UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Anbindung von OpenWrt Routern an VNUML-virtualisierte Netzwerke

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informatik

vorgelegt von

Ansgar Taflinski

Erstgutachter:	Prof. Dr. Christoph Steigner
	Institut für Informatik
Zweitgutachter:	Dipl. Inf. Frank Bohdanowicz
	Institut für Informatik

Koblenz, im Januar 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Seit der Erfindung von HTML von Tim Berners-Lee im Jahr 1989 [BL89] ist das Internet immer weiter in den Alltag insbesondere in der westlichen Welt vorgedrungen. Rechnernetze sind eine tragende Säule der Kommunikationsinfrastruktur unserer Gesellschaft geworden. Um den dementsprechend hohen Ansprüchen an die Verfügbarkeit der Zugänge sowie die Zuverlässigkeit der Datenübertragung nachzukommen ist Redundanz im Leitungsnetz erforderlich. Damit die Redundanz im Leitungsnetz eine Sicherheit darstellt und nicht zum Problem wird, ist der Einsatz von Routingalgorithmen erforderlich, die Schleifen im Netzwerk verhindern. Routingalgorithmen haben dabei die Aufgabe der Wegfindung von einem Ausgangsknoten zu einem Zielknoten über eine Menge von dazwischenliegenden Knoten. Um im Fall des Aufalls eines Knotens die Erreichbarkeit eines Zielknotens gewährleisten zu können, muss ein Routingalgorithmus in der Lage sein, den Ausfall zu erkennen und mit dem Ziel weiterzukommunizieren, das Netzwerk zu reorganisieren. Die Problemstellung ist als Anwendungsfall des *Pathfinding* in Graphen zu verstehen, deshalb liegt es nahe, die die auf dieser abstrakten Betrachtungsebene bekannten Algorithmen für die Organisation von Netzwerken zu verwenden. Die Routing-Algorithmen BGP und RIP sind zum Beispiel Implementierungen des Bellman-Ford-Algorithmus, OSPF und IS-IS basieren auf dem Dijkstra-Algorithmus. Jeder Router lernt durch die Nachrichten die er von den anderen Routern erhält die Topologie des gesamten von Routern kontrollierten Netzwerks und überprüft in fesgelegten Abständen die Erreichbarkeit der anderen Router, um im Fall des Ausfalls eines Netzes einen alternativen Weg zu Zieladressen, die hinter einem ausgefallenen Netz liegen zu finden. Beim

RIP Algorithmus kann es in Netzwerkschleifen zu sogenannten *Counting To Infinity* (CTI) Situationen kommen, wenn ein außerhalb der Schleife liegendes Netz ausfällt, innerhalb der Schleife aber die Erreichbarkeit des Netzes weiter propagiert wird. Bei jedem Intervall wird dabei die Entfernung des Netzes inkrementiert, bis ein Maximalwert, der als unendlich interpretiert wird, erreicht ist. Ein Ansatz zur Lösung dieses Problems ist die Weiterentwicklung RMTI (**R**outing with **M**etric based **T**opology **I**nvestigation), die an der Universität Koblenz im Rahmen mehrerer Diplom- und Studienarbeiten entwickelt und evaluiert wurde (vgl. [Boh08], [Kob09], [Gar10]) und sich im Test-Stadium befindet. Mit der Testumgebung für diesen Algorithmus befasst sich auch diese Bachelorthesis.

1.1 Motivation

Der RMTI Algorithmus ist eine Erweiterung des verbreiteten RIP Algorithmus, die an der Universität Koblenz entwickelt und auf der Basis von der Software Routing Suite Quagga implementiert wurde. 2010 wurde der Algorithmus erstmals veröffentlicht, befindet sich aber weiterhin im Teststadium. Den bisher zu RMTI verfassten Arbeiten liegen Tests zu Grunde, die entweder ausschließlich auf mit VNUML, einer Softwareumgebung zum Aufbau von Netzwerken aus virtuellen Maschinen, virtualisierten Netzwerken bzw. auf einem sehr kleinen Netzwerk aus handelsüblichen Routern mit OpenWrt, einer Linux-Distribution, die speziell für Router entwickelt wurde, durchgeführt wurden. Die VNUML-Umgebung bietet durch die Unterstüzung von *ediv* die Möglichkeit, sehr große Netzwerke zu virtualisieren, jedoch hat sie den Nachteil, das beim Testen keine Realbedingungen auftreten, d.h. Latenzen der Übertragungen zwischen einzelnen Knoten sind praktisch nicht vorhanden. Das Einbeziehen von Handelsüblichen Routern in die Testumgebung ist daher ein weiterer Schritt um die Alltagstauglichkeit des RMTI Algorithmus unter Beweis zu stellen. Thema dieser Arbeit ist deshalb die Integration Handelsüblicher, mit OpenWrt betriebener Router in die Testumgebung, um die Vorteile virtualisierter Netzwerke sowie die Eigenschaften realer Netzwerke zu vereinen.

1.2 Verfügbarkeit der Software

Die im Rahmen dieser Arbeit erstelle Software befindet sich auf dem Git-Server der Universität Koblenz und ist öffentlich zugänglich. Der Quelltext für die Erweiterung der grafischen Oberfläche von OpenWrt kann durch folgenden Befehl aus dem Repository geladen werden:

```
1 git clone git://git.uni-koblenz.de/ansgarsbt/luci-ripd.git
```

Das Programm *vnuml2uci*, das Netzwerkkonfigurationen für mit OpenWrt betriebene Linksys WRT54GL Router erstellt, steht ebenfalls auf dem Git-Server der Universität Koblenz zum Download bereit:

```
1 git clone git://git.uni-koblenz.de/ansgarsbt/vnuml2uci.git
```

Kapitel 2

OpenWrt

OpenWrt ist eine Linux-Distribution, die speziell für Embedded Geräte ausgelegt ist. Hervorgegangen ist OpenWrt aus dem von der Firma Linksys veröffentlichten Quellcode der Linuxbasierten Firmware der WRT54G Router. Trotz der Ausrichtung auf leistungsschwache Kleingeräte bietet OpenWrt dem Entwickler wie dem Anwender die von anderen Linuxumgebungen bekannten Werkzeuge. Die für diese Arbeit verwendete OpenWrt-Version ist 10.03 „Backfire“. Backfire basiert auf dem Linux-Kernel 2.6.32, sowie der für Embedded Geräte optimierten, kleinen C-Bibliothek *uClibc*. Neben dem Grundsystem enthält OpenWrt Backfire standardmäßig auch die in Lua geschriebene Weboberfläche „LuCI“ (**Lu**a **C**onfiguration **I**nterface), mittels der ein mit OpenWrt betriebener Router auch ohne Linux-Kenntnisse bedient werden kann. Der Zugriff auf der Kommandozeilenebene über eine SSH-Verbindung ist ebenso möglich. Diese Arbeit beschäftigt sich mit OpenWrt sowohl aus der Perspektive des Anwenders, des Paket-Bereitstellers und des LuCI-Oberflächen-Entwicklers. Im Folgenden wird am Beispiel der modifizierten Software Routing Suite Quagga und einer Erweiterung der Bedienoberfläche LuCI erläutert, wie Pakete für das in OpenWrt enthaltene Paketmanagement OPKG erstellt werden und am Beispiel dieser LuCI-Erweiterung für die Konfiguration des RIP-Daemons und die Einsicht in dessen Protokolldatei, wie Erweiterungen für die Weboberfläche LuCI erstellt werden können.

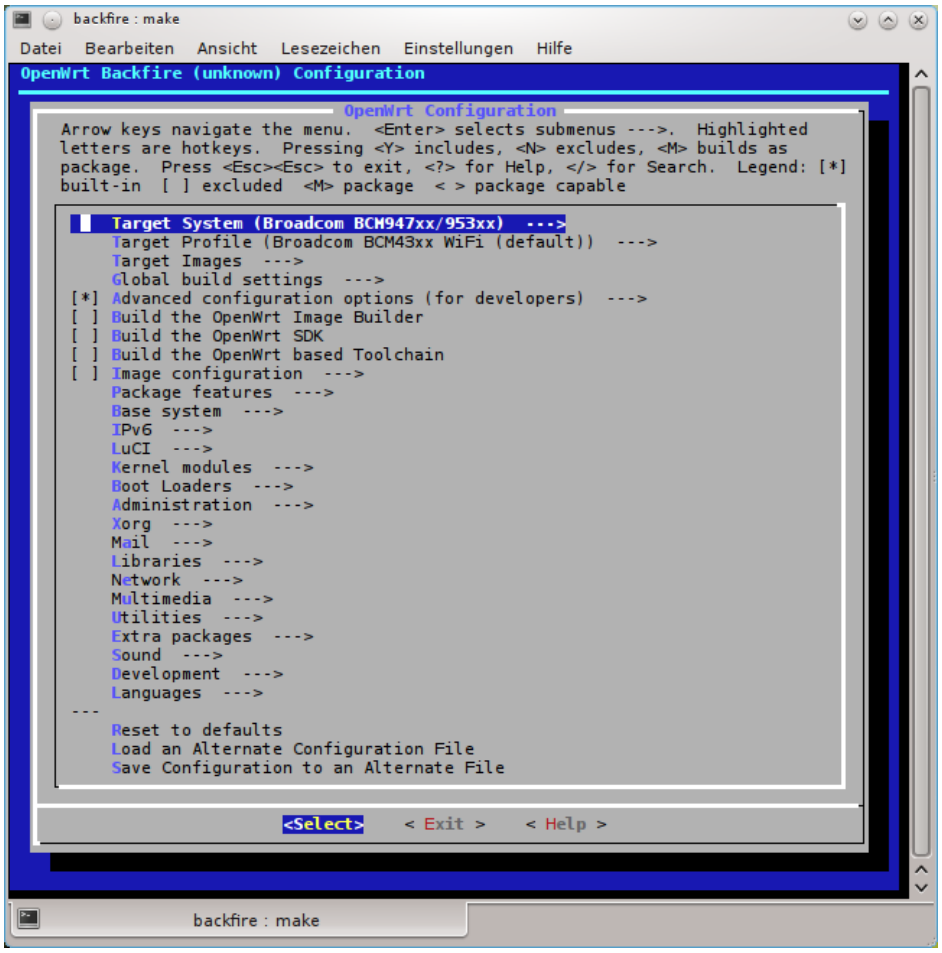

Abbildung 2.1: Hauptmenü der Menuconfig-Oberfläche

2.1 Die OpenWrt Build-Umgebung

Die meisten der Zielplattformen von OpenWrt sind handelsübliche Router, auf denen OpenWrt in Form einer Firmware-Imagedatei, vergleichbar mit einem Firmwareupdate des Herstellers installierbar ist. Da die Zielplattformen meistens über einen sehr begrenzten nicht-flüchtigen Speicher verfügen, dessen Kapazität oft nur 8 Megabyte oder weniger beträgt, müssen die zur Installation benötigten Imagedateien nur mit einer kleinen Auswahl unverzichtbarer Komponenten erstellt werden. Die Konfiguration des Build-Systems erfolgt über die vom Linux Kernel bekannte textbasierte Oberfläche *Menuconfig*, die in Abbildung 2.1 gezeigt wird. Für die wählbaren Pakete müssen jeweils angepasste Makefiles zur Verfü-

gung stehen, die in einem nach dem Paket benannten Unterordner unter dem Ordner „package“ der Build-Umgebung abgelegt sind, optional können in dem Ordner einer Softwarekomponente noch ein Unterverzeichnis „files“ für zusätzliche Dateien, die zum Bauen der Pakete benötigt werden (z.B. Konfigurationsdateien), sowie ein Unterverzeichnis „patches“, das Modifikationen des originalen Quelltextes enthält, die vor der Kompilierung mittels GNU-patch auf den Quelltext angewandt werden. Wahlweise ist auch das Einbinden von sogenannten „feeds“ möglich, die Makefiles für Softwarekomponenten enthalten, die nicht direkt von den OpenWrt Entwicklern unterstützt werden. Das Verwalten von genutzten Feeds geschieht über den Aufruf des Perl-Scripts „feeds“, dass sich im Unterordner „./scripts/“ der Build-Umgebung befindet. Die Feeds für die OpenWrt Build-Umgebung sind Repositories von Versionskontrollsystemen, in denen eine Verzeichnisstruktur vorliegt, die den Konventionen des „package/“ Unterordners der Build-Umgebung entspricht, d.h. für jede Softwarekomponente existiert jeweils ein Unterverzeichnis, dass die Makefile für die Erstellung des Pakets enthält.

2.1.1 Zielplattformen

Bei OpenWrt handelt es sich um eine Linux-Distribution, d.h der überwiegende Teil der Software liegt im Quellcode unter freien Lizenzen vor und kann zum größten Teil für viele vom Linux Kernel und der GNU Compiler Collection unterstützte Plattformen kompiliert werden. Üblicherweise verwendet man für die Kompilierung der OpenWrt Umgebung im Vergleich zur Zeilplattform leistungsstarke Rechnersysteme mit einem großen persistenten sowie nicht-persistenten Speicher und eine für die Zielplattform eingerichtete Crossdev-Umgebung. Eine solche Crossdev-Umgebung stellt alle nötigen Werkzeuge bereit um auf einem System mit einer Prozessorarchitektur A Binärdateien für ein System mit einer Prozessorarchitektur B zu erstellen, A und B seien dabei jeweils beliebige, von der GNU Compiler Collection unterstütze Prozessorarchitekturen.
Die Build-Umgebung bietet 21 Zielsystem-Plattformen an, die meisten davon sind im Embedded-Bereich anzusiedeln, zusätzlich existieren jeweils zu den wählbaren Architekturen wählbare Profile, mittels denen die Build-Umgebung Parameter für ein bestimmtes Gerät mitgeteilt bekommt. Ein Linksys WRT54GL-Router wird beispielsweise beschrieben durch die Wahl der Plattform „Broad-

Plattform	**Architektur**	**Kernel**	
		2.6.32.10	2.4.37.9
AMCC/IBM PowerPC 40x	powerpc	ja	nein
AMCC/IBM PowerPC 44x	powerpc	ja	nein
Atheros AR231x / AR5312	mips	ja	nein
Atheros AR71xx / AR7240 /AR913x	mips	ja	nein
Atmel AVR32	avr32	ja	nein
Broadcom BCM63xx	mips	ja	nein
Broadcom BCM947xx / BCM953xx	mipsel	ja	ja
Cavium Networks Octeon	mips	ja	nein
Cobalt Microservers	mipsel	ja	nein
Infenion Mips	mips	ja	nein
Infenion/ADMtek ADM5120	mips/mipsel	ja	nein
Ingenic XBurst	mipsel	ja	nein
Intel IXP4xx	armeb	ja	nein
Marvell Kirkwood	arm	ja	nein
Marvell Orion	arm	ja	nein
Mikrotik RouterBoard 532	mipsel	ja	nein
RDC 321xn	i386	ja	nein
RMI/AMD AU1x00	mipsel	ja	nein
Texas Instruments AR7	mipsel	ja	nein
User Mode Linux	i386	ja	nein
x86	i386	ja	nein

Abbildung 2.2: Liste aller von OpenWrt 10.03 unterstützen Zielplattformen und ihrer Architekturen

com BCM947xx / BCM953XX“ und des Profils „Broadcom BCM43xx WiFi“. Eine Übersicht aller von OpenWrt 10.03 unterstützten Zielplattformen und deren Prozessorrchitekturen ist in Abbildung 2.2 zusammengestellt. Die angegebenen Prozessorarchitekturen entsprechen dabei jeweils den nach Auswahl der Zielplattform in der Menuconfig-Umgebung in der Konfigurationsdatei unter „CONFIG_ARCH=“ eingetragenen Architekturen.

Unter dem Menüpunkt „Target Images“ werden das Dateiformat und die Art der Ausgabedateien festgelegt. Zur Auswahl stehen die Einbettung des Root-Dateisystems in ein Ramdisk-Image, die Ausgabe des Root-Dateisystems in Form eines Archives, sowie die Generierung einer Festspeicher-Imagedatei, mit der der persistente Speicher des Zielgerätes überschrieben werden kann. Der konventio-

nelle Weg der Installation der OpenWrt-Umgebung auf dem Linksys WRT54GL erfordert Imagedateien im Format „SquashFS“. SquashFS ist ein mit dem Lempel-Ziv-Markow-Algorithmus (LZMA) komprimiertes Dateisystemabbild, das ausschließlich lesende Zugriffe erlaubt.

2.1.2 Ablauf des Build-Vorgangs

Eine OpenWrt-Umgebung wird durch das Build-System aus Quelltext erzeugt, d.h. alle vom Benutzer gewählten Softwarekomponenten werden einzeln kompiliert, wahlweise mit dem Ziel als .ipk-Paket für die Paketverwaltung OPKG bereitgestellt zu werden oder mit OPKG direkt in die Imagedatei installiert zu werden. Standardmäßig erzeugt das Build-System von OpenWrt die Softwarekomponenten, die für den Build-Vorgang benötigt werden, ebenfalls aus Quelltext, anstatt die auf dem Hostrechner installierten Versionen einzusetzen. Dadurch wird eine zusätzliche Abstraktionsschicht erstellt, ein Buildsystem, das die OpenWrt Build-Umgebung von den auf dem Host installierten Versionen der benötigten Werkzeuge und der verwendeten C-Bibliothek unabhängig macht. OpenWrt verwendet standardmäßig die für Embedded-Linux-Systeme Entwickelte, kleine C-Bibliothek uClibc, die Build-Umgebung bietet aber auch die Möglichkeit, eine andere C-Bibliothek zu verwenden, zur Wahl stehen neben uClibc auch die umfangreichen und in den meisten für den Desktop- sowie Serverbereich ausgerichteten Linux-Distributionen verwendeten GNU C-Library (*glibc*) und *EGLIBC*, ein Projekt, dass aus einem Fork der glibc entstanden ist und heute eine zur glibc parallel verlaufende Entwicklung darstellt. Das *EGLIBC*-Projekt beschreibt sich auf seiner Webseite selbst als Erweiterung des *glibc*-Projektes um eine bessere Unterstützung von Embedded-Systemen.

