UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Verteilte Simulation großer Netzwerke mit VNUML und EDIV

Bachelorarbeit

zur Erlangung des Grades eines
Bachelor of Science
im Studiengang Informatik

vorgelegt von

Nicolas Schönfeld

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

Koblenz, im November 2011

Erklärung

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

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

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Verteilung großer virtueller Rechnernetze auf mehrere physische Hosts unter Verwendung der beiden Virtualisierungstools VNUML und EDIV. Dabei ist VNUML für die eigentliche Simulation des Netzwerks zuständig, während EDIV in erster Linie für die entsprechende Verteilung sorgt. Nach einer kurzen Erklärung grundlegender Begriffe und Konzepte aus dem Gebiet der Netzwerksimulation wird zunächst ausführlich auf die beiden erwähnten Virtualisierungstools eingegangen. Dies beginnt jeweils mit einer detaillierten Beschreibung hinsichtlich der korrekten Installation und Konfiguration, gefolgt von einer Demonstration der wichtigsten Funktionalitäten, wie das Starten oder Beenden einer Simulation. Auch auf das Erstellen eines geeigneten Netzwerkszenarios und auf die von EDIV bereitgestellten Skripte zur Überwachung dieser Szenarien wird in diesem Zusammenhang näher eingegangen. Um die vorgestellten Möglichkeiten der beiden Tools auch in der Praxis anwenden zu können, wird zum Schluss ein eigenes Netzwerkszenario entworfen und auf mehrere Rechner verteilt, sodass die verschiedenen Funktionen von EDIV vorgeführt und beschrieben werden können.

Abstract

This thesis deals with the distribution of large virtual networks to multiple physical hosts using the virtualization tools VNUML and EDIV. While VNUML is responsible for simulating the network, EDIV has its focus primarily on the distributiion. After a short explanation of basic terms and concepts from the field of network simulation, the two previously mentioned virtualization tools are described in detail. This starts in both cases with a detailed description regarding the proper installation and configuration, followed by a demonstration of the most important functions, such as starting or stopping a simulation. The creation of a suitable network scenario and the information scripts provided by EDIV are two further points described in this context. To use the presented possibilities of those tools in practice, an own network scenario is designed and distributed to several hosts, so that the different functions of EDIV can be shown and described accurately.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Große Rechnernetze bestehen für gewöhnlich aus einer großen Anzahl an Routern und Hostsystemen, die teuer und aufwendig zu warten sind und nur eine geringe Entscheidungsfreiheit im Einsatz lassen. Um die Kosten und den Aufwand zur Wartung zu vermeiden, aber auch um den Einsatz neuer Software in einem Rechnernetz zu testen oder diese weiterzuentwickeln, wird Virtualisierungssoftware wie VNUML (siehe Kapitel 3) eingesetzt. Diese Software ermöglicht es, relativ große Linux-Rechnernetze innerhalb eines physischen Hostsystems zu simulieren. Die einzelnen Linux-Rechner basieren dabei auf User-Mode Linux (siehe Kapitel 2.1), was die Emulation eines voll-funktionalen Linux-Rechners ermöglicht.
Mithilfe der einfachen Topologie-Beschreibungssprache, die VNUML bietet, können beliebige Netzwerk-Topologien spezifiziert und aufgebaut werden. Diese virtuellen Linux-Rechnernetze können als übersichtliche Testumgebung zum Analysieren und Entwickeln von Netzwerkprogrammen und -protokollen verwendet werden und bieten damit gerade im Hochschulbereich eine sehr gute Möglichkeit, Interessierte an das Thema Rechnernetze heranzuführen und damit experimentieren zu lassen. Größe und Komplexität der virtuellen Rechnernetze sind allerdings durch die vorhandenen, freien Ressourcen des physischen Hostsystems begrenzt.
Da die Ressourcen eines einzelnen Hostsystems zur Erzeugung sehr großer Netze nicht ausreichen, wurde von den VNUML Autoren die Software EDIV (siehe Kapitel 4) entwickelt. Sie ermöglicht es, ein VNUML-Netzwerk-Szenario auf mehrere, über ein Netzwerk miteinander verbundene, physische Hostsysteme zu verteilen und steht daher im Mittelpunkt dieser Bachelorarbeit.

