Hochschule für Telekommunikation Leipzig (FH)
Institut für Telekommunikationsinformatik

Abschlussarbeit zur Erlangung des akademischen Grades Bachelor of Engineering

Thema: # Integration von WebRTC in einen All-IP-Telefonanschluss der Deutschen Telekom

vorgelegt von: Ferdinand Malcher

geboren am: XX.XX.XXXX
in: XXX

Erstprüfer: Dipl.-Ing (FH) Michael Maruschke
Hochschule für Telekommunikation Leipzig
Gustav-Freytag-Straße 43, 04277 Leipzig

Zweitprüfer: Kay Hänsge
Telekom Innovation Laboratories
Winterfeldtstraße 21, 10781 Berlin

vorgelegt am: 28. August 2015

Inhaltsverzeichnis

Abkürzungsverzeichnis

3PTY	Three-Party Service
API	Application Programming Interface
BB-RAR	Broadband Remote Access Router
B2BUA	Back-to-back User Agent
CCBS	Call/Communication Completion of Calls to Busy Subscriber
CFU	Call/Communication Forwarding Unconditional
CFNR	Call/Communication Forwarding No Reply
CLIP	Calling Line Identification Presentation
CLIR	Calling Line Identification Restriction
CSS	Cascading Stylesheets
CH	Call/Communication Hold
DiffServ	Differentiated Services
DHCP	Dynamic Host Configuration Protocol
DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line
DTMF	Dual-Tone Multi-Frequency
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
UI	User Interface
HTML	Hypertext Markup Language
IAD	Integrated Access Device
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
ISDN	Integrated Services Digital Network
ITU-T	International Telecommunication Union (Telecommunication Standardization Sector)
IP	Internet Protocol
JSEP	JavaScript Session Establishment Protocol

JSON	JavaScript Object Notation
MSN	Multiple Subscriber Number
NAPT	Network Address Port Translation
NGN	Next Generation Network
OIP	Originating Identification Presentation
OIR	Originating Identification Restriction
P-CSCF	Proxy Call Session Control Function
PHB	Per-Hop Behaviour
PSTN	Public Switched Telephone Network
PSS	PSTN Simulation Subsystem
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for comments
RTP	Realtime Transport Protocol
SCC	Service Command Code
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SRTP	Secure Realtime Transport Protocol
STUN	Session Traversal Utilities for NAT
TLS	Transport Layer Security
TOS	Type of Service
TURN	Traversal Using Relays around NAT
UE	User Equipment
VoIP	Voice over IP
VSC	Vertical Service Code
WebRTC	Web Real-Time Communication
WLAN	Wireless Local Area Network
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Der fortschreitende Ausbau der Breitband-Internetzugänge im Fest- und Mobilfunknetz und der große Wettbewerb auf dem Telekommunikationsmarkt stellt die Provider vor neue Herausforderungen. Während noch vor einigen Jahren hauptsächlich klassische Telefonanbieter den Markt dominierten, können sich heute auch Dienste wie Skype [1] oder Whatsapp [2] durchsetzen, die nicht originär als Telekommunikationsanbieter tätig waren. Mit dem steigenden Kostendruck wurden in den letzten Jahren die klassischen leitungsvermittelten Telefonnetze zunehmend abgebaut, um Platz zu schaffen für paketvermittelte Dienste über ein All-IP-Netz. Der Paradigmenwechsel weg vom leitungsvermittelten ISDN/PSTN hin zu internetbasierter, paketvermittelter Telefonie (VoIP) ermöglicht außerdem die einfache Anbindung von Telefonanwendungen an Computersysteme. Immer leistungsfähigere mobile Endgeräte eröffnen dazu neue Möglichkeiten für die Einführung komplexer mobiler Anwendungen.

Mit Web Real-Time Communication (WebRTC) wurde eine Technologie vorgestellt, die eine Punkt-zu-Punkt-Übertragung von Audio- und Video-Streams zwischen Webbrowsern ermöglicht und damit Multimediafunktionen wie Telefonie direkt in den Browser integriert. Dadurch eröffnen sich neue interessante Felder für Telefonanbieter und Endanwender. Besonders durch die Integration in existierende Telefonanschlüsse können dem Benutzer die Vorteile von WebRTC einfach zugänglich gemacht werden. Damit ist es möglich, einen bestehenden Telefonanschluss über eine browserbasierte Anwendung von Mobilgeräten und Computern aus zu nutzen, ohne dass ein separater Telefonie-Client notwendig ist.

In Deutschland bietet bisher kein Provider einen Zugang zum Telefonnetz über WebRTC an. Die vorliegende Arbeit beschäftigt sich daher mit der Frage, wie WebRTC in einen bestehenden All-IP-Anschluss integriert werden kann, um die vorhandene Infrastruktur auch über WebRTC zugänglich zu machen. Es werden drei Konzepte zur Integration vorgestellt und einer dieser Ansätze zur Demonstration in einer Laborumgebung umgesetzt. Dabei wird vor allem Wert auf die Unterstützung des breiten Leistungsumfangs gelegt, der neben der Telefoniefunktion auch weitere Leistungsmerkmale umfasst, wie sie aus dem ISDN bekannt sind. Obwohl mit WebRTC auch Video- und Dateiübertragung möglich ist, liegt der Fokus dieser Arbeit auf der Unterstützung von Audio-Telefonie.

2 Analyse

2.1 Web Real-Time Communication

Web Real-Time Communication (WebRTC) beschreibt einen Standard zur Audio-Video-Echtzeitkommunikation im Browser. Zum Zeitpunkt dieser Bachelorarbeit befindet sich WebRTC noch in der Standardisierung beim World Wide Web Consortium (W3C) und bei der Internet Engineering Task Force (IETF) [3, 4]. Die Entwicklung wird maßgeblich von Google, Mozilla und Opera vorangetrieben, die WebRTC bereits in ihren Browsern implementieren. Die Technologie wird derzeit von Google Chrome, Mozilla Firefox und Opera Browser unterstützt, teilweise auch in den Versionen für Mobilgeräte [5]. Microsoft plant für den Internet Explorer bzw. Microsoft Edge die ORTC API, die kompatibel zu WebRTC sein soll [6, 7]. Außerdem wurde das Konkurrenzprodukt CU-RTC angekündigt, das nicht mit WebRTC kompatibel ist [8, 9].

WebRTC unterscheidet sich in seiner Architektur grundlegend von anderen Systemen für Audio-Video-Telefonie oder Videokonferenzen. Für WebRTC wird keine zusätzliche Software benötigt, die Schnittstelle für den Zugriff auf Mikrofon und Kamera ist bereits in den Webbrowser integriert.

Während Anwendungen wie Skype, Mumble, Jingle oder Telefonie mit SIP/SDP/RTP den Medientransport über eine zentrale Serverinstanz realisieren, werden in WebRTC die Nutzdaten in einer Direktverbindung zwischen den Endpunkten übertragen[1].

Abbildung 1 zeigt eine mögliche WebRTC-Architektur, das sogenannte WebRTC-Triangle [11].

Zwei Webbrowser sind über eine Peer Connection für den Medientransport direkt verbunden. Der Medientransport findet mittels Secure Realtime Transport Protocol (SRTP) statt, das an das in der SIP-Telefonie verwendete Realtime Transport Protocol (RTP) angelehnt ist, aber die Nutzdaten verschlüsselt überträgt.

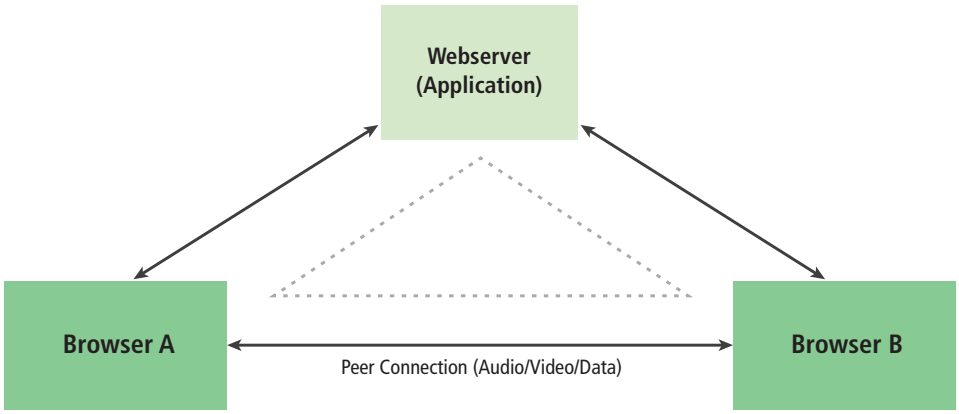

Abbildung 1: WebRTC-Triangle (vgl. [11])

[1] Telefonie mit SIP/SDP/RTP ist auch von Punkt zu Punkt möglich, ist allerdings in Telekommunikationsnetzen nicht der Regelfall. Eine Direktverbindung ist nur möglich, wenn sich die Endpunkte direkt erreichen können und kein Proxy zwischengeschaltet ist. Öffentliche Telefonie im IMS wird immer über eine Proxy-Funktion (Proxy Call Session Control Function (P-CSCF)) vermittelt [10][S. 46].

Die Signalisierung wird über einen Server zentral zwischen den Endpunkten vermittelt. Zur Medienaushandlung wird das Session Description Protocol (SDP) verwendet [12]. Das Modul für das Handling der SDP-Nachrichten ist in den Browser integriert. Für WebRTC ist kein Signalisierungsprotokoll vorgeschrieben, sondern es kann prinzipiell jedes Protokoll verwendet werden, das folgende Anforderungen erfüllt:

- kann über Websocket übertragen werden
- kann eine Sitzung zwischen den Endpunkten vermitteln
- kann SDP-Nachrichten kapseln

Neben proprietären Entwicklungen eignen sich unter anderem folgende Protokolle für die WebRTC-Signalisierung [13, S. 4]:

- Session Initiation Protocol (SIP) (over Websocket) [14]
- Jingle/Extensible Messaging and Presence Protocol (XMPP) [15, 16]
- JavaScript Session Establishment Protocol (JSEP) [17]
- H.323 [18]

Die Applikation zur Darstellung der grafischen Benutzeroberfläche beim Anwender wird von einem Webserver bezogen. Es handelt sich um eine Web-Applikation auf Basis von HTML5 und Javascript.

Der WebRTC-Standard beschreibt ein Application Programming Interface (API), das im Browser den Zugriff auf die Hardware und die Verbindung zu anderen WebRTC-Endpunkten ermöglicht. Dafür sind im Browser drei Javascript-Komponenten vorgesehen [3, 19, 20]:

- `getUserMedia`: Zugriff auf Kamera und Mikrofon des Client-Geräts
- `RTCPeerConnection`: Verbindung zu einem anderen WebRTC-Endpunkt für die Übertragung von Audio und Video
- `RTCDataChannels`: Datenübertragung zwischen WebRTC-Endpunkten über einen sogenannten *DataChannel*

Die grundlegende Architektur eines WebRTC-Clients, wie vorliegend beschrieben, ist in Abbildung 2 zur Veranschaulichung dargestellt.

WebRTC definiert, welche Codecs von Endgeräten mindestens unterstützt werden müssen [21]. Für die Audio-Übertragung sind dabei folgende Codecs vorgeschrieben:

- G.711 A-law (PCMA) und μ-law (PCMU)
- Opus
- Telephone Event[2]

[2] Dabei handelt es sich nicht um einen Codec zur Sprachkomprimierung, sondern um ein Verfahren zur Übertragung von Mehrfrequenzwahltönen (DTMF) über (S)RTP [22].

Der Opus-Codec soll vorzugsweise verwendet werden. Für Video-Codecs sind verschiedene Eigenschaften zur Auflösung und Frame-Rate vorgeschrieben. Der Codec VP8 wird empfohlen.

Darüber hinaus kann jeder weitere geeignete Codec unterstützt werden. Die WebRTC-Endpunkte handeln die Medieneigenschaften während des Verbindungsaufbaus aus.

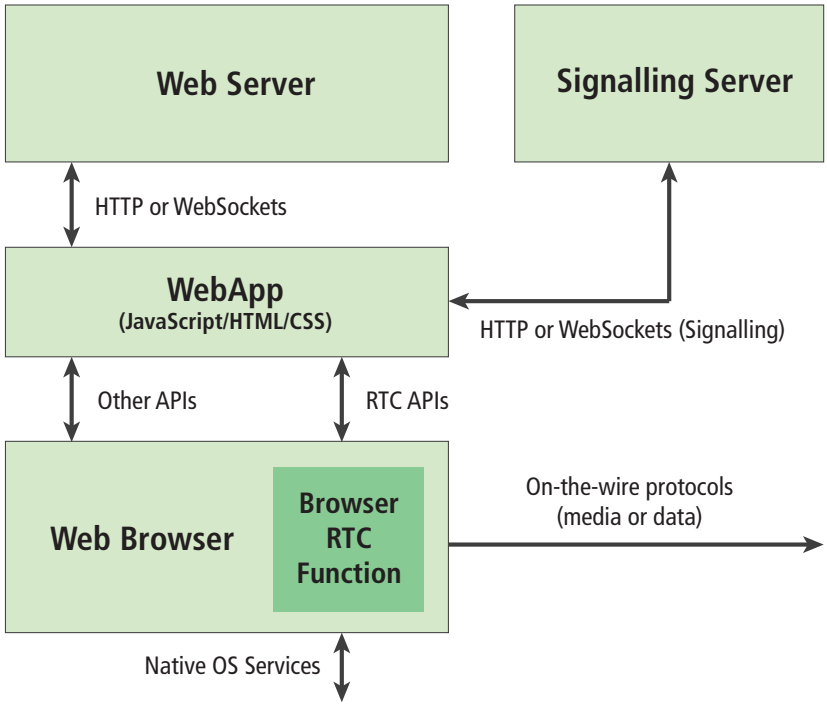

Abbildung 2: Echtzeitkommunikation im Browser (vgl. [11])

Da sich WebRTC-Clients im Home-Network in der Regel hinter einem Router mit Network Address Port Translation (NAPT) befinden und nicht über eine öffentliche IP-Adresse erreichbar sind, kommen die Mechanismen Session Traversal Utilities for NAT (STUN) und Traversal Using Relays around NAT (TURN) zum Einsatz, um eine direkte Kommunikation zwischen den Endpunkten zu etablieren [3]. Für die Verwendung im Zusammenhang mit dem Offer/Answer-Modell, wie es bei SIP/SDP Anwendung findet, wird die Bezeichnung Interactive Connectivity Establishment (ICE) verwendet. ICE greift auf die Technologien STUN und TURN zurück, um das NAPT Traversal zu ermöglichen [23].

2.2 All-IP-Anschluss

Der Begriff des Next Generation Network (NGN) beschreibt ein Telekommunikationsnetz, in dem alle Dienste paketorientiert ausliefert werden. Das Leistungsspektrum umfasst unter anderem Telefonie, Internet und Fernsehen und integriert die bestehenden leitungsvermittelten Technologien in das paketorientierte Netz. Dadurch erfolgt der Übergang von einer vertikalen Architektur mit einzelnen gewachsenen Technologien zur horizontalen Architektur mit einem Kernnetz für alle Dienste.

NGNs sind von der International Telecommunication Union (Telecommunication Standardization Sector) (ITU-T) und European Telecommunications Standards Institute (ETSI)

standardisiert und in verschiedenen Standards beschrieben [24–26].

Das NGN-Kernnetz wird nach Definition der ETSI als IP Multimedia Subsystem (IMS) bezeichnet [25]. Die Telefonie-Dienste im IMS basieren auf SIP (mit SDP und RTP) und sind in ETSI TS 24.229 [27] standardisiert. Durch die vollständige Realisierung aller Dienste über das Internet Protocol (IP) wird ein Anschluss an ein IMS auch All-IP-Anschluss genannt [28][S. 460].

Da mit der Telefonie Echtzeitanwendungen in die paketvermittelte Domäne gebracht werden, sind Quality of Service (QoS)-Mechanismen notwendig. QoS ist im Unterabschnitt 2.4 beschrieben.

Der Übergang in andere Netze wird durch Gateways realisiert. Die Unterstützung leitungsvermittelter Netze ist auch weiterhin erforderlich, da der Ausbau des NGN-Kernnetzes nur sukzessive erfolgt und eine sofortige Abschaffung gewachsener Technologien auch die Umstellung der Endgeräte beim Kunden erfordert. Durch die Gateway-Funktion ist der Übergang zunächst für den Benutzer nicht bemerkbar, da Technologien wie Integrated Services Digital Network (ISDN) und analoge Telefonie (Public Switched Telephone Network (PSTN)) im Home-Network weiterhin verwendbar sind.

Ein Integrated Access Device (IAD) bildet den Netzabschluss in einem NGN und übernimmt aus Sicht des Kernnetzes die Rolle des User Equipment (UE). Es ist als Residential Gateway beim Kunden installiert. Der funktionale Aufbau eines IAD ist in Abbildung 3 zu sehen. Ein DSL-Modem[3] stellt die Verbindung zum Internet her. Ein Router mit Firewall regelt die Vermittlung zwischen dem privaten Netzwerk beim Endnutzer (Home-Network) und dem öffentlichen Internet. Für die Vermittlung innerhalb des Home-Networks verfügt ein IAD in der Regel über einen Switch zum Anschluss von Computern und anderen Netzwerkgeräten. Meist ist auch ein WLAN-Modul integriert. Ein DHCP-Server sorgt für die Adressvergabe im Home-Network.

