Universität Koblenz Landau
Abteilung Koblenz
Fachbreich Informatik

Diplomarbeit

Simulation von Interior Gateway Protokollen in virtuellen Netzen

Martin Krechel

Küttiger Str. 8a

56295 Rüber

06.06.2006

Betreuer:
Prof. Dr. Christoph Steigner
Dipl. Inf. Harald Dickel

Inhaltsverzeichnis

1 Über diese Arbeit **4**

2 VNUML - Virtual Network User-Mode-Linux **6**

2.1 Installation . . . 7

2.2 Arbeiten mit VNUML (Arbeitszyklus) . . . 8

2.3 Die VNUML-Sprache . . . 9

2.4 Benutzung des Parsers . . . 19

2.4.1 Problem-Behandlung . . . 22

3 Die Routing Suite Zebra/Quagga **23**

3.1 Architektur . . . 23

3.2 Umgang mit Quagga . . . 24

4 RIP **28**

4.1 Ein erstes einfaches Beispiel . . . 28

4.1.1 Definition und Implementierung des Szenarios . . . 28

4.1.2 Starten der Simulation (Ausführungsphase) . . . 32

4.2 Erweiterung des Beispiels . . . 38

4.3 RIP-Szenario mit Ausfall einer Leitung . . . 46

4.4 Das “Counting To Infinity” Problem . . . 60

5 OSPF **70**

5.1 Beispiel 1: Kaltstart eines OSPF-Szenarios (Initialisierungsphase) . . . 70

5.2 Beispiel 2: OSPF im Fehlerfall (Ausfall einer Leitung) . . . 84

5.3 Hierarchisierung . . . 92

6 Vergleich RIP vs. OSPF 101
6.1 Initialisierungsphase: 101
6.2 Normalbetrieb: 109
6.2.1 Normalbetrieb in einer kleinen Topologie 109
6.2.2 Normalbetrieb in einer größeren Topologie 114
6.3 Im Fehlerfall: 116

7 Nachbemerkungen und Ausblick 122

1 Über diese Arbeit

Ziel der Arbeit ist es, verbreitete Interior-Gateway Routing-Algorithmen anhand virtueller, simulierter Netzwerke vorzustellen.

Die Simulationen werden mit Hilfe von User Mode Linux bzw. VNUML durchgeführt. Hiermit werden die benötigten Router virtuell auf einem physischen Host-Rechner generiert. Auf den virtuellen Routern kommt GNU Zebra/Quagga als Routing-Suite zum Einsatz.

Die Arbeit beschreibt kurz die Installation und Inbetriebnahme der benötigten Software. Darüber hinaus werden durch geeignete Beipiele die Routing Protokolle RIP und OSPF, als praxisnahe Vertreter der Vektor-Distanz sowie Link-State Familien, vorgestellt und verglichen. Die GNU Zebra/Quagga Routing Software ist ein Open-Source Projekt, das nach Entwicklerangaben RIP und OSPF unterstützt. Die erhältliche Dokumentation ist allerdings sehr spärlich ausgeführt, so dass diese Arbeit einem Leser die benötigte Grundlagen zur Konfiguration der Routing-Software vermittelt. Diese Erläuterungen sowie die Vorstellung der Routing-Suite und des Simulators, sollen einen Leser der Arbeit in die Lage versetzen weiterführende eigene Simulationen zu erstellen.

Die Möglichkeit mehrere virtuelle Rechner auf einer gemeinsamen Hardware laufen zu lassen bietet zum einen eine erhöhte Kostenersparnis zum konventionellen Labor mit Hardware-Routern und echten Netzwerkkomponenten. Zum anderen bietet diese Art der Simulation dennoch ein hohes Maß an Praxisbezug, denn es wird keine spezielle Simulationssoftware, in der ein von der Realität abstrahierendes Modell läuft, benutzt, sondern weit verbreitete Standardsoftware.

Diese Arbeit kann keine vollständige Abhandlung über RIP oder OSPF sein. Es wird die Kenntniss der theoretischen Grundlagen zu beiden Algorithmen vorausgesetzt. Wenn diese Vorkenntnisse nicht oder nur teilweise vorhanden sind, empfehle ich die entsprechenden Kapitel aus dem Buch Routing TCP/IP Volume 1 aus dem Ciscopress Verlag [DOY98] oder andere einführende Literatur zum Thema. Erst dann ist es sinnvoll, weiterführende, praktische Simulationen wie sie im Folgenden durchgeführt werden durchzuarbeiten.

Weiterhin beschränken sich die folgenden Beispiele auf Szenarien innerhalb eines autonomen Systems. Redistribution über Grenzen eines autonomen Systems wird nicht behandelt.

Kapitel 2 stellt den VNUML-Parser vor. Nach einem groben Überblick über die Funktionsweise und den Arbeitszyklus zur Benutzung wird die Installation erläutert.

Einen Überblick über die Funktionalitäten sowie die Architektur der Routing-Suite GNU Zebra/Quagga vermittelt Kapitel 3. Danach werden in Kapitel 4 erste praktische Beispiele schrittweise anhand von einfachen RIP Szenarien gezeigt. Um nach den ersten Schritten und praktischen Erfolgen zu etwas komplizierteren Beispielen zu kommen, die Eigenschaften des Routing Information Protocols aufzeigen.

Kapitel 5 behandelt das Link State Protokoll OSPF. In Kapitel 6 werden die beiden Routing-Verfahren anhand mehrere Beispiele verglichen.

2 VNUML - Virtual Network User-Mode-Linux

Das Virtual Network User-Mode-Linux Projekt (VNUML) wurde anlässlich des Euro6IX Forschungsprojekts entwickelt, um IPv6 Szenarien zu simulieren. Prinzipiell kann damit jede Netzwerkapplikation oder jeder Service auf Linux basierenden Systemen auch über mehrere virtuelle Knoten hinweg ablaufen. Es wird sowohl IPv4 als auch IPv6 unterstützt. Unter anderem ist es auch möglich mit Linux-Software-Routern zu arbeiten.

VNUML wurde als Open-Source Projekt realisiert und ermöglicht relativ einfach und schnell virtuelle Netzwerk-Szenarios zu designen. Dabei basiert VNUML auf dem User-Mode-Linux Projekt. Die Benutzung von UML wird aber durch die VNUML-Software vor dem Benutzer versteckt. Dazu teilt sich VNUML in zwei Hauptbestandteile. Zum einen in die beschreibende VNUML-Sprache, mit der die Szenarien modelliert werden, zum anderen in den Interpreter, der die XML-Beschreibung parst und die notwendigen Aktionen ausführt um das Szenario darzustellen. Dabei bootet VNUML die nötige Anzahl an virtuellen Rechnern, in dem Linux Kernels als normale User-Prozesse gestartet werden. Weiterhin werden die Netzwerkschnittstellen der virtuellen Maschinen entsprechend der Szenariobeschreibung eingerichtet und virtuelle Netzwerkverbindungen zwischen den Knoten konfiguriert. Optional können danach auch Applikationen durch den VNUML-Parser gestartet werden und sogar Konfigurationsdateien in die Dateisysteme der künstlichen Rechner kopiert werden. Beim Beenden der Simulation können auf dieselbe Weise Prozesse angehalten werden. Beim Starten des Szenarios werden zusätzliche Managementschnittstellen angelegt, um sich z.B. vom Hostrechner per ssh auf den virtuellen Rechnern einzuloggen.

Exkurs: User-Mode-Linux

User-Mode-Linux wurde ursprünglich entwickelt, um das Debugging bei der Entwicklung von neuen Linuxkernels zu unterstützen. Dies geschieht, indem ein normaler Kernel auf einem Hostrechner als Wirtssystem arbeitet. Auf diesem Host können Linuxkernels als

normale Prozesse im User-Mode gestartet werden, damit ist es möglich ein Betriebssystem als normalen Prozess auf einem anderen Betriebssystem zu starten. Dazu ist es lediglich notwendig, auf den, zur Simulation geplanten herkömmlichen Linux-Kernel einen Patch anzuwenden.

Außer dem gepatchten Kernel und einem Hostrecher wird noch ein Dateisystem, auf dem die virtuellen Rechner operieren, benötigt. Dabei besteht die Möglichkeit, dass mehrere virtuelle Knoten mit ein und dem selben Basisdateisystem arbeiten. Zur Vermeidung von Konflikten zwischen den einzelnen virtuellen Maschinen beim Zugriff auf das Dateisystem gibt es den sogenannten "COW"-Modus (copy on write). In diesem Dateisystemtyp werden alle Änderungen, also schreibender Zugriff auf das Dateisystem, in eine eigene Datei pro virtueller Maschine geschrieben. So dass das ursprüngliche, gemeinsame Dateisystem nicht verändert wird. Beim Lesen wird dann zuerst in der eigenen Datei nach einem Eintrag gesucht, ist dieser nicht vorhanden, wird im gemeinsamen Dateisystem nach einem entsprechenden Eintrag gesucht. Da ein solches Dateisystem im vorliegenden Fall ca. 500 MB groß ist, macht ein solches platzsparendes Verhalten Sinn. Bei Bedarf kann der "COW"-Modus auch abgeschaltet werden, dann ist es allerdings erforderlich jeder virtuellen Maschine eine eigene Kopie des Dateisystems zuzuordnen.

Somit kann mit Hilfe des UML ein bzw. mehrere virtuelle Linux Rechner auch zur Untersuchung von virtuellen Netzwerken genutzt werden, dazu müssen die virtuellen Rechner nur mit virtuellen Netzwerkkomponenten verbunden werden.

2.1 Installation

Die Installation gelingt am einfachsten an einem Rechner mit breitbandigem Internetzugang. Von einem solchen Rechner kann das mitgelieferte Installationsskript ausgeführt werden. Fehlende Software Pakete sollen dann durch das Skript einfach und automatisch heruntergeladen und installiert werden. In einem Netzwerk das durch eine Firewall geschützt wird, kann dies jedoch zu Problemen führen.

Wenn die Installation mittels des bereitgestellten Skriptes nicht funktioniert, muss auf die manuelle Installation zurückgegriffen werden. Die hier benutzte Version 1.5 der VNUML-Software benutzt einige Perl Bibliotheken, die im Fall einer manuellen Installation aufgrund von Abhängigkeiten zuerst eingerichtet werden müssen. Diese Perl-Module werden im einfachsten Fall, soweit sie nicht vorhanden sind von der benutzten Distribution des Hostrechner nachinstalliert. Wenn die Module kein Teil der Distribution sind, können die jeweils aktuellsten Versionen von CPAN (search.CPAN.org) heruntergeladen

werden. Danach müssen die Module einzeln installiert werden. Das Dateisystem für die virtuellen Rechner befindet sich auf der CD, hier sind bereits alle benötigten Komponenten vorinstalliert.

Nach erfolgreicher Installation muss das Dateisystem von der CD in das Dateisystemverzeichnis von VNUML kopiert werden (/usr/local/share/vnuml/filesystems).

Der, für die virtuellen Rechner benötigte, Kernel ist ebenfalls fertig und muss nur in das entsprechende Verzeichnis kopiert werden (/usr/local/share/vnuml/kernels).

Die Beispiele zu dieser Arbeit sind ebenfalls komplett auf der CD enthalten und sollten unter Beibehalt der Verzeichnisstruktur in das Standardverzeichnis
/usr/local/share/vnuml/examples kopiert werden.

Bei Problemen empfiehlt es sich, die auf der CD befindliche Original-Dokumentation der Installation sowie das Tutorial durchzuarbeiten.

Zusätzlich zu den von VNUML benötigten Software-Paketen werden in dieser Arbeit diverse Werkzeuge benutzt, die gegebenenfalls installiert werden müssen. Um die Beispielsimulationen durchführen zu können, sollten folgende Programme auf dem Hostrechner verfügbar sein:

- ping
- traceroute
- tcpdump
- ethereal
- iptables

2.2 Arbeiten mit VNUML (Arbeitszyklus)

Grundsätzlich teilt sich die Arbeit zur Erstellung eines Netzwerk Szenarios mit VNUML in verschiedene Phasen auf. Zuerst muss die Netzwerktopologie geplant werden. In dieser Phase wird das Netzwerk zuerst unabhängig von jeglicher Simulationssoftware z.B. auf einem Blatt Papier erstellt. Die Anzahl der Hosts, der Netzwerke und deren Adressen, sowie die Art der Routingstrategie sollten festgelegt werden.

Nachdem diese Designphase abgeschlossen ist und als Ergebnis eine Skizze der Netzwerktopologie vorliegt, kann zur nächsten Phase, der Implementierungsphase, geschritten werden, d.h. die Netzwerktopologie wird mit Hilfe der VNUML-Sprache modelliert. Dazu wird eine XML Datei erstellt, in der die Tags des VNUML-Sprachumfangs genutzt werden, um das Szenario abzubilden.

Anschließend kann dann in der Ausführungsphase mit Hilfe des VNUML-Parsers die

XML-Beschreibung umgesetzt werden. Der Parser nimmt dann die nötigen Schritte vor um die virtuellen Maschinen zu starten und die Topologie gemäß der Beschreibung zu erstellen. Die Ausführungsphase besteht aus folgenden Teilen: Dem Starten der virtuellen Rechner, dem Starten von Anwendungen auf den virtuellen Maschinen, dem Stoppen der Anwendungen und dem Herunterfahren der virtuellen Rechner. Während eines solchen Zyklus' darf die XML-Beschreibung nicht geändert werden.

2.3 Die VNUML-Sprache

Die Syntax der VNUML-Sprache ist wie in XML üblich in einer Document-Type-Description (DTD) festgelegt. Diese ist standardmäßig auf dem Host Rechner unter /usr/local/share/xml/vnuml/vnuml.dtd zu finden. Der Parser prüft anhand dieser DTD die syntaktische Korrektheit der XML-Beschreibungen. Eine solche Beschreibung baut auf folgendem Grundgerüst auf.

```
[ 1]<?xml version="1.0" encoding="UTF-8"?>
[ 2]<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
[ 3]<vnuml>
[ 4]    <global>
[ 5]        ...
[ 6]    </global>
[ 7]    <net\>
[10]    <vm>
[11]        ...
[12]    </vm>
[13]    <host>
[14]        ...
[15]    </host>
[16]</vnuml>
```

Die Zeilen 1 und 2 legen die XML-Version, den Zeichensatz und die Dokumentendefinition fest. Diese Informationen beziehen sich wie oben erwähnt auf den syntaktischen Aufbau der folgenden XML Tags. Der XML Parser benötigt diese Informationen dringend.

Die dritte Zeile öffnet das `vnuml-Tag`. Nun folgen die eigentlichen XML Informationen, das `vnuml-Tag` wird erst am Ende der Datei als letztes mit Zeile 16 </vnuml> geschlossen. Alle weiteren Tags müssen sich im `vnuml-Tag` befinden.

Die folgenden vier Hauptabschnitte sind die globalen Definitionen `<global>` ... `</global>`, die Beschreibung der virtuellen Netzwerke mit eventuell mehreren `<net/>` Tags, die Festlegung der virtuellen Maschinen mit `<vm>`...`</vm>` Tags und ergänzend die Einbeziehung des Hostrechners in die Simulation.

Globale Elemente: Mit Hilfe des `<global>` ...`</global>` Tags werden die Elemente geklammert, die allgemeingültige Parameter des Szenarios beschreiben. Die nachfolgenden Tags dürfen innerhalb des globalen Bereichs maximal einmal vorkommen (unique).

`<version>` Benötigtes Element, ohne weitere Attribute. Legt die Version der VNUML-Sprache und des Parsers fest.
Beispiel: `<version>1.4<version>`

`<simulation_name>` Benötigtes Element, ohne weitere Attribute. Legt den Namen des Simulationsszenarios fest. Der Name muss ein-eindeutig sein, um Konflikte mit anderen Simulationen zu vermeiden.
Beispiel: `<simulation_name>Basic_Rip</simulation_name>`

`<ssh_key>` Optionales Element, ohne weitere Attribute. Legt die Datei mit dem öffentlichen SSH-Schlüssel des Host-Rechners fest, mit dessen Hilfe der Parser per SSH auf die virtuellen Rechner zugreifen kann, ohne dass der Benutzer andauernd nach dem Passwort gefragt werden muss. Der Schlüssel muss vorher mit ssh-keygen angelegt werden.
Beispiel: `<ssh_key>/root/.ssh/indentity.pub</ssh_key>`

`<automac/>` Optionales, leeres Element, mit dem möglichen Attribut `offset`. Das `<automac/>` Tag ist ein leeres, dh. selbstschließendes Tag daher steht der / direkt vor der schließenden spitzen Klammer. Dieses Tag weist den Parser an die MAC-Adresse der virtuellen Netzwerkschnittstellen selbstständig zu verteilen. Diese sind dann wie folgt aufgebaut: FE:FD:0:z:xy die ersten drei Byte sind Konstanten, z ist der Wert des `Offset`-Attributes (wenn dieses nicht angegeben wird 0), x ist die Nummer der virtuellen Maschine, entsprechend der Reihenfolge in der Konfigurationsdatei, beginnend bei 1. Y ist die Nummer der entsprechenden Netzwerkschnittstelle.
Es ist nur nötig den `Offset` zu benutzen, wenn mehrere Simulationen gleichzeitig laufen, da die MAC-Adressen ein-eindeutig sein müssen.

Die MAC-Adressen können auch manuell gesetzt werden siehe `<if>` und `<mac>`.
Beispiele:
`<automac/>`
`<automac offset="30"/>`

`<ip_offset>` Optionales Element, mit möglichem Attribut `prefix`. Durch `<ip_offset>` wird es ermöglicht, die IPv4 Adressen der Management-Schnittstellen zu verändern. Der `ip_offset` ist standardmäßig `0` und der `prefix` ist standardmäßig 192.168. Daher werden die Managementschnittstellen, wenn sie nicht durch `ip_offset` überschrieben werden bei 192.168.0.1 beginnen.
Nötig ist eine Änderung des `ip_offset` oder des `prefix`, wenn es Konflikte mit realen Netzwerken des Host-Rechners oder anderen Simulationen gibt.
Beispiele:
`<ip_offset>100</ip_offset>`
`<ip_offset prefix="172.17">50</ip_offset>`

`<net_config/>` Optionales, leeres Element, mit den möglichen Attributen `stp` und `promisc`. Durch die Attribute des `<netconfig>` Tag werden Eigenschaften der virtuellen Netzwerkschnittstellen konfiguriert. Das Attribut `stp` (Spanning Tree Protocol) kann die Werte `on` oder `off` annehmen und bewirkt, dass die virtuellen Bridges, die die Netzwerke verbinden den Spannbaum-Algorithmus anwenden. Das promisc-Attribut kann ebenfalls mit `on` oder `off` belegt werden, dadurch werden die Netzwerkschnittstellen in den entsprechenden Modus konfiguriert. Die standardmäßigen Voreinstellungen sind `stp="off"` und `promisc="on"`.
Beispiel: `<net_config stp="on" promisc="on"/>`

`<host_mapping>` Optionales, leeres Element ohne Attribute. Dieses Tag weist den Parser an, die IP-Adressen der Management-Schnittstellen auf den Hostnamen der virtuellen Maschine abzubilden. Dadurch können die virtuellen Rechner vom Host-Rechner mit ihrem Namen angesprochen werden, anstatt mit der IP-Adresse. Dieser Mechanismus wird dadurch ermöglicht, dass der VNUML-Parser auf die `/etc/hosts` Datei des Wirtsrechners zugreift und diese entsprechend abändert. Die automatischen Einträge in `etc/hosts` sollten nicht von Hand abgeändert werden, da es sonst zu Inkonsistenzen kommen kann.
Beispiel: `<host_mapping/>`

`<shell>` Optionales Element, ohne Attribute. Durch das `<shell>` Tag wird der Shell-Interpreter gewählt, der zur Ausführung der vom Parser erzeugten Skripte benutzt wird. Die Standardeinstellung ist `/bin/bash`, diese Standardeinstellung kann allerdings während der Installation des VNUML-Paketes mit `configure --with-shell` geändert werden.
Beispiel: `/bin/sh`

`<tun_device>` Optionales Element, ohne Attribute. Mit diesem Tag kann das sogenannte Tun-Device gewählt werden, das zum Erzeugen der virtuellen Netzwerk-Schnittstellen benötigt wird. Die Voreinstellung ist `/dev/net/tun`.

`<default_filesystem>` Optionales Element, mit dem benötigten Attribut `type`. Durch dieses Tag wird das Standard-Dateisystem für die Simulation festgelegt. Wird innerhalb eines `<vm>`-Tags kein anderes Dateisystem angegeben, wird das hier definierte benutzt. Das Attribut type kann die Werte `copy` oder `cow` annehmen und legt damit die entsprechende Zugriffsart auf das Dateisystem fest .
Die Voreinstellung ist `.../vnuml/filesystems/root_fs_tutorial`.
Beispiel:
`<default_filesystem type=''cow''> /usr/local/share/vnuml/filesystms/myfs`
`</default_filesystem>`

`<default_kernel>` Optionales Element, ohne Attribute. Analog zum `<default_filesystem>` legt dieses Tag den Standardkernel fest. Voreinstellung beim Fehlen dieses Tag ist `.../vnuml/kernels/linux`.
Beispiel:
`<default_kernel> /usr/local/share/vnuml/kernels/linux-2.6.9 </default_kernel>`

`<basedir>` Optionales Element, ohne Attribute. Mit Hilfe dieses Tags kann ein Pfad im Dateisystem des Hosts angegeben werden, unter dem Konfigurationsdateien abgelegt werden. Dieses Tag arbeitet mit dem `<filetree>` Tag zusammen, welches sich relativ zum Teilpfad in `<basedir>` bezieht. Die standardmäßige Voreinstellung ist `/`.
Beispiel: siehe `<filetree>`.

Definition der virtuellen Netze: Nach den globalen Definitionen werden mit Hilfe der `<net/>` Tags die virtuellen Netzwerke festgelegt. Dabei werden allerdings nur die Namen der Netzwerke und die Anzahl der Netzwerke entsprechend der Anzahl der `<net>` Tags

festgelegt. Die Netzwerkadressen und Netzwerkmasken werden später in den Schnittstellen Tags `<if>` definiert. Durch den eindeutigen Namen, der durch das Attribut `name` definiert wird, kann der Parser die zusammengehörenden Netzwerkschnittstellen zu einem Netz gruppieren. Zusätzlich kann festgelegt werden ob die einzelnen Netzwerke mit Bridges oder Switches zusammengeführt werden. Dies geschieht über das Attribut `type` mit dem zwischen `virtual_bridge` oder `uml_switch` gewählt werden kann. Bei der Wahl von `virtual_bridge` sollte beachtet werden, dass durch die Verwendung des Linux-Kommandos `brctl` die Netzwerknamen auf sieben Zeichen begrenzt sind und zum anderen root-Rechte benötigt werden. Mit *brctl* werden die Brücken im Linuxkernel verwaltet.

Beispielhaft würde `<net name=”netz1” type=”uml_switch”/>` ein Netzwerk mit dem Namen netz1 generieren, das mit weiteren Netzwerken, die auch jeweils durch ein `<net/>` Tag beschrieben werden, mit virtuellen Switches verbunden werden. Es ist möglich, ein Netzwerk mit exakt gleicher Konfiguration in mehreren Simulationen gleichzeitig zu verwenden, ansonsten sollten die Netzwerknamen über alle Simulationen, die gleichzeitig laufen, eindeutig gewählt werden.

Wird `type=”virtual_bridge”` gewählt, besteht die Möglichkeit mit dem weiteren Attribut `external` eine Verbindung zu einem realen Netzwerk zu schaffen. Dazu muss der Schnittstellenname dieses realen Interfaces angegeben werden.
Beispiel: `<net name=”realnet” type=”virtual_bridge” external=”eth1”/>`

Beim `uml_switch` Attribut gibt es ebenfalls eine Erweiterung. Es besteht die Möglichkeit den Switch als Hub zu konfigurieren. Dazu muss das Attribut `hub=”yes”` angegeben werden.

Die Standardeinstellungen ist `type=”virtual_bridge”`.
Beispiele:

```
<net name=”net1”/>
<net name=”net2” type=”virtual_bridge”/>
<net name=”net3” type=”uml_switch”/>
<net name=”net4” type=”virtual_bridge” external=”eth1”/>
<net name=”net5” type=”uml_switch” hub=”yes”/>
```

Definition der virtuellen Maschinen: Den dritten Abschnitt in einer VNUML Konfigurationsdatei bilden die Definitionen der virtuellen Maschinen. Je ein `<vm>... </vm>` Tag klammert die Beschreibung eines Rechners.

Attribute eines `<vm>` Tags sind `name` und `order`, wobei `name` ein Pflichtattribut ist.

Mit dem Attribut `order` kann die Reihenfolge angegeben werden, in der der Parser die `<vm>` Tags abarbeitet. Wird keine Reihenfolge angegeben wird die Konfigurationsdatei von oben nach unten abgearbeitet und die Reihenfolge der `<vm>` Tags als implizite Reihenfolge genommen. Wenn es dagegen erwünscht ist, die Rechner in anderer Abfolge zu starten, kann dies mit Hilfe von `order` geschehen.
Beispiel: `<vm name=”rechner1” order=”2”>...</vm>`.

Zur weitern Beschreibung der virtuellen Maschine sind folgende Tags im Sprachumfang von VNUML enthalten:

`<filesystem>` Optionales Element, mit dem Attribut type. Dieses Tag legt das Dateisystem der virtuellen Maschine fest und überschreibt für diesen virtuellen Rechner den Standard, der mit <default_filesystem> angegeben wurde.
Mögliche type-Attribute sind `direct`, `copy`, `cow` oder `hostfs`. Mit `direct` arbeitet die virtuelle Maschine direkt auf dem angegebenen Dateisystem. Durch `copy` wird das angegebene Dateisystem als Master betrachtet, von dem eine Arbeitskopie gemacht wird. `cow` bewirkt, dass nur Änderungen am Master-Dateisystem in ein eigenes Dateisystem geschrieben werden und ansonsten der Master herangezogen wird. Ein solches Verhalten spart sehr viel Speicherplatz und ist daher die zu bevorzugende Alternative.
Beispiel:
`<filesystem type=”cow”> /usr/local/share/vnuml/filesystems/bsp_fs </filesystem>`

`<mem>` Optionales Element, ohne Attribute. Durch dieses Tag kann die Speichergröße des Arbeitsspeichers der virtuellen Maschine festgelegt werden. Die Standardeinstellung ist `32M` (32 Megabyte).
Beispiel: `<mem>64M</mem>`

`<kernel>` Optionales Element, ohne Attribute. Dieses Tag legt den Kernel der virtuellen Maschinen fest. Hier wird ebenfalls die Standardeinstellung des globalen Bereichs außer Kraft gesetzt. Die aktuellen Versionen des Parsers erlauben allerdings nur Kernels, deren Dateiname mit der Zeichenkette “linux” beginnen.
Beispiel: <kernel> /usr/local/share/vnuml/kernels/linux-2.6.9</kernel>

`<boot>` Optionales Element, ohne Attribute. Mit diesem Tag können dem Kernel beim Starten der virtuellen Maschinen Parameter übergeben werden. In der Dokumentation von VNUML wird jedoch ausdrücklich daraufhingewiesen, dass die Parameter nicht überprüft werden, sondern lediglich an den Kernel weitergereicht werden.

`<con0>` Optionales Unterelement, ohne Attribute. Weiterhin ist ein weiteres Tag innerhalb von `<boot>` möglich. Dieses Tag erlaubt es, eine Konsole anzugeben, auf der die Ausgaben des Kernels geschrieben werden, während dieser bootet. Es ist z.B. möglich *xterm* zu diesem Zweck zu nutzen, wenn eine X11 Umgebung zur Verfügung steht.
Beispiel: `<con0>xterm</con0>`

`<mng_if>` Optionales Element, ohne Attribute. Mit <mng_if>no</mng_if> wird der Parser angewiesen, keine Management-Schnittstellen zu erzeugen.

`<if>` Optionales Element, mit den möglichen Attributen `id` und `net`. Mit `<if>` wird eine virtuelle Netzwerk-Schnittstelle konfiguriert. Von `id` wird der Schnittstellenname abgeleitet, indem aus einer `id=”n”` die Schnittstelle `ethn` erzeugt wird. Bei der Vergabe der `id` muss berücksichtigt werden, dass die kleinste mögliche `id` 1 ist, da `eth0` für die Management-Schnittstelle reserviert ist.
Mit Hilfe des Attributs `net` wird die Schnittstelle einem der Netzwerke zugeordnet, die man vorher mit `<net/>` definiert hat.
Beispiel: `<if id=”1” net”net1”> ... </if>`
Innerhalb eines `<if>` Tags sind folgende weitere Elemente möglich:

`<mac>` Optionales Element, ohne Attribute. Durch das `<mac>` Tag kann die MAC-Adresse der Netzwerkschnittstelle manuell gewählt werden. Wenn das globale Element `<automac>` gesetzt wurde ist dies allerdings nicht nötig, da der VNUML-Parser dann automatisch MAC-Adressen generiert. Wenn nur IPv4 Adressen vergeben werden, ist noch nicht einmal das `<automac>` Tag erforderlich, denn dann vergibt das unter VNUML liegende UML die MAC-Identifier automatisch. Die MAC-Adressen müssen in allen Simulationen, die gleichzeitig laufen, eindeutig sein. Bei Konflikten kann es daher nötig sein die MAC-Adresse manuell zu vergeben.

`<ipv4>` Optionales Element, mit dem Attribut `mask`. Die IPv4 Adresse der Netzwerkschnittstelle wird durch dieses Tag definiert. Mit dem optionalen Attribut `mask` kann die Netzwerkmaske festgelegt werden. Standardmäßig ist diese als Klasse C Netzwerk mit 255.255.255.0 vorbelegt.

`<ipv6>` Optionales Element ohne Attribute. Durch dieses Tag wird die IPv6 Adresse der Netzwerkschnittstelle definiert. In diesem Fall gibt es kein Attribut für die Netzwerkmaske, statt dessen wird die IP-Adresse mit Suffix angegeben.

Beispiel:

```
<if id=”1” net=”net1”>
   <mac> FE:AC:12:154:A6:7A </mac>
   <ipv4 mask=”255.255.255.252”>
      192.168.1.1
   </ipv4>
</if>
```

Allgemein ist es durchaus möglich einer Netzwerkschnittstelle mehrere IP-Adressen zu zuordnen.

`<route>` Optionales Element, mit den Attributen `type` und `gw`. Durch das `<route>` Tag lassen sich statische Einträge in die Kernel-Routingtabellen einfügen. Der Parser führt dazu das Linuxkomando `route` aus. Dies kann auch später über die Kommandozeile nachgeholt werden. Über das Attribut `type` muss dabei die Art der IP-Adressierung angegeben werden. Möglich sind `type=”inet”` für IPv4 und `type=”inet6”` für IPv6 Einträge. Das Gateway, an das die Pakete weitergeleitet werden, muss mit dem Attribut `gw` beschrieben werden.
Beispiel: `<route type=”inet” gw=”192.168.0.16>0.0.0.0/0</route>`

`<forwarding/>` Optionales, leeres Tag, mit den Attribut `type`. Durch das `<forwarding>` Tag wird das Weiterleiten der Pakete eingeschaltet. Soll die virtuelle Maschine ein Router sein muss `<forwarding>` aktiviert werden. Das Attribut `type` kann mit `ip`, `ipv4` oder `ipv6` belegt werden. Standardmäßig ist `type` mit `ip` vorbelegt. Durch `type=”ipv4”` wird nur ein Weiterreichen der IPv4 Pakete erreicht. Analog dazu bewirkt `type=”ipv6”` nur ein Weiterleiten der IPv6 Pakete. Mit dem Typ `ip` werden sowohl IPv4 als auch IPv6 Pakete weitergeleitet.
Beispiel: `<forwarding type=”ipv4”/>`

`<filetree>` Optionales Element, mit den Attributen `root` und `when`. Das Tag `<filetree>` ermöglicht das automatisierte Kopieren von Konfigurationsdateien vom Hostrechner in die virtuelle Maschine. Durch das Attribut `when` kann der Zeitpunkt des Kopierens mit `start`, `stop` und `allways` festgelgt werden. `Start` und `stop` beziehen sich jeweils auf das Starten bzw. Anhalten einer Simulation, durch `allways` werden in beiden Fällen Kopien angelegt. Das Attribut `root` beschreibt den Pfad im Dateisystem der virtuellen Maschine, an dem die Kopien eingefügt werden.