2.1.3 Erstellung von Binärpaketen

Wie bereits erwähnt werden Pakete für OpenWrt durch Makefiles beschrieben. Die Build-Umgebung von OpenWrt stellt eine Menge von Makefiles bereit, die die Basis für die Erzeugung von mit OPKG kompatiblen Paketen bilden. Für die eigentliche Kompilierung der Softwarekomponenten wird im Regelfall eine dem Quelltext beigefügte Makefile ausgeführt, die Steuerung des Kompilierungsvorganges kann jedoch durch die Makefiles der OpenWrt Build-Umgebung

beeinflusst werden. Zum Erstellen eines Paketes ist es erforderlich, der Build-Umgebung neben dem Quelltext der zu kompilierenden Software einige Variablen zu übergeben, die der Build-Umgebung die Anweisungen für die Erzeugung des OPKG-kompatiblen Binärpakets liefern. Der Aufbau der Makefiles in der OpenWrt Build-Umgebung ist hierarchisch und lässt Parallelen zur Vererbung bei der objektorientierten Programmierung erkennen. Die Makefile folgt einer Struktur von Paketen und Verarbeitungsschritten, d.h. die Build-Umgebung unterteilt den Build-Vorgang in mehrere Schritte, die nacheinander abgearbeitet werden, jedoch kann ein Schritt auch, sofern er nicht von vorausgehenden anderen Schritten abhängt, direkt aufgerufen und einzeln ausgeführt werden. Diese Strukturierung des Build-Vorgangs findet sich auch in den Makefiles wieder, wie in Abbildung 2.3 zu sehen, sind sämtliche *define*-Blöcke innerhalb der Makefile mit „Package“ und dem Namen des Pakets („Package/ptpd“) benannt, die „description“ und „install“-Blöcke stellen untergeordnete Aufrufe dar. Der Ablauf des Build-Vorgangs entspricht somit der Traversierung einer Baumstruktur. In einer Makefile der OpenWrt Build-Umgebung werden für die Paketerzeugung erforderliche Einstellungen vorgenommen: Zum Einen werden zur Vorbereitung der Kompilierung Informationen über die Bezugsquellen für den Quelltext, ein hash-Wert für die Verifikation des Quelltextarchives und der Pfad zu dem Verzeichnis, in dem sich der Quelltext nach dem Entpacken befindet. Diese Informationen werden, wie in Abbildung 2.3 erkennbar, den Variablen *PKG_SOURCE_URL* und *PKG_SOURCE*, *PKG_MD5SUM* und *MAKE_PATH* zugewiesen. Die im Beispiel verwendete Kurzform der URL „@SF“ wird von der Build-Umgebung als Kurzform für SourceForge interpretiert. Für die Erstellung einer OPKG-Paketdatei benötigt die Build-Umgebung einen Paketnamen (*PKG_NAME*), Informationen über die Version (*PKG_VERSION* und *PKG_RELEASE*), in der die Software vorliegt, beispielsweise um Update-Funktionen des Paketmanagements zu bedienen. Damit das Finden eines Pakets in einer Paketdatenbank für den Benutzer möglichst einfach zu bewältigen ist, werden alle Pakete mit menschenlesbaren Informationen versehen, die die Software beschreiben: Eine natürlichsprachliche Kurzbeschreibung der Software ist als freier Text in einem *define*-Block, der mit „description“ überschrieben ist, enthalten. Der mit „install“ überschriebene *define*-Block enthält Festlegungen für die Aufnahme bestimmter Dateien in die Paket-Datei nach dem Abschluss der Kompilierung. Die in der 30. Zeile des Beispiels genannte Datei *ptpd.init* befindet sich in dem Unterverzeichnis *files* des Ver-

```
include $(TOPDIR)/rules.mk

PKG_NAME:=ptpd
PKG_VERSION:=1.0.0
PKG_RELEASE:=1

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=@SF/ptpd
PKG_MD5SUM:=b112b2bedc7f6e6e11a838608b9e0357

include $(INCLUDE_DIR)/package.mk

MAKE_PATH:=src

define Package/ptpd
  SECTION:=net
  CATEGORY:=Network
  TITLE:=PTPD
  URL:=http://ptpd.sourceforge.net/
endef

define Package/ptpd/description
 The PTP daemon (PTPd) implements the Precision Time protocol (PTP) as defined by
      the IEEE 1588 standard.
 PTP was developed to provide very precise time coordination of LAN connected
     computers.
endef

define Package/ptpd/install
  $(INSTALL_DIR) $(1)/usr/sbin $(1)/etc/init.d
  $(INSTALL_BIN) $(PKG_BUILD_DIR)/src/ptpd $(1)/usr/sbin/
  $(INSTALL_BIN) ./files/ptpd.init $(1)/etc/init.d/
endef

$(eval $(call BuildPackage,ptpd))
```

Abbildung 2.3: Beispiel einer Makefile für ein OpenWrt-Paket

zeichnisses, das die abgebildete Makefile enthält. Auf diese Weise ist es möglich, nicht aus dem Quelltextarchiv erzeugbare Dateien dem Paket hinzuzufügen, im Fall des Beispiels handelt es sich um ein sog. Init-Skript. Möglich ist auch die Erzeugung mehrerer, einzeln installierbarer Paketdateien aus einer Quelltextdatei, diese Variante wird am Beispiel von LuCI imAbschnitt 2.2.2 erläutert.

2.2 Die Weboberfläche LuCI

Seit der Version 8.09 “Kamikaze” ist die Web-Oberfläche LuCI Teil der vom OpenWrt-Projekt herausgegebenen Firmware-Imagedateien und wird offiziell durch die OpenWrt Entwickler unterstützt. LuCI steht für “**Lu**a **C**onfiguration **I**nterface”.

Seit 2008 wird LuCI im Rahmen des Freifunk-Projekts für OpenWrt entwickelt, um mit den auf der Freifunk-Webseite verteilten OpenWrt Images auch Benutzer ohne Linuxkenntnisse anzusprechen. LuCI ist zum größten Teil in der inter-

Abbildung 2.4: Eröffnungsbildschirm der Weboberfläche LuCI

pretierten Skriptsprache Lua geschrieben, deren Laufzeitumgebung sehr gringe Anforderungen an die Hardware stellt. Lua-Programme werden vor dem Ausführen in einen Bytecode übersetzt, der zur Laufzeit in Maschinencode übersetzt wird. Die Ausgabe in HTML erfolgt über den Webserver *uhttpd*, einen für leistungsschwache Systeme mit wenig Arbeitsspeicher entwickelten Webserver. LuCI präsentiert sich dem Benutzer einfach aufgebaut, der Benutzer benötigt keinerlei Linux-Kenntnisse, um sich in den Menüs zurechtzufinden.

2.2.1 Aufbau von LuCI

LuCI ist eine auf dem Server und dem Client ausgeführte Web-Anwendung, d.h. die serverseitig erzeugte Ausgabe, die an den Client geleitet wird, enthält

JavaScript Code, der lokal auf dem Client ausgeführt wird. Diese Verteilung der Anwendungslogik auf Client und Server ermöglicht das Aufrufen von Funktionen des auf dem Servers ausgeführten Teils der Software und die Integration von vom Server erhaltenen Daten in die angezeigte Seite. Ohne eine Aufteilung der Web-Anwendung in einen Server- und Client-Teil können auf dem Server errechnete Daten nur durch die Ausgabe in eine HTML basierte Webseite, die vom Client geladen wird bereitgestellt werden, der Benutzer erhält Daten also nur durch das erneute Laden der kompletten Seite. In der LuCI-Umgebung beschränkt sich die Auslagerung von Anwendungsteilen auf den Client witgehend auf Steuerelemente, die in den angezeigten Formularen verwendet werden. Durch die Verteilung wird bei LuCI eine hohe Reaktionsgeschwindigkeit der Oberfläche auf Änderungen an voneinander abhängigen Steuerelementen, z.B. kann ein Benutzer in den Netzwerkeinstellungen für ein Netz auswählen ob der Router seine Einstellungen dynamisch über einen DHCP-Server bezieht oder oder ob diese statisch festgelegt sein sollen; entscheidet sich der Benutzer für die statische Festlegung der Einstellungen, werden ohne Rückfrage an den Router weitere Formular-Komponenten angezeigt, in denen Angaben wie die IP-Adresse, der Standard-Gateway und die angefragten DNS-Server eingetragen werden können. Bei der Entwicklung von Erweiterungen ist dies jedoch nicht direkt zu beachten, da die Ausgabe an den Client automatisch generiert wird. LuCI ist nach dem *Model View Controller*-Pattern geschreiben, das eine klare Trennung zwischen Anwendungslogik, Datenbestand und I/O ermöglicht, die in in LuCI nicht strikt eingehalten wird. Die Einordnung der Komponenten als Model, View und Controller wird von LuCI jedoch verlangt und geschieht über die entsprechende Platzierung der Quelltextdateien im Dateisystem. In der LuCI-Umgebung werden die Begriffe Model, View und Controller wie folgt verwendet:

Controller Als Controller werden in LuCI die Module eingeordnet, die die Integration der Konfigurationsseiten in die Menüstruktur, die intern als Baumstruktur des LuCI Dispatchers behandelt wird, beschreiben und Aktionen für den Aufruf durch das Menü vorgeben. Ein Modul kann dabei mehrere Menüeinträge bzw. Knoten im Dispatcher-Baum beschreiben.

Model Als Modell wird im Kontext von LuCI ein CBI-Modell oder ein CBI-Formular bezeichnet. Ein CBI-Modell ist eine in Form von Objektinstanziierun-

gen dargestellte Beschreibung der Struktur einer (UCI-)Konfigurationsdatei und der Darstellung des Formulars im Browser [wikd]. Die Darstellung als HTML- und JavaScript-Formular auf dem Client wird von LuCI generiert und stellt eigentlich die View-Komponente im Model-View-Controller-Pattern dar. Das Darstellen nicht UCI-konformer Konfigurationen ist möglich, wird aber durch die LuCI-Umgebung nicht unterstützt und ist zum jetzigen Zeitpunkt auch nicht in verfügbaren Quellen Dokumentiert. Für die Konfiguration des RIP-Daemons wurde die Darstellung in einem Freitextfeld gewählt.

View HTML-Templates werden in der Verzeichnisstruktur des LuCI Quelltextes als View bezeichnet, die zur Darstellung von durch den Controller erzeugtem Inhalt genutzt werden.

2.2.2 Die LuCI-Entwicklungsumgebung

Das LuCI SDK (Software Development Kit) besteht aus dem LuCI-Quelltext, dem Makefiles beiliegen, die beim Testen der Software hilfreiche Funktionen bereitstellen. Der Quelltext ist durch seine Verzeichnisstruktur logisch unterteilt. Die Bedeutung der für die Entwicklung von Erweiterungen genutzten Unterverzeichnissen des Stammverzeichnisses des LuCI SDK ist in der Tabelle 2.5 erläutert. Das Verzeichnis *applictions* ist der Ort, an dem Erweiterungen der LuCI-Umgebung abgelegt werden. Das LuCi-Quelltextarchiv enthält bereits viele dieser im Späteren Betrieb optional installierbaren Erweiterungen. Für die Entwicklung eigener Erweiterungen existiert in *applications* eine Vorlage im Untervezeichnis *myapplication*, das eine Makefile enthält, die zum Einbinden neuer Erweiterungen in das LuCI Framework genutzt werden kann. Diese Makefile enthält zunächst keine lokal vorzunehmenden Einstellungen für das jeweilige Projekt, sondern verweist auf den Inhalt zweier global genutzter Dateien des *build*-Verzeichnisses, die einen Rahmen für die Behandlung einzelner LuCI-Komponenten durch Make festlegen. Weiterhin befindet sich in dem Verzeichnis *myapplication* ein leerer Unterordner mit dem Namen *luasrc*, der als Heimatverzeichnis für die in Lua geschriebenen Quelltexte und ihrer HTML-Templates der weiteren Arbeit betrachtet werden kann. Es ist zwar auch möglich, in C geschriebenen Quelltext einzubinden, der, sofern er in einem Ordner mit dem Namen *src* abgelegt ist auch durch die LuCI-eigenen Makefiles einem Compiler zugeführt werden kann, jedoch existiert keine

Unterverzeichnis	Bedeutung
applications	In diesem Verzeichnis befindet sich der Quelltext der Erweiterungen, die nach dem Build-Vorgang als separate Pakete zur Verfügung gestellt werden können.
host	Hier wird beim Aufruf der Makefile die Laufzeitumgebung erstellt. Es ist ratsam, dieses Verzeichnis regelmäßig zu löschen, da es durch „make clean“ unberührt bleibt und Ursache für das Wiederauftreten bereits behobener Fehler sein kann.
modules	Dieses Verzeichnis enthält die standardmäßig in LuCI enthaltenen GUI-Komponenten. Der Quelltext der hier zu findenden Dateien ist nach dem gleichen Schema strukturiert, wie der Quelltext im Verzeichnis *applications* und ist daher zur Einarbeitung in die LuCI-Entwicklung nutzbar.
po	In diesem Verzeichnis befinden sich die von den Internationalisierungsfunktionen der Bibliothek *luci.i18n* genutzten Sprachdateien. Die diesem Verzeichnis unterstehende Ordnerstruktur unterteilt die Lokalisierungsdateien nach Sprachen.

Abbildung 2.5: Die Verzeichnisstruktur des LuCI SDK

Dokumentation über das Vorhandensein von in diesem Kontext nutzbaren Programmierschnittstellen und das Zusammenspiel der in Lua und in C geschriebenen Komponenten, so dass diese Möglichkeit im Folgenden außer Acht gelassen wird.

Bei der Arbeit mit dem LuCI-Framework ist es wichtig, dass Namespaces in der Hierarchie der Lua-Packages und Module in Analogie zu der verwendeten Verzeichnisstruktur stehen, da ansonsten Verweise auf die Module fehlschlagen. Bei der Entwicklung der LuCI Anzeige- und Konfigurationskomponente, die im Folgenden *luci-ripd* genannt wird, wurde die in der Abbildung 2.6 gezeigte Verzeichnisstruktur Verwendet (das dem dargestellten Dateisystembaum übergeordnete Verzeichnis ist *applications*). Um unnötige Umschreibungen und Sprachliche Platzhalter für Namen von Erweiterungskomponenten und umständlich verallgemeinerten Verzeichnisnamen zu vermeiden, bezieht sich der folgende Text auf

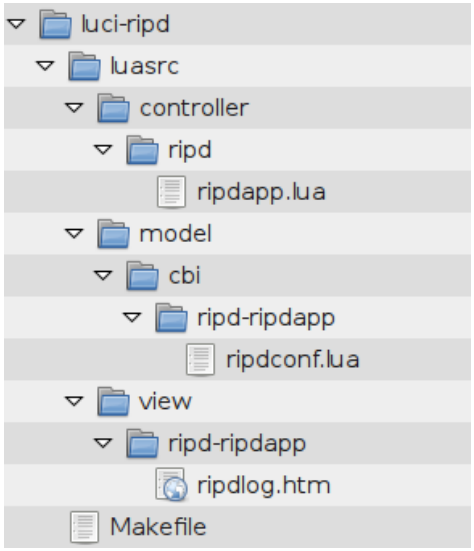

Abbildung 2.6: Dateisystemstruktur des luci-ripd Quelltextes

den konkreten Fall der Implementierung der LuCI Erweiterung „luci-ripd“, deren Verzeichnis- und Quelltextdatei-Namensgebung in der Abbildung 2.6 dargestellt ist.
Der abgebildete Verzeichnisbaum zeigt eine Unterteilung der Quelltextdateien in die drei Rollen *controller*, *model* und *view*.

Controller Ein nach der Applikation benanntes Unterverzeichnis mit dem Namen der Applikation wird im Odener *controller* angelegt, in diesem Fall *ripd*. In dem Controller-Teil einer Applikation erfolgt durch den Aufruf der *entry(...)*-Funktion die Einbindung der Applikation in den Baum des LuCI-Dispatchers. Die *entry(...)*-Funktion benötigt dazu zwei Argumente, zwei weitere können optional übergeben werden:

```
1 entry(path, target, title=nil, order=nil)
```

Dabei bezeichnet *path* die Einordnung in die Baumstruktur des Dispatchers, dieser Parameter wird in Form einer Liste von Knoten angegeben. Der zweite Prameter, *target*, gibt die Aktion an, die ausgeführt werden soll, wenn der Knoten aufgerufen wird, an dieser Stelle kann ein Verweis auf eine Methode (wie in Abbildung 2.7 in Zeile 12), auf ein Template oder ein CBI-Modell bzw ein CBI-Formular (wie in Abbildung 2.7, Zeile 19) angegeben werden. Optional kann

dem *entry(...)*-Aufruf ein Titel, unter dem die Applikation im Menü der LüCI-Oberfläche dem Benutzer angezeigt wird sowie ein ganzzahliger numerischer Wert *order* nach dem Knoten einer Ebene des Dispatcher-Baums sortiert werden (vgl. [wika]). Im Quelltext zu ripdapp.lua (Abbildung 2.7) wird in der elften Zeile „{"admin", "status", "ripdlog"}" als Pfad für die Einordnung in die Baumstruktur angegeben, was im LuCI-Menü einen Eintrag mit dem durch die Lokalisierungsfunktion *luci.i18n.translate* bezogenen Anzeigenamen im Administrations-Menü unter „Status" bewirkt.
Es ist darüber hinaus möglich, Knoten auf allen Ebenen des Dispatcher-Baums durch den Aufruf der Funktion *node* zu erstellen.

```
1 local page  = node()
2 page.target = alias("node1")
```

Der Funktion *node(...)* wird als Argument der Pfad des Dispatcher-Baumes von der Wurzel aus angegeben, unter dem der neue Knoten wird, wird kein Argument übergeben, befindet sich der Knoten auf der höchsten Ebene, für die Einordnung im Menü bedeutet das, dass der Knoten auf der gleichen Ebene steht wie „Essentials" (intern: „mini") und „Administration" (intern: „admin"). Es wird ein lokal gültiges Objekt erzeugt, dem der von der Funktion *node(...)* erzeugte Knoten zugewiesen wird. Im Anschluss werden die Eigenschaften des Knotens durch das Anpassen der Attribute des Objekts festgelegt. Durch das Setzen des Attributs *target* wird der Knoten für die interne Zuordnung benannt. Der Knoten *node1* ist jedoch nur intern ansprechbar, um ihm einen Anzeigenamen zuzuweisen, wird er instanziiert und das Attribut *title* gesetzt:

```
1 local page = node("node1")
2 page.title = "My first node"
```

Weitere Attribute, wie z.B. *target*, was in diesem Fall den bei einem Klick-Ereignis aufzurufenden Knoten bedeutet, können analog an dieser Instanz gesetzt werden. Eine Übersicht der Dispatcher-Funktionen von LuCI ist in der Dokumentation des LuCI API unter http://dev.luci.cyranjo.org/luci-doc/modules/luci.dispatcher.html abrufbar.