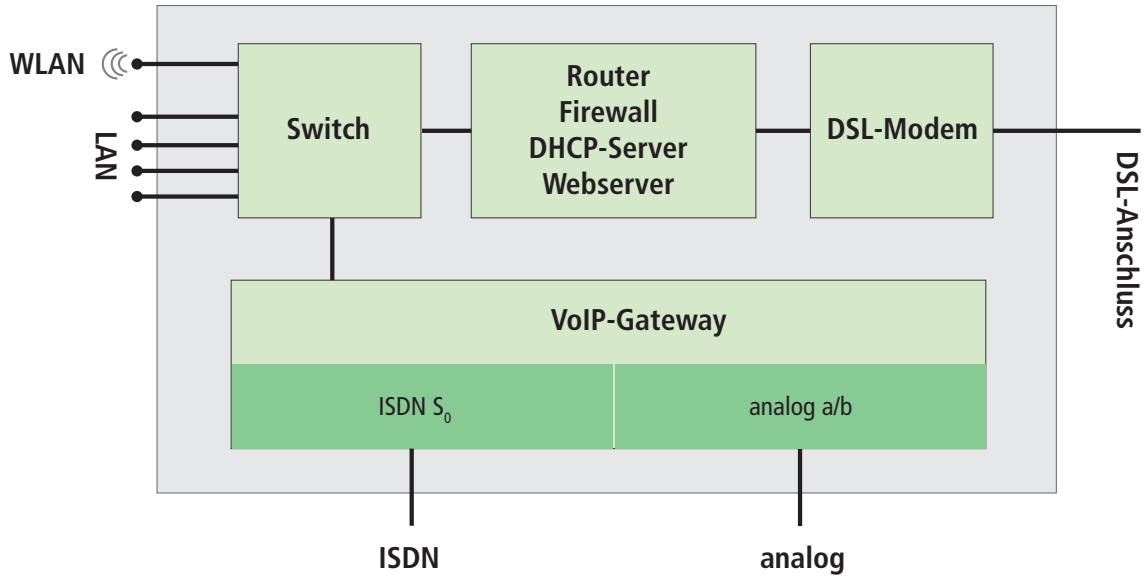

Abbildung 3: Funktionaler Aufbau eines IAD (vgl. [29])

[3]oder anderes Modem, je nach Zugangstechnologie

Eine Besonderheit gegenüber herkömmlichen Heimroutern ist das Voice over IP (VoIP)-Gateway. Es tritt gegenüber dem NGN als VoIP-Client auf und stellt dem Anwender Telefoniedienste über ISDN und analoges PSTN bereit. Das Gateway sorgt für die Umsetzung ins paketvermittelte NGN. Dadurch ist es möglich, ISDN- und PSTN-Endgeräte auch für die paketvermittelte Telefonie weiter zu nutzen. Das IAD übernimmt hierbei sowohl die Funktion eines Media Gateways als auch die eines Signalling Gateways für die Wandlung von ISDN zu SIP.

Handelsübliche IADs sind zum Beispiel die Fritz!Box des Herstellers AVM [30] oder die von der Telekom vertriebene Speedport-Reihe [31].

Next Generation Network der Deutschen Telekom

Die Telefoniedienste im NGN der Deutschen Telekom basieren grundlegend auf den Definitionen nach ETSI TS 24.229 [27]. Das Dokument wurde für die Anwendung in der Telekom überarbeitet und in 1TR114 [32] veröffentlicht.

Im paketvermittelten Festnetz werden folgende Codecs für die Telefonie verwendet [33]:

- G.711 (Narrowband)
- G.722 (Wideband) („HD-Voice“ [34])

2.3 Supplementary Services

2.3.1 Leistungsmerkmale im ISDN

Im ISDN wurden Leistungsmerkmale (Supplementary Services) eingeführt, die die Telefonie-Basisdienste (Teleservices) um zusätzliche Kommunikationsmerkmale erweitern [35]. In Tabelle 1 ist eine Auswahl der standardisierten ISDN-Leistungsmerkmale aufgeführt. Darüber hinaus können von Netzbetreibern weitere nicht standardisierte Dienste eingeführt werden.

Supplementary Services sind in vier Varianten realisiert [29]:

- direkt ins Signalisierungsprotokoll integriert
- mittels Telefontastatur mit dem Keypad-Protokoll, z.B. über Vertical Service Codes (VSC) (vgl. Unterunterabschnitt 2.3.2)
- mit einem funktionalen Protokoll
- mit Sprachsteuerung

Viele Dienstmerkmale wurden mit der Einführung der IP-basierten Telefonie übernommen und werden im NGN vom PSTN Simulation Subsystem (PSS) zur Verfügung gestellt [36]. Sie sind mit den ursprünglichen Diensten aus dem ISDN kompatibel, tragen im NGN aber teilweise andere Bezeichnungen.

Tabelle 1 stellt die ISDN-Dienste und ihre entsprechenden Bezeichnungen im NGN dar.

Bezeichnung	**ISDN**	**NGN (ETSI)**
Rufnummernanzeige	Calling Line Identification Presentation (CLIP) [37]	Originating Identification Presentation (OIP) [38]
Rufnummernunterdrückung	Calling Line Identification Restriction (CLIR) [37]	Originating Identification Restriction (OIR) [38]
Anrufweiterschaltung sofort	Call Forwarding Unconditional (CFU) [39]	Communication Forwarding Unconditional (CFU) [40]
Anrufweiterschaltung bei Nichtmelden	Call Forwarding No Reply (CFNR) [39]	Communication Forwarding No Reply (CFNR) [40]
Mehrfachrufnummer	Multiple Subscriber Number (MSN) [37]	*realisiert durch SIP-Accounts*
Halten	Call Hold (CH)	Communication Hold (CH) [41]
Dreierkonferenz	Three-Party Service (3PTY) [42]	
Automatischer Rückruf bei besetzt	Completion of Calls to Busy Subscribers (CCBS) [41]	Completion of Communication to Busy Subscribers (CCBS) [43]

Tabelle 1: Leistungsmerkmale im ISDN und NGN

2.3.2 Vertical Service Codes

Vertical Service Codes (VSC) sind Ziffernkombinationen, die über die Telefontastatur eingegeben werden können, um Supplementary Services zu steuern. Sie werden aus Sicht des Benutzers wie eine Telefonnummer behandelt. Ein Beispiel für einen VSC ist folgende Kombination zur Aktivierung der *Anrufweiterschaltung sofort*:

```
*21*Zielrufnummer#
```

Der grundsätzliche Aufbau der VSC ist in ETSI ETS 300738 [44] beschrieben. Am genannten Beispiel bedeutet das:

- `*` Service Prefix
  leitet den VSC ein
- `21` Service Code
  adressiert den auszuwählenden Dienst
- `*` Service Separator
  Trennzeichen
- `Zielrufnummer` Supplementary Information
  übergebene Parameter
- `#` Service Suffix
  terminiert den VSC

Die Deutsche Telekom bietet in ihrem Netz zusätzliche Funktionen an, die über die in ETSI ETS 300738 [44] standardisierten Dienstmerkmale hinausgehen. Dafür werden weitere Service Codes eingeführt, die nur im Netz der Deutschen Telekom gültig sind. Teilweise werden auch standardisierte Service Codes mit neuen Funktionen belegt. Eine komplette

Übersicht über die Leistungsmerkmale am IP-basierten Anschluss der Deutschen Telekom, die sich über VSC steuern lassen, befindet sich im Anhang A.

Die Deutsche Telekom verwendet die Bezeichnung Service Command Code (SCC), die synonym zu VSC ist [32].

2.4 Quality of Service

Quality of Service (QoS) beschreibt Mechanismen, die es erlauben, eine Dienstgüte sicherzustellen. Diese Problemstellung ist besonders bei Echtzeitanwendungen wie Telefonie relevant. Eine Möglichkeit zur Realisierung von QoS ist Type of Service (TOS) bzw. Differentiated Services (DiffServ). Der Traffic wird hierbei in Klassen eingeordnet und an den Netzelementen abhängig von der Klasse bevorzugt behandelt.

Die ursprüngliche Spezifikation von IPv4 nach RFC 791 [45] sieht im Header ein 8 Bit langes Feld TOS vor (siehe Abbildung 4). Die Bits 0-2 sind dabei für die *IP Precedence* bestimmt. Damit werden acht Verkehrsklassen definiert, die es möglich machen, den Netzwerktraffic zu klassifizieren, um Datenströme bestimmter Klassen zu priorisieren.

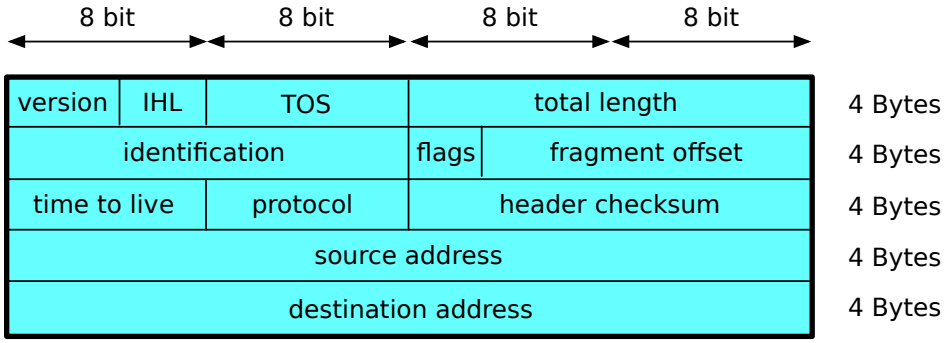

Abbildung 4: Aufbau eines IPv4-Headers [46]

Der spätere RFC 2474 [47] beschreibt DiffServ als ein erweitertes Schema zur Traffic-Klassifizierung, das die 8 Bits des TOS-Felds wiederverwendet. Die Bits 0-5 sind dabei zur Auswahl eines Differentiated Services Code Point (DSCP) zwischen 0 und 63 vorgesehen.

Den DSCP sind sogenannte Per-Hop Behaviour (PHB) zugeordnet. Sie regeln, wie der entsprechend markierte Traffic in den verarbeitenden Netzelementen behandelt werden soll [47, 48]. Die Entscheidung, ob Pakete mit QoS-Klassifizierung tatsächlich priorisiert behandelt werden, obliegt allerdings ausschließlich den verarbeitenden Netzelementen. Je nach Netzsegment können die Vorschriften von den standardisierten PHBs abweichen.

Um die Abwärtskompatibilität mit der ursprünglichen IP Precedence zu gewähren, werden *Class Selector PHB* eingeführt, deren Bits 0-2 den acht Verkehrsklassen nach RFC 791 entsprechen. Außerdem wird ein *Expedited Forwarding PHB* definiert, der für Echtzeitanwendungen optimiert ist und geringe Latenzzeiten und wenig Jitter garantieren soll. Er soll für den Medientransport bei der Telefonie verwendet werden [49][S. 32] und ist dem DSCP $101110_b = 46_d$ zugeordnet [50][S. 9]. Die SIP-Signalisierung soll mit dem DSCP $101000_b = 40_d$ markiert werden, der dem Class Selector PHB CS5 zugeordnet ist [49][S. 34].

Wird der DSCP 0 angegeben oder kein DSCP gesetzt, wird der *Default PHB* verwendet, der auch *Best Effort* genannt wird [47][S. 9].

DiffServ im NGN der Deutschen Telekom

Die Deutsche Telekom veröffentlicht keine offizielle Information, welche DSCP für VoIP in ihrem Netz eingesetzt werden. In einem Dokument zum Octopus F 50, ein von der Telekom eingesetztes IAD, ist folgende Angabe zu finden [51][S. 22]:

> *Zurzeit werden am Call & Surf Comfort IP Anschluss der Deutsche Telekom folgende TOS Werte für VoIP verwendet:*
>
> - *für SIP-Daten TOS-Hexadezimal: "c0"*
> - *für RTP-Daten TOS-Hexadezimal: "a0"*

Diese Angabe entspricht dem Class Selector PHB CS6 für RTP und CS5 für SIP[4]. VoIP-Traffic wird vom IAD nach diesen Vorschriften markiert, um im Netz priorisiert behandelt werden zu können.

Da das DiffServ-Tagging durch das IAD bzw. durch den Broadband Remote Access Router (BB-RAR) im Netz vorgenommen wird, kann die Betrachtung von QoS im Rahmen dieser Arbeit vernachlässigt werden.

2.5 WebRTC-Gateway

2.5.1 Konzept eines WebRTC-Gateways

Die Integration von WebRTC in eine bestehende Telefonieinfrastruktur mit SIP/SDP und RTP wirft folgende Problemstellungen auf:

- In der öffentlichen IP-Telefonie wird SIP zur Signalisierung eingesetzt, für WebRTC ist hingegen kein Signalisierungsprotokoll vorgeschrieben.
- Der Medientransport zwischen WebRTC-Endpunkten findet verschlüsselt über SRTP statt, herkömmliche Internettelefonie setzt wiederum meist auf das unverschlüsselte RTP.
- Für WebRTC werden zum Teil andere Codecs eingesetzt (z.B. Opus) als für die öffentliche Telefonie (z.B. G.711).
- Die Nutzdaten werden bei WebRTC direkt zwischen den Endpunkten übertragen (peer-to-peer), in der öffentlichen Telefonie werden diese zentral vermittelt.

Es muss also ein Modul integriert werden, das die Eigenschaften beider Seiten implementiert und in die jeweils andere wandelt. Man spricht von einem WebRTC-Gateway.

[4] Die Angabe $c0$ bezieht sich auf das gesamte TOS-Byte im IP-Header. Zur Auswahl des DSCP werden die Bits 0-5 verwendet. $c0_h = 11000000_b$, woraus sich der DSCP 110000 ergibt, der dem Class Selector 6 entspricht.

Es lassen sich danach folgende Anforderungen an ein Gateway identifizieren:

- Wandlung der WebRTC-Signalisierung in SIP/SDP
- Medienwandlung von WebRTC-Codecs (insbesondere Opus) in vom Netz unterstützte Codecs (insbesondere G.711)
- WebRTC-spezifische Protokollwandlung von SRTP nach RTP
- Unterstützung der Mechanismen zum NAPT Traversal
- Auslieferung der Webapplikation an den Benutzer

Das Gateway agiert als WebRTC-Endpunkt und Signalisierungsserver (vgl. Unterabschnitt 2.1) aus Sicht des Browsers und als SIP-Client aus Sicht des öffentlichen Netzes und erfüllt damit die Funktion eines Back-to-back User Agent (B2BUA) (vgl. [28][S. 219-221]). Abbildung 5 zeigt den funktionalen Aufbau eines WebRTC-Gateways, das die vorliegenden Anforderungen berücksichtigt.

Ein *Signalling Gateway* sorgt für die Umsetzung der WebRTC-Signalisierung in SIP und SDP und die Terminierung der STUN/TURN-Nachrichten. Die Komponente des *Media Gateway* organisiert die Medienwandlung (Transcoding) der Audiodaten und die Umsetzung von SRTP in RTP. Die für WebRTC obligatorische Webapplikation wird von einem Webserver bereitgestellt.

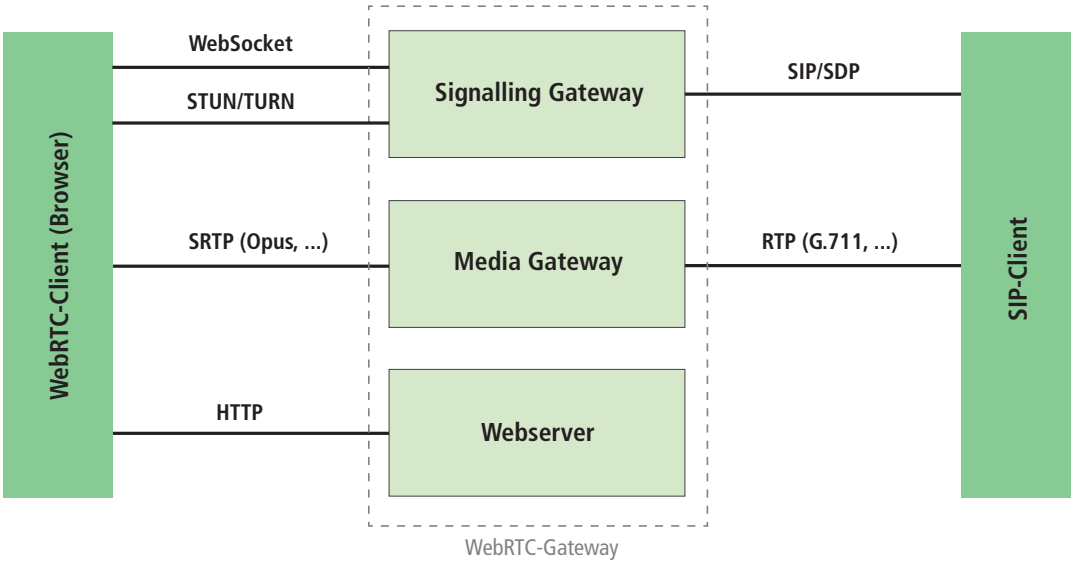

Abbildung 5: Funktionaler Aufbau eines WebRTC-Gateways

2.5.2 Gateway-Lösungen und Frameworks

Es existieren verschiedene freie und proprietäre Implementationen von WebRTC-Gateways. In diesem Abschnitt werden drei dieser Projekte vorgestellt, wobei sich die Auswahl auf Open-Source-Lösungen beschränkt.

Janus Das Projekt Janus [52] implementiert ein „general purpose“ WebRTC-Gateway. Janus ist ein umfangreiches Framework mit Werkzeugen für die Implementation von WebRTC-basierten Anwendungen, bietet aber von sich aus noch keine „Out-of-the-box“-Funktionalität. Unter den Beispielen ist auch ein WebRTC-SIP-Gateway zu finden. Das Projekt wird von der italienischen Firma Meetecho betreut. Auf der Projektwebsite wird Janus wie folgt beschrieben:

> *„The Janus WebRTC Gateway has been conceived as a general purpose gateway. As such, it doesn't provide any functionality per se other than implementing the means to set up a WebRTC media communication with a browser, exchanging JSON messages with it, and relaying RTP/RTCP and messages between browsers and the server-side application logic they're attached to.“ [52]*

webrtc2sip webrtc2sip [53] ist ein Projekt der französischen Firma Doubango Telecom, die sich auf NGN-Technologien spezialisiert hat. Es ist als WebRTC-Gateway konzipiert und bringt als fertiges Produkt bereits vollständige Funktionalität für die Protokoll- und Medienwandlung mit. Es basiert auf dem Doubango IMS Framework und ist kompatibel mit dem JavaScript-SIP-Stack SIPml5 [54], die ebenfalls Produkte der Doubango Telecom sind. Die Entwickler beschreiben webrtc2sip auf der Projektwebsite wie folgt:

> *„webrtc2sip is a smart and powerful gateway using RTCWeb and SIP to turn your browser into a phone with audio, video and SMS capabilities. The gateway allows your web browser to make and receive calls from/to any SIP-legacy network or PSTN.“ [53]*

overSIP Das Projekt overSIP [55] versteht sich als SIP-Proxy und -Server mit Unterstützung für SIP over Websocket. overSIP ist verbreitet und implementiert laut Projektwebsite zahlreiche Standards. Die Software bietet allerdings keine Unterstützung für SIP-Registrierung, SIP-Authentifizierung und Medienwandlung [56].