Beispiel: `<filetree root="/etc" when="start"> /vnuml/examples/conf/vm1 </filetree>` kopiert den Inhalt des Verzeichnisses `/vnuml/examples/conf/vm1` im Dateisystem des Hostrechners in das Verzeichnis `/etc` in der virtuellen Maschine.
Das Tag `<filetree>` kann mit dem globalen Tag `<basedir>` zusammenarbeiten.
Beispiel:
Im globalen Teil steht `<basedir>/vnuml/examples/conf</basedir>`
Im `<vm>` Teil steht `<filetree root="/etc" when="start"> vm1 </filetree>`.
Die Pfadangabe in `<filetree>` bezieht sich relativ auf den `<basedir>` Pfad.
Wenn für alle virtuellen Maschinen ein eigenes Verzeichnis mit Konfigurationsdateien oberhalb von `<basedir>` angelegt wird, kann so eine Schreibabkürzung erfolgen, weil nicht in jedem `<vm><filetree>` Tag der komplette Pfad angegeben werden muss.
Ab Version 1.5 ist es auch möglich den Namen einer Ausführungssequenz (siehe <exec>-Tag) als Wert des Attributes when anzugeben.

`<start>` Optionales Element, mit dem Attribut type. Mit Hilfe des <start> Tags ist es möglich, beim Starten einer Simulation (Parseroption -s) in der virtuellen Maschine Anwendungen zu starten oder Befehle ausführen zu lassen. Mit type="verbatim" wird der Inhalt des Tags als auszuführender Befehl interpretiert. Dabei muss allerdings beachtet werden, dass XML Sonderzeichen vie <, > oder & nicht im Befehl vorkommen dürfen. Als Alternative lässt sich mit type="file" eine Datei angeben, die Zeile für Zeile als Befehl ausgeführt wird. Diese Datei kann auch ein eigenständiges Skript sein. Der Pfad bezieht sich auf das Dateisystem des Hostrechners und muss absolut angegeben werden.
Eine mehrfache Verwendung des Tags ist möglich.
Beispiele:

```
<start type="verbatim"> zebra -f /etc/zebra.conf -d -P 2601 </start>
<start type="file"> /vnuml/examples/startvm1 </start>
```

Achtung: Das Tag <start> ist ab Version 1.5 des VNUML-Parsers veraltet und sollte nicht mehr genutzt werden. Siehe <exec>.

`<stop>` Optionales Element, mit dem Attribut `type`. Analog zum `<start>` Tag gibt es im Sprachumfang der VNUML-Beschreibungssprache auch ein `<stop>` Tag. Die damit angegebenen Befehle werden beim Anhalten einer Simulation ausgeführt (Parseroption -p).

Beispiele:

```
<stop type=”verbatim”> killall zebra</stop>
<stop type=”file”> /vnuml/examples/stopvm1 </stop>
```

Achtung: Das Tag <stop> ist ab Version 1.5 des VNUML-Parsers veraltet und sollte nicht mehr genutzt werden. Siehe <exec>.

`<exec>` Optionales Element, mit dem Attribut `type` und `seq`. Ersetzt die Tags `<start>` und `<stop>`. Das Attribut type kann analog zu `<start>` oder `<stop>` die Werte “`verbatim`” oder “`file`” annehmen.
Mit Hilfe des `<exec>`-Tags können nun beliebig viele Ausführungssequenzen definiert werden, deren Name durch das Attribut `seq` angegeben wird.

```
<exec seq=”start” type=”verbatim”> zebra -f /etc/zebra.conf -d -P 2601
</exec>
```

Diese Ausführungssequenzen werden über den eigenen Schalter -x ausgeführt. Um den zebra-Daemon mittels der obenstehenden Ausführungssequenz zu starten muss der Parser mit folgenden Optionen aufgerufen werden:
`vnumlparser-pl -x start@abc.xml` .

Definition des Hosts: Wenn der Hostrechner als aktiver Teil an der Simulation teilnehmen soll, muss er ähnlich den virtuellen Maschinen konfiguriert werden. Dazu existieren folgende Sprachelemente:

`<hostif>` Optionales Element, mit dem Attribut `net`. Das `<hostif>` Tag besitzt viele Ähnlichkeiten zum `<if>` Tag und dient dazu eine Netzwerkschnittstelle zu konfigurieren. Das Attribut `id` entfällt hier. Stattdessen wird die Schnittstelle nach dem Netzwerk, das durch das Attribut net angegeben wird benannt.
Mit den Unterelementen `<ipv4`> oder `<ipv6>` wird die IP-Adresse der Netzwerkschnittstelle angegeben, diese sind analog zum `<if>` Tag zu verwenden. Die Benutzung von `<mac>` ist nicht möglich. Wenn eine echte Netzwerkkarte auch eine Verbindung zu einem realen Netzwerk schaffen soll, muss dies auch vorher im `<net>` Tag mit dem Attribut `external` angegeben werden.

`<physicalif>` Optionales Element, mit den Attributen `type`, `name`, `ip`, `mask` und `gw`. Damit beim Beenden einer Simulation wieder eine sinnvolle Konfiguration der realen Schnittstelle erfolgen kann, gibt es das `<physicalif>` Tag. Dieses definiert

den Zustand der Schnittstelle, die beim Beenden der Simulation wieder hergestellt wird. Über das Attribut `type` kann zwischen `ipv4` und `ipv6` unterschieden werden. Mit `name` wird der ursprüngliche Schnittstellenname restauriert. Das Attribut `ip` wird mit der IP-Adresse belegt, wobei beachtet werden sollte, dass bei IPv6 Adressen die Netzwerkmaske mit Suffix angegeben wird und bei IPv4 Adressen diese in dezimaler Schreibweise im Attribut `mask` angegeben werden muss. Das Standard Gateway kann mit `gw` wiederhergestellt werden.

2.4 Benutzung des Parsers

Wie bereits zu Beginn dieses Kapitels erläutert, durchläuft ein Szenario verschiedene Abschnitte, die Designphase, die Implementierungphase und schließlich die Ausführungsphase. Die erste Phase ist dabei unabhängig von VNUML und widmet sich nur dem Erdenken des Sezenarios. In der Implementierungphase steht die VNUML Beschreibungssprache, mit den im vorhergehenden Abschnitt erläuterten Sprachelementen, im Mittelpunkt. Der VNUML-Parser, also das eigentliche Werkzeug kommt erst in der Ausführungsphase zum Einsatz. Der folgende Abschnitt erklärt den Aufbau, die Funktion und die Benutzung des Parsers.

Die eigentliche Software des VNUML besteht aus dem ca. 4400 Zeilen großen Perl-Skript `vnumlparser.pl`. Dies leistet mehr als nur das reine parsen der XML-Dateien, vielmehr führt es alle Aktionen aus, um aus einer wohlgeformte XML-Beschreibung die entsprechende Topologie zu erstellen. Dabei werden vom Parser Linux-Kommandos und die UML-Software ausgeführt.

Die Autoren von VNUML reden im Zusammenhang mit der Benutzung des Parsers von einem Simulations Lebenszyklus. Dieser Zyklus besteht aus den Schritten

1. Erstellen des Szenarios
2. Starten von Prozessen und Anwendungen
3. Stoppen von Prozessen und Anwendungen
4. Beenden des Szenarios

Wobei die Schritte 2 und 3 beliebig iteriert werden können.

Um einen dieser Schritte ausführen zu können, muss der VNUML-Parser jeweils einmal aufgerufen werden. Um sicherzustellen, dass immer nur ein Schritt nach dem anderen

ausgeführt wird, dürfen nicht mehrere Instanzen des Parsers aktiv sein. Daher wird beim Ausführen des Parsers in /var/vnuml eine Datei LOCK angelegt, die erst gelöscht wird, wenn der Parser die Ausführung beendet.

vnumlparser -MODE FILENAME [-(OPTIONS)*]

Allgemein sieht ein Parseraufruf wie oben beschrieben aus. Um nähere Informationen zum Parser und den verfügbaren Schaltern zu bekommen, gibt es zwei Pseudomodi -V und -H, die ohne andere Parameter an den vnuml Aufruf angeknüft werden. Durch den Schalter -V zeigt der Parser seine Versionsnummer und beendet sofort seine Ausführung. Der Schalter -H zeigt eine Hilfe im Linux-Stil, in dem alle Modi und Optionen kurz erläutert werden, sowie knappe Hinweise zur Benutzung gegeben werden.

Der folgende Abschnitt geht auf die wichtigsten Parameter, die zur Benutzung des Simulators notwendig sind, ein.
Als Ausführungsmodus stehen folgende Schalter zur Auswahl:

`-t FILENAME` : Die in `FILENAME` beschriebene Topologie wird kreiert, d.h. die virtuellen Rechner werden erstellt und gestartet. Dieser Bootvorgang kann je nach Rechnerleistung des Hosts einige Minuten dauern. Nach erfolgreichem Starten der Rechner sind bereits alle virtuellen Netzwerkschnittstellen vorhanden und konfiguriert. Per ping können erste Verbindungstests erfolgen. Der SSH-Daemon ist ebenfalls auf jeder virtuellen Maschine gestartet und kann ab sofort benutzt werden. Da es allerdings schwer zu sagen ist, wann alle Rechner erfolgreich gestartet wurden, ist es angeraten, ergänzend die Option -B zu benutzen, die für ein Blockieren des Parsers sorgt, bis alle virtuellen Maschinen einsatzbereit sind. Dadurch ist es dem Benutzer möglich mit der Gewissheit weiter zu arbeiten, dass die virtuellen Machinen voll verfügbar sind.

`-s FILENAME` : Die in `FILENAME` beschriebene Simulation wird gestartet. Um diesen Modus zu nutzen, muss die Topologie bereits mit `-t` kreiert worden sein. Das Starten der Simualtion bezieht sich auf die Anwendungen und Netzwerkservices die vom Parser automatisiert aufgerufen werden, d.h. alle `<start>` Tags mit dem Attribut `when=”start”` und `when=”allways”` werden nun ausgeführt.
Achtung: Diese Option sollte ab Version 1.5 nicht mehr benutzt werden.

`-p FILENAME` : Die in `FILENAME` beschriebene Simulation wird gestoppt. Der Aufruf dieses Modus' ist nur möglich, wenn die Topologie zuvor mit -t kreiert wurde.

Analog zum Modus -s werden nun alle `<stop type=”...”>` Tags ausgeführt.
Achtung: Diese Option sollte ab Version 1.5 nicht mehr benutzt werden.

`-r FILENAME` : Die in `FILENAME` beschriebene Simulation wird neu gestartet (restart). Durch diesen Modus wird die Simulation gestoppt und anschließend neugestartet. Der Schalter `-r` bietet somit nur eine Schreibabkürzung zu zwei Aufrufen mit den Modi `-p` und `-s`.
Achtung: Diese Option sollte ab Version 1.5 nicht mehr benutzt werden.

`-d FILENAME` : Die in `FILENAME` beschriebene Topologie wird zerstört. Auch dieser Modus ist nur verfügbar, wenn die Topologie bereits mit -t gestartet wurde. Durch diesen Schalter werden die virtuellen Maschinen heruntergefahren.

`-P FILENAME` : Die in `FILENAME` beschriebene Topologie wird bereinigt. Wenn sich eine Simulation nicht mehr beenden lässt und die virtuellen Maschinen nicht mehr reagieren sollten, kann dieser Modus genutzt werden, um die Prozesse der virtuellen Rechner zwangsweise zu beenden. Weiterhin werden in diesem Modus die kopierten Dateisysteme bzw. die COW-Dateien gelöscht. Dabei sollte beachtet werden, dass alle virtuellen Maschinen abrupt beendet werden, also auch die anderen Simulationen. Daher sollte sichergestellt werden, dass alle einwandfrei funktionierenden Szenarien vorher sauber heruntergefahren werden.

`-x seqName@FILENAME` : Die Ausführungssequenz mit dem Namen seqName aus der XML-Beschreibung FILENAME wird ausgeführt. Diese Option ist erst ab Version 1.5 verfügbar und ersetzt die Optionen -s -p.

Als nützliche Optionen haben sich in der Praxis folgende Schalter bewährt:

`-v` : Diese Option veranlasst den Parser umfangreichere Meldungen auszugeben. Besonders im Fehlerfall kann diese Option helfen, den Grund oder den Ort der Fehlerursache ausfindig zu machen. Die Aktionen die der Parser, z.B. beim Hochfahren eines Szenarios durchführt, können durch die wortreicheren Ausgaben besser nachvollzogen werden.

`-B` : Diese Option ist besonders beim Hochfahren eines Szenarios relevant, da sie ein Blockieren des Parsers zur Folge hat, die erst nach erfolgreichem Start aller virtuellen Rechner aufgehoben wird. Es wird dringend empfohlen diese Option beim Starten zu benutzen.

`-M` : Diese Option funktioniert nur beim Starten, Stoppen oder Neustarten der Simulation (Modi `-s`, `-p` und `-r`) und erlaubt es, eine Liste von virtuellen Rechnern anzugeben. Nur auf diese speziell ausgewählten Rechner wird der entsprechende Aufruf angewendet. Somit ist hiermit ein Starten, Stoppen oder Neustarten von einzelnen Rechnern bzw. Gruppen möglich. Die virtuellen Rechner werden durch Komma getrennt und mit ihrem Namen angegeben.

Die hier aufgeführten Optionen sind die wichtigsten für den praktischen Gebrauch von VNUML. Weitere Optionen sind bei Interesse der Hilfe des Parsers zu entnehmen.

2.4.1 Problem-Behandlung

Es kann vorkommen, dass sich Szenarien nicht mehr beenden lassen. Wenn der Aufruf

VNUML Parser.pl -d FILENAME

nicht die gewünschte Wirkung erzielt, kann man ein Herunterfahren der virtuellen Maschinen mit der zusätzlichen Option -F erzwingen. Diese Option sollte allerdings nur im Notfall benutzt werden, da das Dateisystem dabei beschädigt werden kann.

VNUML Parser.pl -d FILENAME -F

Nachdem eine Simulation in dieser Art beendet wurde, kann es sein, dass anschließende Startversuche nicht mehr erfolgreich sind. In diesem Fall ist sehr wahrscheinlich das Dateisystem nicht mehr intakt. Die einzige Lösung besteht darin die Kopien der Dateisysteme bzw. die COW-Dateien zu löschen. Dadurch gehen alle darin persistent gespeicherten Daten verloren, aber die Simulation lässt sich wieder starten. Eine solche Bereinigung des Szenarios wird mit dem Modus -P durchgeführt (siehe 2.4). Sollte diese Bereinigung nicht erfolgreich sein, sollte überprüft werden ob alle COW-Dateien in /var/vnuml/SIMULATIONSNAME gelöscht wurden. Gegebenenfalls müssen diese dann manuell gelöscht werden. Weiterhin sollte überprüft werden, ob alle virtuellen Rechner, also die gestarteten Kernel auch wirklich beendet wurden. Wenn dies nicht der Fall ist, sollten die entsprechenden Prozesse ebenfalls manuell beendet werden.

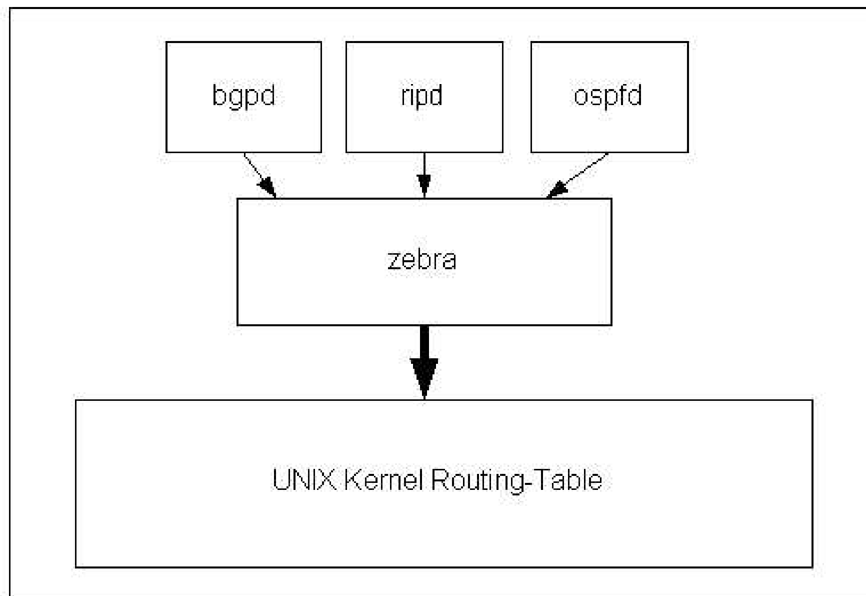
3 Die Routing Suite Zebra/Quagga

Das Software Paket GNU Zebra wurde unter der Leitung von Kunihiro Ishiguro entwickelt um für Unix basierende Systeme eine umfassende Routing-Suite zu schaffen. Das Zebra Projekt wurde dann unter dem neuen Namen Quagga, wieder unter GNU General Public License, weiterentwickelt. Die aktuelle Version von Quagga unterstützt nach Angaben der Entwickler RIPv1, RIPv2, OSPFv2, OSPFv3, BGP-4 und BGP-4+. Es wird sowohl IPv4 als auch IPv6 unterstützt. Das Routing Projekt wurde ins Leben gerufen um neben dem klassischen Routing Werkzeugen unter Unix wie `route`, `ifconfig` und `netstat` eine Alternative für das Weiterleiten mit dynamische Protokollen zu schaffen.

Im Rahmen dieser Arbeit wird die Quagga-Suite auf mehreren virtuellen Rechnern gleichzeitig eingesetzt, um künstliche Netzwerke zu simulieren. Die virtuellen Netze werden sich, in dieser Arbeit auf die Interior-Gateway Protokolle beschränken. Somit kommen nur die RIP und OSPF Protokolle der Quagga Software zum Einsatz.

3.1 Architektur

Die Architektur des Software Paketes läuft funktional verteilt über mehrere Prozesse. Der Hauptprozess ist der Zebra-Daemon, dieser ist der Managementprozess, der die einzelnen Routingprozesse untereinander und mit dem Betriebssystem verbindet. Neben diesem zentralen Prozess laufen dann die jeweiligen Implementationen der Routingalgorithmen als eigenständige Prozesse. Die Kommunikation untereinander, sowie mit dem Kernel, wird mit einem eigenen Protokoll über den Zebra-Daemon abgewickelt. Der modulare Aufbau soll die Erweiterung um neue Protokolle verbessern, da in diesem Fall "nur" ein entsprechender Prozess implementiert und gestartet werden muss. Ein weiterer Vorteil soll nach Angaben der Entwickler eine bessere Wartbarkeit und erhöhte Stabilität sein. Das Anlegen von statischen Routing-Einträgen kann ebenfalls über den Zebra-Daemon geschehen.



3.2 Umgang mit Quagga

Quagga kommt im Rahmen dieser Arbeit als Routingsoftware auf virtuellen Rechnern zum Einsatz, daher müssen die entsprechenden Kernels bereits hochgefahren sein, ehe Quagga gestartet werden kann. Im Dateisystem, mit dem die virtuellen Maschinen arbeiten, ist die Installation von Quagga bereits vorgenommen. Daher wird im weiteren eine korrekte Installation vorausgesetzt und nur die Inbetriebnahme und Nutzung der Software erläutert.

Um Quagga zu nutzen, muss als erstes der zentrale Zebra-Daemon gestartet werden. Dies geschieht über die Kommandozeile der jeweiligen virtuellen Maschine, mit folgendem Aufruf:

```
zebra -f FILE -d -P PORT
```

Mit der Option `-f FILE` kann der Pfad zu einer Konfigurationsdatei angegeben werden, wird die Option `-f` nicht angegeben, versucht die hier benutzte Installation von Zebra die Standardkonfiguration in /usr/local/etc/quagga/zebra.conf zu finden. Kann Zebra keine Konfigurationsdatei finden, wird der Prozess zwar gestartet, es kann aber nicht über die Konsole auf die Konfigurationsschnittstelle zugegriffen werden. Somit kann keine Konfiguration des Zebra-Daemons stattfinden. Die Option -d verursacht, dass Zebra als Daemon gestartet wird. Mit der Option -P PORT wird der Port festgelegt, auf der die Konfigurationsschnittstelle erreicht werden kann.

Der Aufruf von `zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601` sollte auf den virtuellen Maschinen der beigefügten Beispiele immer erfolgreich sein. Die angegebene Konfigurationsdatei ist nur ein rudimentäres Gerüst, das aber zum Starten und Testen der Installation ausreichend ist.
In ähnlicher Weise werden die Routingprozesse mit analogen Optionen gestartet:

```
ripd -f /usr/local/etc/quagga/ripd.conf.sample -d -P 2602
ospfd -f /usr/local/etc/quagga/ospfd.conf.sample -d -P 2603
```

Die obenstehenden Aufrufe sollten bei allen Beispielen dieser Arbeit die entsprechenden Prozesse in den virtuellen Maschinen mit rudimentärer Konfiguration starten. Die manuelle Konfiguration kann danach über die Kommandozeile erfolgen. Dazu besitzt jeder Quagga-Daemon eine eigene Terminalschnittstelle die sogenannte VTY. Ein Benutzer, der auf der dazugehörigen virtuellen Maschine eingeloggt ist, kann eine Verbindung zur VTY öffnen. Um beispielsweise den RIP-Daemon zu konfigurieren, der nach obenstehender Anleitung gestartet wurde, muss mit

```
telnet localhost 2602
```

eine Verbindung zum RIP-Daemon aufgebaut werden. Daraufhin wird ripd mit

```
Hello this is Quagga (version 0.96.4).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verfication

Password:
```

antworten.

Im Konzept von Quagga gibt es zwei unterschiedliche Arten von Benutzern. Zum einen den normalen oder unprivilegierten Benutzer, zum anderen den privilegierten Benutzer. Das erste nun geforderte Passwort ist das normale Benutzerpasswort. Ein solcher unprivilegierter Benutzer darf keine Veränderungen an der Konfiguration des Routers vornehmen. Es ist ihm nur gestattet Informationen nachzufragen. So kann ein normaler Benutzer Einsicht in die Weiterleitungstabelle nehmen oder den Status des Routers einsehen.

Das Standardpasswort ist in den oben angegebenen Konfigurationsdateien `zebra`. Nach erfolgreichem Login zeigt die Terminalschnittstelle die Eingabeaufforderung:

```
ripd>
```

Um vom unprivilegierten Modus in den priveligierten Modus zu wechseln, gibt es den Befehl `enable`. Wenn in der Konfigurationsdatei ein Passwort für den privilegierten Modus vergeben wurde, wird dieses abgefragt. Nach erfolgreichem Wechsel in den privilegierten Modus, zeigt die Terminalschnittstelle folgendes Prompt:

```
ripd#
```

In beiden Modi wird die Eingabe von Befehlen durch einige Komfortfunktionen unterstützt. Nach betätigen der <TAB>-Taste wird versucht, die Eingabe automatisch zu vervollständigen. Die <?>-Taste bewirkt eine kontextsensitive Auflistung der möglichen Befehle bzw. der weiteren Parameter eines Befehlsaufrufs.

```
ripd# s<TAB>
```

So wird beispielsweise die obenstehende Eingabe zu

```
ripd# show
```

vervollständigt. Drückt man nun die <?>-Taste bekommt man Auskunft über weitere Parameter:

```
ripd# show
  debugging          Debugging functions (see also 'undebug')
  history            Display the session command history
  ip                 IP information
  ipv6               IPv6 information
  memory             Memory statistics
  running-config     running configuration
  startup-config     Contentes of startup configuration
  thread Thread      information
  version            Displays zebra version
```

Ergänzt man nun weiter

```
ripd# show r<TAB>
```

vervollständigt die Terminalschnittstelle die Eingabe zu:

```
ripd# show running-config
```

Dieser nun vervollständigte Befehl kann mit der <ENTER>-Taste bestätigt werden. Als Antwort liefert Quagga die aktuelle Konfiguration, die in diesem Beispiel wie folgt aussehen sollte.

```
Current configuration:
!
hostname ripd
password zebra
log stdout
!
router rip
!
line vty
!
end
```

Die Zeilen die mit einem `!` beginnen, werden als Kommentar interpretiert. Die Ausgabe der aktuellen Konfiguration ist nur den privilegierten Benutzern gestattet.

Die Konfiguration anderer Routingalgorithmen erfolgt nach dem selben Muster. Zuerst muss eine Verbindung zur VTY geschaffen werden. Der Port, auf dem die Schnittstelle auf Eingaben horcht, wird wie oben beschrieben beim Starten des Prozesses festgelegt. Danach folgt die Nutzer-Verifikation. Abhängig vom Quagga-Prozesstyp können danach entsprechende Befehle zur Konfiguration eingegeben werden. Die Änderungen an der laufenden Konfiguration werden sofort wirksam. Allerdings werden die Änderungen nicht automatisch in der Konfigurationsdatei gespeichert.

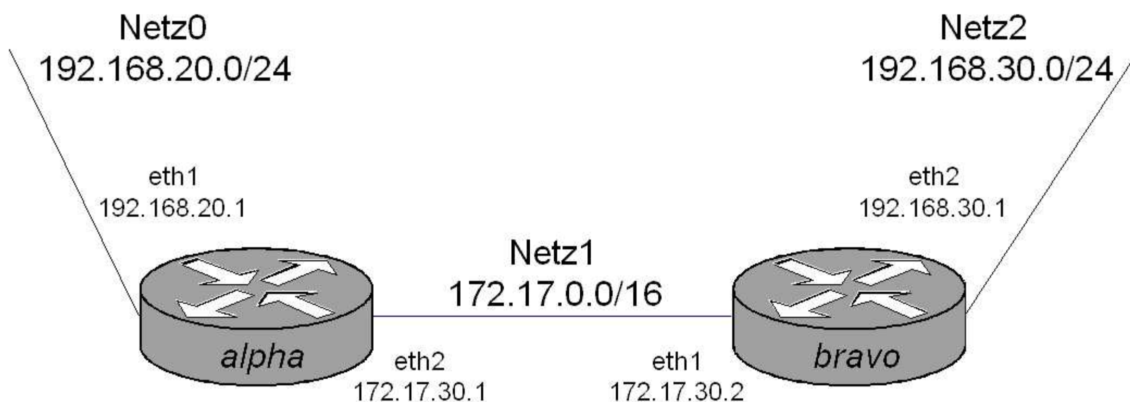
4 RIP

Im anschließenden ersten praktischen Teil dieser Arbeit, werden verschiedene Topologien mit Hilfe von VNUML aufgebaut. Die Erreichbarkeiten über mehrere Netzwerke hinweg werden unter Verwendung der RIP-Implementation von Quagga gewährleistet. Mit dieser Toolchain werden mehrere Beispiele vorgestellt, die in ihrem Schwierigkeitsgrad bzw. ihrer Komplexität stetig ansteigen. Die ersten Beispiele dienen eher dem langsamen Einführen in die Abläufe, die sich auch in den späteren Experimenten ständig wiederfinden lassen. Weiterhin werden aber auch einige wichtige Kernaspekte des RIP-Algorithmus' gezeigt, darunter die Reaktion von RIP auf unvorhergesehene Ereignisse, sowie das für RIP typische Counting to Infinity Problem.

4.1 Ein erstes einfaches Beispiel

4.1.1 Definition und Implementierung des Szenarios

Um VNUML vorzustellen wird ein sehr einfaches Beispiel simuliert. Die Topologie besteht lediglich aus den beiden RIP-Routern alpha und bravo. Jeder der Router ist für ein kleines Netzwerk verantwortlich und soll die Erreichbarkeit dessen sichern. Die beiden Router sind über das dritte Netzwerk 172.17.0.0/16 miteinander verbunden.



VNUML benötigt lediglich eine XML Beschreibung, die dieses Szenario beschreibt.

Aus dieser Beschreibung werden dann die entsprechenden virtuellen Rechner generiert. Um eine virtuelle Maschine zu generieren, benötigt die Software minimal einen Rechner-Namen und Informationen zu den Netzwerkschnittstellen.

Im folgenden wird die XML Beschreibung Schritt für Schritt vorgestellt:

```
[ 1] <?xml version="1.0" encoding="UTF-8"?>
[ 2] <!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
```

Die Zeilen 1 und 2 legen die Dokumentendefinition und die XML-Version fest. Diese Informationen beziehen sich auf den syntaktischen Aufbau der folgenden XML Tags. Der XML Parser benötigt diese Informationen dringend.

```
[ 3] <vnuml>
```

Die dritte Zeile öffnet das `vnuml-Tag`. Nun folgen die eigentlichen XML Informationen. Das `vnuml-Tag` wird erst am Ende der Datei, als letztes, geschlossen. Alle weiteren Tags müssen sich im `vnuml-Tag` befinden.

```
[ 4]     <global>
[ 5]         <version>1.5.0</version>
[ 6]         <simulation_name>rip1</simulation_name>
[ 7]         <ssh_key version="1">/root/.ssh/identity3.pub</ssh_key>
[ 8]         <automac/>
[ 9]         <ip_offset>100</ip_offset>
[10]         <host_mapping/>
[11]         <shell>/bin/sh</shell>
[12]     </global>
```

Die Zeilen 4 bis 12 beschreiben den globalen Teil des Szenarios. Hier stehen allgemeine Informationen für den Simulator. In Zeile 5 wird die Version des VNUML Parsers mit 1.5.0 festgelegt. Frühere Versionen benutzen eventuell andere Attribute. Der Name des Szenarios wird in Zeile 6 angegeben. Dieser Name ist entscheidend für die später angelegten COW-Files der einzelnen virtuellen Maschinen. Jedes Szenario sollte daher einen eigenen Namen bekommen, um Probleme mit den Dateisystemen zu vermeiden. Zeile 7 gibt den Ort des öffentlichen RSA Schlüssels an. Das automac-Tag (Zeile 8) weist den VNUML Parser, an die MAC-Adressen der virtuellen Netzwerkschnittstellen eigenmächtig zu verteilen. Zeile 9 legt den IP-Offset der Managment Schnittstellen fest. Die Management-Schnittstellen sind kein echter Teil des Simulation-Szenarios, sondern

dienen dem Nutzer dazu, sich vom Host per ssh auf den virtuellen Maschinen einzuloggen. Wenn es Konflikte mit echten Netzwerkkarten des Host-Rechners gibt, nutzt man den Offset, um diese Probleme zu beseitigen. Die IP-Adressen der Management-Schnittstellen werden dann anders berechnet. Wenn kein Offset angegeben wird, beginnen die IP-Adressen der Management Schnittstellen bei 192.168.0.1.
Zeile 10 verursacht, dass die Namen der virtuellen Maschinen statt der IP-Adressen zum ansprechen der Management-Schnittstellen benutzt werden können. Der VNUML Parser verändert dann die /etc/hosts Datei des Host-Rechners entsprechend.
Der in Zeile 11 angegebene Interpreter wird benutzt, um Skripts des VNUML Parsers abzuarbeiten.
Schließlich wird in Zeile 12 der globale Teil abgeschlossen.

```
[13]      <net name="Netz0" />
[14]      <net name="Netz1" />
[15]      <net name="Netz2" />
```

Zeile 13 bis 15 teilen dem Parser mit, das insgesamt 3 Netzwerke simuliert werden. Damit die Netzwerk-Schnittstellen den Netzen zugeordnet werden können, benötigt jedes Netz einen eindeutigen Bezeichner.

```
[16]      <vm name="alpha">
[17]         <filesystem type="cow">
/usr/local/share/vnuml/filesystems/root_fs_tutorial </filesytem>
[18]          <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
[19]          <boot><con0>xterm</con0></boot>
[20]          <if id="1" net="Netz0">
[21]              <ipv4 mask="255.255.255.0">192.168.20.1</ipv4>
[22]          </if>
[23]          <if id="2" net="Netz1">
[24]              <ipv4 mask="255.255.0.0">172.17.30.1</ipv4>
[25]          </if>
[26]          <forwarding type="ip" />
[27]      </vm>
```

Die Beschreibung der ersten virtuellen Maschinen wird in den Zeilen 16 bis 27 vorgenommen. Im öffnenden “virtualmachine”-Tag wird als Attribut der Name des virtuellen

Rechners angegeben.
In Zeile 17 wird das Dateisystem angegeben, auf dem die virtuelle Maschine arbeiten soll. Als Typ wird in diesem Beispiel COW angegeben. Der Linux-Kernel der gebootet werden soll, wird in Zeile 18 mit dem Kernel-Tag festgelegt. In Zeile 19 wird die Konsole angegeben, auf der die Ausgaben des Kernels beim Hochfahren angezeigt werden. Sollten Probleme beim Starten dieses ersten Szenarios auftauchen, so bieten diese Ausgaben einen Anhaltspunkt zur Fehlersuche.
Die Zeilen 20 bis 22 beschreiben eine Netzwerkschnittstelle des virtuellen Routers. Das Attribut `id` wird später herangezogen, um das Interface zu benennen, hier wird somit eth1 beschrieben. Dieses Interface verbindet den Rechner mit dem Netz0. Die IPv4-Adresse, sowie die Netzwerkmaske von eth1 wird mit dem ipv4-Tag spezifiziert.
Man sollte beachten, dass eth0 immer die Management-Schnittstelle ist, daher darf als kleinste `id` nur die 1 nicht aber die 0 vergeben werden.
Analog dazu wird in den Zeilen 23 bis 26 eth2 beschrieben.
In Zeile 27 wird schließlich das forwarding aktiviert, ohne dieses Tag würde der virtuelle Rechner kein Routing ausführen.

```
[28]     <vm name="bravo">
   [29]         <filesystem type="cow">
   /usr/local/share/vnuml/filesystems/root_fs_tutorial </filesystem>
   [30]         <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
   [31]         <boot><con0>xterm</con0></boot>
   [32]         <if id="1" net="Netz1">
   [33]             <ipv4 mask="255.255.0.0">172.17.30.2</ipv4>
   [34]         </if>
   [35]         <if id="2" net="Netz2">
   [36]             <ipv4>192.168.30.1</ipv4>
   [37]         </if>
   [38]         <forwarding type="ip" />
   [39]     </vm>
```

Analog wird nun in den Zeilen 28 bis 39 eine weitere virtuelle Maschine mit dem Namen Bravo spezifiziert.

```
[40]</vnuml>
```

In Zeile 40 wird das vnuml-Tag aus Zeile 3 geschlossen und beendet damit die XML-Beschreibung des Szenarios.