Model (CBI-) Modelle und Formulare werden in einem Unterordner des Verzeichnisses *model* angelegt, das nach der Applikation und dem jeweils das Modell aufrufenden Controller-Moduls benannt wird, in diesem Fall *ripd-ripdapp*.

```
module("luci.controller.ripd.ripdapp", package.seeall)

function index()
  require("luci.i18n")
  luci.i18n.loadc("ripd")

  if not nixio.fs.access("/etc/quagga/ripd.conf") then
    return
  end

  local page = entry({"admin", "status", "ripdlog"},
        call("action_ripdlog"),
        luci.i18n.translate("log_ripd"))
  page.i18n = "ripd"
  page.dependent=false
  page.order = 80

  local page = entry({"admin","services","ripdconf"},
        form("ripd-ripdapp/ripdconf"),
        luci.i18n.translate("cfg_ripd"))
  page.i18n = "ripd"
  page.dependent=false
  page.order = 80
end

function action_ripdlog()
  local logfile = luci.sys.exec("cat /var/log/quagga/ripd.log")
  luci.template.render("ripd-ripdapp/ripdlog", {logfile=logfile})
end
```

Abbildung 2.7: Quelltext: ripdapp.lua

Aufgerufen werden CBI- Modelle und Formulare als Aktion von einem Knoten des Dispatcher-Baums. Bei der Verwendung von CBI-Modellen bzw. Formularen wird die im Browser angezeigte Benutzeroberfläche zur Ein- und Ausgabe durch LuCI generiert, das Schreiben eines Templates im Verzeichnis *luasrc/view* entfällt somit für den Entwickler. Unterschieden wird zwischen Modellen und Formularen wie folgt: Ein Modell bezieht sich auf eine UCI-konforme Konfigurationsdatei, ein Formular ist nicht an UCI gebunden und lässt dem Entwickler die Freiheit eine beliebige Datei des Dateisystems als Quelle bzw. Ziel zu behandeln. Da der Quagga RIP Daemon nicht durch UCI konfiguriert wird und die Umstellung auf UCI-konforme Konfiguration auch dem Ziel dieser Arbeit, eine Testumgebung aus realen und virtualisierten Routern zu schaffen, nicht zuträglich ist, da die Umstellung des RIP Daemons die Verwendung zweier unterschiedlicher syntaktischer Schemata für die gleiche Konfigurationsdatei innerhalb der kombinierten Umgebung zur Folge hätte, liegt der Fokus im Folgenden auf dem CBI-Formular. Das CBI-Formular ist eine Instanz des Typs *SimpleForm*. Diesem Objekt

```
1  local fs = require "nixio.fs"
2  local conffile = "/etc/quagga/ripd.conf"
3  local restartdaemon = "/etc/init.d/quagga restart"
4
5  f = SimpleForm("ripcfg", translate("cfg_ripd"), translate("ex_cfg_ripd"))
6
7  t = f:field(TextValue, "rip")
8  t.rmempty = true
9  t.rows = 20
10 function t.cfgvalue()
11   return fs.readfile(conffile) or ""
12 end
13
14 function f.handle(self, state, data)
15   if data.rip then
16     fs.writefile(conffile, data.rip:gsub("\r\n", "\n"))
17     luci.util.exec(restartdaemon)
18   end
19   return true
20 end
21
22 return f
```

Abbildung 2.8: Quelltext: ripdconf.lua

können beliebig viele Felder (Atrribut *field*) zugewiesen werden, die die dargestellten Formularkomponenten repräsentieren. Ein *SimpleForm*-Objekt beinhaltet darüber hinaus zwei Schaltflächen, „Reset“ und „Submit“, die die vom Benutzer im Formular vorgenommenen Einstellungen verwerfen bzw. an LuCI übergeben. Das Formular zur Konfiguration des RIP-Daemons (Abbildung 2.8) enthält ein Textfeld, das den Inhalt der Konfigurationsdatei */etc/quagga/ripd.conf* anzeigt. Im Quelltext wird Das Textfeld durch das Erzeugen eines *TextValue*-Objektes als „Field“-Attribut des *SimpleForm* Objektes realisiert (Zeile 6). Der Inhalt der Konfigurationsdatei in dem Dateisystem-Aufruf *readfile*, der in dem Modul fs der Nixio API enthalten ist, geladen. Die Funktion *SimpleForm.handle* ist die durch einen Klick auf die Schaltfläche „Submit“ aufgerufene Aktion, in diesem Fall beinhaltet die Aktion das Schreiben der geänderten Konfigurationsdatei und den Neustart des Quagga-Daemons.

View Anzeigekomponenten werden, genau wie CBI Modelle und Formulare auch in einem nach dem Package und dem Namen der Aufrufenden Klasse benannten Unterverzeichnis abgelegt, im Beispiel also *luasrc/view/ripd-ripdapp*. Als Anzeigekomponenten werden HTML-Templates verwendet, denen durch die sie aufrudende Controller-Klasse Variablen übergeben werden können (die in Script-

Tags innerhalb des Templates angenommen werden), weiterhin ist es möglich Anwendungslogik in Form von Skripten in Templates zu integrieren. Durch die Skript-Aufrufe <%+*header*%> und <%+*footer*%> wird der dazwischen befindliche Inhalt in die Optik der LuCI Oberfläche integriert. Die Darstellung einer Anzeigekomponente wird durch den Aufruf der Methode *luci.template.render(..)* innerhalb der Controller-Klasse ausgelöst (siehe Abbildung 2.7, Zeile 28). Der Anzeigekomponente für die Protokolldatei des Quagga RIP Daemons wird beim Aufruf die Variable *logfile* übergeben. Da Templates über eigene, von der Controller-Klasse unabhängige Namensräume verfügen, müssen im Template verwendete Werte der *render*-Methode als Variablen übergeben werden. Im Quelltextbeispiel wird dem Template die in der Controller-Klasse gültige Variable *logfile* (Abbildung 2.7) auf die im Template gültige Variable *logfile* (Abbildung 2.9) abgebildet.

```
1 <%+header%>
2 <h2><a id="content" name="content">ripd.log</a></h2>
3 <div id="content_syslog">
4 <textarea readonly="readonly" wrap="off" id="syslog"><% write (logfile) %></
      textarea>
5 </div>
6 <%+footer%>
```

Abbildung 2.9: Quelltext: ripdlog.htm

Lokalisierung LuCI unterstützt die Entwicklung mehrsprachiger Benutzeroberflächen mit dem Paket *luci.i18n*. Die Lokalisierung beruht auf dem simplen Prinzip, ein Alias für einen Text einzuführen und durch die Funktion *luci.i18n.translate(...)*

Sprache	**Kennung**
Deutsch	de
Englisch	en
Französisch	fr
Griechisch	el
Italienisch	it
Japanisch	ja
Katalanisch	ca
Portugiesisch	pt
Portugiesisch (Brasilien)	pt_BR

Abbildung 2.10: Liste der von LuCI 0.9.0 unterstützten Sprachen

den Zieltext in der für LuCI global eingestellten Sprache aus der Lokalisierungsdatenbank abzufragen. Die Nutzung der Lokalisierung erfordert die Angabe der jeweils benutzten Lokalisierungsdatei, im folgenden Beispiel wird die Datei *ripd* in der aktuell eingestellten Sprache geladen:

```
1 luci.i18n.loadc("ripd")
```

Damit die Lokalisierung funktioniert, ist ein Eintrag in der Makefile im Stammverzeichnis der LuCI-Applikation erforderlich:

```
1 PO = ripd
```

Durch diesen Eintrag wird dem LuCI SDK mitgeteilt, welche Lokalisierungsdateien der Applikation zugeordnet werden. Die Lokalisierungsdateien werden im Unterverzeichnis *po* des LuCI Stammverzeichnises in Ordnern abgelegt, die nach den Kennungen der jeweiligen Sprachen (siehe Abbildung 2.10) benannt sind. Im dem Verzeichnis *templates* befinden sich Vorlagen für die Erstellung weiterer Sprachdateien zu bereits für Vielsprachigkeit implementierte LuCI Komponenten. Das Abfragen der Lokalisierung erfolgt durch das Aufrufen der Funktion

```
1 luci.i18n.translate("cfg_ripd")
```

mit der Übergabe des Aliases, unter dem der gewünschte Text in der Datenbank zu finden ist. Die Lokalisierungsdatei besteht aus einer Liste von Paaren, bestehend aus Zeichenketten, die jeweils als *msgid* und *msgstr* bezeichnet werden (siehe Abbildung 2.11). Sofern LuCI in englischer Sprache benutzt wird, liefert die oben erwähnte Anfrage die Zeichenkette „Quagga ripd configuration“ zurück.

Ausführung der LuCI-Testumgebung Das LuCI SDK gibt dem Entwicker die Möglichkeit, die Innerhalb des Frameworks entwickelte Software zu testen. Der Vorteil gegenüber dem Ausführen der Software auf einem OpenWrt Router liegt in der Zeitersparnis, da die Kompilierung im OpenWrt SDK das Vorhandensein einer Menge von Makefile-Einträgen voraussetzt und je nach CPU-Leistung des Hostsystems einen Zeitaufwand von mehreren Stunden bedeuten kann, zudem muss nach jeder Kompilierung das Paket auf dem Zielgerät installiert werden. Das SDK bietet drei verschiedene Möglichkeiten, LuCI-Software mittels der Makefile im LuCI-Stammverzeichnis zu testen:

- Ausführen des LuCI Frameworks mit Weboberfläche,

```
msgid ""
msgstr ""

#. Add entry
#: i18n/english/luasrc/i18n/ripd.en.lua:1
msgid "log_ripd"
msgstr "Quagga ripd logfile"

#. Add entry
#: i18n/english/luasrc/i18n/ripd.en.lua:2
msgid "cfg_ripd"
msgstr "Quagga ripd configuration"

#. Add entry
#: i18n/english/luasrc/i18n/ripd.en.lua:1
msgid "ex_cfg_ripd"
msgstr "This text field allows you to adjust the configuration
file of the quagga RIP daemon to your requirements."
```

Abbildung 2.11: Quelltext: ripd.po (englisch)

- starten einer Lua-Konsole in der LuCI-Umgebung
- starten einer Bash-Konsole in der LuCI-Umgebung.

Die Weboberfläche wird auf einem im LuCI SDK enthaltenen Webserver uf dem lokalen Port 8080 bereitgestellt und ermöglicht dem Entwickler das Betrachten der Integration seiner Erweiterungen in das LuCI-Framework, sowie, falls von der Software angesprochene Dateien im Dateisystem des Hosts vorhanden sind und der Benutzer über die für den Zugriff erforderlichen Rechte verfügt auch das Bearbeiten von Konfigurationsdateien. Nicht möglich ist das Starten und Beenden von Daemons aus der Testumgebung heraus, da diese Aktionen nur durch den Systemadministrator *root* durchgeführt werden können. Gestartet wird der Webserverbasierte Test-Modus durch Eingabe von

```
make runhttpd
```

auf der Konsole eines Benutzers im Stammverzeichnis des LuCI SDK.

Das Ausführen der Lua-Konsole in der LuCI-Umgebung bietet die Möglichkeit den geschriebenen Quellcode auf Fehler zu testen. In der Lua-Konsole können Klassen instanziiert werden und Operationen auf Objekten ausgeführt werden, gleichzeitig können die (Fehler-) Meldungen des Interpreters gelesen werden. Gestartet wird diese Umgebung durch einen Aufruf von

```
make runlua
```

im Stammverzeichnis des LuCI SDK.

Das Testen der LuCI-Umgebung von einer Bash-Konsole innerhalb der ausgeführten Umgebung ermöglicht das Abfragen von durch LuCI gesetzten Umgebungsvariablen, die Beeinflussung der LuCI-Umgebung durch seine Schnittstellen zum Betriebssystem. Diese Umgebung wird von einer Benutzerkonsole aus im Stammverzeichnis des LuCI SDK durch den folgenden Aufruf gestartet.

```
make runshell
```

Aufnahme in den OpenWrt-Feed Die für das Erstellen der OpenWrt Pakete erforderlichen Makefiles sind im LuCI SDK bereits vorhanden, müssen jedoch um die hinzugefügte Erweiterung ergänzt werden. Für das Bauen der Pakete unter der OpenWrt Build-Umgebung wird die Makefile unter *contrib/package/luci* verwendet. Dieser Datei müssen folgende Einträge hinzugefügt werden (vgl. [wikb]):
Eine Beschreibung des Pakets, mit einem Titel, der später durch die OpenWrt Build-Umgebung und durch OPKG ausgelesen wird, sowie Abhängigkeiten zu anderen Paketen, die für die Funktion der Software erforderlich sind:

```
define Package/luci-app-ripd
  $(call Package/luci/webtemplate)
  DEPENDS+=+luci-admin-core +luci-admin-full
  TITLE:=LuCI Support for the Quagga RIP daemon
endef
```

Das Build-System benötigt weiterhin einen Eintrag, in dem das Quellverzeichnis der Applikation angegeben ist und dieser zugeordnet wird:

```
define Package/luci-app-ripd/install
        $(call Package/luci/install/template,$(1),applications/luci-ripd)
endefx
```

Eine Build-Anweisung die abgefragt wird, wenn das Paket in der OpenWrt Build-Umgebung ausgewählt wurde und auf das Quelltextverzeichnis der Applikation verweist:

```
ifneq ($(CONFIG_PACKAGE_luci-app-ripd),)
        PKG_SELECTED_MODULES+=applications/luci-ripd
endif
```

Und ein Aufruf zum Bauen des Pakets.

```
$(eval $(call BuildPackage,luci-app-ripd))
```

In der OpenWrt Build-Umgebung steht die Applikation zur Verfügung, wenn der LuCI-Feed eingebunden wurde, die am schnellsten zum Ziel führende Möglichkeit ist die Verwendung des Feeds des LuCI Projekts, dessen LuCI-Makefile ()durch die modifizierte aus dem Verzeichnis *contrib/package/luci* ersetzt wird. Das Einbinden des LuCI-Feeds geschieht durch die Eingabe der folgenden Befehle auf der Kommandozeile im Stammverzeichnis der OpenWrt Build-Umgebung.

```
./scripts/feeds update packages luci
./scripts/feeds install -a -p luci
```

Um den Quelltext des erweiterten LuCI-Frameworks in den Build-Vorgang einzubringen, ist der inhalt des LuCI SDK Verzeichnisses in einem mit Gzip komprimierten Tar-Archiv in den Unterordner *dl* der OpenWrt Build-Umgebung zu kopieren.

2.2.3 ripd-Plugin für LuCI

Ein Teilziel dieser Arbeit ist die Erstellung einer in die LuCI-Umgebung integrierbaren Anzeige- und Konfigurationskomponente für die Logfile bzw. die Konfigurationsdatei des Quagga RIP Daemons, die als Paket für OPKG bereitgestellt werden soll. Nachdem der Code bereits in Abschnitt 2.2.2 erläutert wurde, wird der Fokus nun auf die fertige LuCI Erweiterung gerichtet. Im Folgenden wird die Installation des Plugins auf einem OpenWrt Router, sowie die Software im Betrieb gezeigt. Während der Implementierung wurden Parallelen der Anforderungen zu den Funktionsweisen bereits in LuCI enthaltener Komponenten ausgenutzt, so dass diese als Vorlage dienen konnten.

Installation Die von OPKG vorgesehene Möglichkeit zur Installation von Software ist, diese aus einem Paket-Repository zu beziehen. Die dafür erforderlichen Dateien werden von der OpenWrt Build-Umgebung durch folgenden Aufruf erzeugt:

```
$ make world
```

Die Übergabe von *world* als Parameter weist die OpenWrt Build-Umgebung an, alle in der Konfiguration ausgewählten Komponenten zu bauen. Die erzeugten

installierbaren Pakete, sowie der Datei *Packages* befinden sich im Unterverzeichnis *bin/brcm47xx/packages*, wobei brcm47xx wenn für andere Plattformen als die des WRT54G(L) gebaut wird, durch die jeweils ausgewählte Zielplattform ersetzt werden muss, und können von dort in ein Verzeichis auf einem beliebigen Webserver kopiert werden, zu dem der OpenWrt Router Zugang hat. Anschließend muss dem Paketmanager mitgeteilt werden, das er Pakete von diesem Server beziehen kann, dies geschieht durch folögenden Eintrag in der Konfigurationsdatei des OPKG */etc/opkg.conf*:

```
1 src/gz myrepo http://userp.uni-koblenz.de/~ataflinski/rmti
```

Das Beispiel weist OPKG an, die Datei „Packages“ bzw. „Packages.gz“, die Informationen über die innerhalb des Repositories bereitgestellten Pakete enthält, unter der angegebenen URL zu suchen. Der voreingestellte Eintrag, der auf den Server des OpenWrt-Projekts verweist sollte herauskommentiert werden, da es sonst zu Problemen beim herunterladen des Lokalisierungspakets kommen kann, dass mit identischer Versionskennung ebenfalls in der originalen Quelle verfügbar ist. Mit folgendem Aufruf läd der Paketmanager die Informationen über das eingestellte Repository:

```
1 # opkg update
```

Anschließend ist die Installation der LuCI-Erweiterung und des veränderten Lokalisierungspakets (die für luci-app-ripd erforderlichen Übersetzungen sind in den englischen und deutschen Lokalisierungspaketen enthalten) durch den Aufruf des Paketmanagers von der Kommandozeile möglich:

```
1 # opkg install luci-app-ripd luci-i18n-english
```

Gegebenenfalls überspringt OPKG die Installation der Sprachdatei mit folgender Meldung:

```
1 Package luci-i18n-english (0.9.0-1) installed in root is up to date.
```

In dem Fall muss die Installation der Sprachdatei erzwungen werden:

```
1 # opkg install luci-i18n-english --force-reinstall
```

Anschließend kann die Erweiterung verwendet werden.

luci-ripd im Betrieb In der Menüstruktur ist die Erweiterung an zwei Stellen integriert: Die Seite, auf der die Konfiguration auf der Web-Oberfläche verändert werden kann (siehe Abbildung **??**), ist durch den Eintrag „Quagga ripd

configuration“ unter „Services“ im Administrationsmenü zugänglich (bei englischer Spracheinstellung), die Anzeigekomponente für die Logfile (siehe Abbildung **??**)des RIP Daemons kann durch einen Klick auf „Quagga ripd logfile“ unter Status im Administrationsmenü aufgerufen werden.