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist es, ein Netzwerk-Szenario mit VNUML aufzubauen und es mithilfe von EDIV, auf mehreren physischen Hostsystemen verteilt, zu starten und zu testen. Dazu wird zunächst eine Netzwerkumgebung aus mehreren Linux-Rechnern aufgebaut, die als Netzwerkknoten für das verteilte VNUML-Szenario verwendet werden. Nach dem Entwurf eines Netzwerk-Szenarios mithilfe der dafür vorgesehenen VNUML-Sprache, wird dieses mittels EDIV auf den physischen Hostsystemen der Netzwerkumgebung verteilt, ausgeführt und konfiguriert. Der Umgang mit VNUML und EDIV wird dabei von der Installation bis hin zu den verschiedenen Konfigurationsmöglichkeiten, die EDIV bietet, ausführlich dokumentiert, sodass es für Interessierte leicht zu verstehen und reproduzieren sein wird. Nach Lektüre dieser Arbeit sollte der Leser mit den vorgestellten Tools, VNUML und EDIV, im Wesentlichen umzugehen wissen und in der Lage sein bei Interesse eigene Szenarios entwerfen, ausführen und analysieren zu können.

Kapitel 2

Grundlagen

2.1 User-Mode Linux

Bei User-Mode Linux (UML) handelt es sich um eine von Jeff Dike als Open-Source-Projekt entwickelte Virtualisierungssoftware, die es ermöglicht innerhalb von Linux ein weiteres Linux-System zu simulieren. Dazu wird für das zu simulierende Linux-System einfach ein Kernel als Anwendungsprozess gestartet. Dieser läuft im Userspace, was bedeutet, dass der Kernel nur eingeschränkte Rechte besitzt und nicht die Kontrolle über das gesamte System hat. Es ist sogar möglich in einem Linux-System gleichzeitig mehrere, bei Bedarf auch verschiedene, Linux-Kernel als Prozesse zu starten. So kann beispielsweise ein Server in mehrere virtuelle Linux-Distributionen unterteilt werden, die dann von verschiedenen Usern gleichzeitig genutzt werden können. Besonders nützlich ist dieses Prinzip der Virtualisierung, um die virtuellen Linux-Systeme für Testzwecke oder als Entwicklungsumgebung zu verwenden. Das „echte“ Linux-System wird dabei von Fehlfunktionen oder Abstürzen der virtuellen Systeme nicht beeinflusst. Der für diese Arbeit wichtigste Punkt ist jedoch die Möglichkeit, die virtuellen Linux-Systeme auch untereinander vernetzen zu können, was letztendlich auch die Grundlage für die Entwicklung von VNUML (siehe Kapitel 3) war. [Dik06] [Keu05]

2.2 Overlay-Netzwerk

Als Overlay-Netzwerk wird ein logisches Netz bezeichnet, das auf einem bereits vorhandenem Netzwerk aufbaut. Dies kann entweder ein physisches Netz oder selbst auch wieder ein virtuelles Overlay-Netz sein. Dabei ist es möglich, mehrere Overlay-Netzwerke auf demselben darunterliegenden Netzwerk zu implementieren. Ebenso gut kann sich aber auch ein einzelnes Overlay-Netz über mehrere existierende Netze erstrecken. Jeder Knoten in einem Overlay-Netzwerk existiert auch im darunterliegenden Netz, wobei das darunterliegende Netz darüber hinaus noch viele weitere Knoten besitzen kann. Verbindungen zwischen Knoten im Overlay-Netzwerk

werden dabei als Tunnel durch das darunterliegende Netz implementiert. Overlay-Netzwerke werden hauptsächlich dafür eingesetzt, neue Technologien oder Funktionen zu implementieren, die im darunterliegenden Netz bislang noch nicht verfügbar waren. So gibt es beispielsweise ein Overlay-Netzwerk für IPv6, um IPv6 Pakete über ein IPv4 Netzwerk zu versenden. Auch VPNs (Virtual Private Networks), die dem Verschlüsseln von Datenverkehr dienen, werden als Overlay-Netzwerk implementiert. Das Internet selbst ist sogar ein Overlay-Netzwerk, da es auf dem Telefonnetz aufsetzt. [PD03]