4.1.2 Starten der Simulation (Ausführungsphase)

Starten der Simulation

Mit Hilfe der XML-Beschreibung aus dem vorigen Abschnitt, kann der VNUML Parser nun die virtuellen Rechner jeweils als User-Prozess erzeugen und booten. Dies geschieht durch die Nutzung des User-Mode-Linux Pakets. Der VNUML-Parser kapselt die Benutzung des eigentlichen UML, so dass der Nutzer von VNUML keine besonderen Kenntnisse zu UML benötigt. Es wird nur die XML-Beschreibung benötigt.

Das Szenario startet und VNUML bootet die virtuellen Rechner mit folgendem Befehl:

```
vnumlparser -t /usr/local/share/vnuml/examples/rip1.xml -vB
```

Als Reaktion auf diesen Befehl sollten sich nach einigen Sekunden zwei `xterm`-Fenster öffnen. Allerdings muss der Benutzer dazu als root in der grafischen Oberfläche eingeloggt sein. Denn wenn der VNUML-Parser zwar mit root-Rechten in einer Konsole aufgerufen wird, der Bediener aber in der grafischen Oberfläche nur als normaler Benutzer angemeldet wurde, kann `xterm` wegen Konflikten mit den Nutzerrechten nicht starten.

Die `xterm`-Fenster sollten die typischen Ausgaben eines startenden Linuxsystems zeigen. Die Rechner sind komplett hochgefahren, wenn in den `xterm`-Fenstern nach dem Login gefragt wird. Gleichzeitig sollte die Konsole, von der der Vnuml-Parser gestartet wird, folgende Ausgabe anzeigen:

```
alpha sshd is ready (socket style): 192.168.1.146 (mng_if)
bravo sshd is ready (socket style): 192.168.1.150 (mng_if)
host> /bin/rm -f /var/vnuml/LOCK
Total time eslapsed: 26 seconds
```

Je nach Hardware des Hostrechners kann die Zeit zum Starten der Simulation differieren. Das erstmalige Starten einer Simulation dauert in der Regel länger, da hier meist eine Überprüfung des Dateisystems durchgeführt wird.

Nach erfolgreichem Start kann sich der Benutzer wahlweise direkt über `xterm` auf den virtuellen Maschinen einloggen, oder mit einer beliebigen Konsole des Hostrechners eine Verbindung per `ssh` aufbauen. Das root-Passwort lautet "xxxx".

Da in Zeile 10 der XML-Beschreibung das <host_mapping> Tag gesetzt wurde, ist es möglich, die virtuellen Rechner vom Host aus mit den Namen alpha und bravo anzusprechen.

Mit

```
ssh alpha
```

bzw.

```
ssh bravo
```

können Verbindungen zu den virtuellen Rechnern aufgebaut werden. Diese Namensauflösung ist allerdings auf den Hostrechner beschränkt, so dass die virtuellen Rechner untereinander nur mit den IP-Addressen angesprochen werden können. Als erstes sollte nun die Netzwerkverbindung getestet werden. Dazu empfiehlt es sich ein Login auf dem virtuellen Rechner `alpha` durchzuführen. Ein

```
ping 172.17.30.2 -c3
```

von Rechner `alpha` prüft ob `eth1`von Rechner `bravo` erreicht werden kann und sollte zu einem ähnlichen Ergebnis wie dem Folgenden führen:

```
PING 172.17.30.2 (172.17.30.2) 56(84) bytes of data.
64 bytes from 172.17.30.2: icmp_seq=1 ttl=64 time=0.563 ms
64 bytes from 172.17.30.2: icmp_seq=2 ttl=64 time=0.628 ms
64 bytes from 172.17.30.2: icmp_seq=3 ttl=64 time=0.671 ms
--- 172.17.30.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.563/0.620/0.671/0.052 ms
```

Sollte der Versuch fehlschlagen, ist es naheliegend, dass die Installation unvolllständig war und zumindest das Paket `brctl` nicht richtig installiert wurde. Wahlweise kann auch die entgegengesetzte Richtung getestet werden, indem man von Rechner `Bravo` mit

```
ping 172.17.30.1 -c3
```

testet, ob die Verbindung zu `alpha` gewährleistet ist.

Starten und Konfigurieren von Quagga

Um das Beispiel abzuschließen wird im Folgenden gezeigt, wie auf den virtuellen Maschinen ein Rip-Router gestartet und konfiguriert wird. Zunächst muss auf einem virtuellen Rechner, der mit Quagga als Router konfiguriert werden soll, der zentrale Zebra-Prozess gestartet werden. Dies geschieht auf den beiden Rechnern Alpha und Bravo mit dem Aufruf:

```
zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601
```

Danach kann ebenfalls auf alpha sowie bravo mit dem Aufruf

```
ripd -f /usr/local/etc/quagga/ripd.conf.sample -d -P 2602
```

der eigentliche Rip-Prozess gestartet werden. Beide Router sind jedoch noch nicht weiter konfiguriert und nicht aktiv. Um alpha zu konfigurieren, muss eine Verbindung zur Konfigurationsschnittstelle des Rip-Routers geschaffen werden (siehe 3.2). Dies kann von einer Konsole von alpha selbst mit

```
telnet localhost 2602
```

oder vom Hostrechner mit

```
telnet alpha 2602
```

geschehen. Der Rip-Prozess fragt nach erfolgreichem Aufbau der Verbindung nach einem Passwort, dieses lautet "zebra". Da nur privilegierte Benutzer Änderungen an der Konfiguration vornehmen dürfen muss mit

```
ripd> enable
```

in den entsprechenden Modus gewechselt werden.
Danach muss die Konfiguration mit

```
ripd# configure terminal
```

eingeleitet werden. Schließlich muss noch festgelegt werden, dass Änderungen am RIP-Router selber konfiguriert werden sollen.

```
ripd(config)# router rip
```

Ab diesem Punkt ist es möglich, die angeschlossenen Netzwerke, die über RIP erreicht werden sollen, anzugeben. Alpha ist direkt mit den Netzwerken 172.17.0.0/16 und 192.168.20.0/24 verbunden und der Router soll die entsprechenden Pakete für beide Netze weiterleiten. Zu diesem Zweck gibt es den Quagga-Befehl `network`.

```
ripd(config-router)# network 172.17.0.0/16
ripd(config-router)# network 192.168.20.0/24
```

Der Konfigurationsmodus wird durch

```
ripd(config-router)# end
```

beendet und der Benutzer gelangt in den normalen privilegierten Modus zurück.

Alpha ist nun soweit konfiguriert, dass der RIP-Router die nötigen Informationen mit anderen Routern austauschen könnte und die Erreichbarkeit der angeschlossenen Netzwerke gewährleistet wäre.

```
ripd# show ip rip
```

zeigt die Weiterleitungstabelle von alpha:

```
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop    Metric From      Tag Time
C(i) 172.17.0.0/16    0.0.0.0          1  self       0
C(i) 192.168.20.0/24  0.0.0.0          1  self       0
```

Allerdings ist Quagga auf Rechner bravo noch nicht konfiguriert, daher stehen in der Weiterleitungstabelle bisher nur direkt angeschlossene Netzwerke, die im vorhergehenden Schritt mit dem network-Befehl konfiguriert wurden.

Analog zur Konfiguration von alpha werden nun die Einstellungen auf bravo vorgenommen.

```
telnet bravo 2602
```

Passwort ist ebenfalls “zebra”.

```
ripd> enable
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 172.17.0.0/16
ripd(config-router)# network 192.168.30.0/24
ripd(config-router)# end
```

Beide Router sind jetzt korrekt konfiguriert. Nach einigen Sekunden sollten die Router ihre Tabellen ausgetauscht haben, so dass die Routingtabelle von bravo wie folgt aussieht:

```
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop        Metric      From         Tag Time
C(i) 172.17.0.0/16    0.0.0.0            1        self          0
R(n) 192.168.20.0/24  172.17.30.1        2        172.17.30.1   0  2:48
C(i) 192.168.30.0/24  0.0.0.0            1        self          0
```

Die Tabelle ist folgendermaßen aufgebaut. In der ersten Spalte steht die Art, über die der Router das Netzwerk erlernt hat, bei dem ersten und dritten Eintrag bedeutet C(i), dass der Router direkt mit Schnittstellen an diese Netzwerke angeschlossen ist. Der zweite Eintrag wurde von einem RIP-Router erlernt, daher der Vermerk R(n). In der zweiten Spalte steht die Netzwerkadresse des betreffenden Netzes. Die dritte Spalte verweist auf den Next Hop, um das Netzwerk zu erreichen. Die Metrik, also in diesem RIP Beispiel die Anzahl der Router bis zum Zielnetzwerk, wird in Spalte vier aufgeführt. In Spalte fünf wird der Router, von dem der Eintag gelernt wurde, angezeigt. In der letzen Spalte wird ein Zeitwert angezeigt. Läuft diese Zeit ab, ohne dass Updatepakete zu dem entsprechenden Eintrag ankommen, wird der Eintrag als unerreichbar markiert, indem die Metrik auf den maximalen Wert 16 gesetzt wird.

Die zweite Zeile in der Tabelle zeigt, dass Router bravo von alpha den Weg zum Netz 192.168.20.0/24 gelernt hat, der Next Hop ist Router alpha. Die Weiterleitungstabelle von Router alpha sieht wie folgt aus:

```
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop       Metric      From         Tag Time
C(i) 172.17.0.0/16    0.0.0.0           1        self          0
C(i) 192.168.20.0/24  0.0.0.0           1        self          0
R(n) 192.168.30.0/24  172.17.30.2       2        172.17.30.2   0  02:48
```

Das Netz 192.168.30.0/24 ist über Router bravo erreichbar die Entfernung beträgt 2 Hops.

Nachdem in diesem ersten Beipiel die Designphase und die Implementationsphase durchlaufen wurden, befindet sich das Szenario an diesem Punkt mitten in der Ausführungsphase. Bisher wurden Anwendungen und Prozesse gestartet und konfiguriert. Die Ergebnisse der Konfiguration liegen in Form der Routingtabellen vor. Das eigentliche Ziel dieses Beipiels liegt aber im nachvollziehbaren, schrittweisen Erläutern des Vorgehens beim Erstellen einer Simualtion mit VNUML.

Um den Lebenszyklus des Szenarios zu komplettieren wird im nächsten Abschnitt erklärt, wie die Simulation beendet wird.

Beenden der Simulation

Das saubere Anhalten einer Simulation besteht aus zwei Teilen. Zuerst werden die Anwendungen, die gestartet wurden, beendet. Danach werden die virtuellen Rechner heruntergefahren. Da die Anwendungen in diesem Beispiel manuell gestartet wurden, werden sie auch durch die entsprechenden Befehle von Hand angehalten. Beim Herunterfahren der virtuellen Rechner wird dann wieder auf die Unterstützung des VNUML-Parsers zurückgegriffen.

Die Routing-Prozesse werden in umgekehrter Reihenfolge zum Start angehalten. Sowohl auf Rechner alpha als auch auf Rechner bravo können der ripd-Prozess und der zebra-Prozess einfach per killall Befehl beendet werden. Dadurch werden alle Prozesse mit dem angegebenen Namen beendet.

```
host:# ssh alpha
alpha:# killall ripd
alpha:# killall zebra
alpha:# exit

host:# ssh bravo
alpha:# killall ripd
alpha:# killall zebra
alpha:# exit
```

Nach erfolgreichem Beenden der Routing-Daemons, kann nun das Simulationsszenario heruntergefahren werden. Der VNUML-Parser bietet dies unter der Option -d an.

```
host:# vnumlparser.pl -d /usr/local/share/vnuml/examples/rip1.xml
   -vB
```

Nach diesem Aufruf kann in den `xterm`-Fenstern beobachtet werden, wie die virtuellen Rechner herunterfahren. Die `xterm`-Fenster schließen selbstständig, wenn die dazugehörigen virtuellen Maschinen komplett heruntergefahren sind.

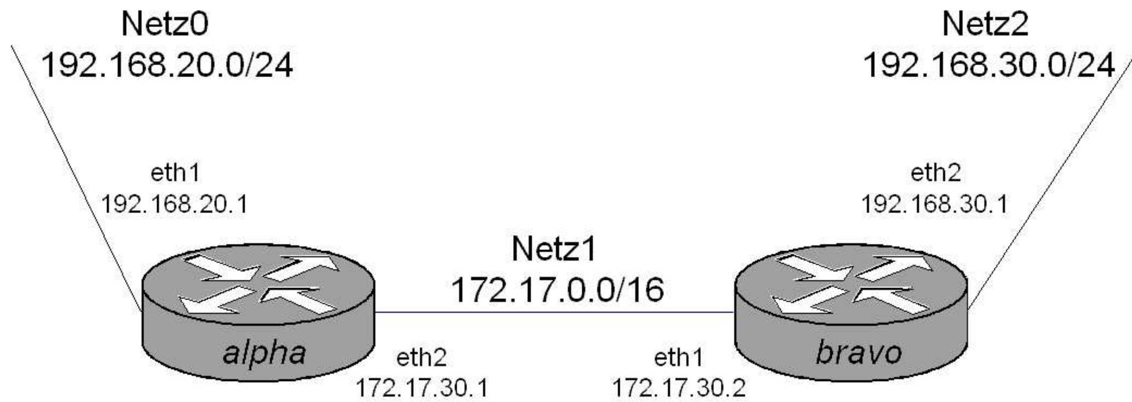
Der Lebenszyklus ist nun einmal komplett durchlaufen worden. Von der Designphase über die Implementierungsphase bis hin zur Ausführungsphase mit den Teilabschnitten, Starten des Szenarios, Starten von Anwendungen, Stoppen von Anwendungen und Herunterfahren des Szenarios, wurden alle VNUML typischen Phasen vorgestellt. Es ist nun möglich, die Topologie abzuändern und das Szenario mit der geänderten XML-Beschreibung erneut zu starten, damit wird dann der Lebenszyklus erneut gestartet.

4.2 Erweiterung des Beispiels

Im vorigen Abschnitt wurden die virtuellen Rechner von VNUML gestartet und später heruntergefahren. Das Starten von Quagga wurde manuell durchgeführt, indem über die Konsole, die entsprechenden Prozesse gestartet wurden. Die Konfiguration der Router wurde ebenso von Hand über die virtuellen Konsolen der Routing-Suite vorgenommen. Da diese manuelle Vorgehensweise allerdings zeitraubend und fehleranfällig ist, gibt es die Möglichkeit, diese Operationen mit Hilfe von VNUML zu teilautomatisieren. Dadurch kann dann eine Simulation wieder und wieder auf exakt die gleiche Weise gestartet und durchlaufen werden.

VNUML bietet dazu die Möglichkeit das Starten und Stoppen von Anwendungen auf den virtuellen Rechnern zu übernehmen. Vielmehr können beliebige Skripte und Befehle gruppiert und dann in zusammenhängenden Blöcken ausgeführt werden. Weiterhin bietet VNUML die Möglichkeit, ganze Verzeichnisse vom Host-Rechner in ein Dateisystem eines virtuellen Rechners zu kopieren. Mit Hilfe dieses Mechanismus' können vorgefertigte Skripte und Konfigurationsdateien zum virtuellen Rechner kopiert werden.

Das Beispiel aus dem letzten Abschnitt wird im Folgenden so abgeändert, dass die beiden beschriebenen Möglichkeiten genutzt werden. Die Netzwerktopologie bleibt dabei unverändert.



Implementierungsphase:

Die Konfigurationsdateien der beiden Router alpha und bravo werden nun bereits im Vorfeld verfasst.

alpha.conf:

```
!
hostname ripd
password zebra
!
router rip
  network 172.17.0.0/16
  network 192.168.20.0/24
!
line vty
```

Alle Zeilen die mit "!" beginnen, sind Kommentarzeilen und werden ingnoriert, dazu muss aber "!" oder "#" zwingend als erstes Zeichen in der Zeile stehen, somit sind Kommentare hinter der eigentlichen Anweisung nicht möglich.

In der zweiten Zeile wird der Hostname des Routers festgelegt. Danach wird das Passwort für den unprivilegierten Benutzer auf "zebra" gesetzt. In dieser Konfiguration wird nicht auf ein zweites Passwort zum Wechseln in den privilegierten Modus bestanden. Daher können alle Benutzer, die das normale Passwort wissen, auch in den privilegierten Modus wechseln und dadurch die Konfiguration abändern. Auf eine Verschlüsselung des Passwortes wird hier ebenso verzichtet, daher steht das Passwort im Klartext in der

Konfigurationsdatei. Prinzipiell ist eine Verschlüsselung jedoch möglich und in Bezug auf die Praxis auch anzuraten. In Zeile 5 wird der RIP-Router aktiviert, dies muss geschehen, bevor weitere RIP Befehle gegeben werden können. Zeile 6 und 7 sind network-Befehle, mit denen festgelegt wird für welche Netzwerke der RIP-Router zuständig ist. In Zeile 6 wird RIP für das Netzwerk 172.17.0.0/16, in Zeile 7 für dass Netzwerk 192.168.20.0/24 aktiviert. Die letzte Zeile erlaubt eine Konfiguration per `telnet`. Die virtuelle Konsole vty wird aktiviert und ripd horcht auf den, beim Starten des Daemons angegebenen, Port.

bravo.conf:

```
!
hostname ripd
password zebra
!
router rip
  network 172.17.0.0/16
  network 192.168.30.0/24
!
line vty
```

Die Konfiguration von bravo unterscheidet sich nur in den network Befehlen von der des Rechners alpha. Da der Router auf Rechner bravo ebenfalls zum Netz 172.17.0.0/16 gehören soll, befindet sich diese Zeile in beiden Konfigurationsdateien. Bravo soll aber die Erreichbarkeit des Netzwerkes 192.168.30.0/24 sicherstellen, daher findet sich in Zeile 7 ein entsprechender `network` Befehl.

Die Konfigurationsdateien befinden sich auf dem Hostrechner unter /usr/local/share/vnuml/examples/rip2/alpha/alpha.conf bzw. unter .../rip2/bravo/bravo.conf jeweils in einem eigenen Unterverzeichnis pro virtueller Maschine. Der Inhalt dieser Verzeichnisse soll vor dem Starten der RIP-Daemons in die Dateisysteme der virtuellen Maschine kopiert werden. Dazu sind einige Erweiterungen der XML-Beschreibung aus dem letzten Abschnitt notwendig.

/usr/local/share/vnuml/examples/rip2.xml:

```
     ...
[ 4] <global>
       ...
[12]   <basedir>/usr/local/share/vnuml/examples/rip2</basedir>
[13] </global>
     ...
[18] <vm name="alpha">
       ...
[29]   <filetree when="start" root="/usr/local/etc">alpha</filetree>
[30]   <exec seq="start" type="verbatim">zebra -f
/usr/local/etc/zebra.conf.sample -d -P 2601</exec>
[31]   <exec seq="start" type="verbatim">ripd -f /usr/local/etc/alpha.conf
   -d -P 2602</exec>
[32]   <exec seq="stop" type="verbatim">killall zebra</exec>
[33]   <exec seq="stop" type="verbatim">killall ripd</exec>
[34] </vm>
```

Die erste Änderung an der Konfigurationsdatei findet sich in Zeile 12. Hier wurde ein <basedir>-Tag in den globalen Bereich der XML-Struktur eingefügt. Dieses Basisverzeichnis wird mit dem Tag auf /usr/local/share/vnuml/examples/rip2 gesetzt. Die verschiedenen Unterverzeichnisse für die virtuellen Rechner und damit die obenstehenden Konfigurationsdateien, sind unter diesem Pfad im Dateisystem des Hostrechners zu finden.

```
[18] <vm name="alpha">
       ...
[29]   <filetree when="start" root="/usr/local/etc">alpha</filetree>
[30]   <exec seq="start" type="verbatim">zebra -f
/usr/local/etc/zebra.conf.sample -d -P 2601</exec>
[31]   <exec seq="start" type="verbatim">ripd -f /usr/local/etc/alpha.conf
   -d -P 2602</exec>
[32]   <exec seq="stop" type="verbatim">killall zebra</exec>
[33]   <exec seq="stop" type="verbatim">killall ripd</exec>
[34] </vm>
```

In Zeile 29 wurde innerhalb des `<vm>`-Tags von Rechner alpha ein <filetree>-Tag ergänzt. Das Verzeichnis alpha, das mit Hilfe des <filtetree> Tags eingeklammert wurde, bezieht sich relativ auf den Pfad des Basisverzeichnises, das durch <basedir> angegeben wurde. Dadurch wird der VNUML-Parser angewiesen, den Inhalt des Verzeichnises /usr/local/share/vnuml/examples/rip2/alpha in das Dateisystem des virtuellen Rechners zu kopieren. Das Attribut `root` gibt das Ziel dieses Kopiervorganges an. Dadurch wird der komplette Inhalt des Verzeichnis' .../rip2/alpha (Hostrechner) in das Verzeichnis /usr/local/etc (virtuelle Maschine) kopiert. In diesem speziellen Fall beinhaltet .../rip2/alpha nur die Datei alpha.conf und diese wird in /usr/local/etc kopiert. Durch das Attribut `when` wird der Zeitpunkt festgelegt, zu dem der Kopiervorgang durchgeführt wird.
Um Quagga mit Hilfe des VNUML Parsers zu starten bzw. anzuhalten, wurden in den Zeilen 30-33 die <exec> Tags ergänzt. Diese <exec> Tags besitzen die Attribute `seq` und `type`. In diesem Beispiel wird `type=''verbatim''` verwendet. Dadurch wird die Zeichenkette innerhalb <exec ...> </exec> wörtlich ausgeführt. In Zeile 30 wird demnach `zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601` ausgeführt und damit der Zebra-Daemon gestartet. Das Attribut `seq=''start''` aus Zeile 30 und 31 definiert dabei den Namen einer Befehlssequenz als Schlüsselwort, bei dem der VNUML-Parser den angegebenen Befehl ausführen soll. Wird der Parser später nach erfolgreichem Starten des Szenarios mit `vnumlparser.pl -x start@rip2.xml` aufgerufen, so werden zuerst alle Kopiervorgänge aus den `<filetree>` Tags ausgeführt, die ein Attribut `when=''start''` haben und danach werden alle `<exec>` Anweisungen abgearbeitet, die ein Attribut `seq=''start''` besitzen. Das Schlüsselwort, hier "start", kann beim Erstellen der XML-Beschreibung frei gewählt werden. In Zeile 30 und 31 wird der VNUML-Parser angewiesen, beim Ausführen der Sequenzen mit dem Schlüsselwort "start" den zebra-Daemon sowie den ripd-Daemon auf alpha zu starten, da die `<exec>` Tags innerhalb des `<vm name=''alpha''>...</vm>` stehen. Beim Starten des ripd wird dabei die Konfigurationsdatei eingelesen, die vorher durch `<filetree>` zu alpha kopiert wurde.
In den Zeilen 32 und 33 stehen die Anweisungen, die zur Sequenz "stop" gehören. Diese Anweisungen beenden sowohl zebra als auch ripd und sind exakt die gleichen Befehle, die im ersten Beispiel von Hand eingegeben wurden (siehe 4.1.2).

Die Zeilen 47-51 beschreiben die `<filetree>` und `<exec>` Einstellungen von Rechner bravo analog dazu.

```
[36] <vm name="bravo">
[47]   <filetree when="start" root="/usr/local/etc">bravo</filetree>
[48]   <exec seq="start" type="verbatim">zebra -f /usr/local/etc/zebra.conf.samp
   -d -P 2601</exec>
[49]   <exec seq="start" type="verbatim">ripd -f /usr/local/etc/bravo.conf
   -d -P 2602</exec>
[50]   <exec seq="stop" type="verbatim">killall zebra</exec>
[51]   <exec seq="stop" type="verbatim">killall ripd</exec>
[52] </vm>
```

Ausführungsphase:

Das Szenario wird wie im vorhergehenden Beispiel mit der Option -t gestartet.

```
host:# cd /usr/local/share/vnuml/examples/
host:/usr/local/share/vnuml/examples# vnumlparser.pl -t rip2.xml
   -vB
```

Die virtuellen Rechner sollten nun gestartet werden. Nach einigen Sekunden öffnen sich wiederum zwei `xterm`-Fenster, in denen der Fortschritt des Bootvorgangs kontrolliert werden kann.

Nachdem Rechner alpha fertig gebootet wurde, sollte ein Login als root erfolgen (wahlweise direkt im xterm Fenster oder über ssh).Das root Passwort lautet "xxxx".

Virtual Console #0 alpha:

```
Debian GNU/Linux 3.1 alpha ttys/0
alpha login: root
Password: xxxx
```

Zum Überprüfen des `<filetree>` Tags wird zunächst in das Verzeichnis /usr/local/etc/ gewechselt. Dort existiert beim ersten Start des Szenarios noch keine Datei `alpha.conf`.

```
alpha:~# cd /usr/local/etc
alpha:/usr/local/etc# ls -l
total 9
-rw-r--r-- 1 root staff 570 Jan 25 2004 bgpd.conf.sample
-rw-r--r-- 1 root staff 2801 Jan 25 2004 bgpd.conf.sample2
```

```
-rw-r--r-- 1 root staff 182 Jan 25 2004 ospfd.conf.sample
drwxr-sr-x 2 quagga quagga 1024 May 23 2004 quagga
-rw-r--r-- 1 root staff 410 Jan 25 2004 ripd.conf.sample
-rw-r--r-- 1 root staff 373 May 18 2004 zebra.conf.sample
alpha:/usr/local/etc#
```

Nun wird der VNUML-Parser angewiesen die Befehlssequenz "start" auszuführen. Dazu muss folgende Eingabe in eine Konsole des Hostrechners gemacht werden:

```
host:/usr/local/share/vnuml/examples# vnumlparser.pl -x start@rip2.xml
   -vB
```

Durch die Option `-x SCHLÜSSELWORT@XML-DATEI` wird der Parser nun beauftragt, alle `<filetree>` und `<exec>` Tags mit entsprechenden Sequenznamen `when=''SCHLÜSSELWORT''` bzw. `seq=''SCHLÜSSELWORT''` in der XML-Datei auszuführen. Das Verzeichnis /usr/local/etc des Rechners alpha enthält nun die Konfigurtaionsdatei alpha.conf:

```
alpha:/usr/local/etc# ls -l
total 9
-rw-r--r-- 1 root staff 103 May 2o 10:06 alpha.conf
-rw-r--r-- 1 root staff 570 Jan 25 2004 bgpd.conf.sample
-rw-r--r-- 1 root staff 2801 Jan 25 2004 bgpd.conf.sample2
-rw-r--r-- 1 root staff 182 Jan 25 2004 ospfd.conf.sample
drwxr-sr-x 2 quagga quagga 1024 May 23 2004 quagga
-rw-r--r-- 1 root staff 410 Jan 25 2004 ripd.conf.sample
-rw-r--r-- 1 root staff 373 May 18 2004 zebra.conf.sample
alpha:/usr/local/etc#
```

Der zebra-Daemon sowie der ripd-Daemon wurden ebenfalls beim letzten Parser Aufruf gestartet und laufen nun bereits auf `alpha` und `bravo`. Eine Verbindung zu ripd kann über `telnet` hergestellt werden:

```
alpha:# telnet localhost 2602
```

Passwort lautet wie in der Konfigurationsdatei angegeben"zebra".

```
ripd> enable
ripd# show running-config
```

```
Current configuration:
!
hostname ripd
password zebra
!
router rip
network 172.17.0.0/16
network 192.168.20.0/24
!
line vty
!
end
ripd#
```

Da das Anzeigen der aktuell verwendeten Konfiguration nur privilegierten Benutzern erlaubt ist, wird der Modus zunächst mit `enable` gewechselt. Danach zeigt `show running-config`, dass die vorher auf dem Host entworfene Konfigurationsdatei korrekt kopiert und eingelesen wurde. Die Routingtabelle von Rechner alpha wird mit `show ip rip` angezeigt:

```
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop     Metric From        Tag Time
C(i) 172.17.0.0/16    0.0.0.0      1      self         0
C(i) 192.168.20.0/24  0.0.0.0      1      self         0
R(n) 192.168.30.0/24  172.17.30.2  2      172.17.30.2  0   02:22
ripd#
```

Da das an bravo angeschlossene Netzwerk 192.168.30.0/24 in der Weiterleitungstabelle von `alpha` aufgeführt wird, können Pakete von alpha dorthin weitergeleitet werden. Im Folgenden wird die Erreichbarkeit der Netzwerkschnittstelle 192.168.30.1 des Rechners bravo von alpha aus getestet.

```
alpha:# ping 192.168.30.1 -c3
PING 192.168.30.1 (192.168.30.1) 56(84) bytes of data.
```

```
64 bytes from 192.168.30.1: icmp_seq=1 ttl=64 time=0.547 ms
64 bytes from 192.168.30.1: icmp_seq=2 ttl=64 time=0.643 ms
64 bytes from 192.168.30.1: icmp_seq=3 ttl=64 time=0.676 ms
--- 192.168.30.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2013ms
rtt min/avg/max/mdev = 0.547/0.622/0.676/0.054 ms
```

Das Ergebnis zeigt, dass die Weiterleitungen zur Schnittstelle 192.168.30.1 und zurück funktionieren.

Da dieses Beispiel die Funktionsweise der `<exec>` und `<filetree>` Tags demonstriert, wird an dieser Stelle nicht weiter auf die Möglichkeiten der Ausführungsphase eingegangen.