Höherwertige Lösungen Darüber hinaus existieren multifunktionale Telefonieanwendungen, die eine WebRTC-Gateway-Funktionalität implementieren. Dazu gehören Asterisk [57], FreeSWITCH [58] und Kamailio [59]. Da diese Lösungen allerdings nicht als reine WebRTC-Gateways ausgelegt sind, werden sie im Rahmen dieser Arbeit nicht näher betrachtet.

3 Konzepte zur Integration

Die zentrale Problemstellung der Integration von WebRTC ins Festnetz wurde vor Beginn der Konzeption als User Story [60] formuliert:

> *Als Kunde eines Telefonanschlusses möchte ich meine Telefonnummern auch über WebRTC nutzen. Dazu soll mir eine Web-Oberfläche zur Verfügung stehen, über die ich mit den Telefonnummern meines Anschlusses Anrufe tätigen und empfangen kann.*

Wie bereits im Unterunterabschnitt 2.5.1 erläutert, kann diese Anforderung durch ein WebRTC-Gateway erfüllt werden.

Im Folgenden werden drei Konzepte vorgestellt, nach denen ein WebRTC-Gateway in die Infrastruktur integriert werden kann. Zu jedem Ansatz werden Vor- und Nachteile tabellarisch aufgeführt und der funktionale Aufbau in einer Grafik[5] dargestellt.

3.1 Vorstellung der Konzeptvarianten

3.1.1 Konzept I: Integration in das Integrated Access Device

Das Gateway wird in das beim Benutzer vorhandene IAD integriert und läuft als Prozess auf dem Betriebssystem des IAD. Mögliche Plattformen sind freetz [61] und OpenWrt [62]. Die Software greift direkt auf die Konfiguration des IAD zu, z.B. um SIP-Zugangsdaten auszulesen. Das Konzept ist in Abbildung 6 verbildlicht, die Vor- und Nachteile sind in Tabelle 2 gegenübergestellt.

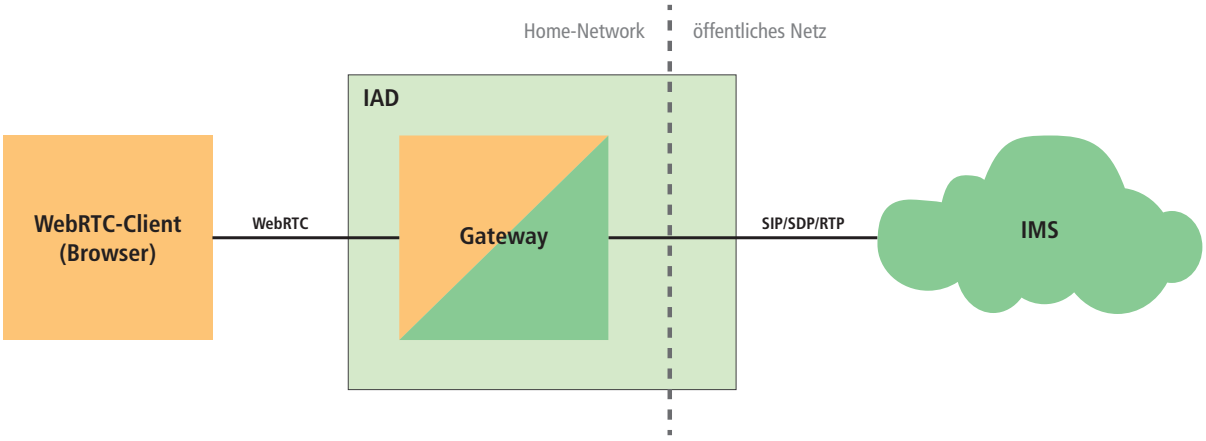

Abbildung 6: Funktionaler Aufbau Konzept I

[5] Im Pfad „WebRTC“ sind jeweils die WebRTC-typischen Protokolle für Signalisierung und Nutzdaten zusammengefasst: Websocket/SDP, SRTP, STUN/TURN

Vorteile	Nachteile
Kein zusätzliches Gerät notwendig, Modul kann durch Firmwareupgrade beim Kunden installiert werden	IAD möglicherweise nicht leistungsstark genug für Transcoding
Gespeicherte Daten (z.B. Accounts) aus dem IAD können verwendet werden	Erfordert Eingriff in das Betriebssystem des IAD
Integration in Weboberfläche des IAD möglich	Wenig Freiheitsgrade bei der Wahl der Programmiersprache und Plattform
Kein NAPT Traversal notwendig, da sich beide WebRTC-Endpunkte im selben Netz befinden	Interne Eigenschaften des IAD müssen berücksichtigt werden

Tabelle 2: Bewertung Konzept I

3.1.2 Konzept II: Zusatzgerät im Home-Network

Das Gateway wird unabhängig vom IAD bereitgestellt und auf einem zusätzlichen Gerät im Home-Network ausgeführt. Dabei kann ein Einplatinencomputer zum Einsatz kommen, z.B. Raspberry Pi [63] oder Radxa Rock [64]. Der Aufbau ist in Abbildung 7 dargestellt und die Vor- und Nachteile in Tabelle 3 aufgeführt.

Teilmodule des Gateways können in das IAD ausgelagert werden. Beispielsweise kann der Webserver auf dem IAD betrieben werden, während das rechenaufwendige Transcoding auf einem anderen Gerät durchgeführt wird. Dieser hybride Ansatz wird allerdings im Rahmen dieser Arbeit nicht weiter betrachtet, da er sich aus den Ansätzen der Konzepte I und II zusammensetzt.

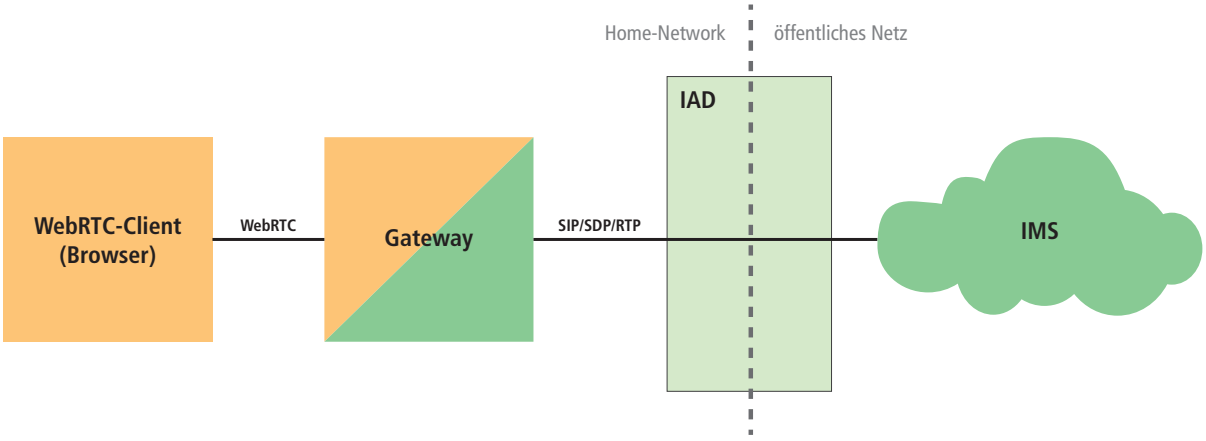

Abbildung 7: Funktionaler Aufbau Konzept II

Vorteile	Nachteile
Mehr Hardwareressourcen möglich und beliebig skalierbar	Zusätzliches Gerät beim Kunden notwendig
Interne Eigenschaften des IAD können vernachlässigt werden	Modul besitzt eigenen Lebenszyklus und muss unabhängig vom IAD gewartet werden
Mehr Freiheitsgrade bei der Wahl der Umgebung	
Kein NAPT Traversal notwendig, da sich beide WebRTC-Endpunkte im selben Netz befinden	

Tabelle 3: Bewertung Konzept II

3.1.3 Konzept III: Gateway im öffentlichen Netz

Das Gateway wird im öffentlichen Netz angesiedelt und ist für den Anwender über eine öffentliche IP-Adresse erreichbar. Der Dienst kann vom Provider bereitgestellt und betreut werden oder es kann auf Cloud-Lösungen zurückgegriffen werden, z.B. twilio[6]. Abbildung 8 zeigt den funktionalen Aufbau des Konzepts, in Tabelle 4 sind die Vor- und Nachteile dargestellt.

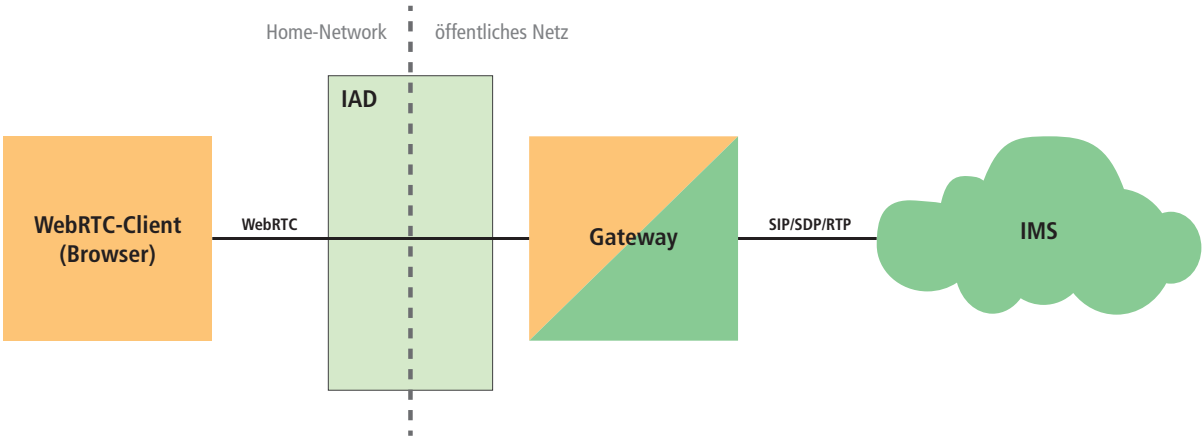

Abbildung 8: Funktionaler Aufbau Konzept III

[6]Das US-amerikanische Unternehmen *twilio* bietet eine breite Auswahl an cloud-basierten Kommunikationsdiensten (IaaS), unter anderem das Produkt „SIP to WebRTC“ [65].

Vorteile	Nachteile
Kein zusätzliches Gerät beim Kunden nötig	Höhere Investitions- und Betriebskosten für den Provider
Kann zentral gewartet werden	NAPT Traversal nötig, da sich ein WebRTC-Endpunkt hinter einem NAPT-Gateway befindet
Kein Kundensupport für Gateway-Hardware notwendig	Vom Provider abhängig und kann nur mit hohem Aufwand vom Kunden selbst betrieben werden
	Single-Point-of-Failure für viele Kunden

Tabelle 4: Bewertung Konzept III

3.2 Gegenüberstellung der Konzepte

Die Konzepte können aus der Sicht verschiedener beteiligter Rollen unterschiedlich bewertet werden.

Aus Sicht des **Providers** überwiegen die Vorteile des Konzepts III (Gateway im öffentlichen Netz). Der Provider integriert das Gateway in die vorhandene Infrastruktur und stellt dem Kunden die Schnittstellen zur Verfügung. Der Endgerätesupport beim Kunden für ein Gateway-Modul entfällt. Um einem Single-Point-of-Failure vorzubeugen, kann die Infrastruktur redundant ausgelegt werden.

Aus Sicht des **Kunden** sticht der Nachteil der Providerabhängigkeit von Konzept III hervor. Die Integration eines WebRTC-Gateways in die Provider-Infrastruktur kann nicht vom Kunden vorgenommen werden. Daher sind aus Sicht des Kunden die Konzepte I und II zu bevorzugen. Das Konzept II (Zusatzgerät im Home-Network) bringt zwar zunächst höhere Investitionskosten mit sich, lässt aber mehr Freiheitsgrade bei der Wahl der Entwicklungsumgebung. Die im Unterunterabschnitt 2.5.2 vorgestellten Gateway-Lösungen sind für die Installation in einer Linux-Umgebung ausgelegt. Die Betriebssysteme freetz und OpenWrt, die für den Betrieb auf Embedded Devices wie IAD entwickelt wurden, basieren zwar auf Linux, allerdings müssen die internen Eigenschaften dieser Systeme berücksichtigt werden.

Bei der Auswahl eines Konzepts steht die Machbarkeit der Umsetzung im Rahmen dieser Bachelorarbeit im Vordergrund. Daher wird im Sinne der einfacheren Implementierung das **Konzept II (Zusatzgerät im Home-Network)** bevorzugt.

4 Integration des Gateways

4.1 Grundvoraussetzungen

4.1.1 Integrationskonzept

Im Unterabschnitt 3.2 wird das Konzept des zusätzlichen Geräts im Home-Network zur Umsetzung in dieser Arbeit ausgewählt. Einen detaillierten Überblick über die Integration eines WebRTC-Gateways nach dem vorgestellten Konzept zeigt Abbildung 9.

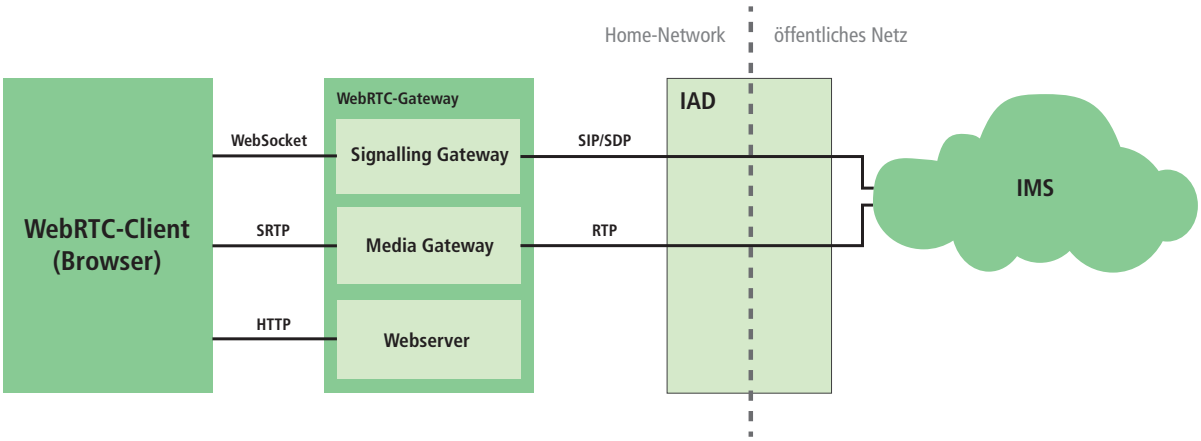

Abbildung 9: Funktionaler Aufbau: Integration eines WebRTC-Gateways in ein Zusatzgerät im Home-Network

4.1.2 Auswahl einer Gateway-Lösung

Im Unterunterabschnitt 2.5.2 werden drei Gateway-Lösungen vorgestellt, von denen eine für die Realisierung ausgewählt wird. Das Projekt overSIP ist nicht für den vorliegenden Zweck geeignet, da es keine Medienwandlung und SIP-Registrierung unterstützt. Während Janus lediglich Schnittstellen und Werkzeuge bereitstellt und damit zunächst Programmieraufwand auf der Serverseite erfordert, bietet webrtc2sip bereits eine vollständige Gateway-Funktionalität, die allen gestellten Anforderungen gerecht wird. Janus verfügt jedoch über eine übersichtlichere Struktur im Dateisystem als webrtc2sip. Außerdem wird eine ausführliche Dokumentation für Entwickler angeboten. Für webrtc2sip gibt es hingegen eine aktive Community. Darüber hinaus existiert mit SIPml5 ein kompatibler JavaScript-SIP-Stack für webrtc2sip, der als Grundlage für die Webapplikation dienen kann.

Aus Sicht des Autors eignet sich daher webrtc2sip am Besten zur Betrachtung im Rahmen dieser Arbeit.

4.2 Doubango webrtc2sip

Die Architektur und Teilmodule von webrtc2sip sind im Anhang B dargestellt. Der *SIP-Proxy* übernimmt die Funktion des Signalling Gateway (vgl. Abbildung 9). Als Signalisierungsprotokoll kommt SIP over Websocket zum Einsatz. Die Terminierung der STUN/TURN-Nachrichten wird vom *RTCWebBreaker* vorgenommen. Das Modul *Media Coder* implementiert das Media Gateway. Ein Webserver ist nicht Teil der Software, kann aber durch Anwendungen wie Node.js [66] oder Apache [67] bereitgestellt werden. Das Modul *click-to-call* ist für das vorliegende Projekt zunächst uninteressant[7].

4.3 Installation der Gateway-Software

4.3.1 Installation auf dem Raspberry Pi

Für den Betrieb von webrtc2sip in der Laborumgebung wird ein Raspberry Pi 2 verwendet. Der Einplatinencomputer verfügt über einen ARM-Prozessor (Quadcore mit 900 MHz) und 1 GB RAM [63]. Mit dem Projekt Raspbian existiert eine Variante der verbreiteten Linux-Distribution Debian, die für den Raspberry Pi optimiert ist [68].

Für webrtc2sip sind keine offiziellen Installationspakete verfügbar. Stattdessen muss der Quellcode selbst kompiliert werden. Es existiert ein inoffizielles Paket-Repository [69], dessen Pakete allerdings nicht für die ARM-Architektur geeignet sind. Für die Installation existiert eine offizielle Anleitung [70], die für die Linux-Distribution CentOS angepasst ist. Darüber hinaus stellt der Herausgeber des inoffiziellen Repositories ebenfalls eine Anleitung zur Verfügung [71]. Auf Basis dieser Dokumente und Hinweisen aus Diskussionsforen wurde die Installation auf die Raspbian-Umgebung angepasst. Die vollständige Installationsbeschreibung ist im Anhang C zu finden.