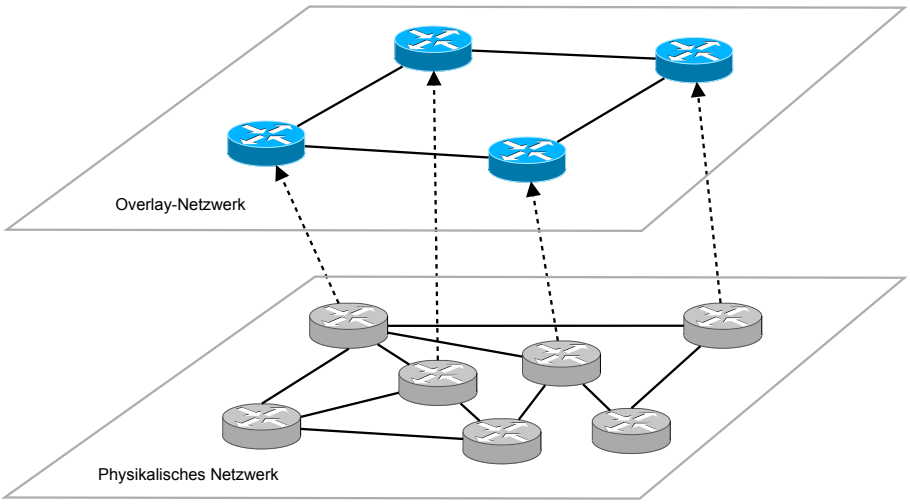

Abbildung 2.1: Overlay-Netzwerk

2.3 Virtual Local Area Networks

Virtual Local Area Networks (VLANs) sind virtuelle Teilnetze eines physischen Netzwerks. Sie unterteilen ein Netzwerk in mehrere getrennte Bereiche, die jeweils eine eigene Broadcast-Domäne darstellen. Broadcasts innerhalb eines VLANs bleiben also stets in diesem VLAN und sorgen für keinen unnötigen Traffic in benachbarten Netzen. Um ein VLAN zu errichten, sind besondere VLAN-fähige Switches notwendig. Wollen zwei unterschiedliche VLANs miteinander kommunizieren, so ist dies nur über einen Router möglich, der die beide VLANs verbindet. Abbildung 2.2 zeigt diese Situation.
Unterschieden werden im Wesentlichen zwei Arten, wie VLANs implementiert werden können:
Bei **implizit getaggten VLANs** erfolgt die Zuordnung der beteiligten Rechner zu ihren jeweiligen VLANs üblicherweise durch deren MAC-Adresse. Diese wird in

den Switches dem jeweiligen VLAN zugeordnet, in dem sich der Rechner befindet. Möchte nun ein Rechner Netzwerkpakete versenden, so wird durch den Switch anhand der Zuordnung von MAC-Adresse und VLAN implizit getaggt, zu welchem VLAN die Pakete gehören.
Bei **explizit getaggten VLANs** nach dem IEEE 802.1Q-Standard[1][oEE98] wird die Zugehörigkeit zu den einzelnen VLANs dadurch gekennzeichnet, dass jedem Netzwerkpaket explizit ein sogenanntes VLAN-Tag angehängt wird, das unter anderem die VLAN-ID enthält. Dieses VLAN-Tag ist eine zusätzliche Markierung, die anzeigt zu welchem virtuellen Netzwerk das jeweilige Paket gehört.
Eingesetzt werden VLANs vor allem, um Netze besser strukturieren zu können. So können Rechner, unabhängig von ihrem Standort im physischen Netzwerk, so gruppiert werden, dass jedes VLAN eine bestimmte Funktion erfüllt oder für eine bestimmte Benutzergruppe eingerichtet wird. Dies sorgt gleichzeitig für ein höheres Maß an Sicherheit, da die Einteilung in verschiedene VLANs neben dem bereits erwähnten Aspekt des reduzierten Broadcast-Traffics auch dafür sorgt, dass vertrauliche Daten innerhalb des jeweiligen VLANs bleiben und nicht von Personen außerhalb dieses VLANs abgehört werden können. [Sys02] [Tan03] [Sch04]