Zum Beenden der Routing-Daemons führt der VNUML-Parser die Befehlssequenz "stop" aus.

```
host:/usr/local/share/vnuml/examples# vnumlparser.pl -x stop@rip2.xml
   -vB
```

Das Szenario wird anschließend wieder mit

```
host:/usr/local/share/vnuml/examples# vnumlparser.pl -d rip2.xml
   -vB
```

heruntergefahren.

4.3 RIP-Szenario mit Ausfall einer Leitung

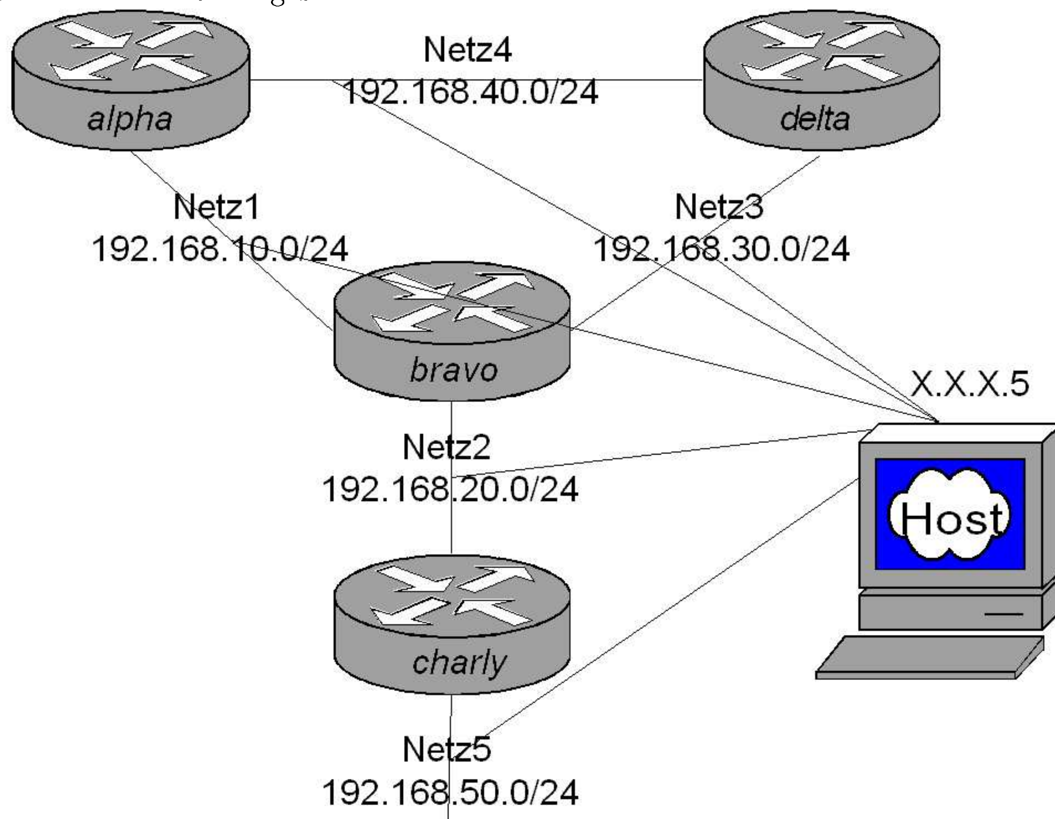
Nun wird eine Netzwerktopologie erstellt, die aus 4 Routern und 5 Netzwerken besteht. Einen ersten Eindruck von Konstruktion und Konfiguration eines Simulationsszenarios mit RIP-Routern hat das vorangehende Beispiel bereits gegeben. Ein wichtiger Aspekt von Routing-Verfahren ist das Verhalten im Fehlerfall. Die zentralen Fragen, die im Folgenden beispielhaft geklärt werden, sind: Wie verhält sich der RIP-Algorithmus beim Wegfall einer Verbindung? Wie wird ein Ausfall bemerkt? Wie rekonvergiert das Netz, wenn die Unterbrechung behoben ist?

Das folgende Szenario wird dazu herangezogen, zu zeigen wie die RIP-Router reagieren, wenn eine Netzwerkverbindung ausfällt. Dieser Ausfall wird daduch simuliert, dass eine Firewall die entsprechende Verbindung blockiert. Da die Firewall so konstruiert

ist, dass die Pakete stillschweigend herausgefiltert werden, kommt dieses Verhalten einer "gebrochenen" Verbindung näher, als der Ansatz die betreffenden Netzwerkschnittstellen einfach zu deaktivieren. Für eine Firewalll macht ein solches Verhalten durchaus Sinn, denn es sollen einem potentiellen Angreifer so wenig Rückmeldungen über den Verbleib seiner Pakete wie möglich gegeben werden.

Designphase:

Außer der eigentlichen Netzwerktopologie, die die Abbildung zeigt, werden weitere Anforderungen an das Szenario gestellt. Es sollen Befehlssequenzen zum Starten und Stoppen der Router, sowie zum Unterbrechen und Wiederherstellen der Verbindung zwischen alpha und delta bereitgestellt werden.



Zum Beobachten der Ereignisse bzw. zum Aufzeichnen des Netzwerkverkehrs und zum Rekonstruieren der unter Umständen schnellen zeitlichen Abläufe, kommt ein sog. Sniffer zum Einsatz. Dieses Werkzeug soll den gesamten Verkehr zwischen den Routern mithören und später anzeigen. Diese Aufgabe übernimmt das Programm `tcpdump`. Da eine grafische Repräsentation der so gewonnen Daten anschaulicher ist, wird `tcpdump` in Kombination mit dem Werkzeug `ethereal` eingesetzt. Dazu ist jedoch ein Zugriff auf die grafische Oberfläche des Hostrechners nötig. Deshalb wird der Mitschnitt und

die Darstellung des Netzwerkverkehrs vom Hostrechner übernommen. Der Hostrechner benötigt dazu jeweils eine Verbindung zu den virtuellen Netzwerken. Im Gegensatz zu den vorhergehenden Beispielen wird der Hostrechner nun, wenn auch nur passiv, mit in die Simulation eingebunden.

Implementierungsphase:

```
<global>
  ...
  <simulation_name>rip3</simualtion_name>
  <basedir>/usr/local/share/vnuml/examples/rip3</basedir>
  ...
</global>
<net name="Netz1" mode="uml_switch"/>
<net name="Netz2" mode="uml_switch"/>
<net name="Netz3" mode="uml_switch"/>
<net name="Netz4" mode="uml_switch"/>
<net name="Netz5" mode="uml_switch"/>

<vm name="alpha">
  ...
  <if id="1" net="Netz1">
    <ipv4 mask="255.255.255.0">192.168.10.1</ipv4>
  </if>
  <if id="2" net="Netz4">
    <ipv4 mask="255.255.255.0">192.168.40.2</ipv4>
  </if>
  ...
</vm>
<vm name="bravo">
  ...
  <if id="1" net="Netz1">
    <ipv4 mask="255.255.255.0">192.168.10.2</ipv4>
  </if><if id="2" net="Netz2">
    <ipv4 mask="255.255.255.0">192.168.20.1</ipv4>
  </if>
```

```
  <if id="3" net="Netz3">
    <ipv4 mask="255.255.255.0">192.168.30.1</ipv4>
  </if>
  ...
</vm>
<vm name="charly">
  ...
  <if id="1" net="Netz2">
    <ipv4 mask="255.255.255.0">192.168.20.2</ipv4>
  </if>
  <if id="2" net="Netz5">
    <ipv4 mask="255.255.255.0">192.168.50.1</ipv4>
  </if>
  ...
</vm>

<vm name="delta">
  ...
  <if id="1" net="Netz3">
    <ipv4 mask="255.255.255.0">192.168.30.2</ipv4>
  </if>
  <if id="2" net="Netz4">
    <ipv4 mask="255.255.255.0">192.168.40.1</ipv4>
  </if>
  ...
</vm>

<host>
  <hostif net="Netz1">
    <ipv4 mask="255.255.255.0">192.168.10.5</ipv4>
  </hostif>
  <hostif net="Netz2">
    <ipv4 mask="255.255.255.0">192.168.20.5</ipv4>
  </hostif>
  <hostif net="Netz3">
```

```
    <ipv4 mask="255.255.255.0">192.168.30.5</ipv4>
  </hostif>
  <hostif net="Netz4">
    <ipv4 mask="255.255.255.0">192.168.40.5</ipv4>
  </hostif>
  <hostif net="Netz5">
    <ipv4 mask="255.255.255.0">192.168.50.5</ipv4>
  </hostif>
</host>
```

Der obenstehende Konfigurationsausschnitt lässt sich direkt aus der Skizze ableiten und stellt eine VNUML Repräsentation der zu simulierenden Topologie dar. Der einzige Unterschied zu den vorangegangenen Beispielen stellt die Verwendung des Attributes "`uml-switch`" des `<net>`-Tags dar. Diese Änderung ist notwendig, da durch die Verwendung der Switch-Technologie ein Mithören des Netzwerkverkehrs einfacherer ist. Die bisher verwendete Alternative sind die Brücken, diese werden durch den Parser so simuliert, dass alle Pakete vom Sender an den Hostrechner (als Brücke) gesendet werden. Von dort aus werden die Pakete dann an die Zielrechner weitergereicht. Beim Mithören des Netzwerkverkehrs vom Hostrechner würden so alle Pakete mehrfach aufgezeichnet werden. Bei der normalen Benutzung der Brücke, die bei VNUML die Standard-Methode zur Darstellung einer virtuellen Netzwerkverbindung ist, geschieht dieser Vorgang für den Benutzer versteckt. In diesem Beispiel würden die beschriebenen Seiteneffekte allerdings stören. Daher wird in diesem Beispiel die UML-Switch Software benutzt, um die virtuellen Maschinen miteinander zu vernetzten. Das Weiterreichen der Pakete geschieht dann durch die Verwendung von Linux-Sockets, die von dem verwendeten Analysewerkzeug nicht abgehört werden. Dadurch wird ein mitgeschnittenes Paket nur einmal aufgezeichnet, was die spätere Analyse vereinfacht und die Übersicht wesentlich fördert.

Das Starten und Stoppen der Prozesse `zebra` und `ripd` übernehmen die bereits bekannten Ausführungssequenzen, jeweils in jeder virtuellen Maschine:

```
<exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601
</exec>
<exec seq="start" type="verbatim">ripd -f /usr/local/etc/bravo.conf
   -d -P 2602</exec>
<exec seq="stop" type="verbatim">killall zebra</exec>
```

```
<exec seq="stop" type="verbatim">killall ripd</exec>
```

Die Konfigurationen von zebra und ripd werden bereits voher vorgenommen und liegen für jede virtuelle Maschine in jeweils einem Unterverzeichnis. Vor der Ausführung der Befehlssequenz "start", werden die Dateien zu den virtuellen Machinen kopiert. Die folgenden Ausschnitte zeigen stellvertretend die entsprechende Zeile in der XML-Bschreibung, sowie die ripd Konfiguration von alpha.

```
<filetree when="start" root="/usr/local/etc">alpha</filetree>
```

```
alpha.conf:

!
hostname ripd
password zebra
!
router rip
network 192.168.10.0/24
network 192.168.40.0/24
!
line vty
```

Während der Simulation soll es möglich sein eine Verbindung gezielt zu unterbechen und wieder herzustellen. Mit der Firewall `iptables`, die im Dateisystem der virtuellen Rechner bereits vorinstalliert ist, können alle eingehenden und ausgehenden Pakete gefiltert werden. Wird die Firewall so konfiguriert, dass **alle** Pakete herausgefiltert werden, kommt dies einer Unterbrechung gleich. Durch das Entfernen der Filterregeln wird die Verbindung wieder hergestellt. Die Konfiguration der Firewall geschieht über kurze Shell-Skripte, die durch eine Befehlssequenz vom Parser aufgerufen werden.

```
cutline.sh:

#!/bin/bash
iptables -I FORWARD -i $1 -j DROP
iptables -I FORWARD -o $1 -j DROP
iptables -I INPUT -i $1 -j DROP
iptables -I OUTPUT -o $1 -j DROP
iptables -L -n -v
```

Das Skript `cutline.sh` unterbricht eine Verbindung. Der Aufruf geschieht mit Angabe der Netzwerkschnittstelle, die unterbrochen werden soll, als Parameter. Soll z.B. die Verbindung zwischen alpha und beta unterbrochen werden, so kann dies geschehen, indem alle Pakete an und von `eth1` von Rechner alpha herausgefiltert werden.

```
alpha:# cutline.sh eth1
```

Dieser Aufruf würde bewirken, dass durch das Skript alle eingehenden und ausgehenden Pakete, sowie alle Pakete, die über eth1 weitergeleitet werden sollen, in das Ziel DROP gefiltert werden, wodurch sie gelöscht werden.

Allgemein beginnen die Aufrufe von `iptables` mit dem ersten Parameter, hier -I, zusammen mit dem Namen der sogenannten chain, z.B. INPUT. Dieser gibt an, dass eine Regel zur Chain INPUT hinzugefügt wird (-I = Insert). Mit dem zweiten Parameter -i $1oder -o $1 wählt man das sogenannte in-interface bzw. das out-interface. $1 ist eine Variable mit dem Inhalt des ersten Parameters beim Aufruf des Skriptes, erfolgt der Aufruf
`cutline.sh eth1` hat $1 den Wert "eth1" an. Der letzte Parameter -j DROP legt das Ziel des Paketes fest, auf das die vorher genannte Regel zutrifft. Mit -j DROP werden die Pakete gelöscht.

```
repair.sh:

#!/bin/bash
iptables -D INPUT -i $1 -j DROP
iptables -D OUTPUT -o $1 -j DROP
iptables -D FORWARD -i $1 -j DROP
iptables -D FORWARD -o $1 -j DROP
iptables -L -n -v
```

Das Skript `repair.sh` ist fast identisch aufgebaut, lediglich der erste Parameter eines jeden `iptables` Aufrufs lautet jetzt -D (D = Delete). Dadurch wird die angegebene Filterregel gelöscht.

In beiden Skripten bewirkt die letzte Zeile eine detailierte Ausgabe der aktuellen Filterregeln. Der Aufruf der Skripte erfolgt durch die Befehlssequenzen "cut" und "repair" in der XML-Beschreibung, analog zu den bekannten Befehlssequenzen "start" und "stop".

```
<filetree when="cut" root="/usr/local/bin">scripts</filetree>

<exec seq="cut" type="verbatim">cutline.sh eth1</exec>
<exec seq="repair" type="verbatim">repair.sh eth1</exec>
```

Ausführungsphase:

Wie die vorangegangenen Beispiele, wird das Szenario mit der Option -t hochgefahren:

```
host:# cd /usr/local/share/vnuml/examples/
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -t rip3.xml
   -vB
```

Nachdem alle vier virtuellen Maschinen hochgefahren sind, können die Routing-Daemons gestartet werden:

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x start@rip3.xml
   -vB
```

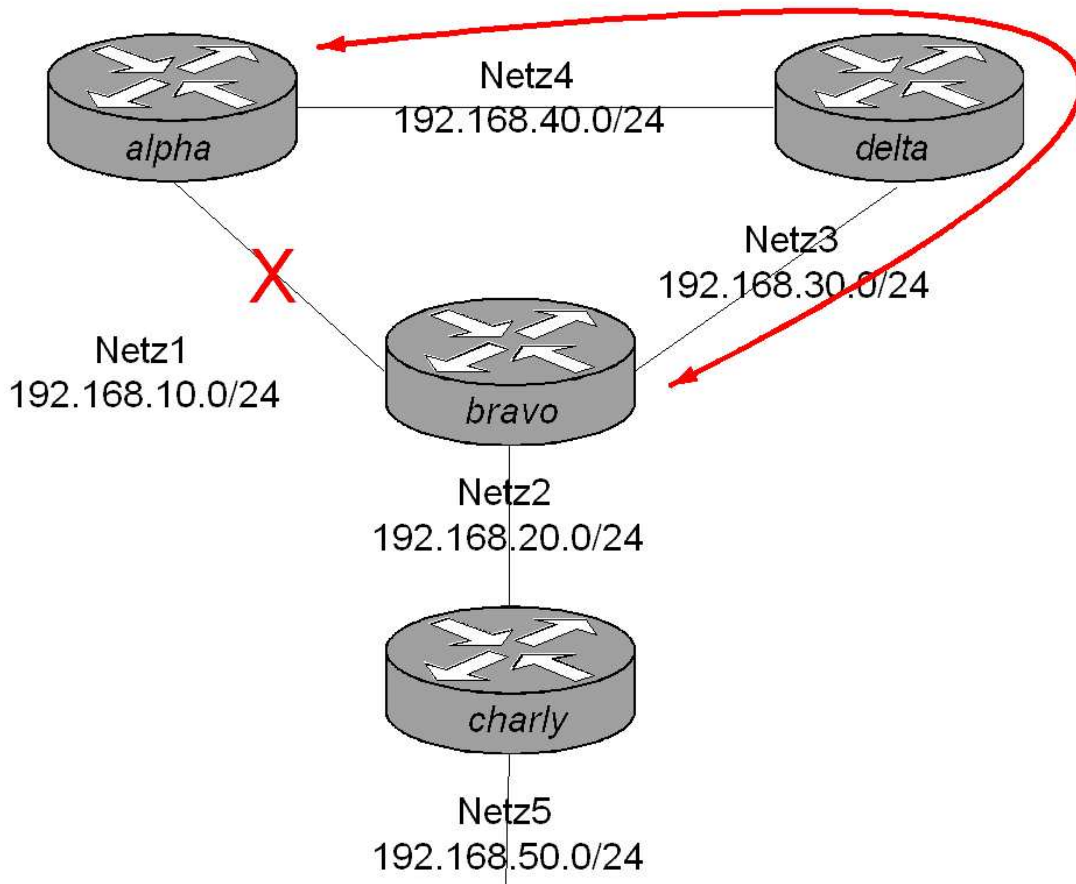
Nach einigen Sekunden sollten die Routingtabellen bereits konvergiert sein.

Die Weiterleitungstabelle von alpha sollte wie folgt aussehen:

```
alpha:~# telnet localhost 2602
Trying ::1...
Connected to localhost.
Escape character is '^]'.

Hello, this is quagga (version 0.96.4).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

     Network          Next Hop       Metric From         Tag Time
C(i) 192.168.10.0/24 0.0.0.0             1    self         0
R(n) 192.168.20.0/24 192.168.10.2        2    192.168.10.2 0 02:53
```

```
R(n) 192.168.30.0/24 192.168.10.2     2    192.168.10.2 0 02:53
C(i) 192.168.40.0/24 0.0.0.0          1    self         0
R(n) 192.168.50.0/24 192.168.10.2     3    192.168.10.2 0 02:53
```

Die Tabelle zeigt, dass alpha alle Pakete für die Netze 192.168.20.0/24, 192.168.30.0/24 und 192.168.50.0/24 über den Next Hop 192.168.10.2 schickt. Dies bedeutet, dass diese Pakete von alpha über bravo geroutet werden. Wenn nun aber die Verbindung zu bravo unterbrochen würde, dann würde immer noch die Route über delta existieren. Über diese Ausweichroute könnten dann immer noch alle Netze erreicht werden.



Damit auf ein solches unvorhergesehenes Ereignis richtig reagiert werden kann, muss der RIP-Algorithmus einige Leistungen erbringen.

Zunächst muss der Ausfall der Verbindung überhaupt bemerkt werden. Danach sollte diese Route als unbrauchbar markiert werden. Wenn eine alternative Route existiert, muss diese gefunden werden, bevor der Verkehr den neuen Weg benutzen kann.

Um den RIP Algorithmus in genau diesen Kriterien zu fordern und zu beobachten wie das Verhalten in einer solchen Situation konkret aussieht, wird nun die Verbindung zwischen alpha und beta unterbochen. Die Reaktionen der einzelnen Router werden

zentral vom Hostrechner aufgezeichnet. Dazu wird das Programm ethereal auf dem Host gestartet:

```
host :# ethereal
```

Die Aufzeichnung wird gestartet, indem aus dem Menü "Capture" der Punkt "Start" aufgerufen wird (oder Strg+K). Nun erscheint das Fenster "Capture Options". In diesem Fenster wird unter dem Punkt "Interface" "any" ausgewählt. In das Feld "Filter" wird "udp" eingetragen. Danach kann die Aufzeichnung mit "OK" begonnen werden. Es werden nun alle UDP-Pakete im Szenario mitgeschnitten. Alternativ können die Aufzeichnungen auch mit tcpdump gemacht werden (tcpdump -i any -s0).

Um die Netzwerk-Verbindung zwischen alpha und beta zu zerstören, wird die Befehlssequenz "cut" auf dem Hostrechner ausgeführt:

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x cut@rip3.xml
   -vB
```

Die Pakete zwischen alpha und beta werden nun herausgefiltert. Die RIP-Router senden in einem Zeitintervall von ca. 30 Sekunden Update-Pakete an Ihre Nachbarn, diese Updates enthalten die Einträge ihrer Weiterleitungstabelle. Bleibt dieses Update von einem Nachbarn für eine längere Zeit aus, nimmt der RIP-Router an, dass dieser Nachbar nicht mehr erreicht werden kann. In der Standardkonfiguration beträgt die Wartezeit das sechsfache des Updateintervalls, im Normalfall also 3 Minuten. Diese 3 Minuten werden heruntergezählt, trifft ein Updatepaket ein, so wird der Timer wieder auf 3 Minuten zurückgesetzt. Der momentane Wert des Timers, wird in der Weiterleitungstabelle angezeigt:

Weiterleitungstabelle von alpha:

	Network	Next Hop	Metric	From	Tag	Time
C(i)	192.168.10.0/24	0.0.0.0	1	self	0	
R(n)	192.168.20.0/24	192.168.10.2	2	192.168.10.2	0	01:53
R(n)	192.168.30.0/24	192.168.10.2	2	192.168.10.2	0	01:53
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	
R(n)	192.168.50.0/24	192.168.10.2	3	192.168.10.2	0	01:53

In der oben abgebildete Weiterleitungstabelle ist zu sehen, dass der Timer für die Updates von bravo einen Wert von 1 Minute und 53 Sekunden hat. Dies bedeutet, dass

bereits seit 1 Minute und 7 Sekunden keine Updates mehr von bravo empfangen wurde. Ein oder zwei Updates sind in diesem Zeitraum bereits nicht korrekt angekommen.

Weitere Informationen über die Einstellungen des Update-Mechanismus' können mit folgendem Befehl abgefragt werden:

```
ripd auf alpha:

ripd> show ip rip status
Routing Protocol is "rip"
Sending updates every 30 seconds with +/-50%, next due in 7 seconds
Timeout after 180 seconds, garbage collect after 120 seconds
Outgoing update filter list for all interface is not set
Incoming update filter list for all interface is not set
Default redistribution metric is 1
Redistributing:
Default version control: send version 2, receive any version
Interface Send   Recv    Key-chain
eth1        2    1 2
eth2        2    1 2
Routing for Networks:
192.168.10.0/24
192.168.40.0/24
Routing Information Sources:
Gateway      BadPackets BadRoutes Distance Last Update
192.168.10.2        0          0       120    00:01:07
192.168.40.1        0          0       120    00:00:23
Distance: (default is 120)
```

Die Statusabfrage zeigt unter anderem, dass Router alpha seine Updates in 7 Sekunden verschicken wird. Darüber hinaus wird die momentane Konfiguration der Timer angezeigt. In diesem Fall die Standardwerte von 180 Sekunden für den Timeout und weitere 120 Sekunden bis der Eintrag komplett aus der Weiterleitungstabelle genommen wird. In den beiden vorletzen Zeilen wird angezeigt, wieviel Zeit bereits seit dem letzten erfolgreich empfangenen Update verstrichen ist. Hier ist zu sehen, dass das letzte Update von Bravo vor 1 Minute 7 Sekunden eingegegangen ist.

Routingtabelle alpha kurz vor Timeout:

	Network	Next Hop	Metric	From	Tag	Time
C(i)	192.168.10.0/24	0.0.0.0	1	self	0	
R(n)	192.168.20.0/24	192.168.10.2	2	192.168.10.2	0	00:06
R(n)	192.168.30.0/24	192.168.10.2	2	192.168.10.2	0	00:06
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	
R(n)	192.168.50.0/24	192.168.10.2	3	192.168.10.2	0	00:06

Routingtabelle alpha kurz nach Timeout:

	Network	Next Hop	Metric	From	Tag	Time
C(i)	192.168.10.0/24	0.0.0.0	1	self	0	
R(n)	192.168.20.0/24	192.168.10.2	16	192.168.10.2	0	02:00
R(n)	192.168.30.0/24	192.168.10.2	16	192.168.10.2	0	02:00
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	
R(n)	192.168.50.0/24	192.168.10.2	16	192.168.10.2	0	02:00

Die obenstehenden Momentaufnahmen der Weiterleitungstabellen von Router alpha zeigen jeweils Schnappschüsse kurz vor und kurz nachdem der Timeout wegen der unterbrochenen Verbindung zu Rechner Bravo aufgetreten ist. Nachdem die 3 Minuten des Wartens auf ein Update Paket verstrichen sind, setzt der RIP-Router die Metriken für die entsprechenden Einträge auf 16 hoch. Dies ist per Definition die größte Metrik für RIP. Eine 16 als Metrik ist gleichbedeutend mit unerreichbar. In der Spalte Time der Weiterleitungstabelle wird nun statt des "Timeout" der sogenannte "Garbage Collection Timer" aufgeführt. Wenn dieser Timer abgelaufen ist und das entsprechende Zielnetzwerk immer noch unerreichbar ist, dann wird die komplette Zeile aus der Weiterleitungstabelle gelöscht.

Da in diesem Beispiel nur die Verbindung zwischen alpha und beta unterbrochen wurde, sind immer noch alle Rechner von alpha aus über delta erreichbar. Sobald nun delta seine komplette Weiterleitungstabelle mittels Update an alpha sendet, erkennt der RIP-Daemon von alpha, dass eine alternative Route existiert, deren Metrik kleiner als 16 ist. Die resultierende Weiterleitungstabelle von alpha sieht wie folgt aus:

	Network	Next Hop	Metric	From	Tag	Time
C(i)	192.168.10.0/24	0.0.0.0	1	self	0	
R(n)	192.168.20.0/24	192.168.40.1	3	192.168.40.1	0	02:59
R(n)	192.168.30.0/24	192.168.40.1	2	192.168.40.1	0	02:59
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	
R(n)	192.168.50.0/24	192.168.40.1	4	192.168.40.1	0	02:59

Alle Routen, die vorher als unerreichbar markiert waren, laufen nun als Next Hop über Router delta. Die Metriken sind dementsprechend angepasst.

Der komplette Vorgang wurde von ethereal aufgezeichnet, so dass der "Capture Mode" nun gestoppt werden kann. Da eine große Anzahl an Paketen mitgeschnitten wurden und die Übersicht darunter leidet, sollte die Anzeige auf die wesentlichen Pakete reduziert werden. Dazu bietet ethereal eine Filtermöglichkeit. Der benötigte Filter, um nur die Pakete zwischen alpha und delta anzuzeigen, lautet:

```
ip.src == 192.168.40.2 or ip.src == 192.168.40.1
```

Der Mitschnitt des Netzwerkverkehrs beginnt mit einigen der normalen Update Pakete. Im exemplarischen Durchlauf, dessen Mittschnitt auf der CD zu finden ist, wird 164 Sekunden, nachdem die Aufzeichnung gestartet wurde, ein Paket mit einer 16 als Metrik von alpha nach delta protokolliert.

Update von alpha an delta

192.168.40.2.router > 224.0.0.9.router: RIPv2-resp [items 1]:
{192.168.20.0/255.255.255.0}(16)

Unmittelbar darauf wird ein Update-Paket von alpha gesendet, in dem alle Routen, die vorher über bravo führten als unerreichbar markiert sind (metrik = 16):

Update von alpha an delta

192.168.40.2.router > 224.0.0.9.router: RIPv2-resp [items 4]:
{192.168.10.0/255.255.255.0}(1)
{192.168.20.0/255.255.255.0}(16)
{192.168.30.0/255.255.255.0}(16)
{192.168.50.0/255.255.255.0}(16)

Im darauffolgenden Update Paket, das von delta geschickt wird, bekommt alpha die Informationen über die alternative Route.

Update von delta an alpha:

192.168.40.1.router > 224.0.0.9.router: RIPv2-resp [items 4]:
{192.168.10.0/255.255.255.0->192.168.40.2}(2)
{192.168.20.0/255.255.255.0}(2)
{192.168.30.0/255.255.255.0}(1)
{192.168.50.0/255.255.255.0}(3)

Durch diese Informationen wird alpha gezeigt, dass delta noch alle Netzwerke erreichen kann. Die Weiterleitungstabelle von alpha wird dementsprechend verändert. Die Pakete für die Netzwerke 192.168.20.0/24, 192.168.30.0/24 und 192.168.50.0/24 werden ab jetzt über delta geleitet.

```
     Network          Next Hop      Metric From          Tag  Time
C(i) 192.168.10.0/24  0.0.0.0          1    self          0
R(n) 192.168.20.0/24  192.168.40.1     3    192.168.40.1 0    02:59
R(n) 192.168.30.0/24  192.168.40.1     2    192.168.40.1 0    02:59
C(i) 192.168.40.0/24  0.0.0.0          1    self          0
R(n) 192.168.50.0/24  192.168.40.1     4    192.168.40.1 0    02:59
```

Analog werden in den anderen Routern beta und charly ebenfalls Änderungen an den Weiterleitungstabellen vorgenommen, da ja alpha nicht mehr über beta erreicht werden kann, sondern ebenfalls der Weg über delta genommen werden muss.

Was geschieht nun, wenn beta wieder von alpha direkt erreicht werden kann, also wenn die Firewall wieder deaktiviert wird.