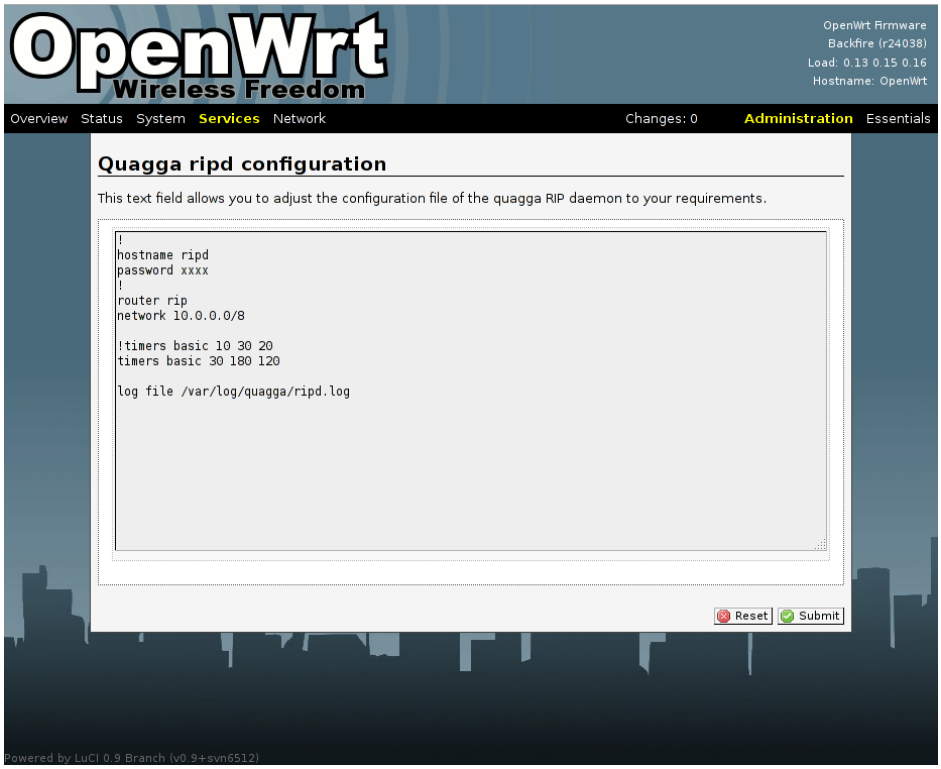

Abbildung 2.12: Konfigurationskomponente für den RIP Daemon

2.3 UCI

Konventionell werden die Einstellungen von Diensten (Daemons) in Unixartigen Betriebssystemen in Konfigurationsdateien gespeichert, die im Unterverzeichnis „/etc“ des Wurzeldateisystems abgelegt sind. Dieser Konvention folgt auch OpenWrt, jedoch wird die Konfiguration einiger wichtiger Daemons in OpenWrt mit dem Werkzeug „UCI“ (**U**nified **C**onfiguration **I**nterface) zentralisiert. Zentralisiert bedeutet, dass die Konfiguration der Dienste durch im Verzeichnis „/etc/config“ abgelegte Konfigurationsdateien, die einem einheitlichen Schema entsprechen, geschieht und permanente Änderungen an den von UCI verwalteten

Abbildung 2.13: Anzeigekomponente für die Protokolldatei des RIP Daemon

Konfigurationen bevorzugt durch UCI vorgenommen werden sollen. UCI bietet die Möglichkeit, durch den Aufruf eines Kommandozeilen-Programms die Konfiguration eines Dienstes zu verändern. Die logische Struktur jeder durch UCI verwalteten Konfigurationsdatei entspricht der eines Baums.

2.3.1 Der Konfigurationsbaum

Die von UCI verwaltete Kofiguration ist objektorientiert und in einer Baumstruktur aufgebaut, die auch im Dateisystem und in der Syntax der UCI konformen Konfigurationsdateien umgesetzt ist: Wenn man das Verzeichnis */etc/config* als Wurzel der Baumstruktur betrachtet, stellt die Konfigurationsdatei in dieser Struktur einen der Wurzel untergeordneten Kind-Knoten dar. Die Konfigurationsdateien teilen sich jeweils in einzelne Abschnitte wie folgt aufgebaut sind (vgl. [wikc]):

```
1 config 'identifier' 'name'
2         option 'another_identifier' 'value1'
```

```
3         option 'another_identifier2' 'value2'
4         list 'list_identifier' 'item1'
5         list 'list_identifier' 'item2'
```

Die Blattknoten der Baumstruktur entsprechen den *option*- bzw. *list*-Einträgen eines *config*-Abschnitts. Unter der objektorientierten Betrachtung entspricht die Konfigurationsdatei einer Klasse, jeder *config*-Abschnitt einem Objekt vom Typ *identifier*, dem der Name *name* gegeben wird, die *option*- und *list*-Zeilen setzen die Attribute dieses Objekts. Die Anweisung *option* ordnet einem Bezeichner einen Wert zu, mit *list* können einer oder mehreren Listen, die durch ihren Bezeichner unterschieden werden, Einträge zugewiesen werden.

```
1  config 'switch' 'eth0'
2    option 'enable' '1'
3
4  config 'switch_vlan' 'eth0_0'
5    option 'device' 'eth0'
6    option 'vlan' '0'
7    option 'ports' '1 2 3 4 5'
8
9  config 'interface' 'loopback'
10   option 'ifname' 'lo'
11   option 'proto' 'static'
12   option 'ipaddr' '127.0.0.1'
13   option 'netmask' '255.0.0.0'
14
15 config 'interface' 'lan'
16   option 'type' 'bridge'
17   option 'ifname' 'eth0'
18   option 'proto' 'static'
19   option 'ipaddr' '192.168.1.2'
20   option 'netmask' '255.255.255.0'
21   option 'gateway' '192.168.1.1'
22   option 'dns' '8.8.8.8'
```

Abbildung 2.14: Datei einer Netzwerkkonfiguration

Die Bedienung der Konfiguration eines OpenWrt-Hosts ist, außer durch die Manipulation der Konfigurationsdateien mit einem Texteditor auch mit dem Kommandozeilentool von UCI möglich, LuCI greift ebenfalls auf die Programmierschnittstellen von UCI zurück. Das Kommandozeilenprogramm UCI wird im Folgenden, um die Verwechselung mit UCI als Name des Konfigurations-Frameworks zu verhindern „UCI-CLI“ (‘Command Line Interface“, engl. Kommandozeilen-Schnittstelle) genannt. UCI-CLI stellt dem Benutzer folgende Grundfunktionen zum Manipulieren des Konfigurationsbaumes zur Verfügung:

- Hinzufügen von *config*-Objekten (add)

```
network.eth0=switch
network.eth0.enable=1
network.eth0_0=switch_vlan
network.eth0_0.device=eth0
network.eth0_0.vlan=0
network.eth0_0.ports=1 2 3 4 5
network.loopback=interface
network.loopback.ifname=lo
network.loopback.proto=static
network.loopback.ipaddr=127.0.0.1
network.loopback.netmask=255.0.0.0
network.lan=interface
network.lan.type=bridge
network.lan.ifname=eth0
network.lan.proto=static
network.lan.ipaddr=192.168.1.2
network.lan.netmask=255.255.255.0
network.lan.gateway=192.168.1.1
network.lan.dns=8.8.8.8
```

Abbildung 2.15: Netzwerkkonfiguration mit UCI-CLI

- Hinzufügen von Listen (add_list)
- Abfragen eines *option*-Wertes (get)
- Setzen eines *option*-Wertes (set)
- Löschen eines Elements (delete)
- Umbenennen eines Objekts oder Attributs(rename)
- Zurücknehmen eines Objekts oder Attributs (revert)
- Ordnen der Objekte (reorder)

2.4 Portierung des RMTI auf OpenWrt

Die Software Routing Suite Quagga ist ein mächtiges Softwarepaket, dass mehrere Daemons mitbringt, die jeweils dem Austausch von Routinginformationen dienen und Implementierungen verschiedener Routingalgorithmen darstellen. Der an der Universität Koblenz entwickelte Routingalgorithmus ist als Modifikation des RIP Daemons der Quagga Software implementiert. Weiterhin enthält Quagga Implementierungen folgender Algorithmen: BGP, IS-IS, und OSPF. Quagga wird in dem standarmäßig eingestellten Repository auch als Paket für OpenWrt bereitgestellt, jedoch ohne die RMTI-Implementierung. Die Grundlage zum bauen

des Quagga-Pakets bildet die Makefile, mit der das Quagga-Paket des OpenWrt-Projekts erstellt wurde. Da diese nicht in der Build-Umgebung von OpenWrt enthalten ist, muss ein seperates SVN-Repository mit Makefiles für die Erstellung weiterer Pakete ausgecheckt werden. Aus praktischen Gründen bietet es sich an, den Ordner, der die Vorlagen zum Erzeugen von Netzwerktools beinhaltet direkt in das *package*-Verzeichnis der OpenWrt Build-Umgebung auszuchecken:

```
$ cd package
$ svn co svn://svn.openwrt.org/openwrt/packages/net/
```

Nun bedarf die Makefile im Unterverzeichnis *package/net/quagga* noch einer Bearbeitung, damit die Build-Umgebung das Quelltextpaket mit dem RMTI akzeptiert. Zunächst muss im Kopf der Makefile die Prüfsumme der *quagga-<Versionsnummer>.tar.gz* benannten, komprimierten Datei, die den Quelltext enthält eingetragen werden:

```
PKG_NAME:=quagga
ifneq ($(CONFIG_QUAGGA_OLD),)
  PKG_VERSION:=0.98.6
  PKG_RELEASE:=8
  PKG_MD5SUM:=b0d4132039953a0214256873b7d23d68
  PATCH_DIR:=./patches-old
else
  PKG_VERSION:=0.99.17
  PKG_RELEASE:=5
  PKG_MD5SUM:=37b9022adca04b03863d2d79787e643f
endif
```

Der erste Teil ist uninteressant, der RMTI steht für aktuelle Quagga-Versionen zur Verfügung, in dieser Makefile ist die Versionsnummer 0.99.17 angegeben. Die md5-Prüfsumme erhält man, in dem man md5 einen Hashwert über die Datei bilden lässt:

```
$ md5sum quagga-0.99.17.tar.gz
```

Anschließend ist noch eine weitere Modifikation der Makefile notwendig, um Quagga mit dem RMTI kompilieren zu können. In dem Abschnitt *CONFIGURE_ARGS*

```
CONFIGURE_ARGS+= \
  --localstatedir=/var/run/quagga \
  --sysconfdir=/etc/quagga/ \
  --enable-shared \
  --disable-static \
  --enable-user=quagga \
  --enable-group=quagga \
```

```
  --enable-pie=no \
  --enable-multipath=8 \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-libzebra,zebra) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-libospf,ospfd) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-bgpd,bgpd) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-isisd,isisd) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-ospf6d,ospf6d) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-ripd,ripd) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-ripngd,ripngd) \
  $(call autoconf_bool,CONFIG_PACKAGE_quagga-vtysh,vtysh) \
```

muss folgende Zeile im oberen Teil hinzugefügt werden:

```
  --with-cflags=gnu99
```

Durch die Übergabe dieses Arguments an das Konfigurations-Skript, dass die Kompilierung des Quagga-Quelltextes vorbereitet, wird der Compiler angewiesen, C-Quelltext nach dem Standard C99 zu akzeptieren. Nun bietet es sich an, noch eine kleine Modifikation am Init-Skript des Quagga Daemons vorzunehmen, da das Verzeichnis */var/log/quagga*, in dem der RIP Daemon seine Logdatei ablegt unter OpenWrt in einem flüchtigen Speicherbereich liegt und nach jedem Bootvorgang neu erstellt werden muss. Das Init-Skript liegt im Unterverzeichnis files des Ordners, in dem sich die Makefile befindet. Die Datei heißt *quagga.init* und hat folgenden Inhalt:

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2006 OpenWrt.org

START=60
start() {
  /usr/sbin/quagga.init start
}

stop() {
  /usr/sbin/quagga.init stop
}
```

Damit beim Start von Quagga auf die Existenz des Verzeichnisses Überprüft wird, bietet sich folgende Erweiterung des *start()*-Abschnitts an:

```
start() {
  if [ -d /var/log/quagga ]; then
    chown quagga:quagga /var/log/quagga
  else
    mkdir /var/log/quagga
    chown quagga:quagga /var/log/quagga
  fi
```

```
8   /usr/sbin/quagga.init start
9 }
```

Damit wird sichergestellt, dass das Verzeichnis, sofern es existiert, dem Benutzer Quagga gehört, mit dessen Rechten Quagga ausgeführt wird und dass es, wenn es nicht existiert mit den erforderlichen Benutzerrechten erstellt wird. Nun wird noch der Inhalt des Ordners *patches* gelöscht, da die mit OpenWrt ausgelieferten Patches zum Teil nicht mit dem modifizierten Quelltext der Universität Koblenz kompatibel sind. Anschließend wird das komprimierte Quelltextpaket in den Unterordner *dl* der OpenWrt Build-Umgebung kopiert. Beim Aufruf der Menuconfig-Oberfläche zur Konfiguration des Build-Vorgangs mittels

```
1 $ make menuconfig
```

im Stammverzeichnis der OpenWrt Build-Umgebung wird nun die Erstellung der Software Quagga als Modul ausgewählt. Als Modul bauen bedeutet im OpenWrt-Kontext lediglich, dass das Paket als *.ipk*-Paket zur Verfügung gestellt wird, die Software jedoch beim Aufruf von *make world* nicht in die erzeugte Imagedatei hineingeschrieben wird. Der RIP Daemon der Quagga-Software wird in der Menuconfig-Umgebung an folgender Stelle angezeigt:

```
1 --- Network
2       <M> quagga  --->
3         <M> quagga-ripd
```

Der Letzte Schritt ist nun, das Paket durch die OpenWrt Build-Umgebung zu erzeugen. Um die Installation auf einem Router einfacher zu gestalten bietet es sich an, hier wie auch beim Erzeugen des Pakets für die LuCI-Erweiterung vorzugehen und ein die gesamtes OpenWrt-System gemäß der Konfiguration bauen zu lassen.