4.3.2 Konfiguration

webrtc2sip verfügt über eine Konfigurationsdatei im XML-Format, über die sich das Verhalten des Gateways steuern lässt. Eine Beispielkonfiguration ist im Unterabschnitt C.7 aufgeführt. Wichtige Konfigurationsparameter sind im Folgenden erläutert. Eine ausführliche Dokumentation ist im *Technical Guide* zu webrtc2sip verfügbar. [72]

transport legt die zu verwendenden Transportprotokolle, IP-Adressen und Ports fest.

enable-media-coder aktiviert das Modul *Media Coder*, das für das Transcoding im Gateway verantwortlich ist. Unterstützen Browser und öffentliches Netz einen selben Codec, kann das Transcoding auch deaktiviert werden.

[7] Mit diesem Modul lassen sich Click-to-call-Funktionen in den Browser integrieren. Das Gateway agiert als SIP-Client gegenüber dem Netz, der vom Browser gesteuert wird. Dadurch müssen SIP-Zugangsdaten nicht im Browser hinterlegt werden, was für eine Anruffunktion auf einer öffentlichen Website interessant ist.

codecs legt die zu verwendenden Codecs fest. Dabei gelten die eingetragenen Codecs auf beiden Seiten des Gateways gleichermaßen. Unter anderem sind folgende Einträge möglich: `pcma;opus;g722`

enable-icestun aktiviert die Mechanismen zum NAPT Traversal. Diese Eigenschaft kann deaktiviert werden, wenn das Gateway im Home-Network angesiedelt ist und die WebRTC-Endpunkte gegenseitig direkt erreichbar sind.

ssl-certificates legt die TLS-Zertifikate zur Verschlüsselung des SRTP-Traffics fest. Sie müssen vorher erstellt und im Dateisystem hinterlegt werden (z.B. unter Verwendung von `openssl` [73]).

4.4 Entwicklung der Webapplikation

Neben einem reinen Gateway-Server ist für die Interaktion mit dem Benutzer eine grafische Benutzeroberfläche (GUI) nötig. Da WebRTC browserbasiert arbeitet, wird ein Client grundsätzlich als Webapplikation realisiert.

Als Framework für die SIP-Signalisierung kommt SIPml5 zum Einsatz, das ebenfalls ein Projekt der Doubango Telecom und offiziell kompatibel mit webrtc2sip ist [54]. Außerdem wird das Javascript-Framework AngularJS[8] [74] sowie das CSS-Framework Bootstrap[9] [75] eingesetzt. Als serverseitige Umgebung wird Node.js[10] [66] ausgewählt.

Vor der Implementation der WebGUI wurden zunächst die Anforderungen als User Stories formuliert. Sie dienen als Grundlage für die Funktionsbeschreibung in den folgenden Abschnitten.

4.4.1 Rufaufbau

> *Als Benutzer möchte ich einen Anruf zu einer öffentlichen Telefonnummer tätigen und Anrufe empfangen können, um meinen Browser wie ein Telefon verwenden zu können.*

Die Anruffunktion ist das zentrale Element der Anwendung. Dem Benutzer steht nach der SIP-Registrierung ein Feld zur Eingabe der Zielrufnummer zur Verfügung (Abbildung 10). Über einen Button kann der Anruf gestartet werden. Der Rufaufbau wird dem Benutzer durch einen Rufton signalisiert. Eingehende Anrufe werden akustisch durch einen Klingelton und optisch in der UI angezeigt. Der Benutzer hat die Möglichkeit, den Anruf anzunehmen oder abzuweisen (Abbildung 11). Ist eine Anrufsitzung aufgebaut, kann diese über einen Button beendet werden (Abbildung 12).

[8] AngularJS, ein JavaScript-Framework von Google für die Entwicklung von Single-Page-Applikationen

[9] Bootstrap, eine umfangreiche Style-Bibliothek von Twitter. Mit Bootstrap lässt sich die Anforderung eines „Responsive Designs“ gut umsetzen, das sich an die Bildschirmgröße verschiedener Endgeräte anpasst.

[10] Node.js, eine Javascript-basierte Serverumgebung für Netzwerkanwendungen, basiert auf der Google V8 JavaScript Runtime

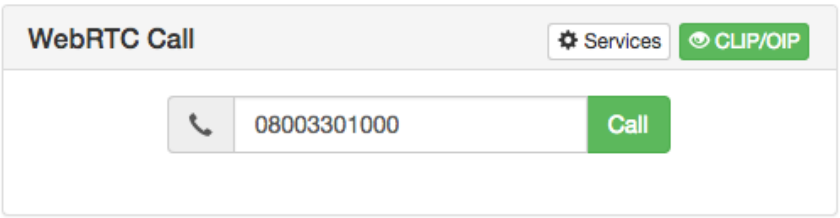

Abbildung 10: Wählfeld

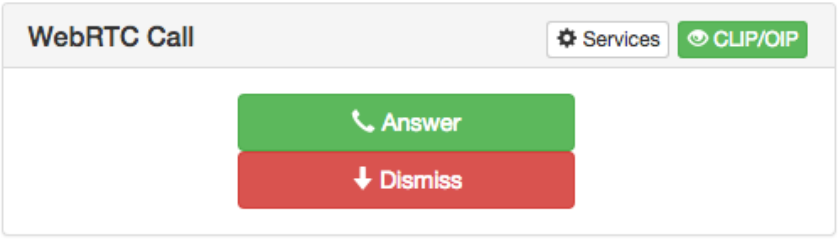

Abbildung 11: Eingehender Anruf

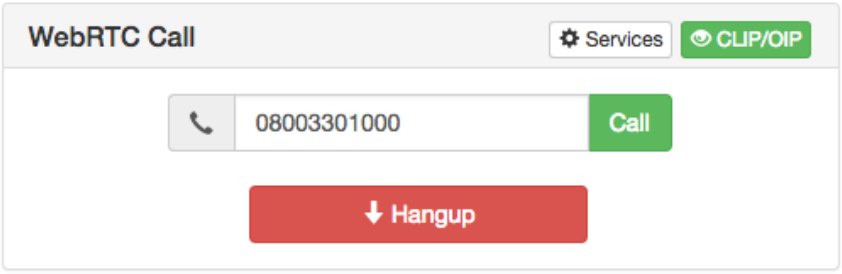

Abbildung 12: Aktiver Anruf

4.4.2 Supplementary Services

> *Als Benutzer möchte ich zusätzliche Dienstmerkmale über die Oberfläche verwalten können, ohne Codes über die Tastatur eingeben zu müssen.*

Supplementary Services können unter anderem mittels VSC gesteuert werden, die über die Telefontastatur vom Benutzer eingegeben werden (vgl. Unterunterabschnitt 2.3.2). Ausgewählte Dienstmerkmale können über die Oberfläche aktiviert werden. Bei Auswahl eines Dienstes wird eine `INVITE`-Nachricht mit dem entsprechenden VSC an das Netz abgesetzt. Exemplarisch wurde dafür der Dienst Anrufweiterschaltung in den Varianten „Sofort“, „bei Nichtmelden“ und „bei besetzt“ implementiert (Abbildung 13). Weitere Dienste sind möglich, da sich die Steuerung nur in den verwendeten Service Codes unterscheidet.

Die Rufnummernunterdrückung (OIR) wird über das Signalisierungsprotokoll gesteuert. Zur fallweisen Unterdrückung der eigenen Rufnummer kann vor dem Aufbau eines ausgehenden Anrufs der Button „CLIR/OIR“ aktiviert werden (Abbildung 10). Diese Funktion sorgt dafür, dass in die `INVITE`-Nachricht die erforderlichen Header (vgl. Unterabschnitt 4.5) eingefügt werden, um das Feature OIR zu aktivieren.

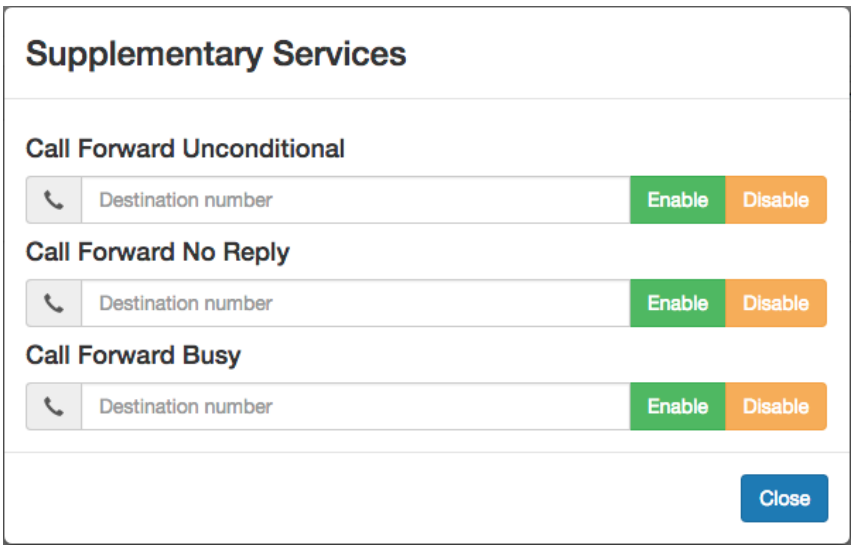

Abbildung 13: Steuerung der Supplementary Services

4.4.3 Account-Verwaltung

> *Als Benutzer möchte ich mehrere Accounts verwalten können, um eine Auswahl zwischen verschiedenen Nummern zu haben. Die Accounts sollen zentral auf dem Gateway gespeichert werden.*

Für diese Anforderung wurde eine separate Oberfläche in der UI angelegt. Die eingetragenen SIP-Accounts werden tabellarisch aufgelistet (Abbildung 14). Es können neue Accounts eingetragen (Abbildung 15) und vorhandene bearbeitet und gelöscht werden. Gespeicherte Accounts können vom Benutzer ausgewählt werden, um damit die SIP-Registrierung durchführen zu können. Dazu existiert ein Dropdown-Menü und Buttons zum Registrieren (Register) und Abmelden (Unregister) in der Menüleiste (Abbildung 16).

Die Accountdaten werden persistent auf dem Webserver gespeichert. Das Backend wurde als REST-API mit Node.js umgesetzt und speichert die Daten in einer JSON-Struktur als Datei auf dem Server.

Abbildung 14: Liste von Accounts

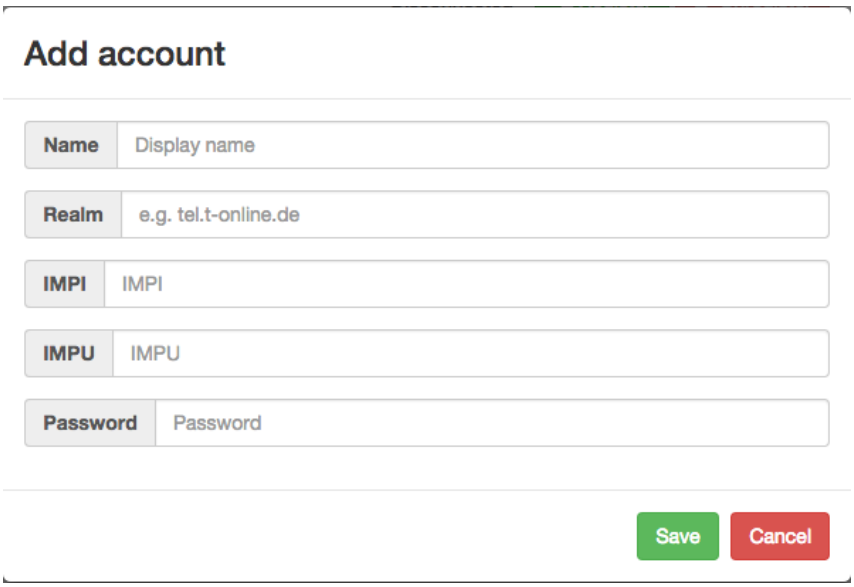

Abbildung 15: Hinzufügen eines neuen Accounts

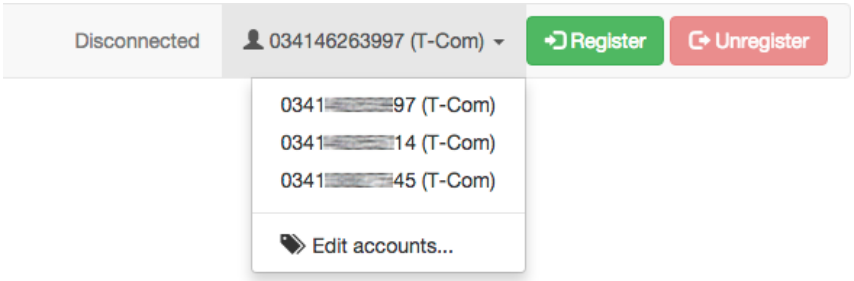

Abbildung 16: Auswahl des zu verwendenden Accounts

4.5 Erweiterung von webrtc2sip für OIR

Die Dienstmerkmale Rufnummernanzeige (OIP) und Rufnummernunterdrückung (OIR) werden direkt durch das Signalisierungsprotokoll gesteuert (vgl. Unterunterabschnitt 2.3.1). Während OIP ohne zusätzliche Angaben automatisch aktiviert ist, wird OIR durch zusätzliche SIP-Header in der `INVITE`-Nachricht fallweise aktiviert, wie Abbildung 17 zeigt [76, 77].

Abbildung 17: SIP-Nachricht `INVITE` mit markierten Header-Feldern für OIR

Das SIP-Proxy-Modul in webrtc2sip arbeitet allerdings statusbehaftet und übernimmt zusätzliche Header-Felder nicht transparent in die ausgehenden SIP-Nachrichten. Stattdessen werden nur die Header weitergeleitet, für deren Verwendung webrtc2sip eine entsprechende Regel vorliegt. Das Dienstmerkmal OIR mit den angegebenen Header-Feldern wird vom Gateway nicht unterstützt.

webrtc2sip wurde daher um die Funktionalität erweitert, auch die für OIR nötigen Header erkennen und weiterleiten zu können. Die Erweiterungen in der Datei `mp_wrap.cc` sind in Listing 1 aufgeführt.

```
        //keep "P-Preferred-Identity" and "Privacy" headers
472     char* headerPPI = tsk_null;
        char* headerPrivacy = tsk_null;
474     headerPPI = const_cast<SipMessage*>(pcMsgLeft)->
  getSipHeaderValue("P-Preferred-Identity");
        headerPrivacy = const_cast<SipMessage*>(pcMsgLeft)->
  getSipHeaderValue("Privacy");
476     if(!tsk_strnullOREmpty(headerPPI)){
          const_cast<SipSession*>(oCallSessionRight->getWrappedSession
  ())->addHeader("P-Preferred-Identity", headerPPI);
478     }

480     if(!tsk_strnullOREmpty(headerPrivacy)){
          const_cast<SipSession*>(oCallSessionRight->getWrappedSession
  ())->addHeader("Privacy", headerPrivacy);
482     }
```

Listing 1: Erweiterungen in der Datei `mp_wrap.cc` zur Unterstützung der OIR-Header

4.6 Modularisierung mit Docker

Das Gateway und die Webapplikation wurden nativ auf einem Raspberry Pi installiert (vgl. Unterunterabschnitt 4.3.1). Um das Deployment der Software im Produktivumfeld zu vereinfachen, sollten alle Komponenten in einen abgeschlossenen Container integriert werden. Dazu wurde die Virtualisierungslösung Docker [78] ausgewählt. Docker ermöglicht die Isolation von Anwendungen in Containern. Ein Container läuft als einzelner Prozess auf dem Hostbetriebssystem. Dabei wird für einen Container kein komplettes Betriebssystem virtualisiert, sondern ein Minimalsystem und die jeweilige zu virtualisierende Anwendung. Den Kernel kann das virtualisierte Betriebssystem mit dem Hostsystem teilen[11].

Neben Containern führt Docker den Begriff der Images ein. Ein Image ist ein Abbild einer Anwendung, aus dem Container erstellt werden können. Ein Container ist eine Instanz eines Images, die eine Anwendung ausführt.

Docker wird über die Kommandozeile bedient [80]. Anweisungen zum Erstellen eines Docker-Images können in einem `Dockerfile` zusammengefasst werden. Daraus kann automatisiert ein Image generiert werden. Viele gebräuchliche Anwendungen und Betriebssysteme sind als Images zum Download in einer Registry, dem sogenannten DockerHub [81] verfügbar.

Die grundlegende Architektur von Docker ist in Abbildung 18 beschrieben.

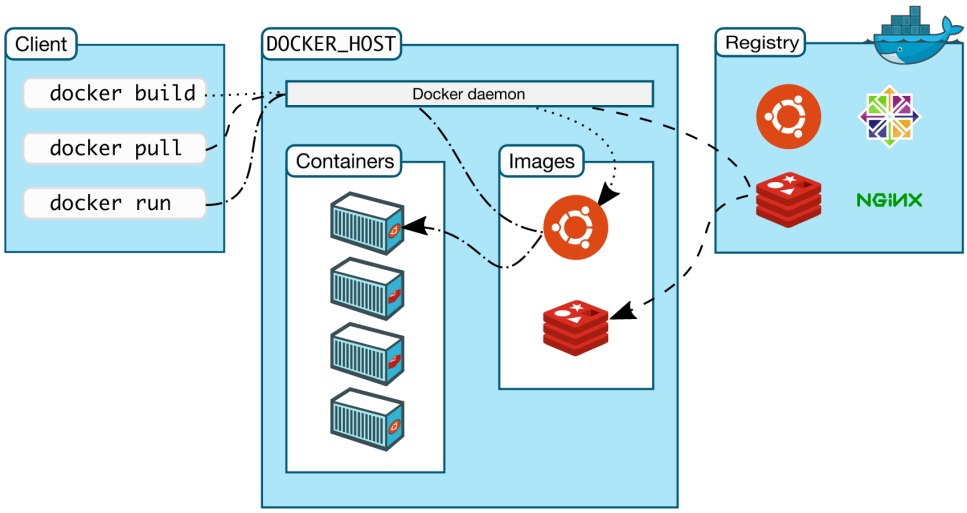

Abbildung 18: Docker-Architektur [78]

Eine ausführliche Dokumentation zur Erstellung eines Docker-Images mit webrtc2sip und der dazugehörigen Webapplikation befindet sich im Anhang D.

