus[0].lat - 21.2 < 0.00001);
}
BOOST_AUTO_TEST_CASE(five_rsu_in_mock_file3)
{
BirdcParser parser(parser_regularRouter, parser_rsuMetric, parser_network, remoteRsus);
string file = birdOutput(MUCK_FILE3);
parser.setOutput(file);
parser.parse();
BOOST_CHECK(remoteRsus.size() == 5);
}
BOOST_AUTO_TEST_CASE(correct_coordinates_rsu_tree_mock_file_3)
{
BirdcParser parser(parser_regularRouter, parser_rsuMetric, parser_network, remoteRsus);
string file = birdOutput(MUCK_FILE3);
parser.setOutput(file);
parser.parse();
BOOST_CHECK(remoteRsus[2].lon - 12.92 < 0.00001 && remoteRsus[2].lat - 50.83 < 0.00001);
}
C++ source file length : 3.806 lines : 147 Ln : 34 Col : 42 Sel : 0 | 0 Unix (LF) UTF-8 INS

X

Anhang 5. : *Unit Test: Ergebnisse*



```
Running 18 test cases...
Entering test suite "RoutingClassModule"
Entering test suite "BirdcParserClass"
Entering test case "one_rsu_in_mock_file2"
../src/test/t_BirdParser.cpp(45): info: check remoteRsus.size() == 1 passed
Leaving test case "one_rsu_in_mock_file2"; testing time: 378mks
Entering test case "correct_ip_rsu_mock_file_2"
../src/test/t_BirdParser.cpp(56): info: check remoteRsus[0].ip ==
"122.138.21.1" passed
Leaving test case "correct_ip_rsu_mock_file_2"; testing time: 180mks
Entering test case "correct_coordinates_rsu_mock_file_2"
../src/test/t_BirdParser.cpp(66): info: check remoteRsus[0].lon - 193.168 <
0.00001 && remoteRsus[0].lat - 21.2 < 0.00001 passed
Leaving test case "correct_coordinates_rsu_mock_file_2"; testing time:
183mks
Entering test case "five_rsu_in_mock_file3"
../src/test/t_BirdParser.cpp(78): info: check remoteRsus.size() == 5 passed
Leaving test case "five_rsu_in_mock_file3"; testing time: 269mks
Entering test case "correct_coordinates_rsu_tree_mock_file_3"
../src/test/t_BirdParser.cpp(88): info: check remoteRsus[2].lon - 12.92 <
0.00001 && remoteRsus[2].lat - 50.83 < 0.00001 passed
Leaving test case "correct_coordinates_rsu_tree_mock_file_3"; testing time:
310mks
Entering test case "correct_coordinates_rsu_five_mock_file_3"
../src/test/t_BirdParser.cpp(98): info: check remoteRsus[5].lon - 44.168 <
0.00001 && remoteRsus[5].lat - 21.5 < 0.00001 passed
Leaving test case "correct_coordinates_rsu_five_mock_file_3"; testing time:
260mks
Entering test case "correct_ip_rsu_one_mock_file_3"
../src/test/t_BirdParser.cpp(108): info: check remoteRsus[0].ip ==
"192.168.21.17" passed
Leaving test case "correct_ip_rsu_one_mock_file_3"; testing time: 255mks
Entering test case "correct_ip_rsu_four_mock_file_3"
../src/test/t_BirdParser.cpp(117): info: check remoteRsus[3].ip ==
"192.168.21.1" passed
Leaving test case "correct_ip_rsu_four_mock_file_3"; testing time: 344mks
Entering test case
"three_rsu_in_mock_file3_if_regular_routers_starts_with_13"
../src/test/t_BirdParser.cpp(129): info: check remoteRsus.size() == 3
passed
Leaving test case
"three_rsu_in_mock_file3_if_regular_routers_starts_with_13"; testing time:
231mks
Entering test case
"four_rsu_in_mock_file3_if_regular_routers_starts_with_22"
../src/test/t_BirdParser.cpp(139): info: check remoteRsus.size() == 4
passed
Leaving test case
"four_rsu_in_mock_file3_if_regular_routers_starts_with_22"; testing time:
248mks
Leaving test suite "BirdcParserClass"
Entering test suite "RsusClass"
Entering test case
"distance_between_chemnitz_and_dresden_bigger_than_calculated"
../src/test/t_Rsus.cpp(35): info: check myRsus.isInRange(rsuChemnitz,
64406.4+1) passed
Leaving test case
"distance_between_chemnitz_and_dresden_bigger_than_calculated"; testing
time: 150mks
Entering test case
"distance_between_chemnitz_and_dresden_same_as_calculated"
../src/test/t_Rsus.cpp(41): info: check !myRsus.isInRange(rsuChemnitz,
64406.4) passed
```