```
1 $ make world
```

Die Installation der Software auf dem Router erfolgt dann analog zur Installation der LuCI-Erweiterung (siehe Abschnitt 2.2.3)von der Shell des OpenWrt-Routers.

Kapitel 3

Uhrensynchronisierung

Ziel des Testens ist es, das Verhalten des Algorithmus unter festgelegten Bedingungen zu beobachten, Fehlfunktionen zu erkennen und analysieren, sowie Vergleiche mit anderen Algorithmen unter identischen Testbedingungen durchführen zu können. Die im Rahmen dieser Arbeit erweiterte Testumgebung soll eine Evaluation der Kommunikation einzelner Router anhand der von ihnen aufgezeichneten Log-Dateien ermöglichen und mit den in ihnen enthaltenen Informationen das Verhalten des Alg̈orithmus im Testsystem nachvollzogen werden. Um anhand der in Log-Dateien enthaltenen Informationen das Verhalten des Routingverfahrens nachvollziehen zu können, ist eine Synchronisierung der Uhren der einzelnen Systeme notwendig, die die Rekonstruktion der zeitlichen Abfolge aufgetretener Ereignisse aus den Log-Dateien aller Teilnehmenden Systeme ermöglicht. Ein im Desktop- und Serverbereich weit verbreitetes Protokoll zur Synchronisation von Uhren über ein Netzwerk ist das serverbasierte *Network Time Protocol* (NTP), das z.B. als standardmäßiges Protokoll zur Uhrensynchronisierung unter aktuellen Windows Betriebssystemen Verwendung findet. Zur Synchronisierung einer Menge von Uhren in einer Laborumgebung existiert jedoch mit dem Precision Time Protocol (PTP), das in der IEEE-Norm 1588 standardisiert ist eine geeignetere Lösung. Geeigneter, weil das Precision Time Protocol selbstorganisierend ist, d.h. PTP legt selbstständig eine der teilnehmenden Uhren als Referenzuhr fest, zu der die anderen Teilnehmer synchronisiert werden. Für die Auswertung von Logdateien ist es unerheblich, auf welche Zeit die Uhren der Teilnehmenden Systeme eingestellt sind, sofern alle Uhren zueinander synchron

sind. Ein zentraler Server ist nicht manuell einzurichten, auf die Nutzung externer Resourcen kann ebenfall verzichtet werden.

3.1 IEEE 1588 - PTP

Das *Precision Time Protocol* (IEEE 1588) ist ein Protokoll zur Uhrensynchronisierung, dass für Uhrensynchronisierung in Test- und Messystemen entwickelt wurde [CBB05]. PTP ist daher nicht auf die Synchronisierung von Uhren über große Distanzen ausgelegt, wie beispielsweise das Network Time Protocol (NTP), bei dem ein Host über das Internet mit einem Zeit-Server verbunden wird, sondern beschränkt sich auf die Synchronisation der Uhren innerhalb eines Netzes. Weiterhin ist das PTP symmetrisch ausgelegt, d.h. es arbeitet nicht mit einer statischen Unterteilung zwischen Servern und Clients, sondern ermittelt aus der Menge zur Verfügung stehender Uhren mit dem „Best Master Clock"-Algorithmus eine aufgrund ihrer Präzision und Stabilität als Referenzuhr für die Synchronisierung der Uhren innerhalb des Netzwerkes geeignete Uhr [CBB05]. PTP überprüft die Synchronität der Uhren in regelmäßigen Zeitabständen in einem Nachrichtenaustausch. Beim Austausch der Nachrichten werden auf der Seite der Master-Clock sowie der Slave-Clock die Aus- und Eingangszeiten beim Versand der Sync- und Delay-Request-Nachrichten gespeichert und zur Berechnung von der Übertragungsverzögerung und den Zeitdifferenzen zwischen Master-Clock und Slave-Clock genutzt (siehe Abschnitt 3.1.1). Nach der Berechnung der Zeitdifferenzen der Slave-Clocks zur Master-Clock werden diese laut der dem Quelltext des PTPd in der Version 1.0.0 im Unterverzeichnis *doc/* beigefügten Dokumentation durch Verringern (für negative Differenzen) bzw. durch Erhöhen der Zählgeschwindigkeit der Uhr angepasst. W erden Diferrenzen größer als $1s$ errechnet, wird die Zeit der Slave-Clock neu gesetzt.
Der Quelltext (*src/dep/sys.c*) der freien Implementierung offenbart, dass das Precision Time Protocols neun Zustände kennt, die (zumindest in der Implementierung) wie folgt benannt sind:

- PTP_INITIALIZING
- PTP_FAULTY
- PTP_LISTENING

- PTP_PASSIVE
- PTP_UNCALIBRATED
- PTP_SLAVE
- PTP_PRE_MASTER
- PTP_MASTER
- PTP_DISABLED

Da der Fokus dieser Arbeit in der Netzwerktechnik liegt, wird im Folgenden der Kommunikationsablauf des Precision Time Protocols näher betrachtet.

3.1.1 Nachrichtenaustausch

Nach der Festlegung der Master- und Slave Clocks nehmen die Teilnehmenden Hosts ihre Rolle als *Master-* oder *Slave-Clock* ein. Der Nachrichtenaustausch setzt die Zuweisung der Rollen der einzelnen Hosts zwingend voraus, da der Nachrichtenaustausch zur Feststellung der Übertragungsverzögerung und der Zeitdifferenzen zwischen den Uhren der Slave-Clocks zur Master-Clock von dem als Master-Clock festgelegten Host aus beginnt. Dieser versendet, wie in Abbildung 3.1 eine sogenannte Sync Nachricht zu einem Zeitpunkt t_1 der Master Clock an eine Multicast-Adresse, mit kurzen zeitlichen Abstand folgt die Follow_Up Message, die den Zeitstempel t_1 enthält. Der Slave erhält die Sync Nachricht zum Zeitpunkt t_2 der Slave Clock und antwortet dem Master zum Zeitpunkt t_3 (Slave Clock) mit der Delay_Req Nachricht, die dieser zum Zeitpunkt t_4 gemäß der Master Clock erhält und mit der Delay_Resp Nachricht, der der Zeitstempel t_4 angehängt ist, bestätigt. Aus den vier Zeitstempeln t_1, t_2, t_3 und t_4 berechnet jeder Slave die durch die Netzwerkkommunikation gegebene Verzögerung sowie einen Ausgleichswert. Im folgenden werden diese beiden Größen mit ihren englischsprachigen Bezeichnungen, Delay und Offset benannt, um eine möglichst geringe Entfernung zu der Terminologie in den Publikationen, die das Precision Time Protocol betreffen, zu wahren. Delay und Offset sind wie folgt definiert (vgl.

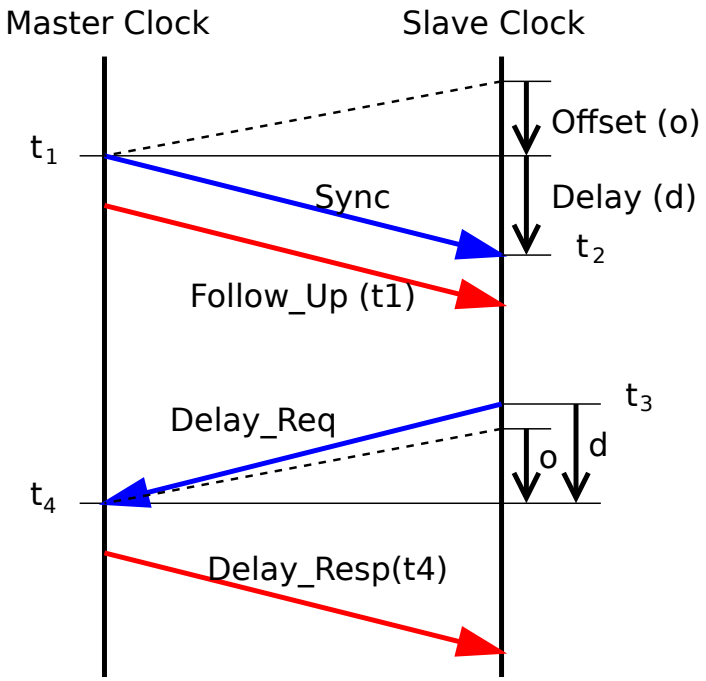

Abbildung 3.1: PTP Nachrichtenaustausch ([WB04])

[WB04] und [CBB05]):

$$d_{prop} = \frac{((t_2 - t_1) + (t_4 - t_3))}{2}$$
$$\Delta t = (t_2 - t_1) - d_{prop}$$

Als Propagation Delay für die Netzwerkkommunikation wird also das Mittel der Zeitdifferenzen der Übertragung vom Master zum Slave und der Übertragung in umgekehrte Richtung angenommen. Dieser Berechnung geht die Annahme voraus, dass die Verzögerungen in beide Übertragungsrichtungen als ungefähr gleich betrachtet werden können.

3.2 PTPd

Der *Precision Time Protocol Daemon* (PTPd) ist eine unter BSD-Lizenz veröffentlichte Software-Implementierung des Precision Time Protocols für GNU/Linux basierte Betriebssysteme. Von PTPd existieren derzeit zwei aktuelle Versionen, die

unterschiedlichen, parallel geführten Entwicklungssträngen entspringen. Der in dieser Arbeit verwendete PTPd ist die in der Version 1.0.0 vorliegende Implementierung des Precision Time Protocols der Version 1. PTPd ist eine in C geschriebene reine Software-Implementierung des Prcision Time Protocol, die keine besonderen Fähigkeiten Hardware erfordert. Die verwendeten Zeitstempel bezieht PTPd durch Systemaufrufe direkt vom Linux Kernel. Die Analyse der Kommunikation zwischen den mit PTPd synchronisierten Hosts mit Hilfe der Software Wireshark offenbart, dass die Nachrichten der PTPd Instanzen über die Ports 319 und 320 auf dem verbindungslosen Protokoll UDP ausgetauscht werden, wobei die in der Abbildung 3.1 durch blaue Pfeile dargestellten Nachrichten beidseitig auf dem Port 319 kommuniziert werden, die in rot dargestellten Nachrichten werden auf dem Port 320 ausgetauscht. Alle Nachrichten werden ausschließlich über die Multicast-Adresse 224.0.1.129 sofern dieser verfügbar ist, kommuniziert. Anderenfalls inkrementiert PTPd das letzte Oktett der Adresse und prüft erneut die Verfügbarkeit. Das Versenden der Sync-Nachrichten mit jeweils fol-

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.10	224.0.1.129	PTPv1	Sync Message
2	0.000720	192.168.0.10	224.0.1.129	PTPv1	Follow_Up Message
3	0.001300	192.168.0.14	224.0.1.129	PTPv1	Delay_Request Message
4	0.002164	192.168.0.10	224.0.1.129	PTPv1	Delay_Response Message
5	1.999998	192.168.0.10	224.0.1.129	PTPv1	Sync Message
6	2.000759	192.168.0.10	224.0.1.129	PTPv1	Follow_Up Message
7	4.000076	192.168.0.10	224.0.1.129	PTPv1	Sync Message
8	4.000790	192.168.0.10	224.0.1.129	PTPv1	Follow_Up Message
9	4.001253	192.168.0.22	224.0.1.129	PTPv1	Delay_Request Message
10	4.002108	192.168.0.10	224.0.1.129	PTPv1	Delay_Response Message
11	6.000120	192.168.0.10	224.0.1.129	PTPv1	Sync Message
12	6.000816	192.168.0.10	224.0.1.129	PTPv1	Follow_Up Message
13	8.000141	192.168.0.10	224.0.1.129	PTPv1	Sync Message
14	8.000853	192.168.0.10	224.0.1.129	PTPv1	Follow_Up Message
15	10.000176	192.168.0.10	224.0.1.129	PTPv1	Sync Message
16	10.000879	192.168.0.10	224.0.1.129	PTPv1	Follow_Up Message
17	10.001621	192.168.0.30	224.0.1.129	PTPv1	Delay_Request Message
18	10.002452	192.168.0.10	224.0.1.129	PTPv1	Delay_Response Message

Abbildung 3.2: Kommunikation des PTPd (Wireshark Mitschnitt)

genden Follow-Up-Nachrichten durch den als Master in Erscheinung tretenden Host geschieht, wie im Wireshark-Mitschnitt (Abbildung 3.2) zu sehen ist, standardmäßig im Abstand von $2s$. PTPd bietet vielfältige Möglichkeiten der Anpassung an die Anwendungssituation durch die Übergabe von Kommandozeilenparametern. Die einfachste Möglichkeit ist, PTPd durch folgenden Befehl auf der

Kommandozeile zu starten (Root-Rechte sind erfordelich, um die Einstellung der Systemuhr zu ändern):

```
1 # ptpd
```

Ohne die Angabe von Parametern beim Aufruf wird PTPd als Daemon gestartet und kommuniziert über alle zur Verfügung stehenden Netzwerke.
Im Folögenden wird eine kleine Auswahl der Konfigurationsmöglichkeiten vorgestellt, eine vollständige Übersicht zulässiger Kommandozeilenparameter wird durch den Aufruf des PTPd mit dem Parameter *-?* angezeigt. PTPd lässt sich, in dem Fall der Anbindung eines Hosts an mehrere Netzwerke einer bestimmten Netzwerkschnittstelle zuweisen, die Multicast-Kommunikation erfolgt dann ausschließlich über das an der Schnittstelle angeschlossene Netzwerk, auch die Angabe einer virtuellen Schnittstelle ist möglich. Im folgenden Beispiel wird der PTPd mit der Bindung an die Netzwerkschnittstelle *eth0* gestartet.

```
1 # ptpd -b eth0
```

Obwohl PTPd selbstständig eine Uhr als Master- oder Slave-Clock festlegen kann, besteht dennoch die Möglichkeit, per Komanndozeilenparameter eine Rolle festzulegen. Mit dem Parameter *-p* wird PTPd als Master-Clock („preferred“) gestartet, als Slave-Clock wird eine PTPd-Instanz mit dem Parameter *-g* gestartet.

```
1 # ptpd -p
2 # ptpd -g
```

Weiterhin ist es möglich, für den Fall das eine PTPd-Instanz als Master betrieben wird, das Zeitintervall zwischen den Sync-Nachrichten manuell festzulegen:

```
1 # ptpd -y 3
```

Das Zeitintervall wird als Exponent einer Potenz mit der Basis 2 angegeben, $2^x, x \in \mathbb{N}$, im obingen Beispiel werden Sync-Nachrichten demnach im Abstand von $2^3 = 8s$ an die Multicast-Adresse verschickt.
Um die Arbeit des PTPd genauer zu beobachten, besteht die Möglichkeit, PTPd nicht als Daemon zu starten, sondern durch den parameter *-c* als Anwendung, die die Textkonsole als Ausgabemedium verwendet.

```
1 # ptpd -c
```

Der aktuelle Zustand und die Messwerte werden durch das Hinzufügen des Parameters *-d* im Konsolen-Modus angezeigt.

```
1 # ptpd -c -d
```

Kapitel 4

Aufbau eines Testnetzwerkes mit OpenWrt Routern

Für Labordemonstrationen im Rahmen von Aktionstagen an der Universität Koblenz ist eine Netzwerkumgebung bestehend aus acht Routern, die mittels mehrerer Netze teilweise redundant miteinander verbunden sind. Die dabei verwendeten Router sind Geräte von Linksys der Modellreihe WRT54GL. Als Betriebssystem kommt auf diesen Routern OpenWrt 10.03 “Backfire” zum Einsatz. Die Routing-Funktionalität wird von der Software-Routing Umgebung Quagga bereitgestellt, deren Implementierung des Routing Information Protocols als Basis für den an der Universität Koblenz entwickelten Routingalgorithmus RMTI (Routing with Metric based Topology Investigation) dient und für Vorführungen mit der Software XTPeer in der Version 0.99.4, sowie für Tests ohne XTPeer in der Version 0.99.17 zum Einsatz kommt. OpenWrt eignet sich als Betriebssystem für die Router des Testnetzwerks, da es seinen Ursprung im GPL-lizensierten Quelltext der von Linksys ausgelieferten Firmware eben dieser Geräte hat und speziell für Kleingeräte weiterentwickelt wird. Darüber hinaus stehen dem Anwender eines OpenWrt Systems alle in Linux-Umgenbungen üblichen Werkzeuge hinsichtlich der für die Errichtung eines Netzwerkes relevanten Funktionen bereits als Paket zur Verfügung oder können mit der Quelloffenen Build-Umgebung von OpenWrt als installierbares Paket für den Paketmanager OPKG gebaut werden. Eine nähere Erläuterungen zu der Build-Umgebung und der Erstellung von installierbaren Paketen für OpenWrt findet sich in Abschnitt 2.1. Die Konfiguration der Komponenten der OpenWrt-Umgebung gestaltet sich zudem einfach, da

OpenWrt viele Konfigurationsdateien mit dem Kommandozeilen-Werkzeug UCI (Unified Configuration Interface) zentralisiert, dessen Konfigurationsdateien eine einheitliche, simpel aufgebaute Syntax haben und im Verzeichnis */etc/config* abgelegt sind. UCI-Konfigurationsdateien (vgl. Abschnitt 2.3) können auch während dem Betrieb von der Kommandozeile durch den Aufruf von UCI verändert werden, ein Neustart der von den Änderungen betroffenen Dienste ist jedoch in den meisten Fällen erforderlich.
In der Praxis erfolgt der Zugriff auf den Router mit installiertem OpenWrt entweder über die Webbasierte Konfigurationsoberfläche LuCI oder auf der Kommandozeile mittels einer SSH-Sitzung.

4.1 Demonstrationsnetzwerk

Das für die Labordemonstrationen aufgebaute Netzwerk besteht aus acht Linksys Routern der Baureihe WRT54GL, die in der in Abbildung 4.1 dargestellten Topologie miteinander verbunden sind. Die Topologie ist so aufgebaut, dass die

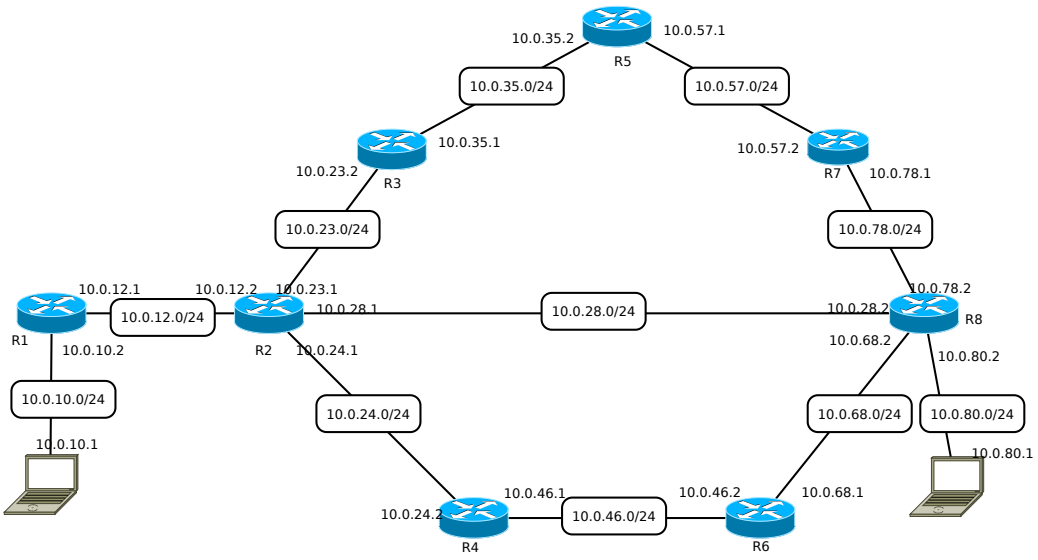

Abbildung 4.1: Topologoie des Demonstrationsnetzwerkes mit 8 Routern

Nachrichten, die von den an den Routern R1 und R8 angeschlossenen Hosts über verschiedene Wege mit jeweils unterschiedlicher Anzahl von Hops geleitet werden können. Wenn eine Netzwerkverbindung auf dem aktuell benutzten

Weg getrennt wird, wird durch den RIP-Daemon, wenn möglich, eine alternative Route festgelegt. Die Abgebildete Topologie stellt das Netzwerk, auf dem der Routingalgorithmus ausgeführt wird, dar. Tatsächlich ist es um ein Management-Netzwerk erweitert, an das alle Router angeschlossen sind. Das Management-Netzwerk besteht aus je einem Subnetz mit zwei Hostadressen pro Router (IP-Netz 192.168.0.XXX/30), in dem sich jeweils der Router und der zu Konfigurations- und Diagnosezwecken angeschlossene PC befinden. Die physikalische Netzwerkverbindung zwischen dem Host und den Routern wird durch einen Switch bereitgestellt, die logischen Verbindungen durch die Definition virtueller Schnittstellen auf dem Hostsystem:

```
1 # ifconfig eth0:1 192.168.0.1 netmask 255.255.255.252
2 # ifconfig eth0:2 192.168.0.5 netmask 255.255.255.252
3 # ifconfig eth0:3 192.168.0.9 netmask 255.255.255.252
4 # ifconfig eth0:4 192.168.0.13 netmask 255.255.255.252
5 # ifconfig eth0:5 192.168.0.17 netmask 255.255.255.252
6 # ifconfig eth0:6 192.168.0.21 netmask 255.255.255.252
7 # ifconfig eth0:7 192.168.0.25 netmask 255.255.255.252
8 # ifconfig eth0:8 192.168.0.29 netmask 255.255.255.252
```

Den Management-Schnittstellen der Router wird jeweils eine um eins höhere IP-Adresse innerhalb des gleichen Subnetzes zugewisen. Die Ausbreitung von Nachrichten des Routingalgorithmus über das Management-Netzwerk wird durch die Netzwerkeinstellung der Konfigurationsdatei des RIP Daemons auf