Da bisher kein offizieller Build von Docker für die ARM-Architektur existiert [82], wurde für diese Arbeit auf das Projekt hypriot zurückgegriffen. hypriot ist ein Raspbian-Fork, in den Docker und nötige Werkzeuge bereits integriert sind [83].

[11] Docker verwendet Linux Containers (LXC), die diese Funktionalität ermöglichen [79].

5 Auswertung

5.1 Signalisierungsabläufe mit verschiedenen Szenarien

Nachfolgend sind drei Signalisierungsabläufe für verschiedene Anwendungsfälle als Message Sequence Chart dokumentiert. Als Grundlage dient der erfasste Netzwerkverkehr an der Netzwerkschnittstelle des webrtc2sip-Gateways.

5.1.1 Registrierung

Vor dem Aufbau einer Call Session ist zunächst die SIP-Registrierung am Netz notwendig. Die Registrierung wird vom Webbrowser angestoßen und vom Gateway weitergeleitet. Der Ablauf entspricht dem Basic Call Flow nach RFC 3665 [84][S. 5].

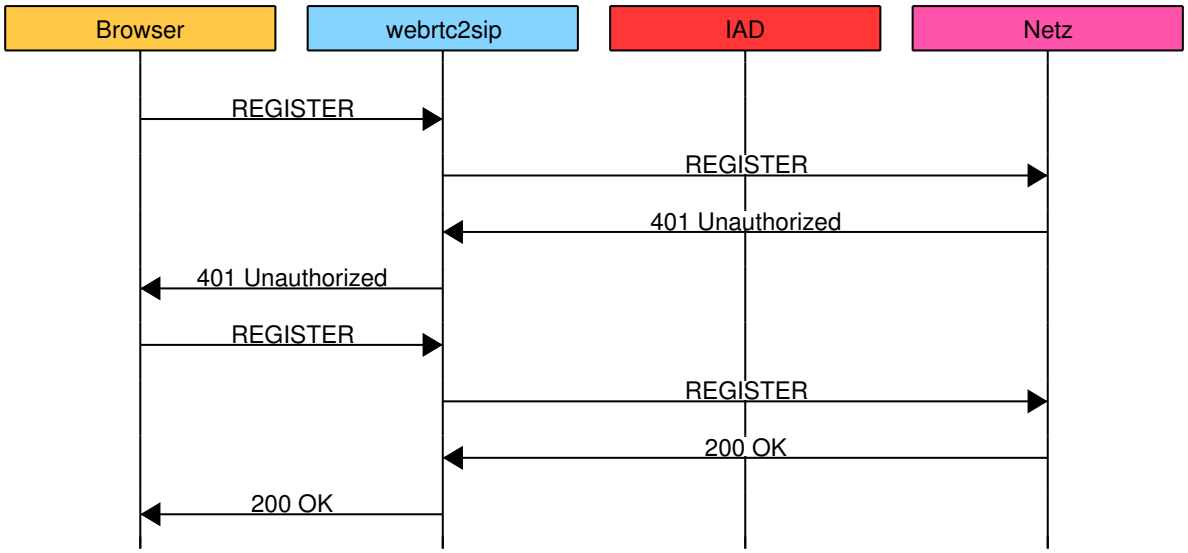

Abbildung 19: Signalisierungsablauf SIP-Registrierung

5.1.2 Call I: Browser (Opus) zum Festnetz (G.711)

Dieser Ablauf zeigt einen erfolgreichen Call vom Browser unter Verwendung des Opus-Codecs zu einem öffentlichen Telefonanschluss mit dem Codec G.711. Im Gateway ist das Transcoding der Nutzdaten aktiviert.

Der Call wird vom Browser eingeleitet und vom WebRTC-Gateway weitergeleitet. Es ist erkennbar, dass das Gateway statusbehaftet arbeitet und die Nachrichten nicht transparent weiterleitet. Stattdessen agiert es als User Agent jeweils gegenüber dem Browser und dem Netz. Beispielsweise wird die Nachricht `100 Trying` an den Browser bereits abgesetzt, bevor die entsprechende Nachricht vom Netz empfangen wurde.

Der Ablauf entspricht grundsätzlich dem Basisablauf nach RFC 3665 [84][S. 46] mit der

Einschränkung, dass die Signalisierung zwischen Netz und dem B-Teilnehmer im vorliegenden Beispiel nicht dokumentiert ist.

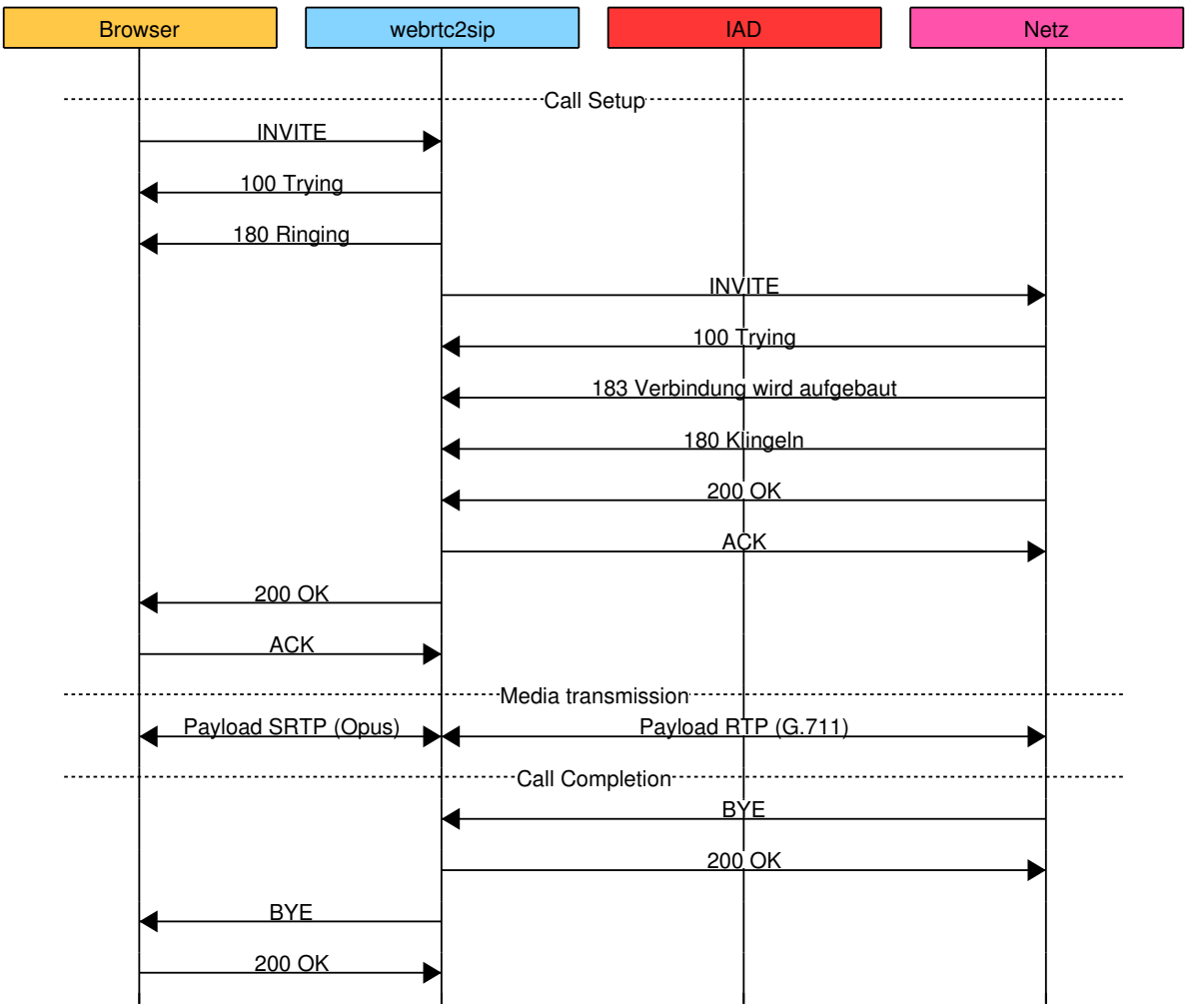

Abbildung 20: Signalisierungsablauf für Call I

5.1.3 Call II: Browser (Opus) zu Browser (Opus)

Im dritten Testfall wird ein Rufaufbau zwischen zwei Browsern unter Verwendung des Opus-Codecs eingeleitet. Nach dem vollständigen Rufaufbau wird die Sitzung vom Netz beendet und dem A-Teilnehmer mit der Meldung `500 Server Internal Error` quittiert. Der B-Teilnehmer wird mit `BYE` über das Ende der Sitzung benachrichtigt.

Die Ursachen für den Sitzungsabbruch konnten nicht ermittelt werden. Nach Vermutung des Autors wird der Opus-Codec vom Netz der Deutschen Telekom nicht unterstützt und aus diesem Grund die Sitzung beendet. Fraglich ist, wieso der Sitzungsabbruch nicht bereits bei der Medienaushandlung mittels SDP (in den Nachrichten `INVITE` und dem daraus folgenden `200 OK`) eingeleitet wird, sondern erst beim Aufbau der RTP-Verbindung.

Die Eigenschaft des Gateways, statusbehaftet zu arbeiten, ist auch in diesem Beispiel erkennbar.

Im Sinne der Übersichtlichkeit sind die einzelnen SIP-Transaktionen farblich markiert. Den Nachrichten ist außerdem die Sequenznummer der jeweiligen Transaktion (CSeq) angehängt. Der komplette Mitschnitt im PCAP-Format ist auf der Begleit-DVD (Anhang E) zu finden.

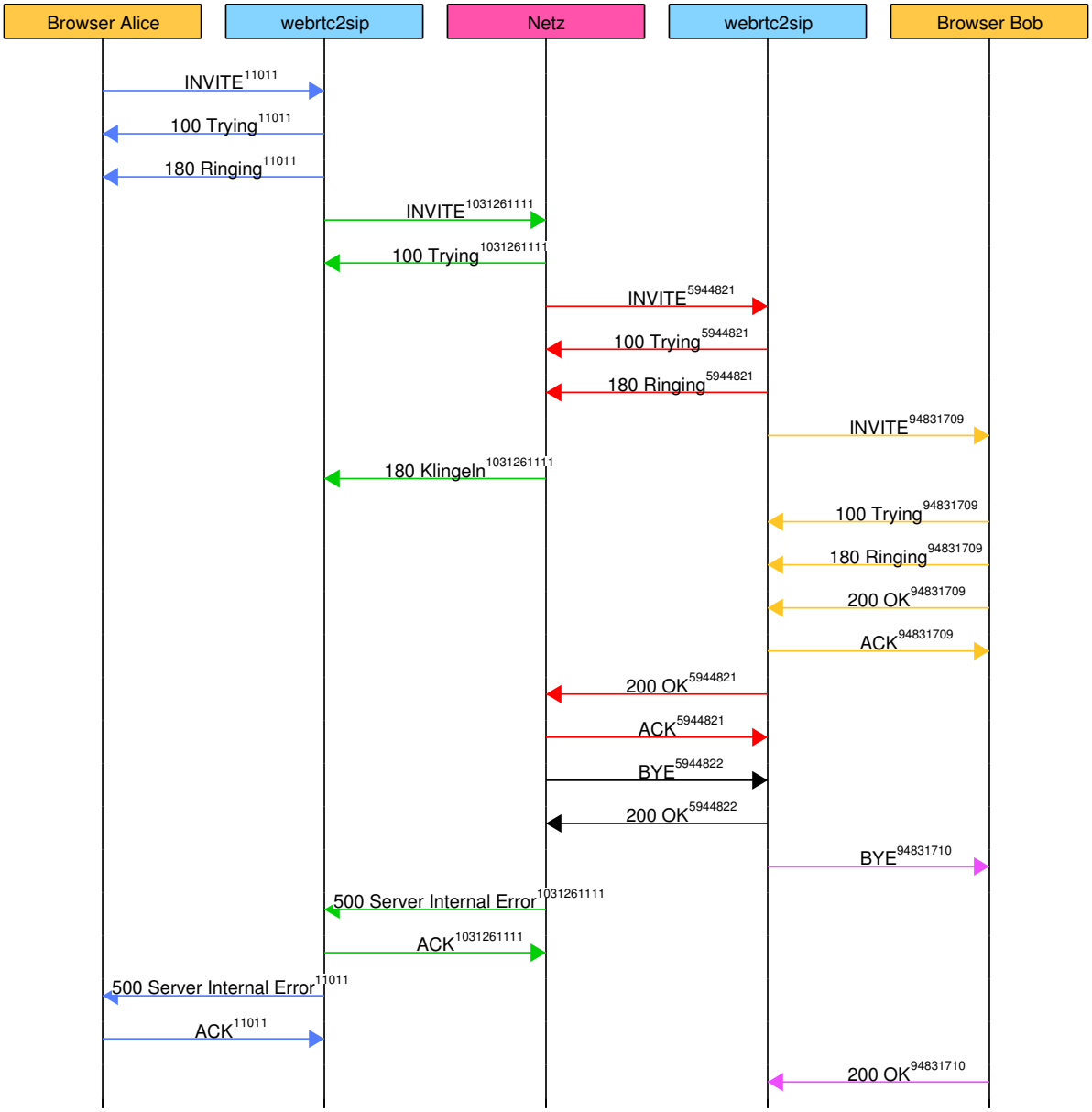

Abbildung 21: Signalisierungsablauf für Call II

5.2 Performance

Zur Betrachtung der Performance wurde der zeitliche Verlauf der CPU-Auslastung in drei Testfällen erfasst und grafisch dargestellt. Zur Erfassung kam das Shell-Skript aus Listing 2 zum Einsatz. Die Messungen wurden auf einem Raspberry Pi 2 ausgeführt, auf dem die Gateway-Software nativ installiert ist.

```
#!/bin/sh

# USAGE:
# ./trackcpu.sh PID FILE
# Example: ./trackcpu.sh 12345 cputrack1.txt

PID=$1
FILE=$2
TIME=0

STEP=0.5

while true; do
    clear

    CPU=$(ps -p $PID -o %cpu | awk 'FNR == 2' | sed -e 's/^[[:space
    :]]*//')
    TIME=`echo "$TIME+$STEP" | bc`
    echo "$TIME $CPU" >> $FILE
    echo "Time: $TIME sec"

    sleep $STEP
done

# NOTE: The time basis is not accurate at all! Calculactions take their
# time which should be added to the actual step time. An asynchronous
# approach would be quite more suitable here. However, since time
# accuracy is not primary important in these test cases we will ignore
# this here.
```

Listing 2: Shell-Skript zur zeitlichen Erfassung der CPU-Auslastung eines Prozesses

5.2.1 Testfall I: Call mit aktiviertem Transcoding

Im ersten Testfall wird ein Call von der Webapplikation zu einem entfernten Festnetztelefon aufgebaut. Das Transcoding im Gateway ist aktiviert, die Nutzdaten werden zwischen Opus und G.711 transcodiert. Der zeitliche Verlauf der CPU-Auslastung ist in Abbildung 22 dargestellt.

Der Call beginnt bei $t = 17s$. Die CPU-Last steigt zunächst stark bis zu einem Wert von 20 % und wächst bis zum Ende der Sitzung weiter an bis zu einem Wert von 30 %. Nach dem Ende des Calls ($t = 522s$) fällt die Last nur langsam ab und erreicht nach etwa 3 Minuten einen Wert von 23 %.

Auffällig ist, dass die CPU-Last nicht über die Dauer der gesamten Sitzung auf dem selben Niveau bleibt, sondern kumuliert wird. Außerdem werden die Ressourcen nach Beendigung der Sitzung nicht sofort freigegeben, obwohl kein Transcoding mehr nötig ist. Nach Vermutung des Autors werden die Ressourcen für das Transcoding nicht wiederverwendet, sondern für Blöcke von Nutzdaten neue Ressourcen beansprucht, die nach der Verwendung nur langsam abgebaut werden. Das führt dazu, dass die CPU-Last im Laufe der Sitzung steigt.

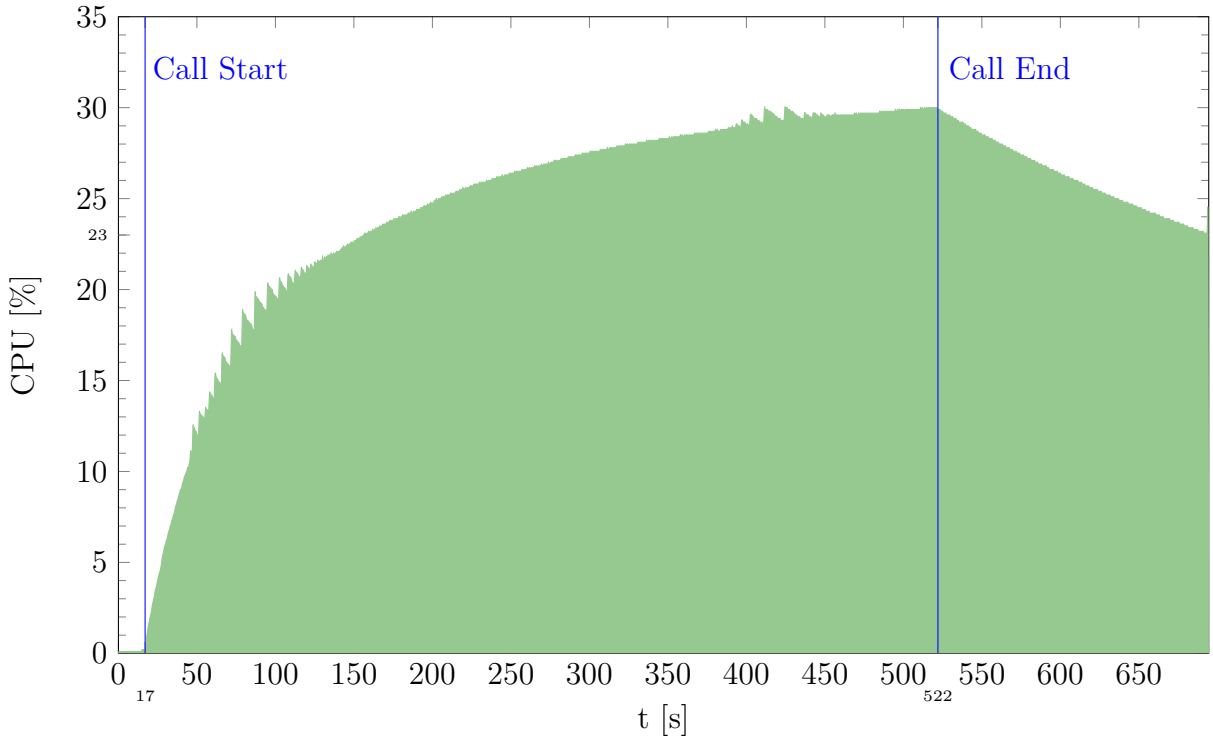

Abbildung 22: CPU-Auslastung mit einem Call und Transcoding (Opus ↔ G.711)

5.2.2 Testfall II: Zwei parallele Calls mit aktiviertem Transcoding

Im zweiten Testfall werden zwei Calls von WebRTC zu externen Festnetztelefonen zeitversetzt aufgebaut. Das Transcoding im Gateway ist aktiv und die Nutzdaten werden von Opus zu G.711 transcodiert. Abbildung 23 zeigt den zeitlichen Verlauf der CPU-Auslastung für diesen Testfall. Zu beachten ist die Skalierung der Achsen, die sich von den anderen Testfällen unterscheidet.