Mit

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x repair@rip3.xml
   -vB
```

wird die Firewall wieder deaktiviert, so dass die Update Pakete zwischen alpha und beta wieder ausgetauscht werden können.

Auch diesmal sollte ethereal genutzt werden, um den Datenverkehr mitzuschneiden.

Als Resultat stellt sich nach einer Update-Periode folgende Weiterleitungstabelle in Router alpha ein:

	Network	Next Hop	Metric	From	Tag	Time
C(i)	192.168.10.0/24	0.0.0.0	1	self	0	
R(n)	192.168.20.0/24	192.168.10.2	2	192.168.10.2	0	03:00
R(n)	192.168.30.0/24	192.168.40.1	2	192.168.40.1	0	02:28
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	
R(n)	192.168.50.0/24	192.168.10.2	3	192.168.10.2	0	03:00

Der Weg von alpha zu den Netzwerken 192.168.20.0 und 192.168.50.0 ist über beta wieder einen Hop kürzer. Deshalb werden Pakete für diese Netzwerke nun wieder über beta geroutet. Die Anzahl Hops von alpha zum Netzwerk 192.168.30.0 ist aber über beta oder delta identisch, weswegen der Algorithmus keine Veranlassung gesehen hat, die vorhandene Route zu ändern. Daher unterscheiden sich die Weiterleitungstabellen vor der Unterbrechung und nach Wiederherstellung der Verbindung, obwohl die Topologie identisch ist.

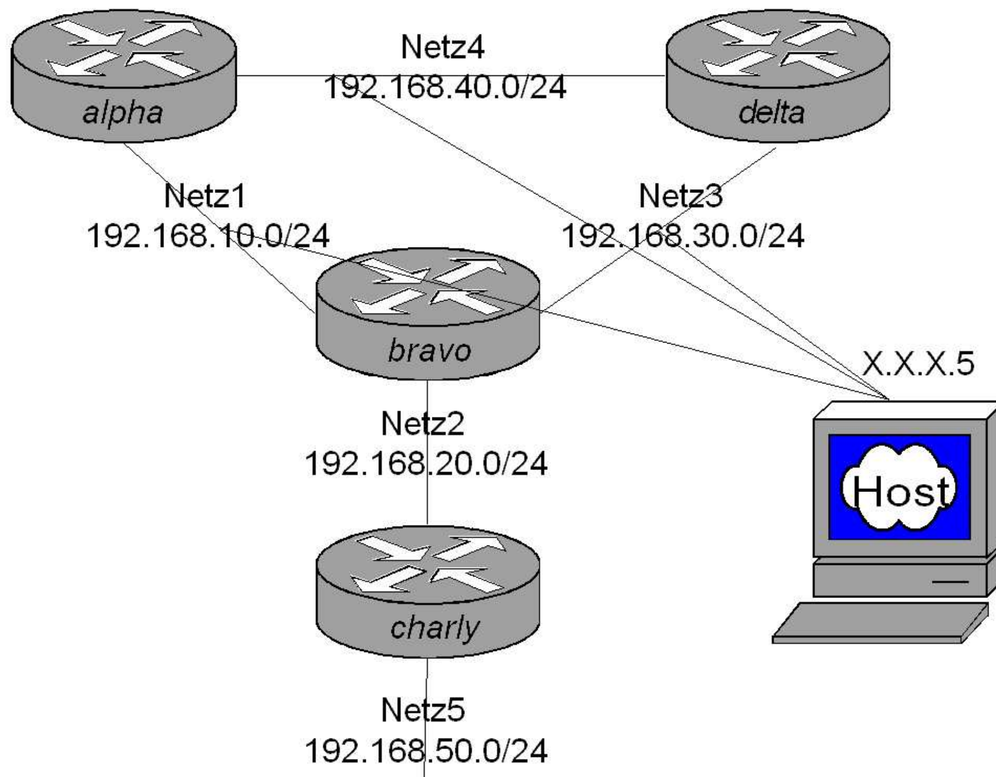
Die Mitschnitte, die sich auf der CD befinden (rip3.dump, rip3b.dump und rip3c.dump) lassen sich mit tcpdump oder ethereal jeweils mit der Option -r einsehen (z.B. ethereal -r rip3c.dump).

4.4 Das "Counting To Infinity" Problem

Der RIP-Algorithmus bemerkt den Wegfall einer Verbindung zu einem direkten Nachbarn durch das Ausbleiben der regelmäßigen Update-Nachrichten von den direkten Nachbarn. Gleichzeitig werden durch die selben Update Nachrichten auch neue bzw. wiederhergestellte Verbindungen bemerkt. In einer solchen Update Botschaft wird der komplette Inhalt der Weiterleitungstabelle des sendenden Routers verschickt. Hat ein RIP-Router so von mehreren alternativen Wegen zu einem Netzwerk erfahren, wird die Möglichkeit bevorzugt, die anhand der einfachen Metrik "Hop Count" die kürzeste zu sein scheint (, wobei die maximale Metrik 16 mit unerreichbar gleichzusetzen ist). Leider birgt der RIP Algorithmus genau hier einen Nachteil, der erst auf den zweiten Blick sichtbar wird. Das folgende Beispiel zeigt, wie ungünstige Verhältnisse im zeitlichen Ablauf zu Fehlverhalten führen können.

Designphase:

Die Netzwerktopologie des letzten Beispiels wird mit einer kleinen Änderung übernommen.



Neben dem Ausfall der Verbindung zwischen beta und charly ist es auch noch erforderlich, dass der Router alpha zu einem gewissen Zeitpunkt ein Update von Router delta nicht erhalten kann. Die Simulation soll so ablaufen, dass die Verbindung zwischen beta und charly unterbrochen wird. Nachdem der Timer von beta abgelaufen ist, wird beta die Unerreichbarkeit von Netzwerk 192.168.50.0 mit der Metrik 16 weitergeben. Genau zu diesem Zeitpunkt ist es notwendig, dass der Router alpha dieses Update nicht zugestellt bekommt, bevor alpha nicht seine noch veraltete Weiterleitungstabelle in einem Update an beta und delta sendet. Es wäre auch denkbar, dass ungünstige Zeitverhältnisse und unterschiedliche Laufzeitverhalten, sowie zeitgleiches Senden der Updates in der Realität dasselbe bewirken. Da dieses Verhalten aber unter den gegebenen Voraussetzungen und mit den hier benutzten Softwarewerkzeugen nicht ohne weiteres reproduzierbar zu simulieren ist, wird in dieser Simulation abstrahiert. Mit der Firewall `iptables` soll verhindert werden, dass die Updates von beta am Router alpha eintreffen können. Dazu werden die in alpha eingehenden Pakete für einen kurzen Zeitraum (ca. ein Update-Intervall) herausgefiltert. Nachdem dann alpha unwissentlich seine veraltete Weiterleitungstabelle verbreitet hat, kommt es in der Schleife alpha-beta-delta zu einem gegenseitigen Hochzählen der Metrik, die bis ins Unendliche reichen würde, wäre nicht genau aus diesem Grund das künstliche Maximum von 16 definiert worden. Diese Situation wird "counting to infinity"-Problem genannt. Da das Hochzählen der Metriken unter Umständen sehr

rasch geschehen kann, sollen auch hier ethereal und tcpdump alle Pakete mitschneiden, die zwischen alpha, beta und delta ausgetauscht werden.

Implementierungsphase:

Da die Netzwerktopologie fast identisch bleibt, sind nur einige kleinere Veränderungen an der XML-Beschreibung notwendig.

Die erste Veränderung zur Beschreibung des vorangegangenen Beispiels findet sich in Zeile 6. Dort wird der Name für die neue Simualtion festgelegt.

```
<simualtion_name>rip4</simulation_name>
```

In Zeile 12 wird der Pfad für die Konfigurationsdateien, entsprechend dem Namen der Simualtion, auf das richtige Verzeichnis gesetzt.

```
<basedir>/usr/local/share/vnuml/examples/rip4</basedir>
```

An der Beschreibung für alpha müssen Veränderungen vorgenommen werden, da die Verbindung zu alpha kurzzeitig unterbrochen werden soll. Dazu wird das Skript cutalpha.sh durch die Befehlssequenz cutalpha gestartet. Dieses Skript liegt oberhalb des Basispfades im Verzeichnis scripts/alpha. Zum Wiederherstellen der Verbindung soll das Skript repairalpha.sh genutzt werden, dass durch die Ausführungssequenz repairalpha durch den VNUML-Parser gestartet wird.

```
<vm name="alpha">
  ...
  <filetree when="cutalpha" root="/usr/local/bin">scripts/alpha</filetree>
  <exec seq="cutalpha" type="verbatim">cutalpha.sh</exec>
  <exec seq="repairalpha" type="verbatim">repairalpha.sh</exec>
  ...
</vm>
```

Das Skript cutalpha.sh konfiguriert die Firewall `iptables` auf dem Rechner alpha, so dass alle ankommenden Pakete von beta und delta verworfen werden. Andersherum bleibt aber eine Verbindung von alpha zu beta oder delta offen. Nach dem Start dieses Skriptes können von alpha nur noch Pakete erfolgreich versandt aber nicht mehr empfangen werden. Dadurch werden die, zum erfolgreichen Ablauf der Simualtion notwendigen, Eingriffe minimal gehalten. Es wird versucht nicht mehr Pakete als nötig verschwinden zu lassen.

cutalpha.sh:

```
#!/bin/bash

iptables -I INPUT -s 192.168.10.2 -j DROP

iptables -I INPUT -s 192.168.40.1 -j DROP
```

Das Skript repairalpha.sh hebt die Filterregeln, die durch cutalpha entstanden sind wieder auf.

repairalpha.sh:

```
#!/bin/bash

iptables -D INPUT -s 192.168.40.1 -j DROP

iptables -D INPUT -s 192.168.10.2 -j DROP
```

Die aus dem letzten Beispiel bekannten Skripte cutline.sh und repair.sh werden in dieser Simualtion wieder benutzt, um die Verbindung zwischen den Rechnern beta und charly zu unterbrechen. Zu diesem Zweck werden die Skripte auf Rechner beta durch die bekannten Ausführungssequenzen mit dem Parameter eth2 gestartet.

```
<vm name="bravo">
  ...
  <filetree when="cut" root="/usr/local/bin">scripts/beta</filetree>
  <exec seq="cut" type="verbatim">cutline.sh eth2</exec>
  <exec seq="repair" type="verbatim">repair.sh eth2</exec>
  ...
</vm>
```

Die letzen Änderungen an der XML-Beschreibung werden an der Konfiguration des Hostrechners gemacht. Da die später auftretende Schleife nur die Netzwerke 192.168.10.0, 192.168.30.0 und 192.168.40.0 betrifft soll auch nur der Netzwerkverkehr dieser Netze durch `ethereal` aufgezeichnet werden. Deshalb wird der Host nur in diese Netze eingebunden.

```
<host>
  <hostif net="Netz1">
    <ipv4 mask="255.255.255.0">192.168.10.5</ipv4>
  </hostif>
```

```
  <hostif net="Netz3">
    <ipv4 mask="255.255.255.0">192.168.30.5</ipv4>
  </hostif>
  <hostif net="Netz4">
    <ipv4 mask="255.255.255.0">192.168.40.5</ipv4>
  </hostif>
</host>
```

Ausführungsphase:

Wie in allen bisherigen Beispielen werden die virtuellen Rechner gestartet, indem der Parser mit der Option -t aufgerufen wird.

```
host:# cd /usr/local/share/vnuml/examples/
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -t rip4.xml
   -vB
```

Die Routingsoftware kann gestartet werden, sobald alle virtuellen Maschinen vollständig hochgefahren sind. Dies geschieht ebenfalls mit der bekannten Ausführungssequenz start.

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x start@rip4.xml
   -vB
```

Um sicher zu stellen, dass die Router korrekt funktionieren und das Netzwerk bereits konvergent ist, wird die Routingtabelle von alpha aufgerufen. Bisher wurde die Weiterleitungstabelle immer direkt im virtuellen Terminal von `Quagga` angezeigt. Alternativ kann auch die echte Weiterleitungstabelle des Linux-Kernels angezeigt werden. Da der RIP-Daemon über den zebra-Daemon mit dem Kernel kommuniziert, sollten die Weiterleitungstabellen übereinstimmen.

```
alpha:~# route
Kernel IP routing table
Destination   Gateway       Genmask         Flags Metric Ref Use
   Iface
192.168.1.144    *          255.255.255.252   U      0    0   0  eth0
192.168.20.0  192.168.10.2  255.255.255.0     UG     2    0   0  eth1
192.168.50.0  192.168.10.2  255.255.255.0     UG     3    0   0  eth1
192.168.30.0  192.168.10.2  255.255.255.0     UG     2    0   0  eth1
```

```
192.168.10.0     *           255.255.255.0    U      0    0    0  eth1
192.168.40.0     *           255.255.255.0    U      0    0    0  eth3
```

Die Weiterleitungstabelle zeigt, dass alle konfigurierten Netzwerke von alpha erreicht werden können.

Nun wird die Verbindung zwischen bravo und charly unterbrochen. Dies geschieht über die Ausführungssequenz “cut”.

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x cut@rip4.xml
   -vB
```

In der internen Weiterleitungstabelle des RIP-Daemons von Router bravo wird der Updatetimer angezeigt, der ab dem Zeitpunkt der Unterbrechung nicht wieder zurückgesetzt wird. Erst nachdem die Zeitspanne von 3 Minuten abgelaufen ist, wird charly als unerreichbar markiert und diese Information an die anderen Router weitergegeben. Genau diese Information soll alpha aber vorenthalten werden, dazu ist es notwendig die Ausführungssequenz “cutalpha” auszuführen, kurz bevor das Update von bravo gesendet wird. Ein guter Zeitpunkt zum Ausführen von “cutalpha” ist, wenn der Timer von Router bravo bei zirka 20 Sekunden angelangt ist.

Weiterleitungstabelle von bravo 20 Sekunden vor Timeout:

```
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop      Metric From         Tag Time
C(i) 192.168.10.0/24 0.0.0.0         1     self          0
C(i) 192.168.20.0/24 0.0.0.0         1     self          0
C(i) 192.168.30.0/24 0.0.0.0         1     self          0
R(n) 192.168.40.0/24 192.168.10.1    2     192.168.10.1 0 02:52
R(n) 192.168.50.0/24 192.168.20.2    2     192.168.20.2 0 00:20
```

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x cutalpha@rip4.xml
   -vB
```

Nachdem Router bravo die Unterbrechung bemerkt hat und die Updates verschickt wurden, muss gewartet werden, bis Router alpha seine veralteten und damit falschen Informationen weitergibt.

In der Weiterleitungstabelle von bravo wird nach dem Timeout, Netzwerk 192.168.50.0./24 noch als unerreichbar geführt:

```
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop      Metric From         Tag Time
C(i) 192.168.10.0/24 0.0.0.0         1     self          0
C(i) 192.168.20.0/24 0.0.0.0         1     self          0
C(i) 192.168.30.0/24 0.0.0.0         1     self          0
R(n) 192.168.40.0/24 192.168.10.1    2     192.168.10.1 0 02:47
R(n) 192.168.50.0/24 192.168.20.2   16     192.168.20.2 0 01:59
```

Die Firewall blockt nur die bei alpha eingehenden Pakete ab, die von alpha gesendeten Pakete passieren iptables ungehindert. Sobald alpha die falschen Informationen über die vermeintliche Erreichbarkeit verbreitet, baut auch Router bravo die Tabelle um, da alpha ja immer noch der Meinung ist, dass er das bereits abgeschnittene Netzwerk erreichen kann.

Weiterleitungstabelle von bravo, nachdem alpha ein falsches Update sendete:

```
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop      Metric From         Tag Time
C(i) 192.168.10.0/24 0.0.0.0         1     self          0
C(i) 192.168.20.0/24 0.0.0.0         1     self          0
C(i) 192.168.30.0/24 0.0.0.0         1     self          0
R(n) 192.168.40.0/24 192.168.10.1    2     192.168.10.1 0 02:23
R(n) 192.168.50.0/24 192.168.30.2    5     192.168.30.2 0 02:39
```

Nun hat bravo die Weiterleitungstabelle abgeändert, obwohl das Netzwerk 192.168.50.0/24 nicht mehr erreicht werden kann, existiert ein falscher Eintrag. Nach diesem Eintrag soll das Netzwerk 192.168.50.0/24 über den Next Hop 192.168.30.2 zu erreichen sein. Dieses Interface gehört zu delta. Ein Blick in die Weiterleitungstabelle von delta zeigt, dass hier ein Eintrag existiert, dem zu Folge das Netzwerk 192.168.50.0/24 über den Next Hop 192.168.40.2, also alpha, erreicht werden könnte.

Weiterleitungstabelle delta:

```
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop      Metric From         Tag Time
R(n) 192.168.10.0/24 192.168.30.1   2      192.168.30.1 0   02:35
R(n) 192.168.20.0/24 192.168.30.1   2      192.168.30.1 0   02:35
C(i) 192.168.30.0/24 0.0.0.0        1      self         0
C(i) 192.168.40.0/24 0.0.0.0        1      self         0
R(n) 192.168.50.0/24 192.168.40.2   4      192.168.40.2 0   02:32
```

Schaltet man nun die Filterregeln, die Rechner alpha betreffen, aus, entsteht eine rege Kommunikation, die das Hochzählen der Metriken innerhalb der Schleife verursacht. Um diese zeitlich rasch ablaufende Kommunikation analysieren zu können, wird der Vorgang mitgeschnitten.

```
host:/usr/local/share/vnuml/examples/# tcpdump -w rip4.dump -i any
   -s0 -vv
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x repairalpha@rip4.xml
   -vB
```

Nachdem Router alpha sein erstes Update-Paket gesendet hat, wird das "counting to infinity" ausgelöst. Durch mehrmaliges Abrufen der Weiterleitungstabellen in den nächsten 10-15 Sekunden, können einige Momentaufnahmen, die das Hochzählen zeigen, gemacht werden. Ist der Maximalwert von 16 erreicht, kann tcpdump gestoppt werden. Dies geschieht durch gleichzeitiges Drücken der Tasten <Strg> und <c>. Anschließend kann der Mitschnitt wieder wahlweise mit tcpdump oder ethereal angeschaut werden.

Im Folgenden werden die für das "counting to infinity" relevanten Ausschnitte der Aufzeichnung erläutert.

192.168.10.1.router > 224.0.0.9.router: RIPv2-resp [items 1]:
{192.168.50.0/255.255.255.0->192.168.10.2}(6)

192.168.40.2.router > 224.0.0.9.router: RIPv2-resp [items 1]:
{192.168.50.0/255.255.255.0}(6)

Alpha sendet auf allen Schnittstellen ein Update, in dem angegeben wird, dass das Netzwerk 192.168.50.0/24 über alpha mit einer Metrik von 6 erreichbar ist. Dieses Update ist somit eine Reaktion auf die Mitteilung von alpha mit der Metrik 6.

192.168.30.2.router > 224.0.0.9.router: RIPv2-resp [items 1]:
{192.168.50.0/255.255.255.0}(7)

192.168.40.1.router > 224.0.0.9.router:RIPv2-resp [items 1]:
{192.168.50.0/255.255.255.0->192.168.40.2}(7)

Die Reaktion von delta besteht aus diesen beiden Paketen. Darin wird mitgeteilt, dass sich die Entfernung zu 192.168.50.0/24 über alpha auf eine Metrik von 7 erhöht hat.

192.168.10.2.router > 224.0.0.9.router:RIPv2-resp [items 1]:
{192.168.50.0/255.255.255.0}(8)

192.168.30.1.router > 224.0.0.9.router:RIPv2-resp [items 1]:
{192.168.50.0/255.255.255.0->192.168.30.2}(8)

Da Router bravo glaubt, das Netz 192.168.50.0/24 über delta erreichen zu können, muss auch bravo entsprechend reagieren und die Metrik dementsprechend anpassen. Als Reaktion werden die obenstehenden "triggered updates" auf allen Schnittstellen gesendet.

192.168.10.1.router > 224.0.0.9.router:RIPv2-resp [items 4]:
{192.168.20.0/255.255.255.0->192.168.10.2}(2)
{192.168.30.0/255.255.255.0->192.168.10.2}(2)
{192.168.40.0/255.255.255.0}(1)
{192.168.50.0/255.255.255.0->192.168.10.2}(9)

192.168.40.2.router > 224.0.0.9.router:RIPv2-resp [items 4]:
{192.168.10.0/255.255.255.0}(1)
{192.168.20.0/255.255.255.0}(2)
{192.168.30.0/255.255.255.0}(2)
{192.168.50.0/255.255.255.0}(9)

Der Zyklus schließt sich mit den beiden obigen Update Paketen von Router alpha, denn dieser ist immer noch überzeugt, dass er das Netzwerk 192.168.50.0/24 über Router bravo erreichen kann. Da dieser jedoch mitteilte, dass sich die Anzahl Hops zum Ziel vergrößert hat, muss auch alpha wiederum nachbessern und erhöht die Metrik entsprechend. Dadurch wird wiederum eine Reaktion von delta angestoßen, worauf auch bravo wieder reagieren muss. Dieses Hochzählen durch die Schleife in der Topologie wird solange fortgesetzt, bis der festgesetzte Maximalwert von 16 Hops erreicht wird. Da diese Metrik von 16 von den Routern als unerreichbar interpretiert wird, stimmt nun die Realität und die Abbildung der Topologie in den Weiterleitungstabellen wieder überein. Jedoch wurde durch das Eintreten des "counting to infinity" mehr Zeit als nötig verloren bis dieser korrekte Zustand erreicht wurde. Der Zeitraum wird jedoch durch den Mechanismus der "triggered updates" gegenüber dem Warten auf die normalen periodischen Updates erheblich verkürzt.

5 OSPF

Als weiteres Intra-Domain Routing Protokoll wird im Folgenden das weit verbreitete OSPF (Open Shortest Path First) untersucht. Dieses Routingprotokoll gehört zu den Link-State-Algorithmen und verfolgt damit einen anderen Ansatz, um Pakete dynamisch weiterzuleiten. Die folgenden Beispiele erläutern einige Kernaspekte der OSPF-Router.

Im ersten Abschnitt wird die Initialisierungsphase nach dem gleichzeitigen Start mehrerer Router beobachtet und analysiert. Im zweiten Abschnitt wird gezeigt, wie ein Ausfall einer Verbindung bemerkt wird und welche Reaktionen wieder zu einer korrekten Weiterleitungstabelle führen. Der letzte Abschnitt dieses Kapitels bietet einen exemplarischen Ausblick auf die Hierarchisierungsmöglichkeiten von OSPF, die in der praktischen Anwendung des Protokolls einen hohen Stellenwert haben.

Obwohl sich dieses Kapitel mit einem gänzlich neuen Protokoll beschäftigt, sollte dennoch der Inhalt von Kapitel 4 bekannt sein. Dort wird ein Schwerpunkt auch auf die Benutzung von VNUML gelegt, ohne deren Kenntnis Verständnisprobleme auftauchen könnten. Einige der dort erklärten Schritte wiederholen sich bei jedem Szenario fast identisch und werden fortan vorausgesetzt.

5.1 Beispiel 1: Kaltstart eines OSPF-Szenarios (Initialisierungsphase)

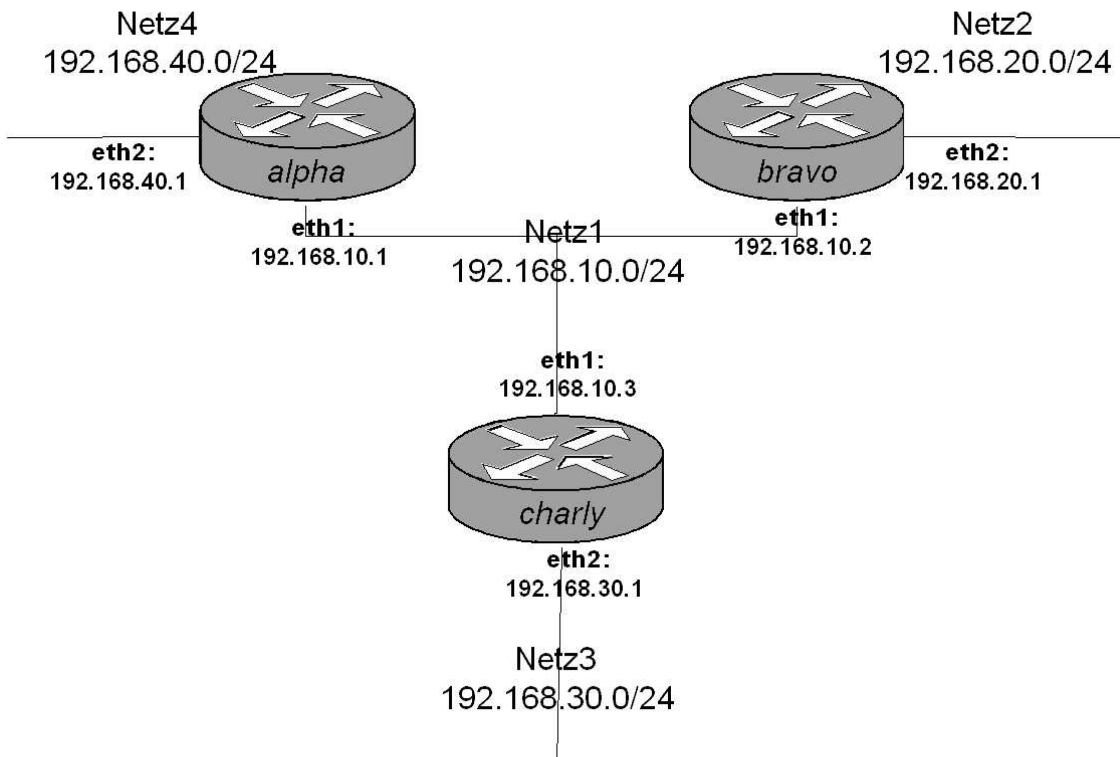
Zu Beginn eines neuen Netzwerkszenarios sind standardmäßig noch alle Router ausgeschaltet. Ausgehend von der Situation, dass alle virtuellen Maschinen bereits hochgefahren sind, werden im nächsten Schritt alle Router fast gleichzeitig gestartet. Dieser Ausgangspunkt ist bei fast allen Szenarien, die mit VNUML simuliert werden zu Beginn anzutreffen. Für ungeduldige VNUML Neulinge, die mit Hilfe von Quagga OSPF-Szenarien untersuchen, birgt diese Ausgangssituation jedoch die Gefahr, dass die Initialisierungsphase der OSPF-Router durchschnittlich etwa 40-50 Sekunden dauert. Vor Ablauf dieser Zeit können die OSPF-Router, auch in sehr kleinen Topologien, keine Weiterleitungstabelle bieten. Das folgende Beispiel soll daher zeigen weshalb der Kaltstart

eines OSPF-Szenarios länger dauert als erwartet und wie dieser abläuft, obwohl OSPF den Ruf hat wesentlich schneller zu sein als RIP. Ein vergleichbares RIP-Szenario konvergiert im Vergleich dazu bereits wenige Sekunden nach Kaltstart. In einem größeren Szenario benötigt RIP zwar ebenfalls länger bis zur endgültigen Konvergenz, allerdings wird bis dahin eine unvollständige Weiterleitungstabelle als "Provisorium" benutzt. Dadurch "sieht" der Nutzer, dass die RIP-Router ihre Arbeit aufgenommen haben.

Da es für den Link-State Algorithmus allerdings von großer Bedeutung ist, dass alle Link-State-Datenbanken synchronisiert sind, um eine korrekte Weiterleitung zu ermöglichen, wird die Erstellung der Weiterleitungstabelle verzögert.

Designphase:

Die Topologie in diesem Beispiel setzt sich aus drei OSPF-Routern zusammen, die über das gemeinsame Netzwerk 192.168.10.0/24 verbunden sind. Jeder der Router hat darüber hinaus die Verantwortung, ein weiteres Netzwerk erreichbar zu machen.



In der initialen Startphase werden in sogenannten "Broadcast Networks" der "Designated Router" und der "Backup Designated Router" bestimmt. Da in dieser Arbeit ausschließlich mit virtuellen Ethernets gearbeitet wird, liegen auch allen OSPF Beispielen "Broadcast Networks" zugrunde. Darüber hinaus wird bereits hier sichtbar, dass beim

OSPF-Protokoll eine Vielzahl von verschiedenen Nachrichtenformaten ausgetauscht werden.

Implementierungsphase:

Die Ableitung der XML-Beschreibung aus der Skizze der Topologie, sollte für einen Leser, des Kapitels 4 keine Schwierigkeit darstellen. Es werden nur bereits bekannte Sprachelemente genutzt.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
<vnuml>
<global>
<version>1.5.0</version>
<simulation_name>ospf1</simulation_name>
<ssh_key>/root/.ssh/identity.pub</ssh_key>
<automac/>
<ip_offset>100</ip_offset>
<host_mapping/>
<shell>/bin/sh</shell>
</global>

<net mode="uml_switch" name="net1"/>
<net mode="uml_switch" name="net2"/>
<net mode="uml_switch" name="net3"/>
<net mode="uml_switch" name="net4"/>

<!-- Nodes -->

<vm name="alpha" >
  <filesystem type="cow" >
    /usr/local/share/vnuml/filesystems/root_fs_tutorial

  </filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
  <boot>
    <con0>xterm</con0>
  </boot>
  <if id="1" net="net1" >
```

```
      <ipv4 mask="255.255.255.0" >192.168.10.1</ipv4>
    </if>
    <if id="2" net="net4" >
      <ipv4 mask="255.255.255.0" >192.168.40.1</ipv4>
    </if>
    <forwarding type="ip" />
    <filetree root="/usr/local/etc" when="start">
      /usr/local/share/vnuml/examples/ospf1/alpha
    </filetree>
    <exec seq="start" type="verbatim">
      zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601
    </exec>
    <exec seq="start" type="verbatim">
      ospfd -f /usr/local/etc/alpha.conf -d -P 2604
    </exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

  <vm name="bravo" >
    <filesystem type="cow" >
      /usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
    <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
    <boot>
      <con0>xterm</con0>
    </boot>
    <if id="1" net="net1" >
      <ipv4 mask="255.255.255.0" >192.168.10.2</ipv4>
    </if>
    <if id="2" net="net2" >
      <ipv4 mask="255.255.255.0" >192.168.20.1</ipv4>
    </if>
    <forwarding type="ip" />
    <filetree root="/usr/local/etc" when="start">
      /usr/local/share/vnuml/examples/ospf1/bravo
```

```
  </filetree>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601
  </exec>
  <exec seq="start" type="verbatim">
    ospfd -f /usr/local/etc/bravo.conf -d -P 2604
  </exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ospfd</exec>
</vm>

<vm name="charly" >
  <filesystem type="cow" >
    /usr/local/share/vnuml/filesystems/root_fs_tutorial
  </filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
  <boot>
    <con0>xterm</con0>
  </boot>
  <if id="1" net="net1" >
    <ipv4 mask="255.255.255.0" >192.168.10.3</ipv4>
  </if>
  <if id="2" net="net3" >
    <ipv4 mask="255.255.255.0" >192.168.30.1</ipv4>
  </if>
  <forwarding type="ip" />
  <filetree root="/usr/local/etc" when="start">
    /usr/local/share/vnuml/examples/ospf1/charly
    </filetree>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601
  </exec>
  <exec seq="start" type="verbatim">
    ospfd -f /usr/local/etc/charly.conf -d -P 2604
  </exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
```

```
  <exec seq="stop" type="verbatim">killall ospfd</exec>
</vm>

</vnuml>
```

Den zweiten Teil der Implementierung bilden die Konfigurationsdateien für die OSPF-Router. Die einfachste Art, ein OSPF Netzwerk zu konfigurieren, besteht lediglich aus dem `network` Kommando. Stellvertretend für alle drei Konfigurationen wird die von alpha detailiert vorgestellt.

alpha.conf:

```
[ 1]!
[ 2]hostname ospfd
[ 3]password zebra
[ 4]log stdout
[ 5]!
[ 6]router ospf
[ 7]  router-id 1
[ 8]  network 192.168.10.0/24 area 1
[ 9]  network 192.168.40.0/24 area 1
[10]line vty
[11]!
```

Die Zeilen 1 und 5 bestehen nur aus dem Ausrufezeichen, dass als Kennzeichnung für einen Kommentar steht. Im vorliegenden Fall dienen diese beiden Zeilen zur Auflockerung und Verbesserung der Lesbarkeit. In den Zeilen 2-4 werden allgemeine Einstellungen vorgenommen. Die hier gezeigten Einstellungen wurden aus den Beispielkonfigurationen der Quagga Autoren übernommen, für weitere Möglichkeiten verweise ich auf das Handbuch von Quagga.
In Zeile 6 beginnt die eigentliche Konfiguration des OSPF Routers. Zeile 7 legt die sogenannte Router ID fest, durch diesen eindeutigen Bezeichner werden die OSPF Router später unterschieden. Wenn die Router ID nicht explizit angegeben wird, wird die höchste IP-Addresse einer konfigurierten Netzwerkschnittstelle als Router ID herangezogen. Aus Gründen der Fehlervermeidung und der Stabilität sollte die Router ID jedoch explizit vergeben werden oder per “Loopback-Interface“ festgelegt werden.

Die network Befehle folgen in Zeile 8 und 9. Durch diese Zeilen , wird der OSPF-Router angewiesen die Routingtätigkeiten für die Netzwerke 192.168.10.0/24 und 192.168.40.0/24

zu übernehmen. Der Zusatz `area 1` legt fest, dass beide Netzwerke in dem Bereich 1 liegen. Auf die Möglichkeiten der Hierachisierung von OSPF -Netzwerken in verschiedenen Bereiche (areas), wird später in diesem Kapitel noch näher eingegangen.

Die Unterschiede in den Konfigurationen der einzelnen Router bestehen nur in den `network` Kommandos, deren Form sich aber direkt aus einem Blick auf die Netzwerk-Skizze ableiten lässt.

Ausführungsphase:

Das Starten des VNUML-Parsers und das Hochfahren der virtuellen Rechner wird mit dem Befehl `vnumlparser.pl -t ospf1.xml -vB` erreicht. Nachdem die virtuellen Knoten komplett hochgefahren sind, können die Router mit

```
vnumlparser.pl -x start@ospf1.xml -vB
```

gestartet werden. Im Folgenden wird der Netzwerkverkehr im Netz 192.168.10.0/24, der aus dem fast gleichzeitigen Start der Router resultiert, analysiert. Der Netzwerkverkehr wird dazu mit dem Sniffer tcpdump mitgeschnitten.

Alpha beginnt die Kommunkation und sendet ein erstes Hello-Paket, ohne zu wissen wer antworten wird.:

```
[1]192.168.10.1 > 224.0.0.5:
[2]  OSPFv2-hello 44: rtrid 0.0.0.1 area 0.0.0.1 E mask 255.255.255.0
[3]  int 10 pri 1 dead 40
```

Der Mitschnitt des ersten Hello-Paketes besteht aus drei Zeilen. In der ersten Zeile wird mitgeteilt, dass die Nachricht vom Interface 192.168.10.1 an die Multicast Adresse 224.0.0.5 geschickt wird. Diese Multicast Adresse ist die sogenannte AllSPFRouter-Adresse unter der alle Shortest-Path-First Router im Netzwerk zu erreichen sind.
Zeile 2 und 3 des Mittschnittes zeigen den Inhalt des Paketkopfs. Dabei wird zuerst mitgeteilt, dass es sich um ein Hello-Paket des Protokolls OSPF Version 2 handelt. Die weiteren Inhalte in der gleichen Reihenfolge wie im Mitschnitt sind die Router ID des sendenden Routers, die Bereichsnummer, die Netzwerkmaske, das Intervall für den Versand der Hello-Pakete, die Priorität des Routers und das sogenannte Dead-Intervall, beide Zeitangaben jeweils in Sekunden.

Beta erkennt durch das erste Paket den Absender (alpha) als Nachbarn und hängt diese Information an sein Hello-Paket an. Erkennt ein Router sich selbst in einem Hello-Paket eines anderen als Nachbar wieder, so weiß er, dass Hin- und Rückweg zu einem direkt verbundenen Router existieren und damit eine Nachbarschaftsbeziehung möglich ist.