```
1 network 10.0.0.0/8
```

verhindert. Die Einstellung hat zur Folge, dass der RIP Daemon seine Routinginformationen ausschließlich in diesem Netzwerk und seinen Subnetzen austauscht, über die dem Management-Netzwerk zugeordnete Netzwerkschnittstelle werden keine Multicast-Nachrichten verschickt. Über das Management-Netzwerk ist auch die Verwendung der Software XTPeer möglich, das, wenn ein mit dieser Software kompatibler RIP Daemon benutzt wird, über eine Diagnoseschnittstelle des RIP Daemons den jeweiligen Zustand der Routingtabellen der Router visualisiert darstellen kann. Die Uhrensynchronisation mit PTPd, der unter BSD-Lizenz veröffentlichten, quelloffenen Implementierung des Precision Time Protocols (siehe Kapitel 3), erfolgt ebenfalls über das Management-Netzwerk. Da auch PTPd durch Multicast-Nachrichten kommuniziert, stellt die Zuweisung einzelner Subnetze für die Kommunikation der PTP Daemons keine Hürde dar. Zu diesem Zweck werden die Init-Skripte des PTPd auf den beteiligten OpenWrt Routern

angepasst, dem PTPd wird die Schnittstelle *eth0.0*, die an das Management-Netz angeschlossen ist, wie folgt durch die Ergänzung der Datei */etc/init.d/ptpd.init* zugewiesen:

```
1  #!/bin/sh /etc/rc.common
2
3  START=50
4
5  start() {
6    ptpd -b eth0.0
7  }
8
9  stop() {
10   killall ptpd
11 }
```

4.2 Netzwerkeinstellungen

Die IP-Adressen, die die verwendeten Router auf ihren Anschlüssen vorweisen müssen, sind aus der grafischen Darstellung der Topologie in Abbildung 4.1 ersichtlich. Die Konfiguration der Netzwerkschnittstellen erfolgt durch die Anpassung der Datei */etc/config/network*, die nach dem simplen Schema der UCI-Konfigurationsdateien aufgebaut ist. Alternativ ist die Konfiguration auch mit UCI oder LuCI möglich, die entsprechende Einstellungsseite findet sich im Administrationsbereich unter dem Eintrag „Switch“ der Netzwerkeinstellungen. Bei der Umsetzung dieser Topologie gibt es eine Eigenheit der eingesetzten Router zu beachten: Die fünf vorhandenen Netzwerkanschlüsse sind hardwareseiting wie ein einziger Switch verschaltet, dessen einzelne Anschlüsse jedoch softwareseitig als VLANs konfigurierbar sind [AP07]. Die Konfigurationsdatei für die Netzwerkeinstellungen gliedert sich in einzelne „config“-Sektionen, die mit einer Angabe über den Betreff der Einstellungen und einer Geräte bzw. Netz-Kennung in Form von Zeichenketten versehen sind. Eine „config“-Sektion besteht aus „option“-Statements, die jeweils mit einem Namen und einem ihm innerhalb der Konfiguration zugewiesenen Wert als Zeichenketten übergeben werden. Die Netzwerkkonfigurationsdatei eines WRT54G beinhaltet „config“-Sektionen der der Typen „switch“, „switch_vlan“ und „interface“. Durch das setzen der Option „enable=1“ einer Switch-Sektion, wird der Switch des Routers aktiviert, anderenfalls verbleibt die Konfiguration der VLANs wirkungslos. Die Konfiguration

eines VLANs geschieht in der Konfigurationsdatei durch die Festlegung eines Namens, die Zuweisung einer physikalischen Netzwerkschnittstelle (auf Routern der WRT54G-Reihe von Liksys steht nur eine physikalische Netzwerkschnittstelle *eth0* zur Verfügung), des Setzens einer VLAN-Nummer und dem Hinzufügen von Ports des Switches bzw dem Zugriff des Betriebssystems des Routers auf das VLAN. Im Folgenden Beispiel wird das VLAN für die Management-Schnittstelle eines Routers aus dem Demonstrationsnetzwerk definiert:

```
config 'switch_vlan' 'eth0_0'
    option 'device' 'eth0'
    option 'vlan' '0'
    option 'ports' '4 5'
```

Diese Konfiguration kann ebenfalls mit dem Kommandozeilentool *uci* durchgeführt werden, die dazu erforderlichen Aufrufe lauten wie folgt:

```
# uci set network.eth0_0=switch_vlan
# uci set network.eth0_0.device=eth0
# uci set network.eth0_0.vlan=1
# uci set network.eth0_0.ports='4 5'
```

In dem Beispiel werden dem VLAN die Ports 4 und 5 zugeordnet. Port 5 bedeutet in der Konfiguration von OpenWrt die Schnittstelle zum Betriebssystem, möglich ist auch das Konfigurieren von VLANs auf dem Switch eines WRT54G(L)-Routers auf die Kein Zugriff durch das Betriebssystem besteht. Bei der Konfiguration von VLANs auf Linksys WRT54GL Routern ist zu beachten, dass die intern genutzte Nummerierung der Ports nicht der Beschriftung auf dem Gehäuse entspricht, die Zuordnung der Beschriftungen zu den intern verwendeten Portnummern ist in Abbildung 4.2 dargestellt.

Beschriftung	**interne Port-Nummer**
Internet	4
1	3
2	2
3	1
4	0

Abbildung 4.2: Switch-Ports des Linksys WRT54GL v1.1

Die Konfigurierten VLANs werden *interface*-Konfigurationen durch die Einstellung des Attributs *ifname* zugewiesen (sofern das Betriebssystem auf das be-

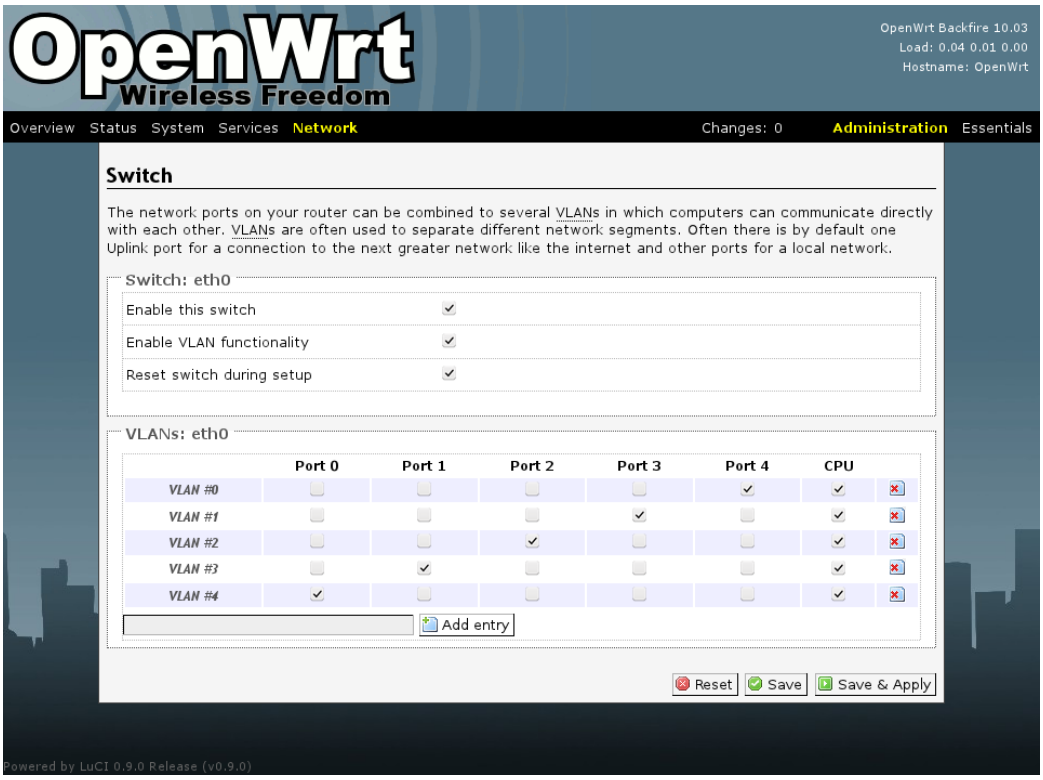

Abbildung 4.3: VLAN-Konfiguration des Switches unter LuCI

treffende VLAN Zugriff hat), in denen Festlegungen der Netzwerke auf der IP-Ebene gemacht werden. In der *interface*-Konfiguration wird durch die Einstellung des Attributs *proto* festgelegt, ob das Gerät in diesem Netz seine auf das Netz bezogenen Einstellungen dynamisch durch das Protokoll DHCP zugewiesen bekommt, oder ob sie statisch festgelegt werden. Im Fall der statischen Festlegung folgen Einstellungen der in dem Netz verwendeten IP-Adresse (Attribut *ipaddr*) und der Subnetzmaske (*netmask*), optional sind weitere Einstellungen wie das Setzen einer Standard-Route, der Angabe eines DNS-Servers etc. möglich. Das folgende Beispiel zeigt die Konfiguration der an das Management-Netzwerk angeschlossenen Schnittstelle eines Routers aus dem Demonstrationsnetzwerk:

```
1 config 'interface' 'mgmt'
2     option 'proto' 'static'
3     option 'ifname' 'eth0.0'
4     option 'ipaddr' '192.168.0.2'
5     option 'netmask' '255.255.255.252'
```

Diese Konfiguration kann mit dem Kommandozeilentool *uci* durch folgende Aufrufe erzeugt werden:

```
# uci set network.mgmt=interface
# uci set network.mgmt.proto=static
# uci set network.mgmt.ifname=eth0.0
# uci set network.mgmt.ipaddr=192.168.0.2
# uci set network.mgmt.netmask=255.255.255.252
```

Die Web-Oberfläche LuCI bietet ebenfalls die Möglichkeit, die Netzwerkeinstellungen vorzunehmen, die entsprechenden Konfigurationsseiten finden sich im Menü „Administration“ unter Network und sind mit „Interfaces“ und „Switch“ benannt, Abbildung 4.3 zeigt einen Screenshot der Konfigurationsseite für die VLANs auf dem Switch.

Kapitel 5

Anbindung von OpenWrt-Routern in VNUML-Netze

OpenWrt-Router stellen eine kostengünstige Plattform zum Testen von Routingalgorithmen unter Realbedingungen dar. Jedoch stößt die alleinige Nutzung dieser Router beim Skalieren der Szenarien schnell an Grenzen: Mit zunehmender Größe werden reale Netzwerke für den Benutzer unüberschaubar und die Suche nach Fehlern in den Konfigurationen der einzelnen Geräte nimmt viel Zeit in Anspruch. An der Universität Koblenz wird in der Lehre, sowie auch in der Forschung das Netzwerksimulationssystem VNUML eingesetzt, das auf der Basis von User-Mode Linux Kernels Netzwerke nachbildet (eine nähere Beschreibung dieser Software findet sich in Abschnitt 5.1 und in [Keu05]). Der Nachteil dieses Systems besetht darin, dass es dem Tester keine Realbedingungen zur Verfügung stellt.

5.1 VNUML

VNUML ist ein ursprünglich an der Technischen Universität Madrid entwickeltes Werkzeug zur Simulation von miteinander vernetzten Linux-Systemen. Die VNUML-Umgebung nutzt dafür die Möglichkeit, leicht modifizierte Linux Kernel im User-Mode zu betreiben. Die Modifikation der im User-Mode ausführ-

```
R1:~# ping -c 10 10.0.12.2
PING 10.0.12.2 (10.0.12.2) 56(84) bytes of data.
64 bytes from 10.0.12.2: icmp_seq=1 ttl=64 time=20.0 ms
64 bytes from 10.0.12.2: icmp_seq=2 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=3 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=4 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=5 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=6 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=7 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=8 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=9 ttl=64 time=0.000 ms
64 bytes from 10.0.12.2: icmp_seq=10 ttl=64 time=0.000 ms

--- 10.0.12.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9140ms
rtt min/avg/max/mdev = 0.00/2.000/20.000/6.000 ms
R1:~#
```

Abbildung 5.1: Ausgabe des Ping-Befehls in einer VNUML-Konsole

baren Linux-Kernel umfasst die Weiterleitung von Systemaufrufen an den darunterliegenden Kernel des Hostsystems, anstatt auf die Hardware zuzugreifen. VNUML stellt dem Benutzer zwar entsprechend seiner Szenario-Vorgabe vernetzte virtualisierte Linux-Systeme zur Verfügung, jedoch treten Effekte wie Verzögerungen in den auf diese Art virtualisierten Netzwerken in nicht mit realen Netzwerken vergleichbaren Maßen auf. So zeigt sich, wenn VNUML auf einem Host mit einem SKAS3 (Separate Kernel Adress Space)-gepatchten Kernel einer Version unter 2.6.32 ausgeführt wird, der Effekt, dass die RTTs mit 0.000 ms angegeben wird, wie es in der Konsolenausgabe des Ping-Befehls in Abbildung 5.1 zu sehen ist. Das Vorhandensein von Latenzen, die in der Regel mit steigender Anzahl von Knoten auf dem Weg eines Pakets wachsen gehört zum Verhalten realer Netzwerke.

Für die Netzwerkkonfiguration der User-Mode Kernels untereinander sowie mit dem Hostsystem nutzt VNUML Ethernet-Brücken und sog. TAP-Schnittstellen. TAP-Schnittstellen sind virtuelle Netzwerkschnittstellen die den Datenaustausch zwischen dem Betriebssystemkernel und auf Benutzerebene ausgeführten Anwendungen durch Gerätedateien (*/dev/tapX*) ermöglichen, aus denen Datenströme von Anwendungen gelesen werden können. Der Kernel erhält die Daten aus der, der TAP-Schnittstelle zugehörigen Netzwerkschnittstelle (i.d.R. *tapX*), ebenso wird ein vom Kernel durch die TAP-Netzwerkschnittstelle kommunizierter Datenstrom auf die Gerätedatei geleitet und kann von dort gelesen werden [Keu05]. VNUML liest die zur Nachbildung einer Netzwerktopologie benötigten Informa-

tionen aus einer Beschreibung, die in einer XML-basierten Beschreibungssprache vorliegt. In der VNUML- Beschreibungssprache wird die Topologie eines IPv4-Netzwerks üblicherweise durch folgende Merkmale festgelegt:

Tag	Attribute	Bedeutung
net	name, mode	Netzwerkverbin
vm	name	Virtuelle Maschi
if	id, net	legt eine Netzwe Identifikationsnu Ethernet-Verbind
ipv4	mask	gibt die IPv4-Ad Subnetzmaske a
4	0	

5.2 Ethernet Bridges

Auf der logischen Ebene steht der Koppelung realer Router an virtualisierte Netzwerke nichts im Weg - die Management-Interfaces der einzelnen virtuellen Maschinen stellen mit den auf dem Hostsystem entsprechend konfigurierten virtuellen Schnittstellen eine „Durchreiche“ zum Hostsystem dar, die sich aus der Sicht des Anwenders auf dem Hostsystem nahtlos in die Logik der Netzwerkinterfaces integriert. VNUML greift für die Vernetzung der User-Mode Kernels untereinander auf die Funktionalität des Linux Kernel, Ethernet-Brücken zu erstellen zurück. Ethernet-Brücken verbinden, wie ein Switch Netzwerkschnittstellen auf der Ebene des Ethernet-Protokolls, d.h sie sind keine Netzwerkknoten im Sinne eines IP-Netzwerkes. So bleibt beispielsweise die TTL der über eine Ethernet-Brücke übertragenen Pakete von dieser unbeeinflusst. Ethernet-Brücken verbinden Netzwerkgeräte wie ein Switch, indem ihre Schnittstellen mit der Ethernet-Brücke verbunden werden. Die Fähigkeit, Ethernet-Brücken zu errichten ist im Funktionsumfang der Linux-Kernel-Pakete der meisten Distributionen enthalten. Anderenfalls ist dieses Feature vor der Kompilierung des Kernels wählbar, in der Menuconfig-Oberfläche findet sich die Option wie in Abbildung 5.2 gezeigt. Ethernet-Brücken können auch reale Netzwerkschnittstellen des Hostsystems hinzugefügt werden und somit Netzwerkverbindungen aus dem VNUML ausführenden Host heraus. In der XML-basierten VNUML-Beschreibungssprache sind Ethernet-Brücken in Form des Netz-Tags („<net>“) mit den Attributen „na-

```
--- Networking support
      Networking options  --->
        <M> 802.1d Ethernet Bridging
```

Abbildung 5.2: Kernelmodul: 802.1d Ethernet Bridging

me“ und „mode“ integriert, das Interface-Tag („<if>“) besitzt ein Attribut „net“, dem eine mit dem Netz-tag definierte Ethernet-Brücke zugeordnet werden kann. Der in Abbildung 5.3 dargestellte Ausschnitt aus der Beschreibung einer VNUML-virtualisierten Netzwerktopologie zeigt die Verbindung zweier virtueller Maschinen „R1“ und „R2“ über die Ethernet-Brücke „Net1“, an der beide virtuelle Maschinen mit jeweils einer Netzwerkschnittstelle verbunden sind. Jedoch ist zu be-

```
  <net name="Net0" mode="virtual_bridge" />
  <net name="Net1" mode="virtual_bridge" />
  <net name="Net2" mode="virtual_bridge" />

  <vm name="R1">
    <if id="1" net="Net0">
      <ipv4 mask="255.255.255.0">10.0.10.2</ipv4>
    </if>
    <if id="2" net="Net1">
      <ipv4 mask="255.255.255.0">10.0.12.1</ipv4>
    </if>
  </vm>

  <vm name="R2">
    <if id="1" net="Net1">
      <ipv4 mask="255.255.255.0">10.0.12.2</ipv4>
    </if>
    <if id="2" net="Net2">
      <ipv4 mask="255.255.255.0">10.0.23.1</ipv4>
    </if>
  </vm>
```

Abbildung 5.3: VNUML-Beschreibung zweier vernetzter Router

rücksichtigen, dass das Verbinden mehrerer virtueller Netzwerkkontaktpunkte mit realen oder virtuellen Netzwerkkontaktpunkten außerhalb des Hosytsystems über eine einzige Ethernet-Brücke und eine Netzwerkschnittstelle des Hostsystems zu ungewollten Effekten im Betrieb des RMTI Algorithmus führt, da dieser seine Routingtabelle über Multicast-Nachrichten an andere Router weitergibt und Multicast-Nachrichten auf Switches unabhängig von den eingestellten Subnetzmasken der beteiligten Schnittstellen kommuniziert werden können.

5.2.1 Einrichtung von Ethernet Bridges

Die Fähigkeit, mit Ethernet-Brücken umzugehen, ist in aktuellen Versionen des Linux-Kernels integriert, zur Einrichtung wird ein Programm namens *brctl* benutzt, dass in den verbreiteten Linux-Distributionen durch das Paket *bridge-utils* installiert werden kann. Linux behandelt Ethernet-Brücken wie Netzwerkschnittstellen, d.h. sie können mit dem Befehl *ifconfig* aktiviert- und deaktiviert werden. Da Brücken durch ein Kernel-Modul zur Verfügung gestellt werden, sind zur Bearbeitung der Brücken mit *brctl* Systemverwaltungsrechte notwendig. Wenn man eine Brücke mit *brctl* erstellt, bindet diese zunächst keine Netzwerkschnittstellen und befindet sich im deaktivierten Zustand.