Wie im ersten Testfall steigt die CPU-Last nach dem Start des ersten Calls ($t = 21s$) steil an. Mit dem Aufbau der zweiten Sitzung ($t = 82s$) erhöht sich die Last und steigt mit dem Faktor 2 an. Bis zum Ende des ersten Calls bei $t = 522s$ steigt die Last weiter, erreicht einen Wert von 57,8 % und fällt schließlich langsam ab. Mit Beendigung des zweiten Calls bei $t = 582s$ sinkt die CPU-Auslastung schneller und erreicht nach 3 Minuten einen Wert von 43,6 %.

Dieser Verlauf zeigt die selben Auffälligkeiten wie im ersten Testfall. Die Leistung ist nicht konstant, sondern steigert sich im Verlauf der Sitzung. Beide Calls beanspruchen unabhängig voneinander eine ähnliche CPU-Last von jeweils etwa 30 %. Dieser Umstand bestätigt die Vermutung des Autors zum ersten Testfall.

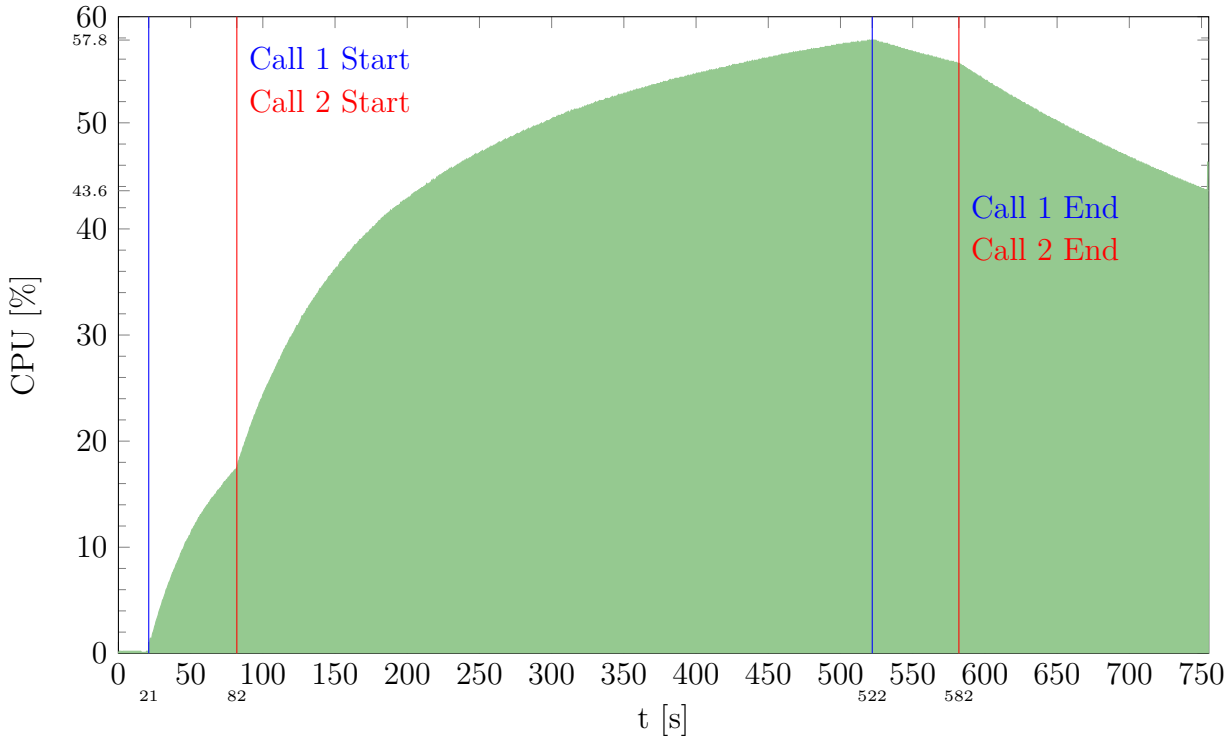

Abbildung 23: CPU-Auslastung mit zwei Calls und Transcoding (Opus ↔ G.711)

5.2.3 Testfall III: Call ohne Transcoding

Im dritten Testfall wird ein Call vom Browser zu einem externen Festnetztelefon aufgebaut. Das Transcoding im Gateway ist nicht aktiviert und es wird auf beiden Seiten der Codec G.711 verwendet. Der Verlauf der CPU-Auslastung ist in Abbildung 24 dargestellt. Die Achsen sind zur besseren Vergleichbarkeit genauso skaliert wie im ersten Testfall.

Zum Beginn des Calls bei $t = 18s$ steigt die CPU-Auslastung sofort auf einen Wert von 2,7 % an und bleibt bis zum Ende des Calls ($t = 522s$) auf diesem Niveau. Anschließend fällt die Last innerhalb von 3 Minuten auf einen Wert von 2,1 %.

Da das Transcoding im Gateway deaktiviert ist, werden in diesem Testfall nur Ressourcen für die Signalisierung und die Umwandlung zwischen SRTP ↔ RTP benötigt. Die Leistung ist konstant und erhöht sich während der Sitzung nicht. Nach dem Abbau der Sitzung werden die Ressourcen wiederum nur langsam freigegeben, obwohl keine Verarbeitung mehr nötig ist. Dieses Verhalten ähnelt dem der anderen Testfälle.

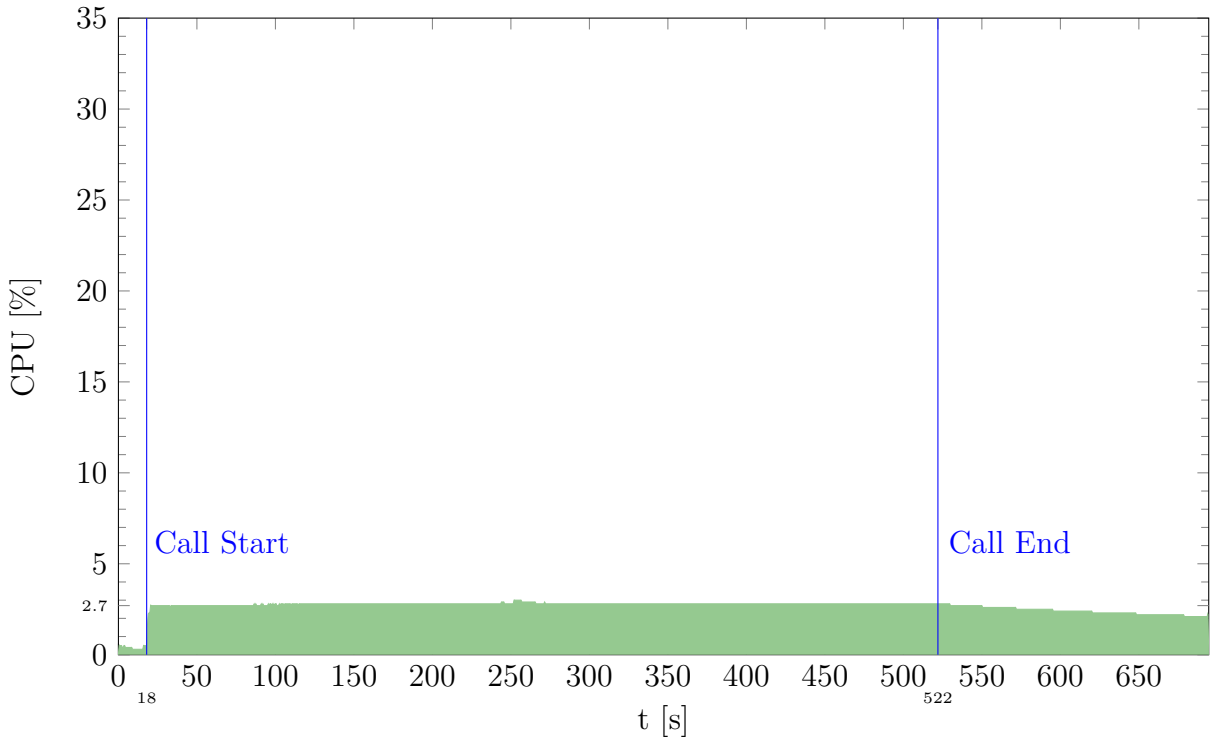

Abbildung 24: CPU-Auslastung mit einem Call ohne Transcoding

5.2.4 Schlussfolgerung

Das Gateway beansprucht ohne Transcoding nur wenig Ressourcen und ist auch auf Minimalhardware wie dem Raspberry Pi 2 problemlos lauffähig. Bei aktiviertem Transcoding verhält sich die Software nach Ansicht des Autors nicht optimal, da die Ressourcen nach Verwendung nicht freigegeben werden und die CPU-Last über die Zeit der Sitzung kumuliert wird. Für jeden aktiven Call wird auf dem Raspberry Pi 2 eine Last von 30 % pro Call erreicht. Pro CPU-Kern sind also nur maximal drei parallele Sitzungen möglich.

Die vorstehenden Betrachtungen beziehen sich allerdings ausschließlich auf den Raspberry Pi 2. Das Verhalten auf anderen Architekturen wurde im Rahmen dieser Arbeit nicht geprüft. Es kann nicht ausgeschlossen werden, dass sich die Software beispielsweise auf einem x86-System effizienter verhält.

6 Zusammenfassung und Ausblick

Die Integration von WebRTC-Funktionalitäten in die bestehende Telefonieinfrastruktur ist für Provider ein interessantes Feld, um Telefoniedienste auch endgeräteunabhängig anzubieten. Zur Integration ins NGN ist ein Gateway nötig, das den Übergang vom WebRTC-basierten Protokollstapel in die rein SIP-basierte Welt schafft. Dafür existieren bereits funktionierende Lösungen, sowohl proprietär als auch im Open-Source-Umfeld. Die vorliegende Arbeit stellt drei Integrationskonzepte vor, von denen eines unter Laborbedingungen demonstriert wurde. Die verwendete Gateway-Software ist auch auf Minimal-Hardware wie dem Raspberry Pi lauffähig und damit im Rahmen eines privaten Telefonanschlusses gut einsetzbar. Obwohl der Fokus dieser Arbeit primär auf der Unterstützung von Audio-Telefonaten lag, lässt sich die Video-Funktionalität nach Ansicht des Autors problemlos ergänzen. Die für das Video-Transcoding nötige Leistung kann allerdings vermutlich mit dem Raspbery Pi 2 nicht erbracht werden. Mit der Integration von Data-Channels könnte die bestehende Anwendung außerdem um Dienste wie Textnachrichten oder Dateiübertragung bereichert werden.

Die WebRTC-Unterstützung ist bisher von führenden Telekommunikationsanbietern noch nicht umgesetzt. Die Integration eines Gateways ins Home-Network nach dem vorgestellten Ansatz (Zusatzgerät im Home-Network) kann allerdings auch unabhängig von der Unterstützung im Netz bereits jetzt vom Kunden selbst vorgenommen werden. Ein solches lokales Gateway könnte auch vom Provider zur Verfügung gestellt werden. Die Telekom Innovation Laboratories beschäftigen sich seit 2012 mit dem Thema WebRTC [85]. Wann die Technologie für die Deutsche Telekom und andere führende Internetprovider Marktreife erreicht hat, bleibt nach aktuellem Stand allerdings noch abzuwarten.

A Übersicht Leistungsmerkmale am IP-basierten Anschluss der Deutschen Telekom

Die Tabelle gibt einen Überblick über die verfügbaren Leistungsmerkmale und Funktionen am IP-basierten Anschluss der Deutschen Telekom [86]. Die dritte Spalte gibt an, ob das jeweilige Leistungsmerkmal in ETSI ETS 300 738 [44] standardisiert ist. Klammern geben an, dass der jeweilige Service Code für eine andere Funktion nach ETSI ETS 300 738 vorgesehen ist, hier also abweichend von der Spezifikation verwendet wird.

Beschreibung	**Service Code**	**Standard nach ETSI ETS 300 738**
Zurücksetzen aller Leistungsmerkmale in den Auslieferungszustand für die genutzte Rufnummer	001	ja
Alle abgehenden Anrufe sperren	03	(nein) aber 33
Abgehende Anrufe nur zu Rufnummern auf Positivliste zulassen	04	(nein)
Aktivierte Anrufsperren zu bestimmten Rufnummerngassen gesammelt aktivieren oder deaktivieren	05	nein
Abgehende Anrufe zu 0900-Rufnummern sperren	051	nein
Abgehende Anrufe zu 0137-Rufnummern sperren	052	(nein)
Abgehende Anrufe zu 0180-Rufnummern sperren	053	nein
Abgehende Anrufe zu Auslands-Rufnummern sperren	054	nein
Abgehende Anrufe zu Interkontinental-Rufnummern sperren	055	nein
Sofortige Anrufweiterschaltung (AWS-sofort)	21	ja
Selektive Anrufweiterschaltungen (AWS) deaktivieren	211	(ja)
Sofortige Anrufweiterleitung selektiv	212	ja
Selektive Anrufweiterschaltung (AWS) bei Nichtmelden	213	nein
Selektive Anrufweiterleitung (AWS) bei besetzt	214	nein
Rufnummernanzeige des Anrufers (CLIP)	30	ja
Eigene Rufnummer fallweise übermitteln oder unterdrücken (CLIR)	31	ja
Eigene Rufnummer übermitteln oder unterdrücken (CLIR)	32	(nein)
Alle eingehenden Anrufe abweisen	335	nein
Alle anonymen Anrufe abweisen	336	nein

Alle weitergeleiteten Anrufe abweisen	337	nein
Anrufer auf Negativliste für kommende Anrufe abweisen	338	nein
Anrufer auf Positivliste für kommende Anrufe zulassen	339	nein
Rufnummern auf Negativliste für abgehende Anrufe sperren	59	(nein)
Anrufweiterschaltung (AWS) bei Nichtmelden	61	ja
Anrufweiterschaltung (AWS) bei nicht registriert	62	ja
Anrufweiterschaltung (AWS) bei besetzt	67	ja
Abweisen unerwünschter Anrufer mit unterdrückter Rufnummer (virtuelle Negativliste) für alle Rufnummern	934	nein
Ändern der PIN für die Steuerung von Telefonie-Leistungsmerkmalen	99	ja

B Architektur von webrtc2sip

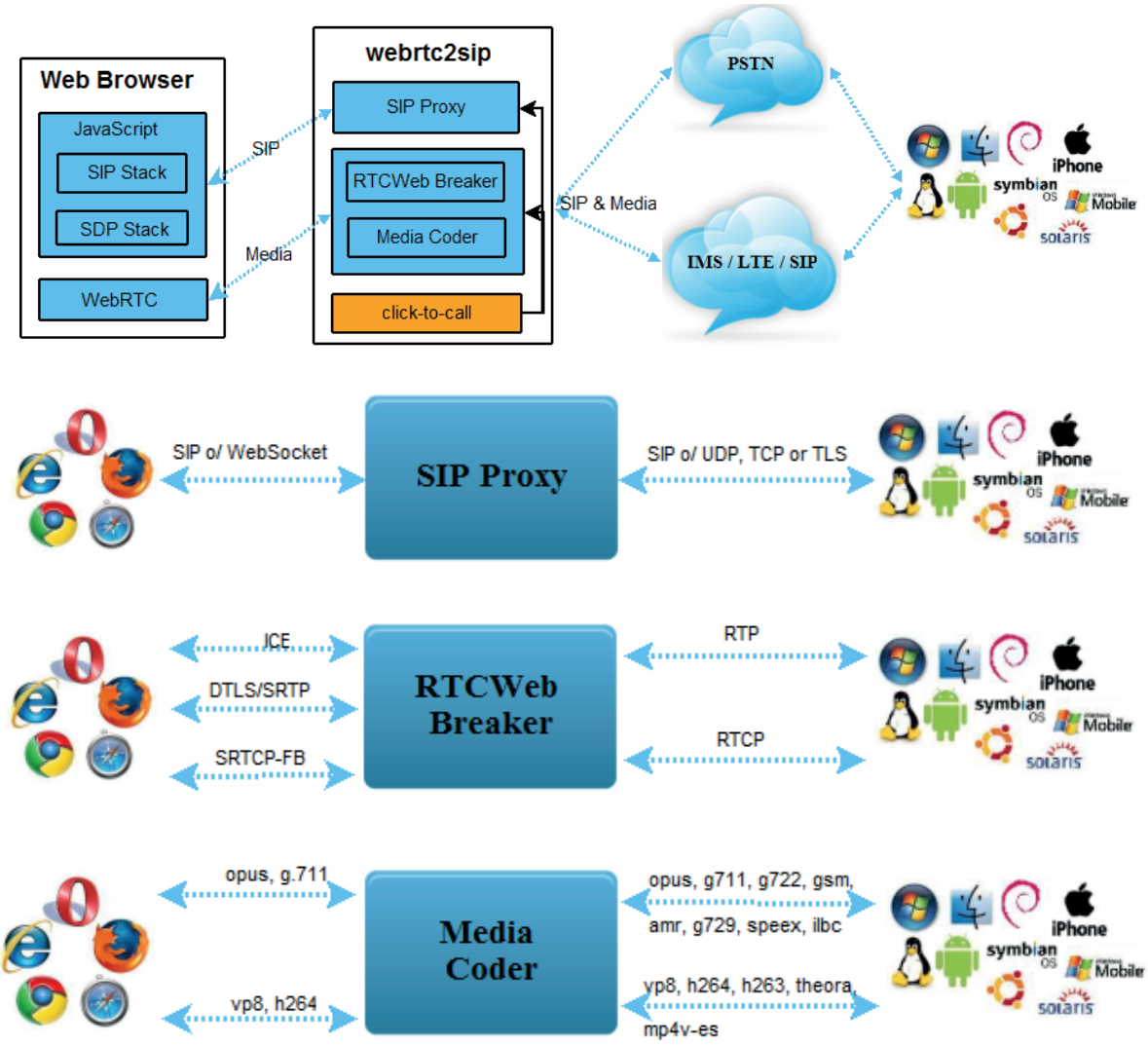

Abbildung 25: Architektur des WebRTC-Gateways webrtc2sip [72]

C Installation von webrtc2sip auf Raspbian

Dieser Abschnitt beschreibt die Installation von webrtc2sip auf dem Raspberry Pi mit dem Betriebssystem Raspbian. Die Anleitung basiert grundlegend auf [71] und [70] und wurde für Raspbian angepasst.