```
192.168.10.2 > 224.0.0.5:
  OSPFv2-hello 48: rtrid 0.0.0.2 area 0.0.0.1 E mask 255.255.255.0
  int 10 pri 1 dead 40
  nbrs 0.0.0.1
```

Auch charly wertet die Nachricht von alpha aus und erkennt diesen als Nachbarn:

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-hello 48: rtrid 0.0.0.3 area 0.0.0.1 E mask 255.255.255.0
  int 10 pri 1 dead 40
 nbrs 0.0.0.1
```

Charly schiebt aber sofort ein Hello nach, in dem auch beta als Nachbar erkannt wird. Diese Infomation war beim Paket zuvor noch nicht ausgewertet:

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-hello 52: rtrid 0.0.0.3 area 0.0.0.1 E mask 255.255.255.0
  int 10 pri 1 dead 40
  nbrs 0.0.0.1 0.0.0.2
```

Alpha hat die Hellos von beta und charly bekommen und weiß nun, dass beide zu seinen Nachbarn gehören:

```
192.168.10.1 > 224.0.0.5:
  OSPFv2-hello 52: rtrid 0.0.0.1 area 0.0.0.1 E mask 255.255.255.0
  int 10 pri 1 dead 40
 nbrs 0.0.0.2 0.0.0.3
```

Auch beta hat durch die Kommunikation erfahren, dass charly ebenfalls ein Nachbar ist:

```
192.168.10.2 > 224.0.0.5:
  OSPFv2-hello 52: rtrid 0.0.0.2 area 0.0.0.1 E mask 255.255.255.0
  int 10 pri 1 dead 40
  nbrs 0.0.0.1 0.0.0.3
```

Jeder der drei aktiven Routern hat zu diesem Zeitpunkt durch die Hello-Pakete Bekanntschaft mit der kompletten Nachbarschaft gemacht. Alle bekannten Nachbarn werden an die Hello-Pakete angehangen. Die erste Phase des Kennenlernens ist nun abgeschlossen. Sicherheitshalber können die Router aber erst zur nächsten Phase übergehen, wenn das Dead-Intervall abgelaufen ist. Kommen in dieser Zeit keine neuen Nachbarn dazu, wird davon ausgegangen, dass alle vollständig erkannt wurden. Da das Dead-Intervall 40 Sekunden beträgt, ist eine schnellere Konvergenzzeit im Initialfall unmöglich.
Jeder Router kennt nun seine Nachbarn, mit denen er an einem Netzwerk hängt. Diese Informationen ermöglichen in Netzwerken, in denen ein Multicast möglich ist, das Wählen des "Designated Router" (DR) und des "Backup Designated Router" (BDR). In diesem Fall wird der Router mit der höchsten Priorität zum DR und der mit der zweithöchsten Priorität zum BDR. Sind die Prioritäten gleich, bildet die Router ID die entscheidende Instanz. Dann wird der Router mit der höchsten Router-ID (also charly) zum DR und der mit der zweithöchsten RID (beta) zum BDR ernannt. Diese Router sind unter einer speziellen Multicast Adresse, der sogenannten AllDRouters 224.0.0.6, erreichbar. Die Designated Router bilden mit den normalen Router logische Verknüpfungen, die sogenannten "adjacences". Nur zwischen Routern mit diesen Verknüpfungen werden "Linkstate Advertisements" ausgetauscht.

Alpha schickt ein erstes LSA Typ-1 an den frisch gekürten DR. Die Typ-1 LSAs enthalten Informationen über die angeschlossenen Netzwerke eines einzelnen Routers. In dem folgenden Paket teilt der Router mit RID 0.0.0.1 (alpha) dem DR mit, dass er direkt an die Netzwerke 192.168.10.0/24 und 192.168.40.0/24 angeschlossen ist.

```
[1]192.168.10.1 > 224.0.0.6:
[2]  OSPFv2-ls_upd 76: rtrid 0.0.0.1 area 0.0.0.1
[3]  { E S 80000002 age 1 rtr 0.0.0.1
[4]    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
[5]    { net 192.168.40.0 mask 255.255.255.0 tos 0 metric 10 }
[6]  }
```

Zeile 1 und Zeile 2 des Mitschnittes des LSA Typ-1 bilden wieder Kopfdaten wie Absender, Empfänger und Art der Nachricht (Linkstate Update). In Zeile 3 beginnt das einzige LSA. Es wird die Sequenznummer 80000002, sowie das Alter und der für den Ursprung des LSAs verantwortliche Router (Router ID 0.0.0.1) mitgeteilt. Der eigentliche Informationsinhalt des LSAs verteilt sich auf zwei Einträge in den Zeilen 4 und 5. In Zeile 4 teilt alpha mit, dass er die Funktion eines OSPF-Routers für das Netzwerk 192.168.10.0 mit der Netzwerkmaske 255.255.255.255.0 ausführt. Der Type of Service (TOS) ist 0 und die OSPF Metrik von alpha zum Erreichen des propagierten Netzes beträgt 10. Analog dazu beschreibt Zeile 5 das zweite Netzwerk, an das alpha angeschlossen ist.

Zunächst teilt charly (RID 0.0.0.3) allen anderen Routern mit, dass er direkt an die Netze 192.168.10.0/24 und 192.168.30.0/24 angeschlossen ist.

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-ls_upd 76: rtrid 0.0.0.3 area 0.0.0.1
  { E S 80000002 age 1 rtr 0.0.0.3
    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
    { net 192.168.30.0 mask 255.255.255.0 tos 0 metric 10 }#
  }
```

Erst dann wird er seiner Funktion als DR gerecht und verteilt die Information, die er im ersten LSA von alpha bekommen hat. Durch dieses Paket wird allen OSPF-Routern (224.0.0.5) mitgeteilt, dass alpha (RID 0.0.0.1) an 192.168.10.0/24 und 192.168.40.0/24 angeschlossen ist.

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-ls_upd 76: rtrid 0.0.0.3 area 0.0.0.1
  { E S 80000002 age 2 rtr 0.0.0.1
    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
    { net 192.168.40.0 mask 255.255.255.0 tos 0 metric 10 }
  }
```

Im Folgenden sendet charly das erste LSA Typ-2 Paket an alle OSPF-Router. In einem Typ-2 LSA wird mitgeteilt, welche Router in einem Netzwerk arbeiten. Nur der DR verschickt Typ-2 LSAs. Da bis zu diesem Zeitpunkt nur alpha und charly über den DR kommuniziert haben, werden auch nur diese beiden als Router bekannt gemacht, beta fehlt noch in der Auflistung. Typ-2 LSAs bieten eine Zusammenfassung von Informationen über ein Netz.

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-ls_upd 60: rtrid 0.0.0.3 area 0.0.0.1
  { E S 80000001 age 1 net dr 0.0.0.3
    if 192.168.10.3 mask 255.255.255.0
    rtrs 0.0.0.1 0.0.0.3
  }
```

Beta reicht die fehlenden Informationen an den DR nach.

```
192.168.10.2 > 224.0.0.6:
  OSPFv2-ls_upd 76: rtrid 0.0.0.2 area 0.0.0.1
  { E S 80000002 age 1 rtr 0.0.0.2
    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
    { net 192.168.20.0 mask 255.255.255.0 tos 0 metric 10 }
  }
```

Alpha schickt eine LSA_Acknowledge an den DR, in dem er bestätigt, dass die Informationen der LSAs erhalten wurden.

```
192.168.10.1 > 224.0.0.6:
  OSPFv2-ls_ack 64: rtrid 0.0.0.1 area 0.0.0.1
    { E S 80000002 age 1 rtr 0.0.0.3 }
    { E S 80000001 age 1 net dr 0.0.0.3 if 192.168.10.3 }
```

Beta hat bisher keine Bestätigung geschickt, dass die LSAs erhalten wurden. Weil es für einen funktionierenden OSPF-Algorithmus unerlässlich ist, dass alle Knoten dieselben Informationen besitzen, wiederholt der DR die beiden LSAs, die von beta nicht quittiert wurden. Im Zweifelsfall würde dies so oft geschehen, bis alle Router die Quittung geschickt haben oder das Dead-Intervall abläuft.

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-ls_upd 76: rtrid 0.0.0.3 area 0.0.0.1
  { E S 80000002 age 1 rtr 0.0.0.3
    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
    { net 192.168.30.0 mask 255.255.255.0 tos 0 metric 10 }
  }
192.168.10.3 > 224.0.0.5:
```

```
  OSPFv2-ls_upd 76: rtrid 0.0.0.3 area 0.0.0.1
  { E S 80000002 age 2 rtr 0.0.0.1
    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
    { net 192.168.40.0 mask 255.255.255.0 tos 0 metric 10 }
  }
```

Endlich kann beta den Empfang der Nachricht bestätigen. In diesem Acknowledge werden aber nicht nur die wiederholten zwei LSAs bestätigt, sondern auch das Typ-2 LSA. Die Bestätigung enthält den Header der zu quittierenden LSAs, um klar zumachen welche Pakete bestätigt werden.

```
192.168.10.2 > 224.0.0.6:
  OSPFv2-ls_ack 84: rtrid 0.0.0.2 area 0.0.0.1
    { E S 80000002 age 1 rtr 0.0.0.3 }
    { E S 80000002 age 2 rtr 0.0.0.1 }
    { E S 80000001 age 1 net dr 0.0.0.3 if 192.168.10.3 }
```

Charly verteilt die Informationen über betas Link-States nicht unter den OSPF-Routern. Sondern schickt lediglich ein Acknowledge an seinen BDR.

```
192.168.10.3 > 224.0.0.5:
  OSPFv2-ls_ack 84: rtrid 0.0.0.3 area 0.0.0.1
    { E S 80000002 age 1 rtr 0.0.0.2 }
```

Als BDR verschickt beta seine Typ-1 LSAs direkt an alle OSPF-Router

```
192.168.10.2 > 224.0.0.5:
  OSPFv2-ls_upd 76: rtrid 0.0.0.2 area 0.0.0.1
  { E S 80000003 age 1 rtr 0.0.0.2
    { net 192.168.10.0 mask 255.255.255.0 tos 0 metric 10 }
    { net 192.168.20.0 mask 255.255.255.0 tos 0 metric 10 }
  }
```

Zur Vollständigkeit fehlt noch das Acknowledge von alpha.

```
192.168.10.1 > 224.0.0.6:
  OSPFv2-ls_ack 84: rtrid 0.0.0.1 area 0.0.0.1
    { E S 80000003 age 2 rtr 0.0.0.2 }
```

Der Mitschnitt zeigt, wie OSPF im Initialfall nach dem Starten der Router die Bekanntschaften mit benachbarten Routern aufbaut und die Link-States über Updates verteilt werden, so dass alle Router eine synchronisierte Link-State-Database haben. Diese kann mit dem Befehl `show ip ospf database` über die vty des OSPF-Daemons eingesehen werden. Eine Verbindung zur virtuellen Konsole von `ospfd` wird mit dem Befehl

`alpha:# telnet localhost 2604` aufgebaut. Dabei sollte beachtet werden, dass in den Beispielen zu dieser Arbeit der OSPF-Daemon immer auf Port 2604 horcht. Im vorangegangenen Kapitel benutzt der RIP-Daemon immer 2602. Da diese Portzuweisung auch in `/etc/services` im benutzten Dateisystem eingetragen ist, kann auch die besser lesbare Variante `telnet localhost ospfd` benutzt werden.

Link-State-Database von alpha:

```
ospfd# show ip ospf database

OSPF Router with ID (0.0.0.1)

Router Link States (Area 0.0.0.1)
Link ID   ADV Router   Age Seq#       CkSum  Link count
0.0.0.1   0.0.0.1      719 0x80000002 0xfb95 2
0.0.0.2   0.0.0.2      719 0x80000003 0xdbe4 2
0.0.0.3   0.0.0.3      719 0x80000002 0xac07 2

Net Link States (Area 0.0.0.1)

Link ID        ADV Router   Age Seq#       CkSum
192.168.10.3   0.0.0.3      719 0x80000001 0x9342
```

Die Tabelle zeigt an, welche LSAs Einfluss auf das Bilden der aktuellen Routing-Tabelle genommen haben. Es werden nur die Headerdaten der LSAs angezeigt, nicht aber die Inhalte. Die Link-State-Database ist die Grundlage für die Durchführung des Dijkstra-Alogorithmus' zum Finden der kürzesten Pfade im Netz. In der Link-State-Database von alpha sind die drei Typ-1 LSAs in der Tabelle Router Link-States aufgeführt. Das eine Typ-2 LSA steht in einer eigenen Net Link Tabelle. Für jeden LSA Typ wird eine eigene Tabelle gepflegt.

Der `show ip ospf database` Befehl bietet noch mehr, wenn man ihn mit einer entsprechenden Erweiterung aufruft. Folgende weitere Spezifizierungen des `database` Befehls zeigt Quagga, wenn man die automatische, kontextabhängige Hilfe über das Fragezeichen benutzt:

Weitere Info-Optionen der Link-State-Database von alpha:

```
ospfd> show ip ospf database ?
  asbr-summary ASBR summary link states
  external External link states
  network Network link states
  router Router link states
  summary Network summary link states
  max-age LSAs in MaxAge list
  self-originate Self-originated link states
  <cr>
```

Z.B. werden die Typ-2 LSAs komplett mit `show ip ospf database network` aufgerufen.

Detailierte Auskunft über die Typ-2 LSAs von Router alpha:

```
ospfd> show ip ospf database network

           OSPF Router with ID (0.0.0.1)

      Net Link States (Area 0.0.0.1)
  LS age: 160
  Options: 0x2 : *|-|-|-|-|-|E|*
  LS Flags: 0x6
  LS Type: network-LSA
  Link State ID: 192.168.10.3 (address of Designated Router)
  Advertising Router: 0.0.0.3
  LS Seq Number: 80000003
  Checksum: 0x9143
  Length: 36
  Network Mask: /24
    Attached Router: 0.0.0.1
    Attached Router: 0.0.0.2
    Attached Router: 0.0.0.3
```

Die daraus abgeleitete Weiterleitungstabelle wird im OSPF-Daemon von Quagga mit dem Befehl `show ip ospf route` angezeigt.

Weiterleitungstabelle des OSPF-Daemons von alpha:

```
ospfd> show ip ospf route

============ OSPF network routing table ============

N 192.168.10.0/24 [10] area: 0.0.0.1
                           directly attached to eth1
N 192.168.20.0/24 [20] area: 0.0.0.1
                              via 192.168.10.2, eth1
N 192.168.30.0/24 [20] area: 0.0.0.1
                              via 192.168.10.3, eth1
N 192.168.40.0/24 [10] area: 0.0.0.1
                           directly attached to eth2

============ OSPF router routing table =============

============ OSPF external routing table ===========
```

Um noch mehr Informationen abzufragen, die der OSPF-Daemon gesammelt hat, gibt es einige weitere Varianten des `show ip ospf` Befehls. Folgende Möglichkeiten bietet Quaggas vty erneut über kontextabhängigen Druck auf "?" an. Einige davon warten wiederum mit mehreren Varianten und Spezialisierungen auf.

```
ospfd> show ip ospf ?
  database      Database summary
  interface     Interface information
  neighbor      Neighbor list
  route OSPF    routing table
  <cr>
```

Die Simulation kann nach der Ausführung mit den bekannten Befehlen heruntergefahren werden.

5.2 Beispiel 2: OSPF im Fehlerfall (Ausfall einer Leitung)

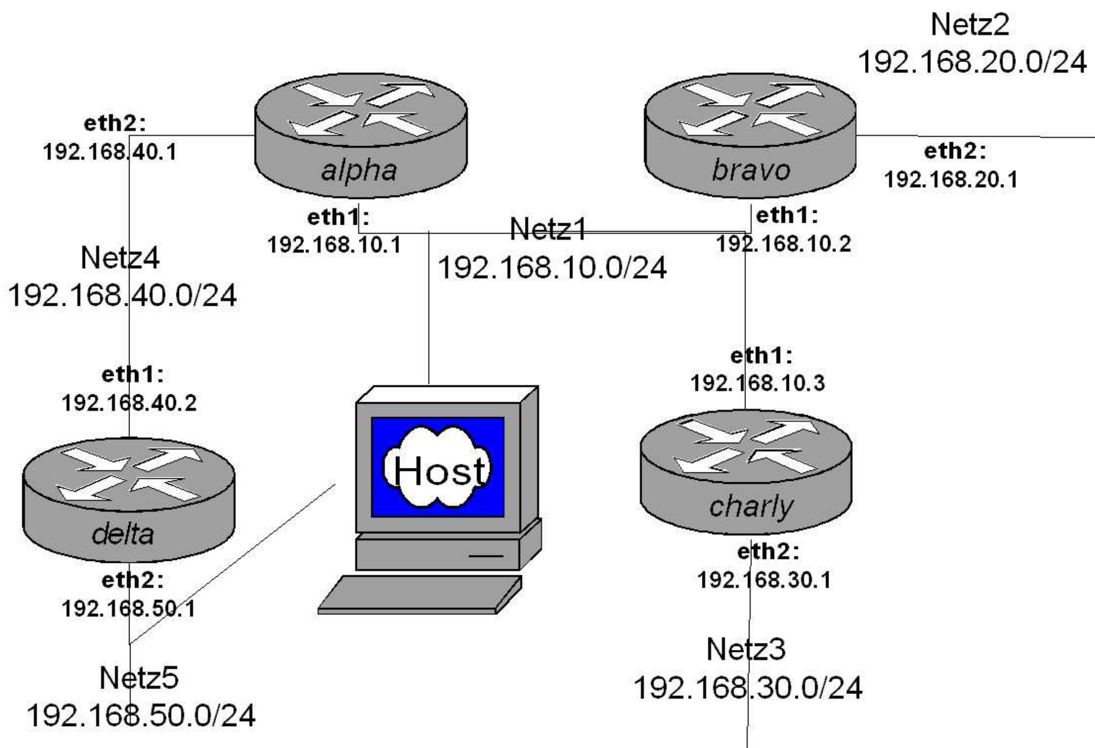
Der Vorteil von dynamischen Routingprotokollen ist, dass Änderungen an der Topologie zur Laufzeit des verteilten Systems selbstständig erkannt werden. Die Reaktion auf ein

unvorgesehenes Ereignis wie Ausfall einer Netzwerkkomponente oder die Erweiterungen des Netzwerks durch zusätzliche Netze bestimmt maßgebend die Qualität eines Routing Algorithmus'.

In den beiden folgenden Abschnitten werden diese Aspekte von OSPF untersucht. Im nächsten Beispiel wird eine leicht erweiterte Topologie aus dem ersten Beispiel simuliert. In dem entstehenden Netzwerk wird dann nach der Initialisierungsphase eine Netzwerkverbindung unterbrochen, um anschließend die Reaktionen von OSPF abzuwarten und aufzuzeichnen. Die Analyse des Mitschnittes macht den Großteil der Ausführungsphase aus.

Designphase:

Die Topologie aus 5.1 wird um einen Router und ein Netzwerk erweitert.



Die Abbildung zeigt die zu simulierende Topologie. Nach der Kaltstartphase soll die Verbindung von alpha ins Netzwerk 192.168.10.0/24 unterbrochen werden, so dass 2 unabhängige Netzwerk-Inseln entstehen. Nach der Unterbrechnung wird die Reaktion der Router bravo und charly mitgeschnitten und analysiert. Die Unterbrechung wird wie zuvor im Kapitel 4 mit Hilfe der Firewall `iptables` erreicht.

Implementierungsphase:

Die XML Beschreibung basiert auf der XML-Datei zu Beispiel 5.1. Deshalb werden nur die notwendigen Änderungen bzw. Erweiterungen der vorangegangenen Implementierung gezeigt.

Neben der komplett neuen Beschreibung der virtuellen Maschine delta gibt es nur einige erklärenswerte Änderungen an der Beschreibung von alpha. Hier werden zwei neue <exec>-Tags eingefügt, um die bekannten Skripte cutline.sh und repair.sh zu starten. Diese Shell-Skripte verändern die Einstellungen der Firewall `iptables` dahingehend, dass die Verbindung von alpha zum Netzwerk 192.168.10.0/24 unterbrochen wird bzw. die Unterbrechung wieder beseitigt wird.

OSPF2.xml

```
<vnuml>
...
<simulation_name>ospf2</simulation_name>
...
<net mode="uml_switch" name="net5"/>
...
<vm name="alpha" >
  ...
  <filetree root="/usr/local/bin" when="cut">#
    /usr/local/share/vnuml/examples/ospf2/scripts
  </filetree>
  ...
  <exec seq="cut" type="verbatim">cutline.sh eth1</exec>
  <exec seq="repair" type="verbatim">repair.sh eth1</exec>
  ...
</vm>
<vm name="delta" >
  <filesystem type="cow" >/usr/local/share/vnuml/filesystems/root_fs_tutorial</f
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
  <boot>
    <con0>xterm</con0>
  </boot>
  <if id="1" net="net4" >
```

```
    <ipv4 mask="255.255.255.0">192.168.40.2</ipv4>
  </if>
  <if id="2" net="net5" >
    <ipv4 mask="255.255.255.0">192.168.50.1</ipv4>
  </if>
  <forwarding type="ip" />
  <filetree root="/usr/local/etc" when="start">
    /usr/local/share/vnuml/examples/ospf2/delta
  </filetree>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">ospfd -f /usr/local/etc/delta.conf
   -d -P 2604</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ospfd</exec>
</vm>
```

Ausführungsphase:

Das Einlesen der XML-Beschreibung und damit das Hochfahren des Szenarios geschieht wieder mit:

```
vnumlparser.pl -t ospf2.xml -vB
```

Nach dem Hochfahren der virtuellen Rechner können die Zebra- und OSPF-Daemons mit

```
vnumlparser.pl -x start@ospf2.xml -vB
```

gestartet werden. Nachdem die Kaltstartphase der OSPF-Router überwunden wurde, sollten die Router synchronisierte Linkstate-Datenbanken haben. Die Weiterleitungstabelle des Kernels von charly sollte wie folgt aussehen, sobald der Zebra-Daemon alle Informationen korrekt propagiert hat.

```
charly:~# route
Kernel IP routing table
```

```
Destination      Gateway       Genmask          Flags Metric Ref
   Use Iface
192.168.1.152    *             255.255.255.252  U     0      0    0
   eth0
192.168.20.0     192.168.10.2  255.255.255.0    UG    20     0    0
   eth1
192.168.50.0     192.168.10.1  255.255.255.0    UG    30     0    0
   eth1
192.168.30.0     *             255.255.255.0    U     0      0    0
   eth2
192.168.10.0     *             255.255.255.0    U     0      0    0
   eth1
192.168.40.0     192.168.10.1  255.255.255.0    UG    20     0    0
   eth1
```

Dabei stehen die Flags U für "up" und G für "use Gateway". Der Eintrag für das Ziel 192.168.1.152 bezieht sich auf das Management Interface von VNUML und ist kein Teil des eigentlichen Szenarios. Es wird nur vom Parser verwendet, um organisatorische Zugriffe zu tätigen.

Der stabile Zustand, in dem sich das Netzwerk-Szenario nun befindet, wird mit dem Abschneiden von alpha vom Netz 192.168.10.0/24 gestört. Damit die Reaktionen per `tcpdump` mitgeschnitten werden können, kann der Sniffer wahlweise auf den virtuellen Maschninen bravo oder charly den Verkehr im Netz 192.168.10.0/24 belauschen.

Folgender Befehl bewirkt ein Mitschneiden des Netzwerkverkehrs im Netz 192.168.10.0/24. Option -s0 veranlasst `tcpdump`, die kompletten Pakete mitzuschneiden. Die Option -v schaltet eine ausführlichere Ausgabe ein.

```
charly:~# tcpdump -i eth1 -s0 -v
```

Jetzt kann die künstliche Unterbrechung mit

```
host:/usr/local/share/vnuml/examples # vnumlparser -x cut@ospf2.xml
```

vorgenommen werden.

Wegen der besseren Übersicht, wurden die folgenden Ausgaben von tcpdump auf die wesentlichen Bestanteile reduziert.

```
charly:~# tcpdump -i eth1 -s0 -v
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture
   size 65535 bytes
192.168.10.2 > 224.0.0.5:
OSPFv2, Hello (1), length: 52 Router-ID: 0.0.0.2, Area 0.0.0.1,
Hello Timer: 10s, Dead Timer 40s, Mask: 255.255.255.0, Priority:
   1
Designated Router charly, Backup Designated Router 192.168.10.2
Neighbor List:
0.0.0.1
0.0.0.3
```

Zuerst wird tcpdump im 10 Sekunden Intervall Hellos von den Routern bravo und charly hören können. Die Hello-Pakete von alpha erreichen nun das Netzwerk 192.168.10.0/24 nicht mehr. Nach Ablauf des Dead-Timers (40 Sekunden) schickt charly folgendes Linkstate-Update. Die diversen Hello-Pakete, die vor Ablauf des Dead-Intervalls geschickt werden, wurden aus Platzgründen nicht aufgeführt. Wie in dem ersten beispielhaften Hello-Paket zu sehen, wird bis dahin auch Router alpha mit der Router-ID 0.0.0.1 noch als Nachbar mit an das Hello angehangen.

Die Reaktion des DR nach Ablauf des Dead-Intervalls, besteht aus einem LSA Typ-2, in dem das Netzwerk mit allen aktiven SPF-Routern beschrieben wird. Charly teilt damit mit, dass nur noch die Router mit der ID 0.0.0.2 und 0.0.0.3 aktiv sind. Die Router ID von alpha (0.0.0.1) wurde entfernt.

```
charly > 224.0.0.5:
OSPFv2, LS-Update (4), length: 60 Router-ID: 0.0.0.3, Area 0.0.0.1,
   1 LSA
LSA #1
Advertising Router: 0.0.0.3, seq 0x80000003, age 1s, length: 12
Network LSA (2), LSA-ID: charly Mask 255.255.255.0
Connected Routers:
0.0.0.2
0.0.0.3
```

Als weitere Maßnahmen schickt charly ein LSA Typ-1 mit Informationen über seinen lokalen Status. Obwohl sich hier nichts geändert hat, wird dennoch die Information

geflutet, dass charly sowohl Teil des Netzwerks 192.168.10.0/24 als auch eine Verbindung zum Netz 192.168.30.0/24 besitzt.

```
charly > 224.0.0.5:
OSPFv2, LS-Update (4), length: 76 Router-ID: 0.0.0.3, Area 0.0.0.1,
   1 LSA
LSA #1
Advertising Router: 0.0.0.3, seq 0x80000004, age 1s, length: 28
Router LSA (1), LSA-ID: 0.0.0.3
Router LSA Options: [none]
Neighbor Network-ID: charly, Interface Address: charly, tos 0, metric:
   10
Stub Network: 192.168.30.0, Mask: 255.255.255.0, metric: 10
```

Auch beta darf als BDR direkt LSAs an die AllSPFRouter Adresse schicken und versendet folgendes LSA Typ-1:

```
192.168.10.2 > 224.0.0.5:
OSPFv2, LS-Update (4), length: 76 Router-ID: 0.0.0.2, Area 0.0.0.1,
   1 LSA
LSA #1
Advertising Router: 0.0.0.2, seq 0x80000004, age 1s, length: 28
Router LSA (1), LSA-ID: 0.0.0.2
Router LSA Options: [none]
Neighbor Network-ID: charly, Interface Address: 192.168.10.2, metric:
   10
Stub Network: 192.168.20.0, Mask: 255.255.255.0, metric: 10
```

Anschließend werden die versendeten LSAs, um dem "reliable flooding" gerecht zu werden, bestätigt. Zunächst bestätigt bravo in einem zusammengefassten Acknowledge sowohl das LSA Typ-2 als auch das LSA Typ-1 von charly. Danach quittiert charly den Empfang des LSA von bravo.

```
192.168.10.2 > 224.0.0.5:
OSPFv2, LS-Ack (5), length: 64 Router-ID: 0.0.0.2, Area 0.0.0.1
  Advertising Router: 0.0.0.3, seq 0x80000003, age 1s, length: 12
```

```
  Network LSA (2), LSA-ID: charly

  Advertising Router: 0.0.0.3, seq 0x80000004, age 1s, length: 28
  Router LSA (1), LSA-ID: 0.0.0.3


charly > 224.0.0.5:
OSPFv2, LS-Ack (5), length: 44 Router-ID: 0.0.0.3, Area 0.0.0.1

Advertising Router: 0.0.0.2, seq 0x80000004, age 1s, length: 28
Router LSA (1), LSA-ID: 0.0.0.2
```

Nach diesen Paketen kann der OSPF-Algorithmus wieder eine korrekte Abbildung der Netzwerktopologie aus den ausgetauschten Informationen bilden. Ab diesem Zeitpunkt kann zum normalen Alltagsgeschäft zurückgekehrt werden. Dies zeigt sich darin, dass nur noch Hello-Pakete ausgetauscht werden.

Die beiden folgenden Hello-Pakete zeigen, dass keiner der zwei verbleibenden OSPF-Router mehr annimmt, dass ein Router alpha mit der Router ID 0.0.0.1 in der Nachbarschaft arbeitet, sondern nur noch jeweils der andere Router aufgeführt wird.