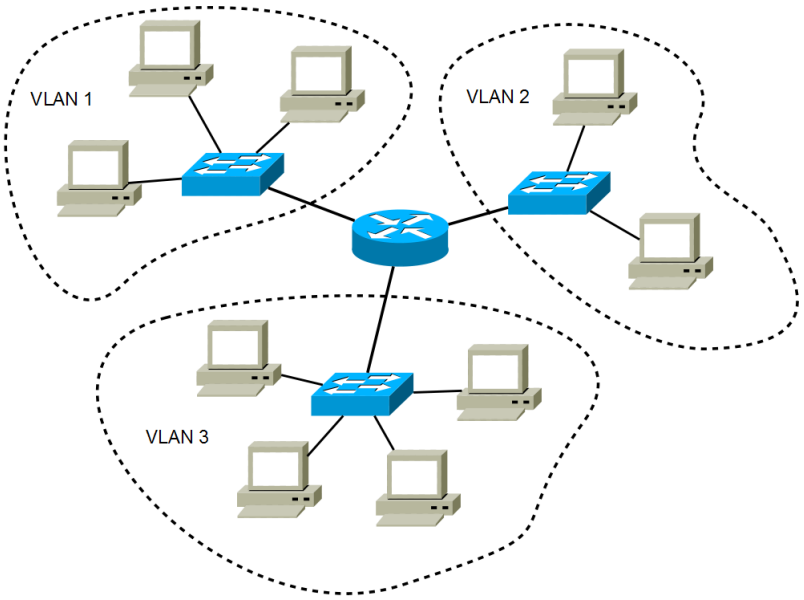

Abbildung 2.2: Netzwerk mit VLANs

[1]Vom Institute of Electrical and Electronics Engineers definierter Standardisierungsvorschlag für *tagged VLANs*.

Kapitel 3

VNUML

Bei VNUML (**V**irtual **N**etwork **U**ser **M**ode **L**inux) handelt es sich um eine Open-Source Virtualisierungssoftware, die als Teil des Euro6IX[1] Forschungsprojekts an der Technischen Universität Madrid entwickelt wurde.
Ziel von VNUML ist es, den Aufbau und Test großer virtueller Linux-Rechnernetze auf nur einem physischen Hostsystem zu ermöglichen. Basierend auf User-Mode Linux werden die einzelnen Linux-Rechner simuliert und mit virtuellen Netzen verbunden. Die simulierten Linux-Rechner, deren Dateisystem als eine Image-Datei auf dem Hostrechner gespeichert wird, können dabei beliebig konfiguriert werden, wobei komplexe UML-spezifische Details bewusst vor dem Benutzer verborgen werden. Somit ist es möglich, mit einfachen und vor allem kostensparenden Mitteln, verschiedenste Netzwerkszenarios zu simulieren. Dabei hilft insbesondere die leicht verständliche Topologie-Beschreibungssprache, durch die Szenarios in XML[2] beschrieben werden können.
Die Einsatzgebiete von VNUML sind insbesondere Universitäten oder andere Forschungseinrichtungen, die einerseits eine möglichst realitätsnahe Umgebung für die Forschung im Bereich der Rechnernetze suchen, andererseits aber auch die Kosten und den Konfigurationsaufwand scheuen, den große physische Rechnernetze mit sich bringen. Für VNUML wird lediglich ein einziges Hostsystem gebraucht, um ein ganzes Netzwerk zu erzeugen. Da die Konfiguration der einzelnen Komponenten des Netzwerks nun in Dateien auf dem Hostsystem gespeichert ist, fällt der Neustart des Szenarios oder das Zurücksetzen auf einen vorherigen Zustand erheblich einfacher als dies in einem physischen Netzwerk der Fall wäre. VNUML bietet somit eine perfekte Spielwiese zum Entwerfen eigener Netzwerk-Szenarios oder zur Entwicklung und Analyse von Netzwerkprotokollen.

[1] Das Euro6IX Projekt wurde ins Leben gerufen, um die Verbreitung von IPv6 in Europa zu beschleunigen. Weiterführende Informationen finden sich im Internet unter: http://www.euro6ix.org

[2] XML steht für Extensible Markup Language und ist eine Auszeichnungssprache, die dazu dient, Dokumente in Form einer Baumstruktur zu erstellen, die sowohl für Menschen, als auch Maschinen lesbar sind.