Die Angabe der zu verwendenden Threads beim `make`-Prozess bezieht sich auf den Raspberry Pi 2. Beim Kompilieren auf anderen Systemen muss dieser Wert angepasst werden!

C.1 Installation der Grundpakete

```
apt-get update && apt-get upgrade -y
apt-get install -y build-essential libtool automake pkg-config
   subversion git-core libsrtp0-dev libssl-dev libspeexdsp-dev yasm
   libvpx-dev libgsm1-dev libxml2-dev libx264-dev libopus-dev
```

C.2 Deinstallation von libav

Debian (und Raspbian) wird standardmäßig mit `libav` ausgeliefert. Dabei handelt es sich um einen Fork von `ffmpeg`. Das Doubango IMS Framework, das Grundlage von webrtc2sip ist, benötigt allerdings die originalen `ffmpeg`-Bibliotheken.

```
apt-get remove libavutil51 libavcodec54 libavformat54 gstreamer1.0-libav
apt-get autoremove
```

C.3 Installation von openh264

Der Schwerpunkt dieser Arbeit liegt auf der Unterstützung von Audio-Calls. Das Paket `openh264` wird also theoretisch nicht benötigt, wird der Vollständigkeit halber aber mit installiert.

```
cd /usr/local/src
git clone https://github.com/cisco/openh264.git
cd openh264
git checkout v1.1
make -j4
make install
```

C.4 Installation von ffmpeg

```
cd /usr/local/src
wget https://ffmpeg.org/releases/ffmpeg-1.2.9.tar.gz
tar xzvf ffmpeg-1.2.9.tar.gz
cd ffmpeg-1.2.9
./configure --extra-cflags="-fPIC" --extra-ldflags="-lpthread" --enable-
    pic --enable-memalign-hack --enable-shared --disable-static --disable
    -network --disable-protocols --disable-pthreads --disable-devices --
    disable-filters --disable-bsfs --disable-muxers --disable-demuxers --
    disable-parsers --disable-hwaccels --disable-ffmpeg --disable-ffplay
    --disable-ffserver --disable-encoders --disable-decoders --disable-
    zlib --enable-gpl --disable-debug --enable-encoder=h263 --enable-
    encoder=h263p --enable-decoder=h263 --enable-encoder=mpeg4 --enable-
    decoder=mpeg4 --enable-libx264 --enable-encoder=libx264 --enable-
    decoder=h264

make -j4
make install
```

C.5 Installation des Doubango IMS Frameworks

```
cd /usr/local/src/
svn co http://doubango.googlecode.com/svn/branches/2.0/doubango doubango
cd doubango
```

Vor der Installation muss folgender Patch angewendet werden, um einen Fehler beim Aufbau der Websocket-Verbindungen zu beheben [87]. Das Patchfile befindet sich auch auf der Begleit-DVD (Anhang E).

```
--- tsip_transport_layer.c      2015-06-23 21:39:25.262921979 +0000
+++ tsip_transport_layer.c.new  2015-06-23 21:38:38.043572556 +0000
@@ -371,8 +371,16 @@
case event_accepted:
case event_connected:
{
-                               TSK_DEBUG_INFO("WebSocket Peer accepted/
    connected with fd = %d", e->local_fd);
-                               return tsip_transport_add_stream_peer(
    transport, e->local_fd, transport->type, tsk_true);
+
+                               tsip_transport_stream_peer_t* peer;
+                               // find peer
+                               if((peer =
    tsip_transport_find_stream_peer_by_local_fd(transport, e->local_fd)))
    {
+                                    // If peer already exists.. do
    nothing :0
+                                        return  0;
+                                } else {
+                                        TSK_DEBUG_INFO("WebSocket Peer
    accepted/connected with fd = %d", e->local_fd);
```

```
+                                                   return
    tsip_transport_add_stream_peer(transport, e->local_fd, transport->
    type, tsk_true);
+                                   }
}
default:{
return 0;
```

./media/code/patch.txt

Anwendung des Patchfiles auf die Datei `tsip_transport_layer.c`:

```
patch tinySIP/src/transports/tsip_transport_layer.c < patch.txt
```

Anschließend kann die Installation fortgesetzt werden:

```
sed -i '1,/==/s/==/=/' autogen.sh
./autogen.sh
./configure --with-ssl --with-srtp --with-vpx --with-speex --with-
    speexdsp --enable-speexresampler --enable-speexjb --enable-
    speexdenoiser --with-gsm --with-ffmpeg --with-openh264 --with-opus --
    prefix=/usr/local

make -j4
make install
ldconfig
```

C.6 Installation von webrtc2sip

```
cd /usr/local/src/
svn co http://webrtc2sip.googlecode.com/svn/trunk/ webrtc2sip
cd webrtc2sip
```

Zur Unterstützung von Originating Identification Restriction (OIR) sollte die Datei `mp_wrap.cc` ergänzt werden, wie in Unterabschnitt 4.5 angegeben.

```
sed -i '1,/==/s/==/=/' autogen.sh
./autogen.sh
./configure --prefix=/usr/local
sed -i 's/LDFLAGS =/LDFLAGS = -ldl/' Makefile
make -j4
make install
```

]

C.7 Erstellen der Konfigurationsdatei

webrtc2sip verfügt über eine Konfigurationsdatei im XML-Format. Eine Beispielkonfiguration befindet sich im Verzeichnis `/usr/local/src/webrtc2sip` und ist in angepasster Form im Listing 3 aufgeführt. Sie sollte an einen geeigneten Ort kopiert werden, z.B. `/usr/local/etc/webrtc2sip/`. Die wichtigsten Parameter der Konfiguration sind im Unterunterabschnitt 4.3.2 beschrieben. Es existiert außerdem eine ausführliche Dokumentation vom Hersteller [72].

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
  <debug-level>INFO</debug-level>

  <transport>udp;*;10060</transport>
  <transport>ws;*;10060</transport>
  <transport>wss;*;10062</transport>

  <enable-rtp-symetric>yes</enable-rtp-symetric>
  <enable-100rel>no</enable-100rel>
  <enable-media-coder>yes</enable-media-coder>
  <enable-videojb>yes</enable-videojb>
  <video-size-pref>vga</video-size-pref>
  <rtp-buffsize>65535</rtp-buffsize>
  <avpf-tail-length>100;400</avpf-tail-length>
  <srtp-mode>optional</srtp-mode>
  <srtp-type>dtls</srtp-type>
  <dtmf-type>rfc4733</dtmf-type>

  <codecs>opus;pcma</codecs>
  <codec-opus-maxrates>48000;48000</codec-opus-maxrates>

  <stun-server></stun-server>
  <enable-icestun>no</enable-icestun>

  <max-fds>-1</max-fds>

  <nameserver>8.8.8.8</nameserver>

  <ssl-certificates>
    /usr/local/etc/webrtc2sip/keys/server.key;
    /usr/local/etc/webrtc2sip/keys/server.crt;
    /usr/local/etc/webrtc2sip/keys/rootCA.pem;
    no
  </ssl-certificates>
</config>
```

Listing 3: Konfigurationsdatei `config.xml` für webrtc2sip

Für die Verschlüsselung des SRTP-Traffics müssen TLS-Zertifikate eingebunden werden. Sie können mithilfe von `openssl` erstellt werden, wie in Listing 4 dokumentiert ist.

```
# CA-Key generieren
openssl genrsa -des3 -out ca.key 8192

# CA-Zertifikat erstellen
```

```
openssl req -new -x509 -extensions v3_ca -key ca.key -out rootCA.pem -
    days 3650

# Private Key fuer Anwendung erstellen
openssl genrsa -out server.key 2048

# Certificate Signing Request (CSR) fuer Zertifikat erstellen
openssl req -new -key server.key -sha256 -out server.csr

# Zertifikat mit CA signieren
openssl x509 -req -in server.csr -CA rootCA.pem -CAkey ca.key -
    CAcreateserial -out server.crt -days 365 -sha512
```

Listing 4: Befehle zum Erstellen von TLS-Zertifikaten mit `openssl`

C.8 Starten von webrtc2sip

Um den Gateway zu starten, muss das Binary `webrtc2sip` ausgeführt werden. Über den Parameter `--config` wird der Pfad zur Konfigurationsdatei angegeben.

```
/usr/local/sbin/webrtc2sip --config=/usr/local/etc/webrtc2sip/config.xml
```

D Erstellung eines Docker-Images

Dieser Abschnitt beschreibt die nötigen Schritte zur Erstellung eines Docker-Images für webrtc2sip und die dazugehörige Webapplikation. Die erstellten Images und benötigten Dokumente befinden sich auf der Begleit-DVD (Anhang E).

D.1 Herunterladen des Basis-Images

Ein Image basiert meist auf einem Grund-Image, das aus dem DockerHub bezogen wird. Als Grundlage für webrtc2sip dient hier Raspbian. Docker lädt benötigte Images automatisch herunter, sie können aber auch manuell auf den Host geladen werden:

```
1 docker pull resin/rpi-raspbian
```

Vorhandene Images können mit folgendem Befehl angezeigt werden:

```
1 docker images
```

D.2 Konzepte zur Erstellung von neuen Images

Es existieren zwei Wege, ein neues Image aus einem vorhandenen Image zu erstellen:

1. Erstellen eines interaktiven Containers aus einem Image und anschließendes Ableiten eines neuen Images aus diesem Container
2. Zusammenfassen der Build-Schritte in einem `Dockerfile` und automatisches Erstellen eines neuen Images auf Basis dieser Anweisungen

Im vorliegenden Fall soll die Gateway-Software webrtc2sip und ihre Abhängigkeiten direkt in einem interaktiven Container kompiliert werden. Die Installation der Webapplikation und die finalen Schritte zum Erstellen eines zum Deployment geeigneten Containers sollen mit einem automatischen Build auf Basis eines `Dockerfile` durchgeführt werden.

D.3 Erstellen eines Containers aus dem Basis-Image

Zunächst wird aus dem Basis-Image ein interaktiver Container erstellt, in dem eine Bash-Shell ausgeführt wird:

```
1 docker run -it resin/rpi-raspbian /bin/bash
```

Der Container startet im Vordergrund und es können Befehle über die Shell innerhalb des Containers ausgeführt werden. Mit dem Befehl `exit` wird der Container verlassen

und heruntergefahren. Soll der Container im Hintergrund weiter ausgeführt werden, kann man mit der Tastenkombination `Ctrl + P, Ctrl + Q` vom Container „detachen“.

Eine Übersicht über die vorhandenen Container und ihren Status kann mit folgendem Befehl abgerufen werden:

```
1 docker ps -a
```

Jeder Container wird durch eine ID eindeutig identifiziert, die in der ersten Spalte der Containerübersicht zu finden ist. Die ID ist auch gleichzeitig der Default-Hostname des Containers.

Auf einen laufenden Container kann mit folgendem Befehl wieder „attached“ werden, wobei `<CONTAINERID>` durch die ID des Containers zu ersetzen ist:

```
1 docker attach <CONTAINERID>
```

D.4 Installation von webrtc2sip im Container

In dem neu angelegten Container kann nun das WebRTC-Gateway webrtc2sip installiert werden. Der Ablauf befindet sich im Anhang C.

Zusätzlich wird NodeJS installiert, das als Webserver für die Webapplikation dient: [88]

```
1 wget http://node-arm.herokuapp.com/node_latest_armhf.deb
dpkg -i node_latest_armhf.deb
```

Die überflüssigen Dateien sollten nach dem Kompilieren wieder entfernt werden, um die Containergröße zu verringern.

D.5 Image aus den Container erstellen

Aus dem modifizierten Container wird ein neues Image abgeleitet. Die Commit-Nachricht ist frei wählbar und dient nur der Dokumentation der Änderungen. Das Ziel-Image wird als letztes Argument des Befehls angegeben und ist nach dem Schema `Benutzer/Paketname` aufgebaut [89]. `<CONTAINERID>` ist durch die ID des im vorherigen Schritt modifizierten Containers zu ersetzen.

```
docker commit -m "Installed webrtc2sip and NodeJS" -a "Ferdinand Malcher
    " <CONTAINERID> fmalcher/webrtc2sip
```

Anschließend ist das neue Image in der Image-Übersicht aufgeführt und es können Container daraus erstellt werden.

D.6 Image mit Dockerfile automatisiert erstellen

Auf Grundlage des erstellten Images `fmalcher/webrtc2sip` kann nun ein weiteres Image gebaut werden, das zusätzlich die Webapplikation und die Konfigurationsdateien einbindet.

Dazu muss zunächst ein `Dockerfile` erstellt werden, in dem die Vorschriften für den Build-Prozess zusammengefasst sind:

```
FROM fmalcher/webrtc2sip
MAINTAINER Ferdinand Malcher
EXPOSE 10060 80

ENV APPPATH=/root/webrtcApp
COPY start.sh /root/start.sh

RUN git clone https://github.com/fmalcher/webrtcApp $APPPATH && \
    cd $APPPATH && npm install && \
    npm install -g forever && \
    chmod +x /root/start.sh

ENTRYPOINT /root/start.sh
```

Listing 5: Dockerfile zum automatisierten Build eines Images

Die wichtigsten Befehle werden im Folgenden erläutert [90]:

- `FROM`: zu verwendendes Basis-Image
- `EXPOSE`: Angabe über die Ports, die standardmäßig außerhalb des Containers verfügbar sein sollen, hier Port 10060 für den Gateway und Port 80 für die Webapplikation
- `ENV`: Anlegen von Umgebungsvariablen im Container, hier der Pfad der Webapplikation
- `RUN`: Befehl, der beim Build des Images einmalig ausgeführt wird, hier Klonen des git-Repositories für die Webapplikation und Installation der Abhängigkeiten
- `COPY`: Kopieren einer Datei vom Host in das Image
- `ENTRYPOINT`: Befehl, der beim Erstellen des Containers ausgeführt wird und damit der Einstiegspunkt der virtualisierten Anwendung ist

Das Skript `start.sh` wird beim Erstellen eines Containers von diesem Image (mit `docker run`) ausgeführt. Hier werden die Konfigurationsdateien innerhalb des Containers vorbereitet und die Komponenten gestartet:

```
#!/bin/bash

# create config files from examples
cp $APPPATH/config.example.js $APPPATH/config.js
cp $APPPATH/frontend/webrtcApp/config.example.js $APPPATH/frontend/
   webrtcApp/config.js

# replace variables
sed -i s/{W2SPORT}/"$W2SPORT"/g $APPPATH/frontend/webrtcApp/config.js
```

```
9  sed -i s/{W2SIP}/"$W2SIP"/g $APPPATH/frontend/webrtcApp/config.js

11 # start nodejs server
   forever start --minUptime=1000 --spinSleepTime=2000 $APPPATH/server.js
13
   # start webrtc2sip
15 /usr/local/sbin/webrtc2sip --config=/w2sconf/config.xml
```

Listing 6: `start.sh` als Einstiegspunkt für Container aus diesem Docker-Image

Auf Grundlage des erstellten `Dockerfile` kann nun ein neues Image automatisiert gebaut werden. Das vorletzte Argument bezeichnet das Ziel des neuen Images. Als letztes Argument wird der Pfad des zugrunde liegenden `Dockerfile` und dessen Abhängigkeiten (Startskript `start.sh`) angegeben. Der Befehl muss also im vorliegenden Fall im selben Verzeichnis ausgeführt werden, in dem sich `Dockerfile` und `start.sh` befinden.

```
1 docker build --no-cache -t fmalcher/webrtc2sipbuild .
```

D.7 Erstellen eines lauffähigen Containers aus dem neuen Image

Aus dem neu erstellten Image kann nun ein lauffähiger Container erstellt werden, in den das Gateway und die Webapplikation integriert sind.

Die Konfigurationsdateien für webrtc2sip sollen in einem Ordner auf dem Hostsystem hinterlegt und als sogenanntes Volume in den Container eingebunden werden.