```
192.168.10.2 > 224.0.0.5:
OSPFv2, Hello (1), length: 48 Router-ID: 0.0.0.2, Area 0.0.0.1
Hello Timer: 10s, Dead Timer 40s, Mask: 255.255.255.0, Priority:
   1
Designated Router charly, Backup Designated Router 192.168.10.2
Neighbor List:
0.0.0.3


charly > 224.0.0.5:
OSPFv2, Hello (1), length: 48 Router-ID: 0.0.0.3, Area 0.0.0.1
Hello Timer: 10s, Dead Timer 40s, Mask: 255.255.255.0, Priority:
   1
Designated Router charly, Backup Designated Router 192.168.10.2
Neighbor List:
0.0.0.2
```

5.3 Hierarchisierung

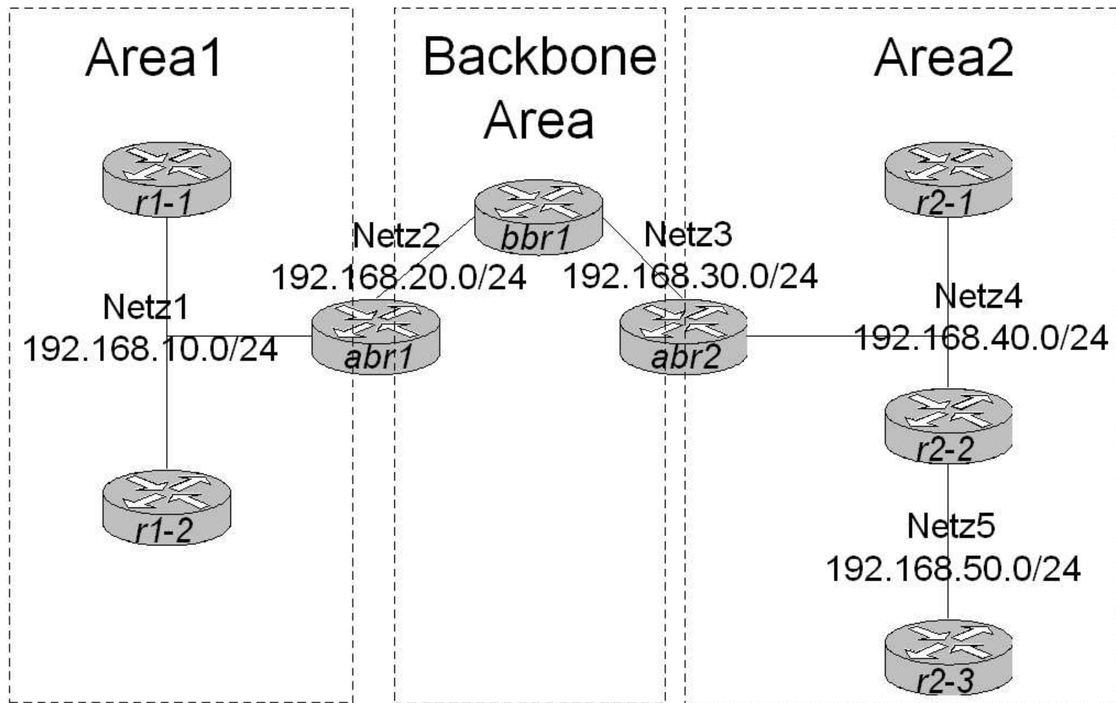
Das letzte Beispiel hat bereits einen ersten Eindruck verschafft, wie hoch der Aufwand für das zuverlässige Verteilen der Informationen durch den OSPF-Algorithmus ist. Diese Kommunikation der OSPF-Router ist dringend erforderlich, damit alle Router, auf dem identischen Wissen über die Topologie basierend, die kürzesten Pfade berechnen können. In diesem erhöhten Kommunikationsbedarf ist ein großer Nachteil zu sehen, da das benötigte Übertragungsvolumen bei wachsenden Netzwerken schnell sehr groß wird.

Um eine bessere Skalierbarkeit zu erreichen, gibt es die sogenannten Areas. Diese Areas stellen eine Möglichkeit zur Hierachisierung der OSPF-Topologie dar. Die bisher gezeigten LSAs vom Typ-1 und Typ-2 werden nicht über die Grenzen einer Area hinaus propagiert. Stattdessen wird eine Erreichbarkeit über mehrere Areas durch sogenannte Network Summary LSAs (Typ-3) realisiert. Gleichzeitig wird die Rechenzeit zum Erstellen der Weiterleitungstabelle verkleinert.

Im Folgenden wird ein Szenario erstellt, das aus 2 Areas und einer Backbone-Area besteht.

Designpase:

Die Topologie besteht aus 5 Netzwerken, die auf 3 Areas aufgeteilt sind. Die Backbone-Area wird traditionell als Area 0 benannt. Die beiden Areas 1 und 2 sind nur durch die Backbone Area verbunden. Der Datenverkehr zwischen Area 1 und Area 2 muss also durch die Backbone Area geleitet werden.



Area 1 besteht aus den Routern r1-1 und r1-2, sowie dem Area-Border-Router abr1. Alle drei Router sind über das Netzwerk 192.168.10.0/24 verbunden. Um die Weiterleitungstabellen klein zu halten, haben die Router r1-1 und r1-2 keine weiteren Netzwerke, für die Sie die Erreichbarkeit sichern sollen. Im realen Leben wären solche Router, ohne angeschlossenes Netzwerk, eher unsinnig. Hier wurde diese Designentscheidung allerdings bewusst getroffen, um die Auswertung der Simulation nicht unnötig komplizierter zu machen. Der Area Border Router abr1 verbindet die Area 1 mit dem Backbone.

Die Backbone Area besteht aus den Area Border Routern abr1 und abr2, sowie dem Backbone-Router bbr1. Zwischen den beiden Routern abr1 und bbr1 liegt das Netzwerk 192.168.20.0/24. Das Netzwerk 192.168.30.0/24 verbindet bbr1 mit abr2. Area 2 schließt sich durch abr2 an den Backbone an.

Area 2 besteht aus zwei Netzwerken und insgesamt 4 Routern. Am Netzwerk 192.168.40.0/24 sind abr1, r2-1 und r2-2 angeschlossen. Zwischen einer weiteren Schnittstelle von r2-2 und r2-3 erstreckt sich das letzte Netzwerk 192.168.50.0/24.

Außer der reinen Netzwerktopologie soll auch die Möglichkeit zum Starten und Stoppen der Router geschaffen werden.

Implementierungphase:

Die XML-Beschreibung des Szenarios beinhaltet keine neuen Elemente und lässt sich zu einem großen Teil direkt aus der Designbeschreibung ableiten. Der hier abgedruckte Teil der XML-Beschreibung umfasst die globalen Attribute, die Definition der Netzwerke, sowie eine stellvertretende virtuelle Maschine. Die Beschreibungen der übrigen virtuellen Rechner lassen sich leicht analog dazu herleiten und beinhalten keine Überraschungen oder Besonderheiten.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
<vnuml>
  <global>
    <version>1.5.0</version>
    <simulation_name>ospf3</simulation_name>
    <ssh_key>/root/.ssh/identity.pub</ssh_key>
    <automac/>
    <ip_offset>100</ip_offset>
    <host_mapping/>
    <shell>/bin/sh</shell>
  </global>
  <net mode="uml_switch" name="net1"/>
  <net mode="uml_switch" name="net2"/>
  <net mode="uml_switch" name="net3"/>
  <net mode="uml_switch" name="net4"/>
  <net mode="uml_switch" name="net5"/>
<!-- Nodes -->
<vm name="bbr1" >
  <filesystem type="cow" >
    /usr/local/share/vnuml/filesystems/root_fs_tutorial
  </filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
  <boot>
  <con0>xterm</con0>
  </boot>
  <if id="1" net="net2" >
```

```
      <ipv4 mask="255.255.255.0" >192.168.20.2</ipv4>
    </if>
    <if id="2" net="net3" >
      <ipv4 mask="255.255.255.0" >192.168.30.1</ipv4>
    </if>
    <forwarding type="ip" />
    <filetree root="/usr/local/etc" when="start">
      /usr/local/share/vnuml/examples/ospf3/bbr1
    </filetree>
    <exec seq="start" type="verbatim">
      zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601
    </exec>
    <exec seq="start" type="verbatim">
      ospfd -f /usr/local/etc/bbr1.conf -d -P 2604
    </exec>
    <exec seq="stop" type="verbatim">
      killall zebra
    </exec>
    <exec seq="stop" type="verbatim">
      killall ospfd
    </exec>
  </vm>

  </vnuml>
```

Die Konfiguration der OSPF-Router kann ebenfalls direkt aus der Skizze abgeleitet werden. Als stellvertretendes Beispiel folgt nun die Konfiguration eines Area-Border-Routers. Die Besonderheit der ABR liegt darin, dass ihre Schnittstellen auf mehrere Areas verteilt werden. Im Fall von ABR1 ist die eine Netzwerk-Schnittstelle Teil von Area 1 und eine andere Schnittstelle Teil der Backbone Area. Ansonsten gibt es keine spezielle Änderung an der Konfiguration, die den Router explizit als Area-Border-Router kennzeichnet. Alleine aus der Tatsache, dass Netze verschiedener Areas mit dem Network-Kommando konfiguriert werden, erkennt der OSPF-Router, dass er als ABR arbeiten muss.

```
ABR1.conf:
!
```

```
hostname ospfd
password zebra
log stdout
!
router ospf
network 192.168.10.0/24 area 1
network 192.168.20.0/24 area 0

line vty
!
```

Ausführungsphase:

Die Ausführung der Simulation wird gestartet, indem das Szenario hochgefahren wird.

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -t ospf3.xml
   -vB
```

Da acht virtuelle Maschinen gestartet werden, kann dieser Vorgang einige Minuten dauern. Beim ersten Start des Szenarios werden zusätzlich noch die Dateisysteme überprüft, wodurch nochmal zusätzliche Zeit benötigt wird.

Um die Kommunikation der Router mitzuverfolgen, wird im Netzwerk 192.168.20.0/24 `tcpdump` eingesetzt. Wahlweise kann natürlich auch `ethereal` genutzt werden. Beide Varianten haben Ihre Vor- und Nachteile. Die textbasierten Ausgaben von tcpdump können besser abgedruckt werden. Die grafische Präsentation von `ethereal` bietet größere Möglichkeiten, die Darstellung auf die benötigten Pakete zu reduzieren. Dazu werden geeignete Filter angelegt, mit deren Hilfe eine unüberschaubare Menge an Paketen auf ein Minimum der interessanten Pakete verkleinert wird. Um die Vorteile von beiden Varianten zu nutzen, wird zuerst ein Mitschnitt über `tcpdump` aufgezeichnet und in einer Datei gesichert. Diese Daten können dann wahlweise mit `tcpdump` oder `ethereal` angezeigt werden. Die Aufzeichnung des Verkehrs im Netzwerk 192.168.20.0/24 wird mit folgendem Befehl vom Rechner bbr1 gestartet.

```
bbr1:/tcpdump -i eth1-s0 -w bbr1.dump
```

Mit der Option -w kann die Datei angegeben werden, in die der Mitschnitt geschrieben wird. In diesem Fall wird die Aufzeichnung in die Datei bbr1.dump gesichert.

Nachdem tcpdump bereit für den Mitschnitt ist, können die Zebra- und OSPF-Daemons gestartet werden.

```
host:/usr/local/share/vnuml/examples/# vnumlparser.pl -x start@ospf3.xml
   -vB
```

Nach ungefähr einer Minute sollten die Router ihr Wissen komplett ausgetauscht haben und damit eine Konvergenz erzielen. Eine stichprobenartige Überprüfung einiger Weiterleitungstabellen sollte darüber Klarheit verschaffen.

```
bbr1:~# route

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref Use
   Iface
192.168.1.144   *               255.255.255.252 U     0       0   0
   eth0
192.168.20.0    *               255.255.255.0   U     0       0   0
   eth1
192.168.50.0    192.168.30.2    255.255.255.0   UG    30      0   0
   eth2
192.168.30.0    *               255.255.255.0   U     0       0   0
   eth2
192.168.10.0    192.168.20.1    255.255.255.0   UG    20      0   0
   eth1
192.168.40.0    192.168.30.2    255.255.255.0   UG    20      0   0
   eth2
```

Die Weiterleitungstabelle von bbr1 zeigt, dass für alle fünf Netzwerke Routen existieren. Das Netzwerk 192.168.1.144 ist Teil von VNUML (Mangement-Schnittstelle) und kann ignoriert werden.

Wenn Klarheit besteht, dass die Router alle Informationen ausgetauscht haben, kann `tcpdump` mit einem Drücken von STRG+C in der entsprechenden Konsole gestoppt werden.

Um die soeben aufgezeichneten Pakete anzusehen, muss tcpdump mit der Option -r aufgerufen werden:

```
bbr1:/tcpdump -r bbr1.dump -vv
```

Wahlweise kann die Datei bbr1.dump nun auch mit `ethereal` geöffnet werden. Da `ethereal` als GUI-Werkzeug auf den X-Server angewiesen ist, müßte ein entsprechendes Tool benutzt werden, das es gestattet von der virtuellen Konsole grafische Anwendungen zu starten. Im Test haben sich diese Clients für den X-Server als sehr unzuverlässig und instabil erwiesen. Deshalb ist ein kleiner Umweg sinnvoller. Dabei wird die Datei bbr1.dump zuerst auf den Host kopiert. Von dort kann `ethereal` ohne Einschränkungen gestartet werden.

```
bbr1:/scp bbr1.dump 192.168.1.145:/
```

Obiger Befehl kopiert die Daten in das Stammverzeichnis / vom Hostrechner. Die Management-Schnittstelle vom Host zu bbr1 ist im vorliegenden Beispiel 192.168.1.145. Bei Konflikten mit echten bereits vergebenen IP-Adressen, müssen die Management-Schnittstellen eventuell mittles <ip_offset> angepasst werden. Entsprechend muss dann auch der Befehl zum Kopieren der Datei abgeändert werden. Auf dem Host kann dann `ethereal` gestartet werden. Die option -r funktioniert auch hier zum direkten öffnen einer Datei.

```
host:/ ethereal -r bbr1.dump
```

Die große Stärke von `ethereal` sind die Filter, die über den Mitschnitt gelegt werden können. Der Displayfilter `ospf.msg.lsupdate` beschränkt die Anzeige auf die Pakete mit LSAs, alle Acknowlegements, Hellos und sonstige Pakete, die zwar zum Protokoll gehören, aber für die Betrachtung momentan eher störend sind, werden nicht angezeigt.

Die OSPF-Pakete können aber noch feiner gefiltert werden. Die Ergänzung des Displayfilters zu

```
ospf.msg.lsupdate && ospf.lsa == 3
```

beschränkt die angezeigten Pakete auf die LSAs vom Typ-3. Diese Pakete werden vom Area-Border-Router verschickt, um die Netzwerke einer Area auch in den anderen Areas bekannt zu machen. Die LSAs vom Typ-1 und Typ-2 werden nicht über die Grenzen einer Area hinaus propagiert. Somit reduziert sich das Wissen eines Routers über eine fremde Area lediglich auf die Kenntnis der Netzwerke, die sich dort befinden. Die genaue Topologie der fremden Area wird jedoch nicht mitgeteilt. Um die Netzwerke erreichen zu können, reicht diese Kenntnis jedoch voll und ganz aus. Im vorliegenden Mitschnitt wurden drei solcher LSA-Typ 3 Nachrichten von bbr1 mitgehört.

Ein LSA vom Typ-3 mit Informationen über die Netze 192.168.20.0/24 oder 192.168.30/24 kann bbr1 nicht mithören, da diese Netze Teil von Area 0 sind und von den Area-Border-Routern ausschließlich in die anderen Areas propagiert werden. Umgekehrt kann nun überprüft werden, ob LSAs vom Typ-1 und Typ-2 mit Inhalten aus Area 1 und Area 2 diese Areas nicht verlassen haben. Wenn die ABRs korrekt arbeiten dürfte bbr1 diese nicht mitgeschnitten haben. Eine Anpassung des Displayfilters zeigt die LSAs vom Typ-1

```
ospf.msg.lsupdate && (ospf.lsa == 1 || ospflsa == 2)
```

Als Resultat zeigt `ethereal` alle LSAs vom Typ-1 und Typ-2. Bei der exemplarischen Ausführung wurden 9 Pakete angezeigt, die alle nur Informationen über die Netzwerke 192.168.20.0/24 und 192.168.30.0/24 beinhalteten. Die Area-Border-Router haben die Erwartungen erfüllt. Die LSAs vom Typ-1 und Typ-2 werden nur innerhalb einer Area geflutet. Die Typ-3 LSAs werden nur außerhalb der Area weitergegeben.

Weiterhin kann noch gezeigt werden, dass die Typ-3 LSAs mit Informationen über Area 1, durch Area 0 hindurch, auch an alle Router in Area 2 weitergeleitet werden. Dazu genügt ein Blick in die Linkstate-Database von Router R2-3, der immerhin zwei Netzwerke "weit weg" vom ABR2 entfernt ist.

```
ospfd# show ip ospf database

OSPF Router with ID (0.0.2.3)

Router Link States (Area 0.0.0.2)

Link ID  ADV Router Age Seq#       CkSum  Link count

0.0.2.0  0.0.2.0    97  0x80000003 0xa576  1
0.0.2.1  0.0.2.1    96  0x80000003 0xa079  1
0.0.2.2  0.0.2.2    95  0x80000003 0x08bb  2
0.0.2.3  0.0.2.3    94  0x80000003 0x53af  1

Net Link States (Area 0.0.0.2)

Link ID       ADV Router Age Seq#       CkSum
192.168.40.3  0.0.2.2    95  0x80000002 0x644f
192.168.50.2  0.0.2.3    99  0x80000001 0xddd0

Summary Link States (Area 0.0.0.2)

Link ID       ADV Router Age Seq#        CkSum   Route
192.168.10.0  0.0.2.0     87 0x80000001  0x6a5f  192.168.10.0/24
```

```
192.168.20.0   0.0.2.0       92 0x80000001  0x9732  192.168.20.0/24
192.168.30.0   0.0.2.0      138 0x80000001  0xc405  192.168.30.0/24
```

Im letzten Abschnitt Summary Link States werden alle Typ-3 LSAs angezeigt, die Router r2-3 empfangen hat. Es werden alle Netzwerke der Topologie, die außerhalb von Area 2 liegen korrekt aufgezählt, darunter auch das LSA mit Informationen über Netzwerk 192.168.10.0/24 aus Area 1. Die LSAs vom Typ-3 werden somit auch über mehrere Areas hinweg propagiert, um die Erreichbarkeit zu ermöglichen.

6 Vergleich RIP vs. OSPF

In den vorangegangenen Kapiteln wurden zwei unterschiedliche Routing-Protokolle vorgestellt. Beide wurden entwickelt, um im Interior Gateway-Bereich die Weiterleitung der Pakete über Netzwerkgrenzen zu ermöglichen. In den folgenden Abschnitten werden die beiden Startegien gegenübergestellt und ihre Leistungen in vergleichbaren bzw. identischen Netzwerken verglichen. Die Kriterien des Vergleichs sind die benötigte Zeit bis das Netzwerk konvergiert und das hierzu benötigte Datenvolumen. Dabei wird das Verhalten der Algorithmen schwerpunktmäßig in typischen Situationen getestet, mit denen ein Router alltäglich konfrontiert wird. Dazu gehört ein reibungsloser Betrieb genauso wie ein Fehlerfall, auf den reagiert werden muss. Zur Durchführung der Messungen werden die Werkezuge `tcpdump` und `ethereal` eingesetzt.

6.1 Initialisierungsphase:

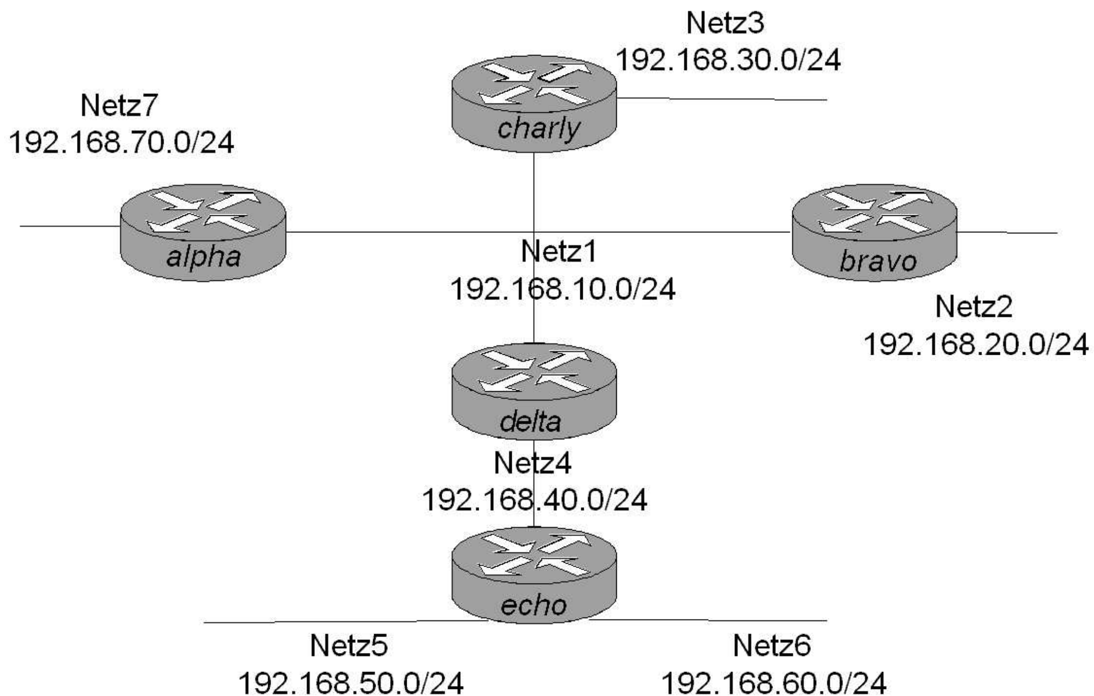
In diesem ersten Vergleich werden die Initialisierungsphasen von RIP und OSPF gegenübergestellt. Diese Phase wiederholt sich mit jedem Starten des Netzwerkszenarios. Für den alltäglichen Einsatz eines realen Routers ist die Bedeutung eher zu vernachlässigen, aber beim Arbeiten mit den Simulationwerkzeugen VNUML und Zebra/Quagga wird diese Phase wesentlich öfter Durchlaufen.

Die Dauer der Initialisierungsphase, also das Zeitkriterium wurde bereits in Abschnitt 5.1 erläutert, daher wird im Folgenden das Datenvolumen bis zur Konvergenz des Netzwerkes betrachtet. Dazu ist es nötig, das vollständige Datenvolumen, das von den beiden Routing-Daemons erzeugt wird, zu messen und zu vergleichen. Ein möglicher Weg, um diese Messungen durchzuführen ist, eine Firewall so zu konfigurieren, dass alle Pakete, die von den Routern erzeugt werden, protokolliert werden. Eine weitere Möglichkeit wäre per SNMP vergleichbare Messungen durchzuführen. Wegen dem erhöhten Konfigurationsaufwand und um nicht mehr Softwarewerkzeuge als notwendig zu installieren, wurde die Entscheidung gegen die SNMP-Methode getroffen, obwohl sie im Testbetrieb gleichwertige Ergebnisse lieferte. Die Volumen-Messungen in diesem Kapitel werden mit der

Firewall `iptables` vorgenommen. Zur Erleichterung der Auswertung gibt es ein Perl-Skript, das die Gesamtsumme bildet. Dabei ist zu beachten, dass die Schnittstelle eth0 standardmäßig auf allen virtuellen Rechnern die Management-Schnittstelle ist, deren Verkehr nicht miteinbezogen werden darf.

Designphase:

Das Szenario soll abwechselnd mit RIP und OSPF-Routern untersucht werden. Deshalb müssen für beide Varianten Konfigurationen erstellt werden. Weiterhin muss die Protokollierung des Netzwerkverkehrs eine Messung des Datenvolumens erlauben.



Implementierungsphase:

Die oben genannten Forderungen an die Simulation können durch Ausführungssequenzen des VNUML-Parsers erfüllt werden. Die Konfigurationen werden in jeweils eigenen Unterverzeichnissen mit den Namen rip und ospf abgelegt. Vor den Ausführungssequenzen “startRip” bzw, “startOspf“ werden dann jeweils die richtigen Konfigurationsdateien zu den virtuellen Maschinen kopiert. Der folgenden Ausschnitt der XML-Datei zeigt die konkrete Umsetzung für die virtuelle Maschine alpha.

```
<filetree root="/usr/local/etc" when="startRip">
```

```
    rip/alpha
</filetree>
<filetree root="/usr/local/etc" when="startOspf">
    ospf/alpha
</filetree>
<exec seq="startRip" type="verbatim">ripd -f /usr/local/etc/alpha.conf
    -d -P 2604</exec>
<exec seq="startOspf" type="verbatim">ospfd -f /usr/local/etc/alpha.conf
    -d -P 2604</exec>
```

Dadurch ist es möglich über die Ausführungssequenzen zu bestimmen, ob die Topologie mit RIP oder OSPF Routern arbeitet.

Das Datenvolumen und die Pakete, die ein Router sendet, werden von `iptables` unter OUTPUT protokolliert, wenn eine entsprechende Filterregel angewendet wird. Die Filterregeln werden ebenfalls über Ausführungssequenzen manipuliert.

```
<exec seq="startIptables" type="verbatim">iptables -A OUTPUT -o eth0</exec>
<exec seq="startIptables" type="verbatim">iptables -A OUTPUT -o eth1</exec>
<exec seq="startIptables" type="verbatim">iptables -A OUTPUT -o eth2</exec>
<exec seq="showCounter" type="verbatim">iptables -L -v</exec>
<exec seq="resetCounter" type="verbatim">iptables -Z</exec>
```

Die obenstehenden Ausführungssequenzen "startIptables" veranlassen die Firewall `iptables`, den gesendeten Netzwerkverkehr des jeweiligen Interfaces zu protokollieren. Die Ausführungssequenz "showCounter" ruft den Bericht der Firewall ab. Um die Messdaten der Firewall zurück auf 0 zu setzen, gibt es die Ausführungssequenz "resetCounter". Diese wird benötigt, um die Zähler zwischen den Messungen von RIP und OSPF zurückzusetzen.

Um den Zebra-Daemon zu starten, gibt es für jede virtuelle Maschine eine weitere Ausführungssequenz:

```
<exec seq="startZebra" type="verbatim">
```

Mit einer gemeinsamen Stop-Sequenz können alle, am Routing beteiligten Daemons angehalten werden.

```
<exec seq="stop" type="verbatim">killall zebra</exec>
<exec seq="stop" type="verbatim">killall ripd</exec>
<exec seq="stop" type="verbatim">killall ospfd</exec>
```

Die weitere VNUML-Konfiguration lässt sich direkt von der Skizze ableiten und dürfte nach Lektüre der vorangegangenen Kapitel keine Schwierigkeiten darstellen.

Die Konfiguration der Router enthält ebenfalls keine Neuerungen. Die RIP und OSPF Konfigurationen von Router alpha werden unten beipielhaft aufgeführt.

```
RIP-Konfiguration: .../rip/alpha/alpha.conf

!
hostname ripd
password zebra
!
router rip
  network 192.168.10.0/24
  network 192.168.70.0/24
!
line vty
```

```
OSPF-Konfiguration: .../ospf/alpha/alpha.conf
!
hostname ospfd
password zebra
!
router ospf
  network 192.168.10.0/24 area 1
  network 192.168.70.0/24 area 1
!
line vty
```

Ausführungsphase:

Um die Simulation durchführen zu können, muss das Szenario zuersteinmal hochgefahren werden. Dies geschieht wieder mit dem Befehl:

```
vnumlparser.pl -t ospfVsRip1.xml -vB
```

Wenn alle fünf virtuellen Maschinen erfolgreich gestartet wurden, sollte als nächstes der Zebra-Daemon gestartet werden. Dies geschieht durch Ausführen der Sequenz "startZebra" vom Hostrechner.

```
vnumlparser.pl -x startZebra@ospfVsRip1.xml -vB
```

Als nächstes wird die Firewall gestartet und die Messung des Datenvolumens initiiert:

```
vnumlparser.pl -x startIptables@ospfVsRip1.xml -vB
```

Im folgenden Schritt wird für den ersten Durchlauf zuerst auf allen virtuellen Rechnern der RIP-Daemon gestartet.

```
vnumlparser.pl -x startRip@ospfVsRip1.xml -vB
```

Die Konvergenzzeit des RIP-Algorithmus beträgt nur wenige Sekunden. Nach dieser Zeit sollten die Weiterleitungstabellen der virtuellen Rechner korrekt sein. Es empfiehlt sich dies rasch durch Stichproben zu überprüfen. Um das Datenvolumen nicht durch die folgenden periodischen Updates der RIP-Router zu verfälschen, müssen die Werte der Messung ausgelesen werden.

```
vnumlparser.pl -x showCounter@ospfVsRip1.xml -vB
```

Dieser Befehl zeigt nacheinander die Aufzeichnungen der Firewall getrennt nach virtuellen Maschinen. Allerdings sind diese Ausgaben nicht sehr leserfreundlich, da sie in einem Kontext von Aufrufen und Ausgaben in der Konsole stehen, die der obige Befehl nachsichzieht. Um eine bessere Lesbarkeit zu erreichen, ist es sinnvoller die Ausgabe in eine Datei umzuleiten und danach von einem kleinen Hilfsskript auszuwerten.

```
vnumlparser.pl -x showCounter@ospfVsRip1.xml -vB > ergebnis1Rip.txt
```

Diese Anweisung bewirkt, dass die Ausgaben in eine Datei mit dem Namen "ergebnis1Rip.txt" umgeleitet werden und nicht auf der Standardausgabe zu sehen sind. Diese Datei wird anschließend an das Hilfsprogramm parse.pl übergeben.

```
usr/local/share/vnuml/examples/parse.pl ergebnis1Rip.txt
```

Das Skript arbeitet mit regulären Ausdrücken und sucht die benötigten Informationen aus der etwas unübersichtlichen Textausgabe heraus, um sie anschließend zu summieren. Die Ergebnisse werden übersichtlicher und geordnet dargestellt. Die Ausgabe gliedert sich ich Abschnitte für jede virtuelle Maschine, wobei der erste Wert hinter jeder Schnittstellenbezeichnung die Anzahl der Bytes ist und die zweite Angabe die Anzahl

der Pakete wiederspiegelt. Jeder Abschnitt wird von einer Zwischensumme abgeschlossen. Diese Zwischensumme beinhaltet nicht den Verkehr über eth0. Da diese Managementschnittstellen kein eigentlicher Teil der zu simulierenden Topologie sind. Aus der abschließenden Gesamtsumme des Datenvolumens sind die Messungen für eth0 ebenfalls herausgerechnet.

```
alpha
eth0 10240 115
eth1 504 8
eth2 328 5
Summe: 832

bravo
eth0 10188 114
eth1 512 7
eth2 348 5
Summe: 860

charly
eth0 9876 108
eth1 448 5
eth2 336 4
Summe: 784

delta
eth0 9824 107
eth1 296 4
eth2 356 4
Summe: 652

echo
eth0 9876 108
eth1 316 4
eth2 244 3
eth3 244 3
Summe: 804
```

```
Gesamtsumme (ohne eth0): 3932
```

Die RIP-Router haben in der beispielhaften Ausführung ein Datenvolumen von insgesamt 3932 Bytes gesendet. Die Messungen sind wegen der manuellen Triggerung nur qualitativ zu bewerten und können durchaus variieren. Ein Vergleich ist aber erst nach der Messung des OSPF-Verhaltens möglich. Dazu müssen die momentan laufenden RIP-Daemons beendet werden. Die Ausführungssequenz

```
vnumlparser.pl -x stop@ospfVsRip1.xml -vB
```

beendet jegliche Routingdaemons.

Danach ist ein erneutes Starten des Zebra-Daemons notwendig:

```
vnumlparser.pl -x startZebra@ospfVsRip1.xml -vB
```

Es ist ebenfalls notwendig, die Zähler der Firewall zurückzusetzen, damit die Ergebnisse nicht auf den vorangegangenen basieren.