3.1 Installation

Es gibt eine Reihe von Möglichkeiten, an eine lauffähige Version von VNUML zu gelangen. So ist es unter anderem möglich, VNUML zu verwenden, ohne es zuvor installieren zu haben. Zu diesem Zweck haben die Entwickler von VNUML auf ihrer Homepage[3] eine *VNUML-Live-DVD* zum Download bereitgestellt. Sie ist besonders für diejenigen geeignet, die VNUML auf einfache und schnelle Art kennenlernen möchten. Neben der Live-DVD ist dazu lediglich ein ausreichend großer Arbeitsspeicher erforderlich.
Eine weitere Möglichkeit VNUML kennen zu lernen, bietet *coLinux/andLinux*. Bei coLinux handelt sich um eine Software, die es erlaubt, virtuelle Linux-Maschinen unter Windows zu starten. Dadurch ist es möglich, VNUML auch unter Windows zu verwenden. Wird zusätzlich auch eine grafische Oberfläche gewünscht, so ist andLinux statt coLinux zu installieren. Eine VNUML-Version von coLinux/andLinux kann von der VNUML-Seite[4] der Universität Koblenz heruntergeladen werden.
Für ein intensiveres Arbeiten mit VNUML ist aufgrund der komfortableren Nutzung eine *statische Installation* auf einem Linux-System allerdings unerlässlich. Sie hat zudem den Vorteil, dass sie, im Gegensatz zu den anderen Installationsmöglichkeiten, stets in der aktuellsten VNUML-Version zur Verfügung steht. Auf die statische Installation wird daher im Folgenden näher eingegangen. Eine recht ausführliche Anleitung in Englisch kann zudem bei [Mad] gefunden werden.

3.1.1 Debianbasierte Linux-Distributionen

Für Benutzer von Debian-Systemen oder darauf basierender Distributionen wie Ubuntu, bietet sich die sehr einfache Installation mittels .deb-Paket an. Der große Vorteil einer Installation durch ein .deb-Paket liegt darin, dass alle Abhängigkeiten automatisch erkannt und mitinstalliert werden. Um VNUML zu installieren, muss lediglich die Liste der Paketquellen (*/etc/apt/sources.list*) mit einem Texteditor geöffnet und um die Zeile

```
1 deb http://jungla.dit.upm.es/~vnuml/debian binary/
```

erweitert werden. Nach einer Aktualisierung der Paketquellenliste kann VNUML nun einfach über die Paketverwaltung installiert werden. Zusätzlich sollten noch die Pakete *vlan*, *xterm*, *bridge-utils* und *screen* installiert werden, da sie im Umgang mit VNUML oftmals Anwendung finden.

[3] Offizielle Homepage der VNUML-Entwickler:
http://jungla.dit.upm.es/~vnuml/

[4] VNUML-Seite der Universität Koblenz:
http://www.uni-koblenz.de/~vnuml/index.de.php

3.1.2 Sonstige Linux-Distributionen

Es gibt zwei Möglichkeiten, VNUML für Linux-Distributionen zu installieren, die nicht auf Debian basieren und daher nicht mit .deb-Paketen umgehen können.
Eine etwas aufwendige und fehleranfällige Methode besteht darin, das *Quellcode-Paket* herunterzuladen und Schritt für Schritt alle notwendigen Pakete und Module selbst zu installieren. Auf der offiziellen VNUML-Seite finden sich zahlreiche Howtos für verschiedene Distributionen, die einen dabei unterstützen. Um diese aufwendige Prozedur zu automatisieren, befindet sich in dem Quellcode-Paket aber auch ein Installations-Skript, das mit den Befehlen

```
./configure
make
make install
```

ausgeführt wird und daraufhin alle notwendigen Schritte automatisch ausführt bzw. dem Benutzer mitteilt, was noch manuell installiert werden muss.
Die zweite Möglichkeit stellt der, von der Universität Koblenz entwickelte, *VNUML-Offline-Installer*[5] dar. Dabei handelt es sich um eine Zusammenstellung aller Pakete, Module und sonstiger Dateien, die für den Betrieb von VNUML benötigt werden. Ähnlich dem Installations-Skript aus dem Quellcode-Paket, enthält auch der Offline-Installer ein Shell-Skript, das sich um die korrekte Installation kümmert. Dazu muss besagtes Skript nur durch die Eingabe von *./install* im Ordner des entpackten Offline-Installers aufgerufen werden.
Neben der automatischen Installation hat der Offline-Installers den Vorteil, dass alle erforderlichen Dateien bereits enthalten sind und nichts mehr aus dem Internet nachgeladen werden muss. Der Nachteil dessen ist jedoch, dass die enthaltenen Pakete unter Umständen nicht auf dem aktuellsten Stand sind.