```
1 docker run -it -e W2SIP=192.168.178.41 -e W2SPORT=10060 -v /root/
     webrtc2sip/:/w2sconf/ -p 10060:10060 -p 80:80 fmalcher/
     webrtc2sipbuild
```

Die Parameter sind im Folgenden erläutert:

- `-e W2SIP=192.168.178.41 -e W2SPORT=10060`
  Definition von Umgebungsvariablen innerhalb des Containers. Sie dienen im Startskript zum Erstellen der Konfigurationsdateien.
- `-v /root/webrtc2sip/:/w2sconf/`
  Einbinden eines Ordners auf dem Hostsystem (Quelle: `/root/webrtc2sip/`) im Container (Ziel: `/w2sconf`)
- `-p 10060:10060 -p 80:80`
  Konfiguration der freizugebenden Ports nach dem Schema `<HOSTPORT>:<CONTAINERPORT>`

E Begleit-DVD

Dieser Arbeit liegt eine DVD bei, auf der folgende Inhalte zu finden sind:

- `bachelorarbeit_webrtc_malcher.pdf`: Diese Arbeit als PDF
- `build/`: `Dockerfile` und Startskript `start.sh`
- `config/`: Konfiguration für webrtc2sip
- `images/`: Docker-Images mit webrtc2sip für Raspbian und Debian
- `patch/`: Patchfile für `tsip_transport_layer.c` und angepasste `mp_wrap.cc` (siehe Unterabschnitt 4.5)
- `traces/`: Wireshark-Traces für Signalisierungsabläufe aus Unterabschnitt 5.1

Quellenverzeichnis

[1] Skype. Skype. https://www.skype.com/de/. [retrieved 21.08.2015].

[2] WhatsApp Inc. WhatsApp. https://www.whatsapp.com. [retrieved 21.08.2015].

[3] A. Bergkvist, D. Burnett, C. Jennings, and A. Narayanan. WebRTC 1.0: Real-time Communication Between Browsers. W3C Working Draft, W3C, February 2015. http://www.w3.org/TR/webrtc/ [retrieved: 10.03.2015].

[4] H. Halvestrand. Overview: Real Time Protocols for Browser-based Applications. IETF Internet Draft <draft-ietf-rtcweb-overview-14>, Internet Engineering Task Force (IETF), June 2015. https://tools.ietf.org/html/draft-ietf-rtcweb-overview-14 [retrieved: 08.08.2015].

[5] caniuse.com. WebRTC Peer-to-peer connections. http://caniuse.com/#feat=rtcpeerconnection. [retrieved 12.08.2015].

[6] R. Raymond, IB. Castillo, JL. Millan, C. Dorn, R. Shpount, and E. Lagerway. Object RTC (ORTC) API for WebRTC. Draft Community Group Report, ORTC Community Group, April 2014. http://ortc.org/wp-content/uploads/2014/04/ortc.html [retrieved: 12.08.2015].

[7] IEBlog. Bringing Interoperable Real-Time Communications to the Web. http://blogs.msdn.com/b/ie/archive/2014/10/27/bringing-interoperable-real-time-communications-to-the-web.aspx, October 2014. [retrieved 12.08.2015].

[8] M. Thomson, B. Aboda, and M. Kaufman. Customizable, Ubiquitous Real-Time Communication over the Web: Real-Time Media and Peer-to-Peer Transport API. Unofficial Draft, Microsoft Open Technologies, Inc., July 2013. http://html5labs.interoperabilitybridges.com/cu-rtc-web/cu-rtc-web.htm [retrieved: 01.08.2015].

[9] B. Schwan. Formatkrieg beim Videochat. *Technology Review*, March 2013. http://heise.de/-1802797 [retrieved 05.11.2014].

[10] 3GPP. IP Multimedia Subsystem (IMS); Stage 2. 3GPP TS 23.228 V13.3.0, 3rd Generation Partnership Project (3GPP), June 2015. http://www.3gpp.org/DynaReport/23228.htm [retrieved: 11.08.2015].

[11] A. Johnston and D. Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. 2013.

[12] S. Nandakumar and C. Jennings. SDP for the WebRTC. IETF Internet Draft <draft-nandakumar-rtcweb-sdp-08>, Internet Engineering Task Force (IETF), August 2015. https://tools.ietf.org/html/draft-nandakumar-rtcweb-sdp-08 [retrieved: 07.08.2015].

[13] P. Hancke. webRTC and XMPP. http://hancke.name/jabber/webrtc-xmpp-summit-13ph.pdf, 2013. [retrieved 28.07.2015].

[14] I. Baz Castillo, J. Millan Villegas, and V. Pascual. The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP). RFC 7118 (Proposed Standard), January 2014.

[15] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), March 2011.

[16] XMPP Standards Foundation. XEP-0166: Jingle. Draft, December 2009. https://xmpp.org/extensions/xep-0166.html [retrieved: 21.08.2015].

[17] J. Uberti, C. Jennings, and E. Rescorla. Javascript Session Establishment Protocol. IETF Internet Draft <draft-ietf-rtcweb-jsep-07>, Internet Engineering Task Force (IETF), July 2014. https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-07 [retrieved: 21.08.2015].

[18] ITU-T. Packet-based multimedia communications systems. ITU-T Recommendation H.323, December 2009. https://www.itu.int/rec/T-REC-H.323-200912-I/en [retrieved: 21.08.2015].

[19] D. Burnett, A. Bergkvist, C. Jennings, and A. Narayanan. Media Capture and Streams. W3C Working Draft, W3C, September 2013. http://www.w3.org/TR/mediacapture-streams/ [retrieved: 10.03.2015].

[20] Wikipedia EN. WebRTC. https://en.wikipedia.org/wiki/WebRTC. [retrieved 23.05.2015].

[21] C. Bran, C. Jennings, and J. M. Valin. WebRTC Codec and Media Processing Requirements. IETF Internet Draft <draft-cbran-rtcweb-codec-02>, Internet Engineering Task Force (IETF), February 2015. https://tools.ietf.org/html/draft-cbran-rtcweb-codec-02 [retrieved: 11.03.2015].

[22] H. Schulzrinne and T. Taylor. RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals. RFC 4733 (Proposed Standard), December 2006. Updated by RFCs 4734, 5244.

[23] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), April 2010. Updated by RFC 6336.

[24] ITU-T. Next Generation Networks – Frameworks and functional architecture models. ITU-T Recommendation Y.2001, December 2004. https://www.itu.int/rec/T-REC-Y.2001-200412-I/en [retrieved: 11.08.2015].

[25] ETSI. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Functional Architecture. ETSI ES 282 001 V3.4.1, European Telecommunications Standards Institute (ETSI), September 2009. http://www.etsi.org/deliver/etsi_es/282000_282099/282001/03.04.01_60/es_282001v030401p.pdf [retrieved: 11.08.2015].

[26] ETSI. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Terminology. ETSI TR 180 000 V1.1.1, European Telecommunications Standards Institute (ETSI), February 2006. http://www.etsi.org/deliver/etsi_tr/180000_180099/180000/01.01.01_60/tr_180000v010101p.pdf [retrieved: 11.08.2015].

[27] 3GPP. IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3 (Release 13). 3GPP TS 24.229 V13.2.1, 3rd Generation Partnership Project (3GPP), June 2015. http://www.3gpp.org/dynareport/24229.htm [retrieved: 11.08.2015].

[28] U. Trick and F. Weber. *SIP, TCP/IP und Telekommunikationsnetze: Next Generation Networks und VoIP - konkret*. Oldenbourg Wissenschaftsverlag, July 2009.

[29] M. Maruschke. *Folien zur Vorlesung Netze 2*. Hochschule für Telekommunikation Leipzig (HfTL), March 2013.

[30] AVM. Unsere FRITZ!Box-Produkte - FRITZ!Box. https://avm.de/produkte/fritzbox/. [retrieved 21.08.2015].

[31] Wikipedia DE. Speedport. https://de.wikipedia.org/wiki/Speedport. [retrieved 21.08.2015].

[32] Deutsche Telekom AG. Technical Technical Specification of the SIP (Gm) interface between the User Equipment (UE) and the NGN platform of Deutsche Telekom. 1TR114 v3.00, June 2013. https://hilfe.telekom.de/dlp/eki/downloads/1/1TR114.zip [retrieved: 03.07.2015].

[33] U. Mansmann. Telekom beginnt mit Umstellung herkömmlicher Telefonanschlüsse auf VoIP. *heise online*, 02 2013. http://heise.de/-1807580 [retrieved 12.08.2015].

[34] Deutsche Telekom AG. Telekom Kunden telefonieren in HD Qualität. 11 2011. https://www.telekom.com/medien/produkte-fuer-privatkunden/30588 [retrieved 20.08.2015].

[35] ITU-T. Specifications of Signalling System No. 7 – ISDN supplementary services. ITU-T Recommendation Q.730, December 1999. https://www.itu.int/rec/T-REC-Q.730/en [retrieved: 06.08.2015].

[36] ETSI. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Multimedia Telephony with PSTN/ISDN simulation services. ETSI TS 181 002 V2.2.5, European Telecommunications Standards Institute (ETSI), November 2007. http://www.etsi.org/deliver/etsi_ts/181000_181099/181002/02.02.05_60/ts_181002v020205p.pdf [retrieved: 17.08.2015].

[37] ITU-T. Specifications of Signalling System No. 7 – Stage 3 Description for number identification supplementary services using signalling system No. 7. ITU-T Recommendation Q.731, March 1993. https://www.itu.int/rec/T-REC-Q.731.3-199303-I/en [retrieved: 17.08.2015].

[38] ETSI. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); PSTN/ISDN simulation services; Originating Identification Presentation (OIP) and Originating Identification Restriction (OIR); Protocol specification. ETSI TS 183 007 V1.4.0, European Telecommunications Standards Institute (ETSI), June 2008. http://www.etsi.org/deliver/etsi_ts/183000_183099/183007/01.04.00_60/ts_183007v010400p.pdf [retrieved: 17.08.2015].

[39] ITU-T. Specifications of Signalling System No. 7 – Stage 3 Description for call offering supplementary services using signalling system No. 7: Call diversion services. ITU-T Recommendation Q.732.2-5, March 1993. https://www.itu.int/rec/T-REC-Q.732.2/en [retrieved: 17.08.2015].

[40] ETSI. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); PSTN/ISDN simulation services: Communication Diversion (CDIV); Protocol specification. ETSI TS 183 004 V1.1.1, European Telecommunications Standards Institute (ETSI), April 2006. http://www.etsi.org/deliver/etsi_ts/183000_183099/183004/01.01.01_60/ts_183004v010101p.pdf [retrieved: 17.08.2015].

[41] ITU-T. Specifications of Signalling System No. 7 – Stage 3 description for call completion supplementary services using Signalling System No. 7. ITU-T Recommendation Q.733, March 1993. https://www.itu.int/rec/T-REC-Q.733.4-199303-I/en [retrieved: 17.08.2015].

[42] ITU-T. Specifications of Signalling System No. 7 – Stage 3 description for multiparty supplementary services using Signalling System No. 7: Three-party service. ITU-T Recommendation Q.734.2, July 1996. https://www.itu.int/rec/T-REC-Q.734.2/en [retrieved: 17.08.2015].

[43] ETSI. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); PSTN/ISDN Simulation Services; Completion of Communications to Busy Subscriber (CCBS), Completion of Communications by No Reply (CCNR); Protocol specification. ETSI TS 183 042 V2.1.1, European Telecommunications Standards Institute (ETSI), January 2009. http://www.etsi.org/deliver/etsi_ts/183000_183099/183007/01.04.00_60/ts_183007v010400p.pdf [retrieved: 17.08.2015].

[44] ETSI. Human Factors (HF); Minimum Man-Machine Interface (MMI) to public network based supplementary services. ETS 300 738, European Telecommunications Standards Institute (ETSI), March 1997. http://www.etsi.org/deliver/etsi_i_ets/300700_300799/300738/01_30_9720/ets_300738e01v.pdf [retrieved: 17.02.2015].

[45] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFCs 1349, 2474.

[46] Wikimedia Commons. Format of IPv4 Header. https://commons.wikimedia.org/wiki/File:IPv4_Header.svg. [retrieved 18.01.2015].

[47] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated

Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260.

[48] D. Black, S. Brim, B. Carpenter, and F. Le Faucheur. Per Hop Behavior Identification Codes. RFC 3140 (Proposed Standard), June 2001.

[49] J. Babiarz, K. Chan, and F. Baker. Configuration Guidelines for DiffServ Service Classes. RFC 4594 (Informational), August 2006. Updated by RFC 5865.

[50] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246 (Proposed Standard), March 2002.

[51] Deutsche Telekom AG. Octopus F50 V1 FAQ-Liste. `https://www.telekom.de/dlp/eki/downloads/Octopus/Octopus%20F_50_FAQ.pdf`, March 2012. [retrieved 09.08.2015].

[52] Meetecho. Janus: the general purpose WebRTC Gateway. `https://janus.conf.meetecho.com/`. [retrieved 14.04.2015].

[53] Doubango Telecom. webrtc2sip: Smart SIP and Media Gateway to connect WebRTC endpoints. `http://webrtc2sip.org/`. [retrieved 14.04.2015].

[54] Doubango Telecom. SIPml5: World's first HTML5 SIP client. `http://sipml5.org/`. [retrieved 03.05.2015].

[55] Versatica. OverSIP - the SIP framework you dreamed about. `http://www.oversip.net/`. [retrieved 04.08.2015].

[56] Versatica. OverSIP Documentation: Overview. `http://www.oversip.net/documentation/2.0.x/overview/`. [retrieved 04.08.2015].

[57] Digium Inc. Asterisk. `http://www.asterisk.org/`. [retrieved 21.08.2015].

[58] FreeSWITCH.org. FreeSWITCH | Communication Consolidation. `https://freeswitch.org/`. [retrieved 21.08.2015].

[59] Kamailio SIP Server Project. Kamailio® – the Open Source SIP Server. `http://www.kamailio.org/w/`. [retrieved 21.08.2015].

[60] Mountain Goat Software. User Stories. `http://www.mountaingoatsoftware.com/agile/user-stories/`. [retrieved 11.03.2015].

[61] freetz. freetz. `http://freetz.org/`. [retrieved 21.08.2015].

[62] OpenWrt. OpenWrt - Wireless Freedom. `https://www.openwrt.org/`. [retrieved 21.08.2015].

[63] Raspberry Pi Foundation. RASPBERRY PI 2 MODEL B. `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`. [retrieved 02.08.2015].

[64] Radxa Limited. Radxa Rock. http://radxa.com/Rock. [retrieved 21.08.2015].

[65] Twilio. SIP to WebRTC. https://www.twilio.com/client/sip-to-webrtc. [retrieved 21.08.2015].

[66] Node.js Foundation. Node.js. https://nodejs.org/. [retrieved 21.08.2015].

[67] Apache Software Foundation. Apache HTTP Server Project. https://httpd.apache.org/. [retrieved 21.08.2015].

[68] Raspbian. Raspbian. https://www.raspbian.org/. [retrieved 07.07.2015].

[69] launchpad. Autostatic Doubango webrtc2sip PPA. https://launchpad.net/~autostatic/+archive/ubuntu/doubango. [retrieved 09.12.2014].

[70] Google Code. webrtc2sip: Building Source v2.0. https://code.google.com/p/webrtc2sip/wiki/Building_Source_v2_0. [retrieved 26.02.2015].

[71] Autostatic. Installing webrtc2sip on Ubuntu 12.04. http://linux.autostatic.com/installing-webrtc2sip-on-ubuntu-1204. [retrieved 04.02.2015].

[72] M. Diop. webrtc2sip - Smart SIP and Media Gateway for WebRTC endpoints. Technical Guide v2.5.0, Doubango Telecom, October 2013. http://webrtc2sip.org/technical-guide-1.0.pdf [retrieved: 13.12.2014].

[73] OpenSSL Software Foundation. OpenSSL - Cryptography and SSL/TLS Toolkit. https://www.openssl.org/. [retrieved 21.08.2015].

[74] Google Inc. AngularJS — Superheroic JavaScript MVW Framework. https://angularjs.org/. [retrieved 21.08.2015].

[75] Twitter Bootstrap. Bootstrap. http://getbootstrap.com/. [retrieved 21.08.2015].

[76] 3GPP. Originating Identification Presentation (OIP) and Originating Identification Restriction (OIR) using IP Multimedia (IM) Core Network (CN) subsystem; Protocol specification . 3GPP TS 24.607 V13.1.0, 3rd Generation Partnership Project (3GPP), June 2015. http://www.3gpp.org/dynareport/24607.htm [retrieved: 03.08.2015].

[77] J. Peterson. A Privacy Mechanism for the Session Initiation Protocol (SIP). RFC 3323 (Proposed Standard), November 2002.

[78] Docker Inc. Docker Docs. https://docs.docker.com/. [retrieved 21.07.2015].

[79] Docker Inc. Docker Docs: Understand the architecture. https://docs.docker.com/introduction/understanding-docker/. [retrieved 05.07.2015].

[80] Docker Inc. Docker Docs: Using the command line. https://docs.docker.com/reference/commandline/cli/. [retrieved 06.07.2015].

[81] Docker Inc. Docker Hub. https://hub.docker.com/. [retrieved 21.08.2015].

[82] G. Roden. Ein Container voller Himbeeren: Docker auf dem Raspberry Pi. *heise Developer*, March 2015. http://heise.de/-2572533 [retrieved 05.07.2015].

[83] hypriot. Docker Pirates ARMed with explosive stuff. http://blog.hypriot.com/. [retrieved 16.06.2015].

[84] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, and K. Summers. Session Initiation Protocol (SIP) Basic Call Flow Examples. RFC 3665 (Best Current Practice), December 2003.

[85] Telekom Innovation Laboratories. WebRTC- Wenn Browser miteinander reden. http://www.laboratories.telekom.com/public/Deutsch/Innovation/Pages/WebRTC.aspx. [retrieved 21.08.2015].

[86] Deutsche Telekom AG. Häufige Fragen und Antworten: Wie lassen sich Leistungsmerkmale am IP-basierten Anschluss mit den Telefontasten steuern? https://hilfe.telekom.de/hsp/cms/content/HSP/de/3378/faq-445419652. [retrieved 22.07.2015].

[87] Google Code. telepresence: Issue 22: ***ERROR: function: tsip_transport_layer_ws_cb(). https://code.google.com/p/telepresence/issues/detail?id=22, December 2013. [retrieved 05.03.2015].

[88] P. Beulque. "Raspberry Pi + NodeJS". *We Work We Play*. http://weworkweplay.com/play/raspberry-pi-nodejs/ [retrieved 07.06.2015].

[89] Docker Inc. Docker Docs: Get started with images. https://docs.docker.com/userguide/dockerimages/. [retrieved 06.07.2015].

[90] Docker Inc. Docker Docs: Dockerfile reference. https://docs.docker.com/reference/builder/. [retrieved 05.07.2015].

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir an der Hochschule für Telekommunikation Leipzig eingereichte Bachelorarbeit zum Thema

„**Integration von WebRTC in einen All-IP-Telefonanschluss der Deutschen Telekom**“

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Leipzig, den 28. August 2015

Ferdinand Malcher