```
vnumlparser.pl -x resetCounter@ospfVsRip1.xml -vB
```

Nun ist alles vorbereitet, um die OSPF-Daemons zu starten. Die Konvergenzzeit ist länger als im vorangegangenen Beispiel, daher müssen die nachfolgenden Schritte nicht ganz so rasch ablaufen:

```
vnumlparser.pl -x startOspf@ospfVsRip1.xml -vB
```

Nach Konvergenz des Netzwerks (Stichproben der Weiterleitungstabellen):

```
vnumlparser.pl -x showCounter@ospfVsRip1.xml -vB > ergebnis1Ospf.txt
usr/local/share/vnuml/examples/parse.pl ergebnis1Rip.txt
```

alpha
eth0 10188 114
eth1 2776 38
eth2 528 9
Summe: 3304

bravo
eth0 9824 107
eth1 1736 23
eth2 528 9
Summe: 2264

charly
eth0 9616 103
eth1 1780 24
eth2 528 9
Summe: 2308

delta
eth0 9772 106
eth1 2448 29
eth2 1920 26
Summe: 4368

echo
eth0 9720 105
eth1 1460 20
eth2 528 9
eth3 528 9
Summe: 2516

Gesamtsumme (ohne eth0): 14760

Auch wenn die Triggerung der Messungen nicht genau erfolgen können und sich so eventuelle Fehler einschleichen, so kann man dennoch an den Ergebnissen eine eindeutige Tendenz ablesen. Für die Initialisierung dieses Beispiels verursacht der OSPF-Algorithmus ein Vielfaches an Datenvolumen. Für die praktische Anwendung stellt dies jedoch keinen Nachteil dar, da normalerweise ein Neustart aller Router in einer Topologie eher selten ist. Die folgenden Beispiele orientieren sich näher an für die Praxis relevanten Ausgangssituationen.

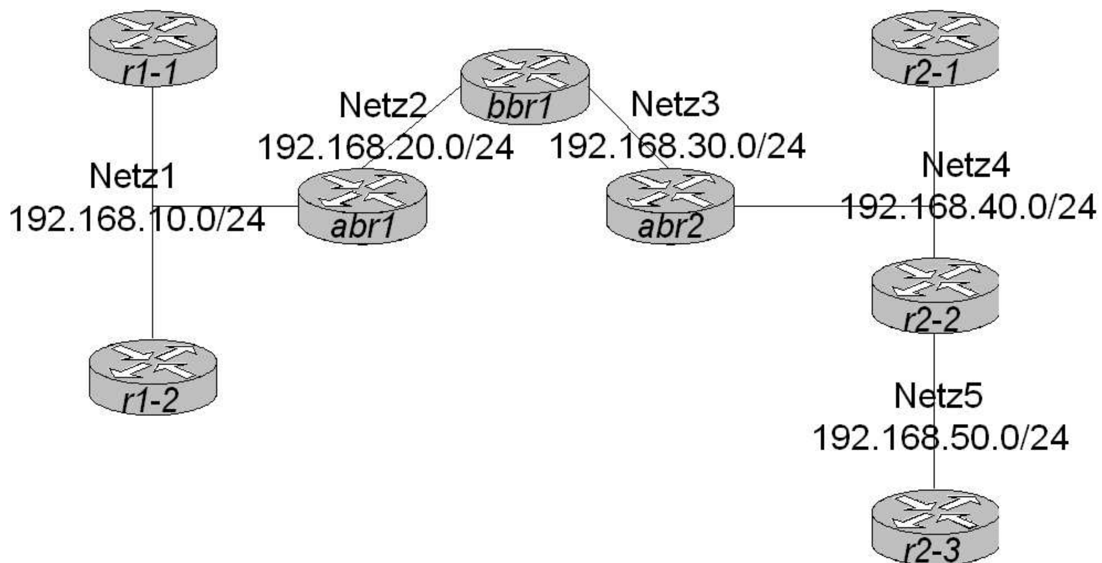
6.2 Normalbetrieb:

Im Alltagsgeschäft eines Routers wird der normale Betrieb einer intakten Netzwerktopologie den Großteil der Betriebszeit ausmachen. Daher darf eine Betrachtung des Aufwands in einem solchen Normalbetrieb nicht fehlen. Die interessanteste Größe bei dieser Untersuchung ist das Datenvolumen, das zusätzlich zur Nutzlast von den Routern erzeugt wird. Das Datenvolumen wird mit den Werkzeugen und Vorgehensweisen aus dem vorigen Abschnitt gemessen.

6.2.1 Normalbetrieb in einer kleinen Topologie

Designphase:

Die Simulation basiert auf der Topologie aus Abschnitt 5.3, allerdings muss die XML-Datei erweitert werden.



Diese Änderungen beziehen sich auf die Ausführungssequenzen, um das Datenvolumen zu messen und die Anpassungen der Start- und Stopp-Tags, um sowohl die RIP als auch die OSPF-Daemons optional zu starten. Die vorhandenen OSPF Konfigurationsdateien sollen so abgeändert werden, dass diesmal keine Hierarchisierung durch Areas vorgenommen werden. Die Konfigurationsdateien von RIP müssen neu erstellt werden.

Implementierungsphase:

An der XML-Beschreibung von Beispiel 5.3 sind einige Erweiterungen notwendig um das Beispiel an die Designvorgaben anzupassen. In jeden Abschnitt einer virtuelle Maschine werden folgende Einträge entsprechend der Schnittstellenanzahl hinzugefügt, um die Firewall zu aktivieren. Dieser Eintrag bewirkt dann dass `iptables` den ausgehenden Verkehr über diese Schnittstelle protokolliert.

```
<exec seq="startIptables" type="verbatim">iptables -A OUTPUT -o eth1</exec>
```

Weiterhin bekommt jede virtuelle Maschine folgende Ergänzungen, die es erlauben die zebra-, ripd- und ospfd-Daemons getrennt zu starten, sowie eine gemeinsame Stopp-Sequenz.

```
<exec seq="startRip" type="verbatim">ripd -f /usr/local/etc/alpha.conf
   -d -P 2604</exec>
<exec seq="startOspf" type="verbatim">ospfd -f /usr/local/etc/alpha.conf
   -d -P 2604</exec>
<exec seq="startZebra" type="verbatim"></exec>

<exec seq="stop" type="verbatim">killall zebra</exec>
<exec seq="stop" type="verbatim">killall ripd</exec>
<exec seq="stop" type="verbatim">killall ospfd</exec>
```

Die Konfigurationen für die RIP- und OSPF-Router werden wieder in getrennten Unterverzeichnissen abgelegt. Die OSPF-Router werden so konfiguriert, dass sich alle Router in der Area 1 befinden.

```
OSPF-Konfiguration von Router ABR1:

!
hostname ospfd
password zebra
log stdout
!
router ospf
network 192.168.10.0/24 area 1
network 192.168.20.0/24 area 1
line vty
```

```
!
RIP Konfiguration von Router ABR1:
!
hostname ospfd
password zebra
log stdout
!
router rip
network 192.168.10.0/24
network 192.168.20.0/24
line vty
!
```

Ausführungsphase:

Das Szenario wird mit folgendem Befehl gestartet:

```
vnumlparser.pl -t ospfVsRip2.xml -vB
```

Nachdem die acht Router erfolgreich hochgefahren sind können die Zebra-Daemons gestartet werden und die Firewall für die Protokollierung der gesendeten Datenvolumina eingerichtet werden.

```
vnumlparser.pl -x startZebra@ospfVsRip2.xml -vB
vnumlparser.pl -x startIptables@ospfVsRip2.xml -vB
```

Um die Simulation mit RIP-Konfiguration zu durchlaufen, müssen anschließend die RIP-Daemons gestartet werden.

```
vnumlparser.pl -x startRip@ospfVsRip2.xml -vB
```

Da die laufende Simulation den Verkehr messen soll, den die Router im Normalbetrieb verursachen, ist es sinnvoll, die Messung erst nach der Initialisierungsphase zu starten. Nachdem ausreichend lange gewartet wurde und die Router die Initialisierungsphase durchlaufen haben, kann die Messung gestartet werden. Dazu werden die Zähler von `iptables` auf null zurückgesetzt.

```
vnumlparser.pl -x resetCounter@ospfVsRip2.xml -vB
```

Nun wird gewartet, bis eine beliebige festgelegte Zeitspanne abgelaufen ist. In diesem exemplarischen Durchlauf wird das Datenvolumen, das von den Routern in einem Zeitraum von 10 Minuten verursacht wird, gemessen.

Nach Ablauf dieser Zeit müssen die Zählerstände der Firewall ausgelesen werden. Die Ausführung von

```
vnumlparser.pl -x showCounter@ospfVsRip2.xml -vB > ergebnisRip2-10m.txt
```

liest die Zähler aus und leitet die Ausgabe in die Datei ergebnisRip2-10m.txt um. Diese Ergebnisdatei kann anschließend wieder mit

```
usr/local/share/vnuml/examples/parse.pl ergebnisRip2-10m.txt
```

in eine lesbarere Form gebracht werden:

```
bbr1
eth0 4312 48
eth1 2240 20
eth2 2240 20
Summe: 4480

r1-1
eth0 4208 46
eth1 2352 21
Summe: 2352

r1-2
eth0 4208 46
eth1 2352 21
Summe: 2352

abr1
eth0 4208 46
eth1 2352 21
eth2 2352 21
Summe: 4704

abr2
eth0 4312 48
eth1 2240 20
```

```
eth2 2240 20
Summe: 4480

r2-1
eth0 4260 47
eth1 2240 20
Summe: 2240

r2-2
eth0 4104 44
eth1 2128 19
eth2 2128 19
Summe: 4256

r2-3
eth0 4052 43
eth1 2352 21
Summe: 2352

Gesamtsumme (ohne eth0): 27216
```

Die zehnminütige Messung hat ergeben, dass die RIP-Router zusammen 27216 Bytes gesendet haben, da es wieder keine Nutzdaten während der Simulation gab, ist dies der reine Verkehr, der durch den Routing-Algorithmus verursacht wurde.

Um das Szenario nun in eine OSPF-Topologie umzubauen, müssen zunächst alle Routing-Daemons angehalten werden.

```
vnumlparser.pl -x stop@ospfVsRip2.xml -vB
```

Anschließend können die Zebra-Daemons und die OSPF-Daemons gestartet werden.

```
vnumlparser.pl -x startZebra@ospfVsRip2.xml -vB
vnumlparser.pl -x startOspf@ospfVsRip2.xml -vB
```

Nachdem die Initialisierungsphase der OSPF-Router abgeschlossen ist, müssen die Zähler der Firewall zurückgesetzt werden, damit die vorhergehenden Datenpakete nicht mit in die Messung eingehen und dadurch das Ergebnis deutlich verändern.

```
vnumlparser.pl -x resetCounter@ospfVsRip2.xml -vB
```

Nachdem die definierte Zeitspanne abgelaufen ist, werden die Ergebnisse wieder in einer Datei gesichert und anschließend mit dem Hilfsprogramm parse.pl in einer lesbareren Form ausgegeben.

```
vnumlparser.pl -x showCounter@ospfVsRip2.xml -vB > ergebnisOspf2-10m.txt
usr/local/share/vnuml/examples/parse.pl ergebnisRip2-10m.txt
```

Im Ergebnis des exemplarischen Durchlaufs hat OSPF in 10 Minuten einen Datenverkehr von 50748 Bytes verursacht.

Dieses Ergebnis ist vielleicht auf den ersten Blick überraschend, denn in der Lehr-Literatur wird immer hervorgehoben, dass der RIP-Algorithmus durch seinen Update-Mechanismus wesentlich mehr Verkehr verursacht. Dies gilt allerdings nur in “größeren” Netzwerken. Daher wird die Topologie im folgenden Versuch vergrößert um anschließend erneut Messungen vorzunehmen.

6.2.2 Normalbetrieb in einer größeren Topologie

Designphase:

Die Topologie aus dem vorhergehenden Beispiel wird um zehn Netzwerke auf insgesamt fünfzehn Netzwerke ausgebaut. Dies geschieht dadurch, dass an die Router r1-1, r1-2, r2-1, r2-2 und r2-3 jeweils zwei weitere Stummelnetzwerke angeschlossen werden. Die Topologie bleibt ansonsten unverändert. Dadurch soll erreicht werden, dass die Update Nachrichten der RIP-Router anwachsen und schließlich der aufkommende Datenverkehr der RIP-Router größer ist als das Datenvolumen der OSPF-Router.

Implementierungsphase:

Da die Topologie fast unverändert bleibt, gibt es nur einige wenige Änderungen bzw. Ergänzungen an der Modellbeschreibung. Die neuen Netzwerkschnittstellen werden jeweils mit einem <if>-Tag zu den Beschreibungen der virtuellen Maschinen ergänzt. Dementsprechend wird die Konfiguration der Firewall ebenfalls um einen Eintrag ergänzt, damit der Datenverkehr über die neu hinzugefügten Netzwerkschnittstellen ebenfalls protokolliert wird.

In den Konfigurationen der RIP und OSPF-Router werden diese neuen Schnittstellen ebenfalls ergänzt, ansonsten wird alles unverändert aus dem vorangegangenen Beispiel übernommen.

Ausführungsphase:

Der Start der Simulation erfolgt wie gewohnt über den Befehl:

```
vnumlparser.pl -t ospfVsRip2b.xml -vB
```

Anschließend werden der Zebra-Daemon und die Firewall `iptables` gestartet.

```
vnumlparser.pl -x startZebra@ospfVsRip2b.xml -vB
vnumlparser.pl -x startIptables@ospfVsRip2b.xml -vB
```

Um das Szenario zuerst mit dem RIP-Algorithmus zu starten, müssen die entsprechenden Prozesse gestartet werden:

```
vnumlparser.pl -x startRip@ospfVsRip2b.xml -vB
```

Nachdem die Initialisierungsphase durchlaufen wurde, können die Zähler der Firewall zurückgesetzt werden.

```
vnumlparser.pl -x resetCounter@ospfVsRip2b.xml -vB
```

Jetzt sollte wieder die Wartezeit verstreichen, bevor die Zählerstände ausgelesen werden. Im exemplarischen Durchlauf wurde wieder der Datenverkehr eines zehnminütigen Intervalls gemessen.

Danach wurden die Ausgaben der Zählerstände in die Datei ergebnisRip2b-10m.txt umgeleitet.

```
vnumlparser.pl -x showCounters@ospfVsRip2b.xml -vB > ergebnisRip2b-10m.txt
```

Diese Resultate können dann wieder mit dem Skript parse.pl in eine übersichtlichere Form gebracht werden.

Resultat RIP 2b 10 Minuten:

135720 Bytes

Um anschließend dasselbe Szenario mit dem OSPF-Algorithmus zu untersuchen, müssen zunächst alle Routing-Prozesse beendet werden und dann die Zebra und OSPF-Daemons gestartet werden.

```
vnumlparser.pl -x stop@ospfVsRip2b.xml -vB
vnumlparser.pl -x startZebra@ospfVsRip2b.xml -vB
vnumlparser.pl -x startOSPF@ospfVsRip2b.xml -vB
```

Die Messung beginnt wieder mit dem Zurücksetzen der Zählerstände, was zweckmäßigerweise erst nach Durchlaufen der Initialisierungsphase stattfindet.

```
vnumlparser.pl -x resetCounter@ospfVsRip2b.xml -vB
```

Um die Ergebnisse zu sichern, werden nach Ablauf des Erfassungszeitraums die Zählerstände wieder in eine Datei umgeleitet, um sie anschließend darzustellen.

```
vnumlparser.pl -x showCounter@ospfVsRip2b.xml -vB > ergebniss2bOspf-10m.txt
./parse ergebniss2bOspf-10m.txt
```

Resultat OSPF nach 10 Minuten: Gesamtsumme 90280 Bytes

Die Ergebnisse zeigen, dass in diesem, etwas größeren Netzwerk, OSPF deutlich weniger Datenverkehr verursacht als RIP. Bei noch größeren Netzwerken wären die Resultate noch eindeutiger. Diese Entwicklung entsteht dadurch, dass in den RIP-Update Paketen immer die kompletten Einträge der Weiterleitungstabelle mit gesendet werden, dadurch steigt der Datenverkehr mit dem Wachstum der Weiterleitungstabelle an. Zusätzlich werden die periodischen Updates auch auf den neu hinzugekommenen Netzwerken gesendet, um enventuelle neue Nachbarn zu bemerken.

Der OSPF-Algorithmus sendet nur seine Hello-Pakete periodisch. Diese sind unabhängig von der Größe der Weiterleitungstabelle und wachsen nur leicht an, wenn ein neuer Nachbar gefunden wird (4 Bytes pro Nachbar). Diese Hello-Pakete werden zwar auch auf den zehn neuen Netzwerken gesendet und das wesentlich häufiger als die RIP-Pakete (RIP-Update Intervall typisch 30 Sekunden OSPF-Hello Intervall 15 Sekunden), trotzdem ist die Gesamtsumme des Datenverkehrs, der durch RIP verursacht wird größer, da die Weiterleitungstabellen weitere zehn Einträge enthalten.

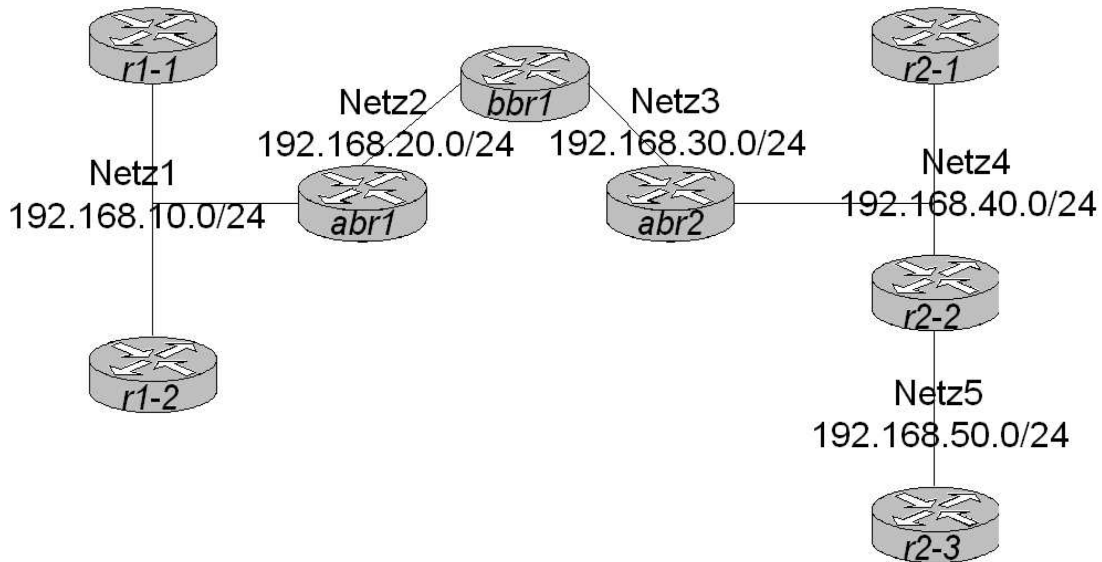
6.3 Im Fehlerfall:

Der Bedarf an dynamischen Routingprotokollen wurde im wesentlichen geweckt, um einen Fehlerfall schnell zu erkennen und die nötigen Reaktionen zu veranlassen, daher sollten die Routingprotokolle gerade in diesen Situationen überzeugen können. Der Vergleich der selbstorganisierenden Algorithmen in diesem Abschnitt erfolgt anhand verschiedener Kriterien. Neben den Datenmengen wird die Zeit gemessen, die benötigt wird, bis die Rekonvergenz wieder erreicht wird.

Als Grundlage für diese Untersuchungen dient die erweiterte Topologie aus dem vorangegangenen Abschnitt. Der Fehlerfall wird durch geeignete Filterregeln der Firewall realisiert.

Designphase:

Die Topologie aus Abschnitt 6.2.2 muss soweit verändert werden, dass es möglich wird Router r1-1 abzuschneiden.



Dazu müssen Ausführungssequenzen eingebaut werden, die es ermöglichen die Firewall `iptables` so zu konfigurieren, dass die Verbindung eth1 von r1-1 zu den restlichen Routern im Szenario unterbrochen werden kann, sowie die Unterbrechung wieder beseitigt werden kann. Gleichzeitig soll beim Ausführen der Unterbrechung auch der Zählerstand der Firewall, die den Datenverkehr protokolliert, zurückgesetzt werden.

Implementierungsphase:

Die geforderten Erweiterungen an der Konfiguration der Firewall betreffen nur die virtuelle Maschine r1-1. Die bekannten Skripte cutline.sh und repair.sh übernehmen das Unterbrechen bzw. Wiederherstellen der Verbindung eth1. Dazu werden die Skripte jeweils in eine Ausführunggssequenz verpackt und zu der XML-Modellierung der virtuellen Maschine r1-1 hinzugefügt.

```
Ausschnitt aus ospfVsRip3.XML:
```

```
<vm name="r1-1" >
  <if id="1" net="net1" >
    <ipv4 mask="255.255.255.0" >192.168.10.1</ipv4>
  </if>
  ...
  <exec seq="cutNet1" type="verbatim">cutline.sh eth1</exec>
  <exec seq="cutNet1" type="verbatim">iptables -Z</exec>
  <exec seq="repairNet1" type="verbatim">repair.sh eth1</exec>
  ...
</vm>
```

Die übrigen virtuellen Maschinen werden nur um den Eintrag zum Zurücksetzen der Zählerstände von iptables erweitert.

```
<exec seq="cutNet1" type="verbatim">iptables -Z</exec>
```

Ausführungsphase:

Die Simulation beginnt mit dem Hochfahren der virtuellen Rechner.

`vnumlparser.pl -t ospfVsRip3.xml -vB`

Danach werden die Router-Daemons für den ersten Durchlauf mit RIP gestartet und die Firewall in den Initialzustand versetzt. Diese Schritte sind bereits aus dem Abschnitt 6.2.2 bekannt.

```
vnumlparser.pl -x startZebra@ospfVsRip3.xml -vB
vnumlparser.pl -x startRip@ospfVsRip3.xml -vB
vnumlparser.pl -x startIptables@ospfVsRip3.xml -vB
```

Wenn die Router ihre Initialisierung abgeschlossen haben und die Weiterleitungstabellen vollständig erzeugt wurden, kann Router r1-1 von dem übrigen Szenario abgeschnitten werden.

```
vnumlparser.pl -x cutNet1@ospfVsRip3.xml -vB
```

Gleichzeitig werden durch diese Ausführungssequenz die Zähler der Firewall zurückgesetzt. Nun sollte die Routing-Tabelle des RIP-Routers r1-2, also dem direkten Nachbarn von r1-1, beobachtet werden.

```
host:~ # ssh r1-2
Password:
r1-2:~# telnet localhost 2604
Trying ::1...
Connected to localhost.
Escape character is '^]'.
Hello, this is quagga (version 0.96.4).
Copyright 1996-2002 Kunihiro Ishiguro.
User Access Verification
Password:
ripd> enable
ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface
     Network          Next Hop      Metric From          Tag Time
R(n) 172.17.0.0/24    192.168.10.3    6    192.168.10.3  0   03:00
R(n) 172.17.10.0/24   192.168.10.1    2    192.168.10.1  0   01:49
R(n) 172.17.20.0/24   192.168.10.1    2    192.168.10.1  0   01:49
C(i) 172.17.30.0/24   0.0.0.0         1    self          0
C(i) 172.17.40.0/24   0.0.0.0         1    self          0
R(n) 172.17.50.0/24   192.168.10.3    5    192.168.10.3  0   03:00
R(n) 172.17.60.0/24   192.168.10.3    5    192.168.10.3  0   03:00
R(n) 172.17.70.0/24   192.168.10.3    5    192.168.10.3  0   03:00
R(n) 172.17.80.0/24   192.168.10.3    5    192.168.10.3  0   03:00
R(n) 172.17.90.0/24   192.168.10.3    6    192.168.10.3  0   03:00
C(i) 192.168.10.0/24  0.0.0.0         1    self          0
R(n) 192.168.20.0/24  192.168.10.3    2    192.168.10.3  0   03:00
R(n) 192.168.30.0/24  192.168.10.3    3    192.168.10.3  0   03:00
R(n) 192.168.40.0/24  192.168.10.3    4    192.168.10.3  0   03:00
R(n) 192.168.50.0/24  192.168.10.3    5    192.168.10.3  0   03:00
```

Die hervorgehobenen Zeilen zeigen, dass zum Zeitpunkt dieser Abfrage der Weiterleitungstabelle schon seit 1:11 Minute keine Updates mehr von Router r1-1 empfangen wurden, wenn diese Timer heruntergezählt wurden, wird der Router r1-2 die Unerreich-

barkeit verbreiten. Der Router, der diese Nachricht als letztes erhalten wird, ist Router r2-3, da dieser am weitesten entfernt ist. Erst wenn dieser Router r2-3 in seiner Weiterleitungstabelle die Netzwerke 172.17.20.0/24 und 172.17.30.0/24 mit einer Metrik von 16 versieht, stimmen die Realität und die Abbildungen in den Tabellen wieder überein.

Genau dieser Zeitpunkt sollte abgepasst werden, um die Zählerstände der Firewall auszulesen.

```
vnumlparser.pl -x showCounter@ospfVsRip3.xml -vB > ergebnisRip3.txt
```

Das Ergebnis eines exemplarischen Durchlaufs ergab, dass die RIP-Router einen Verkehr von 54364 Bytes verursachten. Die Zeit, die der RIP Algorithmus bis zur Rekonvergenz benötigt, hängt in einem gewissen Maß vom Zeitpunkt der Unterbrechung ab. Das Standardintervall zwischen dem Senden der Updates ist bei Quagga auf 30 Sekunden +/- 50% eingestellt. Dies bedeutet, dass zwischen zwei Updates 15 - 45 Sekunden liegen können, im ungünstigsten Fall findet die Unterbrechung unmittelbar nach der Ankunft eines Update Paketes statt. In diesem Fall dauert es volle 3 Minuten bis der Nachbar für unerreichbar gehalten wird. Im günstigsten Fall würde die Unterbrechung unmittelbar vor dem Eintreffen eines Update Paketes stattfinden, auf welches schon 45 Sekunden gewartet wurde. In diesem Fall wären diese 45 Sekunden bereits verstrichen und die Unerreichbarkeit würde bereits 2:15 Minuten nach dem Starten der eigentlichen Unterbrechung propagiert. Die Laufzeit solcher "triggeredUpdates" durch ein so kleines Netzwerk ist zu vernachlässigen. Der entfernteste Router r2-3 bekommt die Unerreichbarkeit der Netzwerke so schnell, dass diese Laufzeit im Vergleich zu den ca. 3 Minuten Wartezeit verschwindendt gering ist.

Um nun im Anschluss die gleichen Messungen mit OSPF-Routern durchzuführen, sollten zuerst wieder alle Routing-Prozesse angehalten werden und die Unterbrechung mittels Firewall behoben werden.

```
vnumlparser.pl -x stop@ospfVsRip3.xml -vB
vnumlparser.pl -x repairNet1@ospfVsRip3.xml -vB
```

Danach werden die zebra- und ospfd-Daemons gestartet.

```
vnumlparser.pl -x startZebra@ospfVsRip3.xml -vB
vnumlparser.pl -x startOspf@ospfVsRip3.xml -vB
```

Hier muss ebenfalls wieder die etwas länger andauernde Initialisierungsphase abgewartet werden. Anschließend kann, nachdem die Weiterleitungstabellen korrekt gebildet wurden, auch hier wieder die Unterbrechung herbeigeführt werden.

```
vnumlparser.pl -x cutNet1@ospfVsRip3.xml -vB
```

Nachdem nun einer der Nachbar Routern r1-2 oder abr1 für 40 Sekunden (Dead-Intervall) kein Hello-Paket von r1-1 erhalten hat, wird die Unerreichbarkeit an die anderen Router per LSA-Mitteilung verschickt. Nachdem der weitest entfernteste Router r2-3 diese Information erhalten hat, werden die Zählerstände der Firewalls abgefragt.

Dieser Vorgang läuft unter OSPF wesentlich früher ab, da hier der Standardwert für das Dead-Intervall bei 40 Sekunden liegt und das Hello-Intervall standardmäßig auf 10 Sekunden eingestellt ist. Dementsprechend wird auf Ausfälle bereits nach 30-40 Sekunden reagiert. Das Datenvolumen lag in den exemplarischen Ausführungen bei ca. 15300 Bytes.

Die Ergebnisse zeigen deutlich, dass OSPF schneller reagieren kann, weil die Timer-Intervalle wesentlich kleiner sind. Eine einfache Verkleinerung der RIP-Intervalle würde jedoch erheblich zur Vergrößerung der Netzwerklast führen. Bereits mit dem Standardwert des Update-Intervalls wird in großen Topologien, wie in Abschnitt 6.2.2 gezeigt, mehr Traffic verursacht als unter OSPF. Daher muss der Kompromiss zwischen dauernder Belastung im Normalbetrieb und Schnelligkeit im Fehlerfall geschlossen werden. Diesen Kompromiss gibt es bei OSPF ebenfalls, nur ist die Startegie deutlich besser und daher können bei vergleichsweiser niedrigerer Belastung der Netze die Timerintervalle erheblich verkürzt werden.

7 Nachbemerkungen und Ausblick

Mit VNUML können nahezu beliebige Netzwerkszenarien kreiert werden. Dadurch ist es möglich eine Laborumgebung zu schaffen, in der auch weitaus mehr untersucht wird als nur die hier bearbeiteten Routingverfahren. Durch diese Arbeit wird ein einfacher Einstieg in die Simulation mit Hilfe von VNUML geboten. Zusätzlich werden diverse Kernaspekte der Interior Gateway Protokolle RIP und OSPF verständlich vorgestellt. Die praktische Durchführung von Simulationen und Experimenten ermöglicht ein einfaches Erlernen und Vertiefen der gewonnen Kenntnisse. Teile dieser Arbeit im speziellen die Erzeugung des "Counting to Infinity" haben sich bereits in weiteren Arbeiten als hilfreich erwiesen. Somit wurde auch ein Teil zur Forschungsarbeit beigesteuert.

Es gibt eine Vielzahl von Werken über die hier beschriebenen Algorithmen, deren Schwerpunkt bei Erläuterung und theoretischen Ausführungen liegen. Weiterhin gibt es Bücher, die deutlich praxisoientierter sind. Leider beziehen sich diese Werke ausschließlich auf Cisco-Produkte. Diese Arbeit versucht einen Brückenschlag zwischen Theorie und Praxis und verirrt sich dabei absichtlich nicht in der enormen Anzahl von erschlagenden Details.

Weiterführende Arbeiten könnten an diesem Punkt ansetzen und noch stärker auf den praktischen und realen Einsatz der Router eingehen. Z.B. wird das für den echten Einsatz der Softwarerouter wichtige Thema Sicherheit in dieser Arbeit gänzlich vernachlässigt. Dabei ist das Thema Sicherheit im Sinne von "security" und im Sinne von "safety" für das Design von Netzwerken, deren Konfiguration und Betrieb essentiell. Neben dem Mechanismus des privelegierten und unpreveliegierten Benutzer gibt es noch eine Reihe weiterer Sicherheitskomponenten in der Routing-Suite Quagga, die nicht genannt werden. Weiterhin könnten noch zusätzliche Anregungen zu den Grundregeln des Networkdesign innerhalb von autonomen Systemen gegeben werden. Außerdem ist es möglich, neben dem Erstellen der Netzwerke, in einer Laborumgebung auf Basis von VNUML das Management von Netzwerken im laufenden Betrieb zu üben.

Die Simulation von Netzwerken mit Hilfe von VNUML sind flexibel und realistisch, so dass hier ein großes Potential liegt. Gleichzeitig kann das Nachstellen von kleinen bis

mittleren Topologien nicht kostengünstiger sein.

Literaturverzeichnis

[THO03] Thomas M. Thomas II: OSPF Network Design Solutions, Second Edition, Ciscopress, 2003

[PAR02] William R. Parkhurst: Cisco OSPF Command ans Configuration Handbook, Ciscopress, 2002

[DOY98] Jeff Doyle: Routing TCP/IP Volume 1, Ciscopress, 1998

[VNUML] VNUML Homepage vom 21.03.2005 URL: http://jungla.dit.upm.es/~vnuml

[tcpdump] Manpages tcpdump, Version vom 03.01.2001, URL: http://www.tcpdump.org

[ethereal] Manpages ethereal, Version vom 23.09.2003, URL: http://www.ethereal.com

[iptables] Manpages iptables, Version vom 09.03.2002, URL: http://www.netfilter.org

Eigenständigkeitserklärung:

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit mit dem Titel “Simulation von Interior Gateway Protokollen in virtuellen Netzen” selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Rüber, 06.06.2006 Martin Krechel