```
1 # brctl addbr myfirstbridge
```

Werden einer Brücke Netzwerkschnittstellen hinzugefügt, stellt sie diesen die Funktionalität eines Verteilers zur Verfügung. An einer Brücke teilnehmende Schnittstellen können untereinander kommunizieren, als ob sie an einem auf Ethernet-Ebene arbeitenden Switch angeschlossen wären. Ethernet-Brücken können darüber hinaus auch ähnlich wie Netzwerkschnittstellen mit einer IP-Adresse konfiguriert werden und damit logisch die Position von Schnittstellen einnhemen, während die an ihnen teilnehmenden Schnittstellen dann die Funktionalität der Ports eines Switches, der an eben dieser Schnittstelle angeschlossen ist, zur Verfügung stellen. Das Hinzufügen von Schnittstellen zu einer Ethernetbrücke geschieht durch die Anweisung *addif*.

```
1 # brctl addif myfirstbridge eth0
2 # brctl addif myfirstbridge eth1
```

Diese Aufrufe fügen der Brücke „myfirstbridge“ die Netzwerkschnittstellen „eth0“ und „eth1“ hinzu. Aktiviert bzw. deaktiviert wir eine Ethernet-Brücke durch *ifconfig*:

```
1 # ifconfig myfirstbridge up
2 # ifconfig myfirstbridge down
```

Wie auf einigen Switches ist auch auf Ethernet-Brücken das starten (und beenden) des Spanning-Tree-Protokolls möglich:

```
1 # brctl stp myfirstbridge on
2 # brctl stp myfirstbridge off
```

Das Entfernen eines Interfaces von einer Ethernet-Brücke geschieht durch:

```
# brctl delif myfirstbridge eth0
```

Die gesamte Brücke wird mit folgendem Aufruf entfernt:

```
# brctl delbr myfirstbridge
```

brctl bietet die Möglichkeit, eine Übersicht aller auf einem Host vorhandenen Ethernet-Brücken und der an ihnen teilnehmenden Schnittstellen anzuzeigen:

```
# brctl show
```

5.3 Kommunikation mit VNUML-Netzen

Das Errichten einer Netzwerkverbindung zwischen einem virtualisierten Netzwerk und realen Routern ist mit den in den letzten Abschnitten vorgestellten Mitteln möglich. Da die Vernetzung der virtuellen Maschinen untereinander auf Ethernet-Brücken basiert, die auf dem Hostsystem erstellt werden, existiert bereits eine Netzwerkverbindung der virtuellen Maschinen zum Kernel des Hostsystems. Einer existierenden Netzwerkbrücke kann mit dem Kommandozeilentool *brctl* eine reale Netzwerkschnittstelle des Hostsystems hinzugefügt werden.

```
# ifconfig Net1 down
# ifconfig eth0 0.0.0.0 up
# brctl addif Net1 eth0
# ifconfig Net1 up
```

Das erste Szenario, das betractet werden soll ist das in Abbildung 5.4 abgebildete: Die Netzwerkschnittstelle einer virtuellen Maschine VM1 wird mittels einer Ethernet-Brücke mit der Netzwerkschnittstelle des Hostsystems verbunden, an der ein Router angeschlossen ist. Eine Verbindung zwischen der virtuellen Maschine und dem Router ist nun möglich, ein von der virtuellen Maschine aus gesendeter Ping wird von dem Router beantwortet. Die Herausleitung der Netzwerkanbindung einer virtuellen Maschine geschieht auf Ethernet-Ebene, das Hostsystem tritt nicht als Netzwerkteilnehmer eines IP-Netzwerkes in Erscheinung. Auch die Weitergabe von Routinginformationen durch den RIP Daemon ist mit dieser Netzwerkverbindung möglich.

Das nächste Szenario (Abbildung 5.5), dass betrachtet werden soll, besteht aus einem virtualisierten Netzwerk in einer VNUML-Umgebung und einem realen

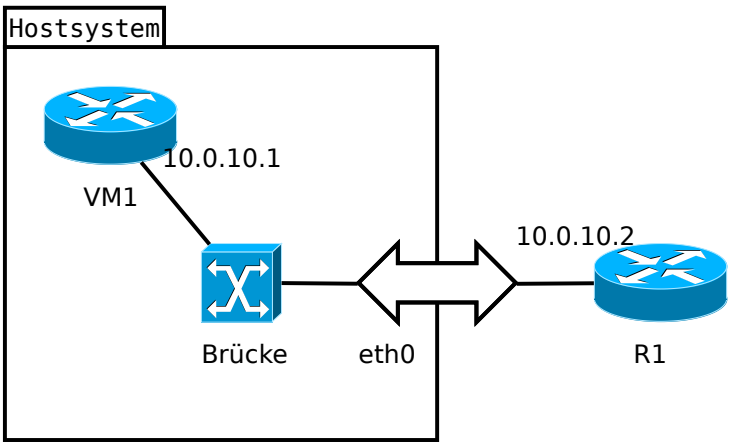

Abbildung 5.4: Szenario 1

Netzwerk bestehend aus Routern außerhalb des Hostsystems. Da mehrere Verbindungen durch eine Netzwerkschnittstelle des Hostsystems geführt werden, wird die Verbindung außerhalb des Hostsystems durch einen Switch gebündelt, innerhalb des Hostsystems sind ebenfalls beide virtuellen Maschinen mit Verbindungen nach außen an einer Ethernet Brücke angeschlossen. Die mit den virtuellen Maschinen kommunizierenden Router sind nicht miteinander Verbunden und befinden sich in unterschiedlichen Subnetzen, ebenso sind auch die mit den Routern außerhalb des Hostsystems kommunizierenden virtuellen Maschinen in unterschiedlichen Subnetzen und nicht direkt miteinander verbunden. Das Problem, das von der Tatsache kommt, dass der RMTI-Algorithmus zum Austausch von Routinginformationen zwischen benachbarten Routern Multicast-Gruppen benutzt, also über die Grenzen eines Subnetzes hinaus mit anderen RMTI-Instanzen kommuniziert, tritt im Testdurchlauf mit diesem Szenario auf. So ist mit der Diagnosesoftwrae Wireshark zu erkennen, dass alle beteiligten Router, sowohl die virtuellen als auch die realen, über den Switch bzw. die Ethernetbrücke durch Multicast-Verbindungen miteinander kommunizieren und Routinginformationen austauschen. Aus diesem Grund eignet sich die Bündelung mehrerer Netzwerkverbindungen zur Kommunikation virtueller Maschinen mit realen Routern nicht als universell einsetzbare Methode zur Verbindung.
Das Erreichen des gewünschten Kommunikationsverhaltens ist durch die Benutzung separater Schnittstellen möglich, wie es in Szenario 3 (Abbildung 5.6) gezeigt ist. Dieses Szenario zeigt im Test das erwartete Verhalten, die Kommuni-

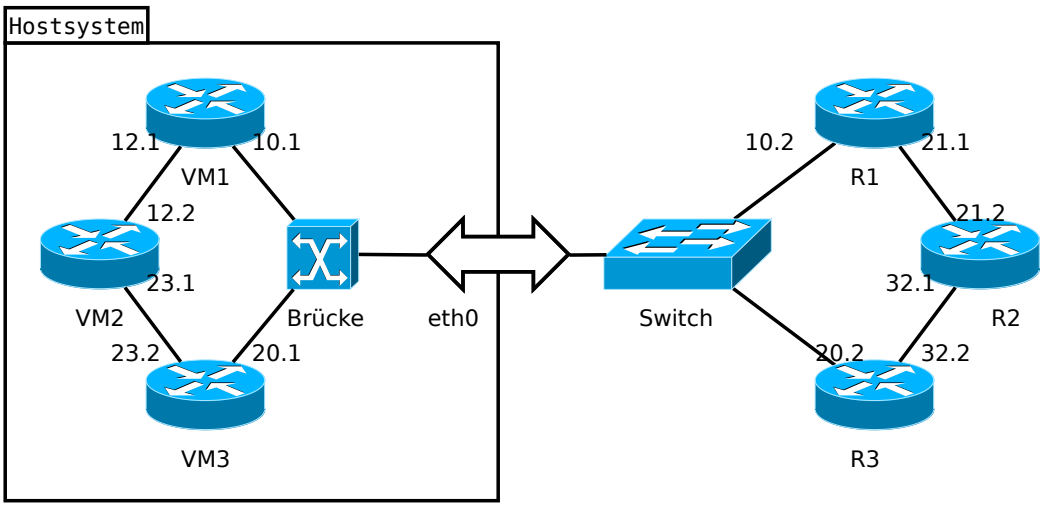

Abbildung 5.5: Szenario 2

kationswege kreuzen sich physikalisch nicht, Routinginformationen werden nur zwischen driekt miteinander verbundenen (virtuellen und realen) Routern ausgetauscht.

Diese Methode der Anbindung externer Router an virtualisierte Netzwerke ist jedoch aufrund der Notwendigkeit von jeweils einer physikalischen Netzwerkschnittstelle im Hostsystem pro Verbindung einer Schnittstelle einer virtuellen Maschine mit einer Schnittstelle eines Routers außerhalb des Hosts schlecht skalierbar.

Die Vortäuschung direkter Ethernetverbindungen zwischen zwei Schnittstellen ist durch getunnelte Layer 2 Netzwerkverbindungen ermöglicht die Errichtung von separater Punkt-zu-Punkt Verbindungen. Szenario 4 (Abbildung 5.7) zeigt zwei durch Tunnel geführte Verbindungen. Für einen Netzwerktunnel können die TAP-Schnittstellen (siehe Abschnitt 5.1) der virtuellen Maschinen genutzt werden, deren aus den Gerätedateien lesbare Datenströme mittels eines Tunnel-Protokolls an das Zielgerät innerhalb des Netzwerks weitergeleitet werden. Auf dem Zielgerät werden die Daten ebenfalls über eine virtuelle Schnittstelle durch das Betriebssystem bereitgestellt. Für die Tunnelverbindung kann grundsätzlich auch das Management-Netzwerk genutzt werden. Eine Kommunikation zwischen verschiedenen Tunnel-Verbindungen durch Multicast-Nachrichten ist nicht möglich, da die getunnelten Verbindungen einen Switch als verbindungsorientiert kom-

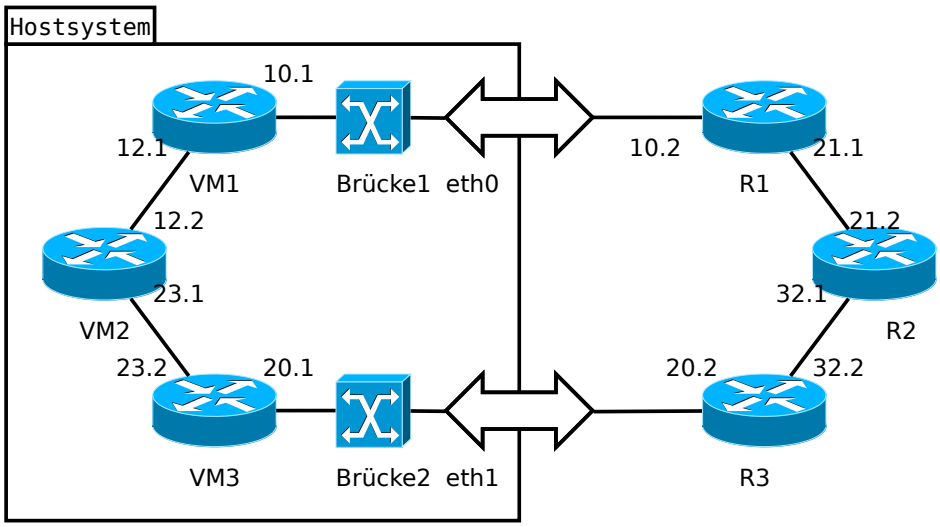

Abbildung 5.6: Szenario 3

munizierte Datenströme durchlaufen. Im Test kam als Verbindung zur Übertragung der Datenströme das verbreitete Protokoll SSH zum Einsatz. Von dem in OpenWrt standardmäßig enthaltenen SSH Daemon *Dropbaer* wird das errichten von Netzwerktunneln nicht unterstützt, für diesen Test muss die unter BSD Lizenz veröffentlichte Implementierung *OpenSSH* verwendet werden. Diese Nutzung von durch Tunnel geführten Layer-2-Verbindungen hat im Vergleich zu der in Szenario 3 (Abbildung 5.6) dargestellten Realisierung separater Verbindungen durch die Nutzung einer physikalischen Netzwerkschnittstelle pro Verbindung zwischen virtuellen Maschinen und realen Routern keine in der Praxis relevante Beschränkung in der Skalierbarkeit, da für einen Tunnel lediglich eine Portnummer reserviert wird. Die Anzahl der auf diese Weise möglichen Verbindungen übersteigt die Anzahl der auf einem Host ausführbaren User-Mode-Linux Kernel weit.

5.4 vnuml2uci

Im Rahmen dieser Arbeit ist das Fern-Konfigurationstool *vnuml2uci* entstanden, das es dem Benutzer ermöglicht, die Netzwerkeinstellungen eines Linksys WRT54GL Routers aus einer VNUML-Beschreibungsdatei zu lesen und auf dem Router einzurichten. *vnuml2uci* ist in Java implementiert und verwendet die Bibliothek *ganymede-*

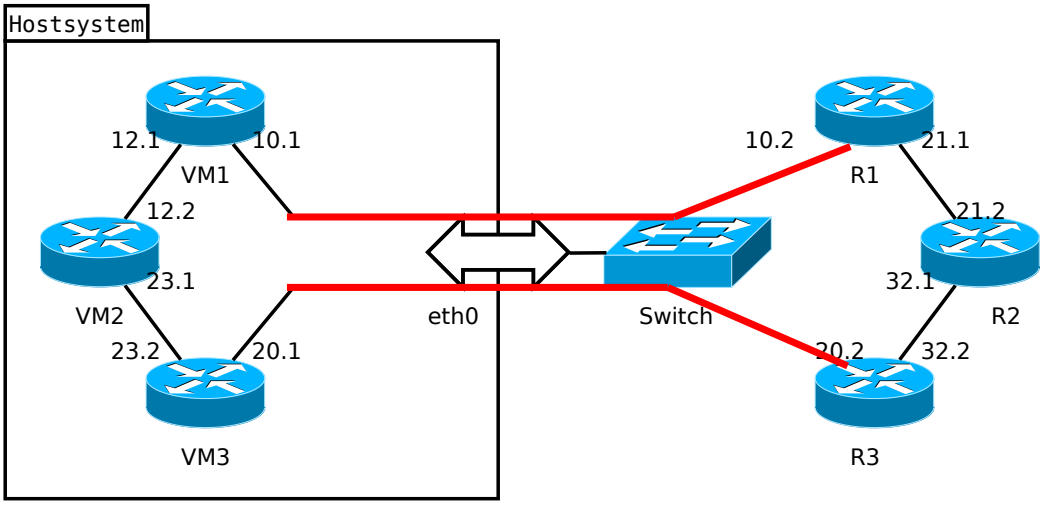

Abbildung 5.7: Szenario 4

SSH2, die ursprünglich an der ETH Zürich entwickelt wurde für den SSH-Zugriff auf den WRT54GL, DOM als Parserumgebung für die XML-basierten VNUML Topologiebeschreibungen, sowie *jargs* zur Interpretation der auf der Kommandozeile übergebenen Argumente. Das Werkzeug ist ausdrücklich nur zum WRT54GL kompatibel, da in anderen Routern die Netzwerkschnittstellen zum Teil anders realisiert sind.

5.4.1 Implementierung

Das Werkzeug *vnuml2uci* ist in Java implementiert. Das Projekt umfasst drei Java-Klassen: *Vnuml2uci*, *DOMvnuml* und *VnumlInterface*. Die Klasse Vnuml2uci stellt

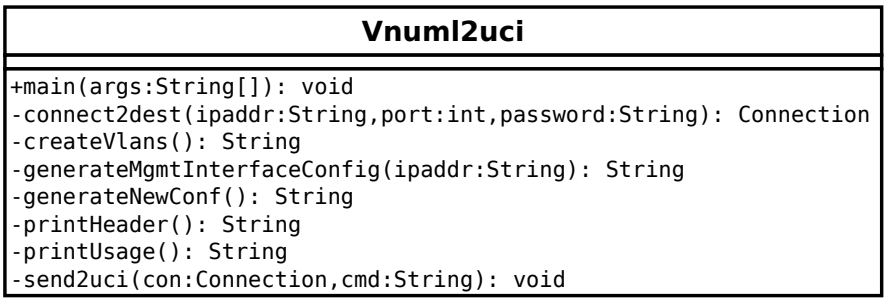

Abbildung 5.8: Diagramm der Klasse Vnuml2uci

neben der *main*-Methode einige Methoden bereit, die zum SSH-Verbindungsaufbau, zum Senden, zum Erzeugen von UCI-Befehlen, zum Ausgeben des Programmnamens sowie einer Hilfestellung zur Angabe der Kommandozeilenparameter.

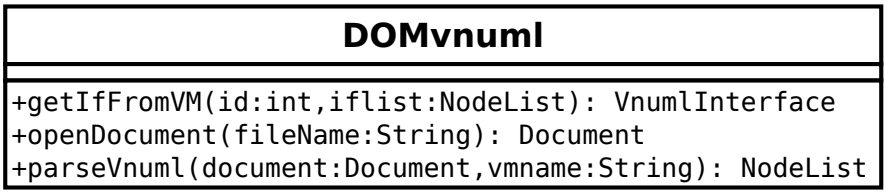

Abbildung 5.9: Diagramm der Klasse DOMvnuml

Die Klasse DOMvnuml stellt die Parserfunktionen für die XML-basierte VNUML-Beschreibungssprache zur Verfügung. Die in dieser Klasse enthaltene Methode getIfFromVM parst die Definition einer Schnittstelle von einer virtuellen Maschine in den Datentyp *VnumlInterface*, openDocument öffnet eine Topologiebeschreibung und gibt sie in From des Typs Document, einer DOM-Klasse zurück. Die Methode parseVnuml gibt eine dem ihr als Zeichenkette übergebenen Namen zuzuordnende virtuelle Maschine als NodeList zurück.

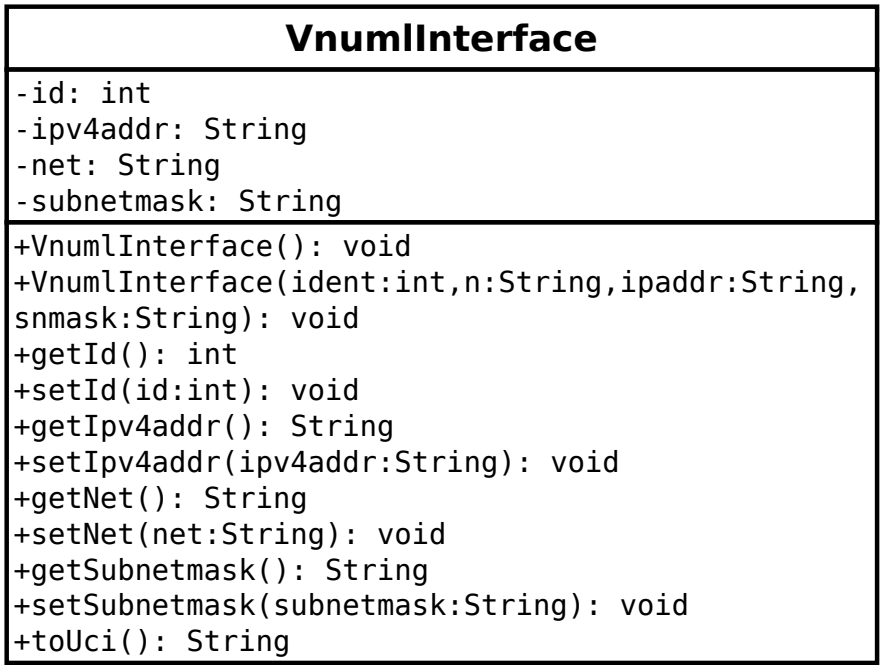

Abbildung 5.10: Diagramm der Klasse VnumlInterface

VnumlInterface ist ein Datentyp, der die Eigenschaften, durch die eine virtuelle Maschine unter VNUML definiert ist in sich aufnimmt. Unter den Methoden steht neben den Getter- und Setter-Methoden die Methode toUci bereit, die die durch die Instanz des Objekts repräsentierte Netzwerkschnittstellenkonfiguration in Form von UCI-Aufrufen als Zeichenkette ausgibt.

5.4.2 Anwendung

Das Java-Programm *vnuml2uci* erzeugt in einer SSH-Verbindung mit einem mit OpenWrt betriebenen WRT54GL eine Netzwerkkonfiguration gemäß der Beschreibung einer Virtuellen Maschine, die in der VNUML-Beschreibungssprache auf dem Hostsystem vorliegt. Obwohl *vnuml2uci* in Java geschrieben ist, ist es nur für den Betrieb auf Linux-Systemen, auf denen VNUML installiert ist, geeignet, da es beim Einlesen der Topologiebeschreibung die angegebene DTD-Datei im Dateisystem sucht, die i. d. R. unter */usr/share/xml/vnuml* abgelegt ist. Auf der Kommandozeile wird *vnuml2uci* mit einigen Argumenten aufgerufen, die im Folgenden kurz beschrieben werden.
Damit virtuelle Maschinen aus VNUML-Topologien eingelesen, geparsed und in Form von UCI-Aufrufen in einer SSH-Sitzung übermittelt werden können, benötigt *vnuml2uci* folgende Informationen:

- den Pfad zu der VNUML Topologiebeschreibung,
- den Namen der virtuellen Maschine, deren Netzwerkkonfiguration auf den Router übertragen werden soll (der Name der VM darf keine Leerzeichen enthalten),
- die IP-Adresse des einzurichtenden Routers,
- den Port, an dem den die SSH-Verbindung vom Router angenommen wird und
- das Passwort des Benutzers *root* auf dem Router.

Der Pfad bzw. Dateiname der VNUML Topologie wird *vnuml2uci* mit dem Parameter „-s [DATEINAME]“ übergeben, alternativ kann auch die lange Form „–source [DATEINAME]“ verwendet werden. Die angegebene Datei muss eine gültige VNUML-Datei sein.

Um *vnuml2uci* mitzuteilen, welche virtuelle Maschine in die Konfiguration des Zielgeräts umgewandelt werden soll, wird der Name der virtuellen Maschine mit dem Parameter „-v [VM-NAME]“ übergeben. Zu beachten ist, dass nur virtuelle Maschinen akzeptiert werden, die über maximal vier Netzwerkschnittstellen verfügen, da auf einem WRT54GL nicht mehr als fünf Schnittstellen definiert werden können und eine durch die Management-Schnittstelle belegt wird. Mit dem Parameter „-d [IP-ADRESSE]“ bzw. „–destination [IP-ADRESSE]“ wird *vnuml2uci* die IP-Adresse des Zielgeräts mitgeteilt.
Der Port, auf dem das Zielgerät SSH-Verbindungen annimmt, wird mit dem Parameter „-p [PORTNUMMER]“ bzw. „–port [PORTNUMMER]“ übermittelt. Wird dieser Parameter nicht angegeben, versucht *vnuml2uci* standardmäßig eine SSH-Verbindung auf Port 22 zu errichten.
Das Root-Passwort, das von *vnuml2uci* für die Errichtung einer SSH-Verbindung zum Zielgerät benötigt, wird *vnuml2uci* optional als Parameter „-r [PASSWORT]“ bzw. „–password [PASSWORT]“ mitgeteilt. Wenn *vnuml2uci* das Root-Passwort des Zielgeräts nicht auf der Kommandozeile übermittelt wird, wird der Benutzer im Programmablauf zur Eingabe des Passworts aufgefordert.
Nach der Ausführung von *vnuml2uci* ist die Konfiguration auf dem Zielgeät eingerichtet, jedoch noch nicht gespeichert und angewendet, damit der Benutzer die vorgenommenen Einstellungen vor dem Übernehmen noch einmal kontrollieren und ggf. modifizieren kann.
Ein gültiger Aufruf auf der Kommandozeile könnte folgendermaßen aussehen:

```
1 $ java -jar vnuml2uci.jar -s scenario.xml -v R3 -d 192.168.0.10 -p 22 -r admin
```

Kapitel 6

Ausblick

Die im Rahmen dieser Arbeit geschehene Portierung des RMTI-Algorithmus auf OpenWrt setzt die Einstiegshürde für die Nutzung des RMTI in realen Netzwerkumgebungen herab, da durch die Bereitstellung von OpenWrt-Paketen keine teure Hardware erforderlich ist, da OpenWrt als Embedded-Linux Umgebung auf vielen preisgünstig erhältlichen Routern lauffähig ist. Die Integration einer Konfigurations- sowie Anzeigekomponente für die Konfigurations- und Logdatei des RMTI ermöglicht die Nutzung ohne tiefe Kenntnis von Linux-Systemen und eignet sich somit auch für Demonstrationszwecke. Um den RMTI noch attraktiver für OpenWrt-Benutzer zu machen bietet sich eine Integration der Konfiguration des RMTI in das zentrale Konfigurationssystem von OpenWrt, UCI an, das wiederum auch das Erstellen einer optisch ansprechenderen Oberfläche für die Konfiguration erleichtert. Aufgrund der während der Arbeit mit OpenWrt gemachten Erfahrungen erscheint mir die Nutzung von OpenWrt-Images als Dateisysteme für VNUML-Maschinen als erstrebenswert, da die in OpenWrt enthaltene Software aufgrund ihrer Orientierung an Embedded-Plattformen sehr geringe Anforderungen an die Hardware stellt und mit UCI und LuCI benutzerfreundliche Konfigurationswerkzeuge enthält, die den Benutzer einen großen Teil der Konfiguration auch ohne Linux-Kenntnisse bewältigen lassen.
Das Benutzen kombinierter Testumgebungen, die die Vorteile beider Ansätze, also die gute Skalierbarkeit auf der einen Seite und die realitätsgemäße Eigenschaften von physikalischen Netzwerkverbindungen auf der anderen Seite für die Analyse von Routingalgorithmen ist aus meiner Sicht ein sehr interessantes Projekt, in dem die Ergebnisse dieser Arbeit weiter genutzt werden können.

Glossar

Bytecode ist ein Hardwareplattform-unabhängiger Zwischen-Code, in den ein Quelltext übersetzt wird. Auf der jeweiligen Zielplattform wird der Bytecode durch einen Bytecode-Interpreter zur Laufzeit in Maschinencode übersetzt. Diese Technik kommt z.B. bei Java, Lua, Python, Perl und dem Microsoft .NET-Framework zum Einsatz. Die Bytecodes dieser genannten verschiedenen Umgebungen sind zueinander inkompatibel.. 11

Daemon ist ein Hintergrunddienst in einem Unixoiden Betriebssystem. Skripte zum Aufruf, Abbruch und Neustart von Daemons stehen in den meisten Linux-Umgebungen im Dateisystem unter */etc/init.d/* bereit.. 4, 13, 17–19, 21, 23, 25, 28, 30, 31, 37, 41, 51, 54

Hop Als Hop bezeichnet man einen Netzwerkknoten, den ein IP-Paket auf seinem Weg durch eine Menge von Netzwerken durchläuft. Da bei der Weitergabe eines Pakets die glsttl dekrementiert wird, lässt sich anhand des TTL-Wertes eines Pakets die von ihm bereits durchlaufene Anzahl von Hops bestimmen.. 40

Imagedatei (engl. für Abbild) ist eine Datei, die andere Dateien enthält. Anders als Archive enthalten Imagedateien nicht nur die Positionen enthaltener Dateien im Dateisystem, sondern auch Informationen über die Position der Daten auf dem Speichermedium bzw. der Partition, mit anderen Worten - das Dateisystem selbst. 5, 7, 8, 31

Makefile Makefiles sind Makros für das Tool Make, das konventionell dafür genutzt wird, die Kompilierung von Software zu steuern.. 5, 6, 8–10, 13, 14, 20, 22, 23, 29, 30

Multicast (auch IP Multicast genannt) ist ein Verfahren, mit dem in der verbindungslosen Netzwerkkommunikation Nachrichten an mehrere Empfänger gleichzeitig versendet werden kann, ohne dass die Nachricht vom Versender mehrfach verschickt werden muss. Der Nutzen dieser Technik liegt vor allem in der effizienten Nutzung der Bandbreite und der Möglichkeit, anders als z.B. beim IP-Broadcast Empfänger-Gruppen festzulegen.. 34, 36, 37, 41, 49, 52, 53

Router ist ein Knoten, der Netzwerke miteinander verbindet, sodass die Kommunikation zwischen Clients verschiedener Netzwerke ermöglicht wird. Anhand der Zieladresse eines am Router eingehenden Paketes und der in seiner *Routing Table* gespeicherten Informationen über die Netzwerktopologie trifft der Router die Entscheidung, an welches Netz das Paket weiterzureichen ist, um sein Ziel auf einem möglichst günstigen Weg zu erreichen. 1, 3–6, 12, 17, 20, 23, 24, 32, 39–43, 46, 48, 51–55, 57, 59

Routingalgorithmus ist ein Algorithmus, der die Selbstorganisation eines Rechnernetzes steuert. Es existieren verscheidene Ansätze, das Routing in einem Netzwerk zu organisieren, die am weitesten verbreiteten sind die Distanz-Vektor-Algorithmen, die Pfad-Vektor-Algorithmen und die Link-State-Algorithmen.. 1, 28, 39, 41

RTT Abkürzung für Round Trip Time (engl. Rundlaufzeit). Bezeichnet die bei der Übertragung eines Datenpaketes von einem Knoten A über einen Knoten B zurück zu Knoten A anfällt. Die RTT ist definiert wie folgt:
RTT = $2 \cdot (\text{PD} + \text{TT} + \text{QD})$, PD steht dabei für Propagation Delay (engl. Ausbreitungsverzögerung), TT steht für Transmit Time (engl. Übertragungszeit) und QD steht für Queue Delay (engl. Warteschlagenverzögerung), die die Summe der Verzögerungen durch das Puffern von Paketen in allesn auf dem Weg zwischen den Knoten A und B liegenden Knoten beschreibt.. 47

SSH Abkürzung für **S**ecure **Sh**ell, bezeichnet eine Protokollfamilie, die einen standardmäßig AES-verschlüsselten Kanal (andere Kryptografische Algorithmen sind ebenfalls einsetzbar) zu einem entfernten System bereitstellt. Die Authentifikation geschieht über ein für diesen Dienst anbietenden Host ausgestelltes RSA-Zertifikat. SSH ermöglicht das Bedienen eines entfern-

ten Hosts über die die Shell oder einen Grafikbildschirm, bietet einen verschlüsselten Kanal zur Dateiübertragung (SFTP). 40, 54–58

Switch (engl. für Schalter) bezeichnet ein Gerät, das Hosts eines Netzwerkes miteinander Verbindet, in dem es eingehenden Traffic gezielt über den Port, an dem der als Ziel angegebene Knoten angeschlossen ist. Dies ist möglich durch die Interpretation der Informationen der Header der eingehenden Rahmen. Switches arbeiten auf der Ethernet-Ebene, in einem IP-Paket enthaltene (Header-) Informationen bleiben also unberührt.. 42, 48

Template (engl. Schablone), ist eine Vorlage für die Darstellung von durch ein serverseitig ausgeführtes Programm generierten Inhalts in einer HTML-Seite. In einigen Fällen ist der Programmcode, durch den der Inhalt generiert wird in Form von Skripten in dem Template enthalten, jedoch wird dies oft als schlechter Stil angesehen.. 13, 15, 17–19

TTL Abkürzung für Time To Live (engl. Lebensdauer). Die TTL beschreibt die Lebensdauer eines IP-Pakets. Wenn ein Paket durch ein auf der Netzwerkschicht operierendes Gerät weitergereicht wird, wird die TTL dekrementiert. Hat die TTL den Wert 0 erreicht, wird das Paket verworfen. Durch ein auf einer niedrigeren Schicht des OSI-Modells arbeitendes Gerät bleibt der TTL-Wert unberührt.. 48

VLAN Abkürzung für **V**irtual **L**ocal **A**rea **N**etwork (engl. virtuelles lokales Netzwerk). VLANs ermöglichen das separieren einzelner Netze auf einem VLAN-fähigen Switch. virtuelle Netzwerke werden auf Switches definiert und bekommen bestimmte Ports des Switches zugewisen, die von der restlichen Kommunikation auf dem Switch getrennt werden, obwohl auf physikalischer Ebene eine Verbindung über den Switch besteht.. 42–45

XML Abkürzung für E**X**tensible **M**arkup **L**anguage (engl. Erweiterbare Auszeichnungssprache) ist eine Sprache, mit der sich Daten strukturiert beschreiben lassen. In der Syntax von XML wird jedes Element von Tags umschlossen, die mit Attributen erweiterbar sind. XML wird auch als Meta-Datenformat bezeichnet, da es als Basis für Datenformate dienen kann.. 47, 48, 55, 56

Literaturverzeichnis

[AP07] Paul Asadoorian and Larry Pesce. *Linksys WRT54G Ultimate Hacking*. Syngress Publishing, Inc. and Elsevier, Inc., Burlington, USA., 2007.

[BL89] Tim Berners-Lee. Hypertext and CERN, März 1989.

[Boh08] Frank Bohdanowicz. Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons. Master's thesis, Universität Koblenz-Landau, Campus Koblenz, Mai 2008. [Online; Stand 02. August 2010].

[CBB05] Kendall Corell, Nick Barendt, and Michael Branicky. Design Considerations for Software Only Implementations of the IEEEE 1588 Precision Time Protocol. 2005. [Online; Stand 22. Juli 2010].

[Gar10] Andreas Garbe. Simulation großer Netzwerke in der VNUML-Umgebung. Master's thesis, Universität Koblenz, September 2010.

[Keu05] Tim Keupen. User-Mode Linux, November 2005. [Online; Stand 13. Juli 2010].

[Kob09] Manuel Kober. Evaluation & Convergence-Analysis of RIP with metric based Topology Investigation . Master's thesis, Universität Koblenz-Landau, Campus Koblenz, August 2009. [Online; Stand 5. Oktober 2010].

[WB04] Hans Weibel and Dominic Béchaz. IEEE 1588 Implementation and Performance of Time Stamping Techniques. Institute of Embedded Systems, Zurich University of Applied Sciences, 2004. [Online; Stand 22. Juli 2010].

[wika] Howto: Write modules. Wiki. Online, Stand 9. November 2010.

[wikb] Reference: Luci modules. Wiki. Online, Stand 9. November 2010.

[wikc] The uci system. OpenWrt-Wiki. Online, Stand 21. Oktober 2010.

[wikd] Writing luci cbi models. Wiki. Online, Stand 9. November 2010.