3.1.3 UML-Kernel und Dateisystem

Zwei Dinge werden nach einer statischen Installation noch benötigt, bevor VNUML verwendet werden kann: Ein UML-Kernel und ein Dateisystem. Andernfalls könnten die virtuellen Maschinen der VNUML-Szenarios nicht gestartet werden. Beide können auf der offiziellen VNUML-Seite[6] heruntergeladen werden. Im Offline-Installer der Universität Koblenz sind sowohl Kernel als auch Dateisystem sogar schon enthalten.
Wurde nun die aktuellste Version des UML-Kernels heruntergeladen, so muss dieser nur noch entpackt und in das dafür vorgesehene Verzeichnis kopiert werden. Je nachdem, ob VNUML über ein Debian- oder Quellcode-Paket installiert wurde, lautet das Verzeichnis */usr/share/vnuml/kernels* bzw. */usr/local/share/vnuml/kernels*.

[5]Download unter: http://www.uni-koblenz.de/~vnuml/down.de.php
[6]Download unter: http://neweb.dit.upm.es/vnumlwiki/index.php/Download

Zusätzlich sollte noch ein symbolische Verknüpfung angelegt werden, damit die Pfadangabe des Kernels in den Beispiel-Szenarios nicht verändert werden muss und beim Erstellen von eigenen Szenarios vereinfacht wird. Im Falle des in dieser Arbeit verwendeten Kernels, sähe das Anlegen einer symbolische Verknüpfung so aus:

```
1 ln -s /usr/share/vnuml/kernels/linux-2.6.28.10-1m /usr/share/vnuml/kernels/linux
```

Die Installation des Dateisystems läuft analog zum Kernel ab. Es muss heruntergeladen, entpackt und an die entsprechende Stelle im Ordner *vnuml* kopiert werden. Noch einfacher geht es mit einem Perl-Skript, das all diese Aufgaben übernimmt und sich zusätzlich um das Anlegen eines symbolischen Verknüpfung kümmert. Dazu müssen lediglich folgende zwei Zeilen in die Konsole eingegeben werden:

```
1 wget http://www.dit.upm.es/vnuml/download/scripts/root-fs-installer
2 perl root-fs-installer
```

Nachdem der Kernel und das Dateisystem erfolgreich installiert wurden, ist VNUML einsatzbereit. Um das Ganze zu testen, liegen unter */usr/share/vnuml/examples* bzw. */usr/local/share/vnuml/examples* bereits vorgefertigte Beispiel-Szenarios bereit, die nun einfach ausgeführt werden können. Wie dies genau funktioniert, wird im Folgenden beschrieben.

3.2 Tutorial

In dieser Einführung sollen nun die wichtigsten Funktionen von VNUML vorgestellt und ein kurzer Einblick in den Aufbau von Szenariodateien gegeben werden. Unterschieden wird bei VNUML prinzipiell zwischen zwei Hauptbestandteilen: Die VNUML-Sprache und der VNUML-Interpreter.

3.2.1 VNUML-Sprache

Bei der VNUML-Sprache handelt es sich um eine spezielle Variante der XML-Sprache. Mit ihr ist es möglich, Netzwerktopologien auf einfache Art zu beschreiben und in Form von XML-Dateien zu speichern. Diese Szenario-Dateien können dann von dem VNUML-Interpreter eingelesen und ausgeführt werden.
Die Strukturierung einer Szenario-Datei erfolgt dabei, wie jede XML-Datei, durch bestimmte Tags. Neben notwendigen globalen Tags, die für das gesamte Szenario gelten und beispielsweise zur Angabe der VNUML-Version, des Szenarionamens oder der Pfadangabe des Dateisystems und des UML-Kernels dienen, gibt es noch weitere, teils optionale Tags, die zur Definition der Netze und virtuellen Maschinen verwendet werden. Ein Beispiel für eine einfache Szenario-Datei, inklusive kurzer Erklärung der XML-Tags in Kommentarform, ist im Folgenden zu sehen. Es wurde aus [Gar10] übernommen und um weitere Kommentare ergänzt.

```
<!-- Standard-Header von XML-Dateien mit Pfadangabe zur DTD-Datei -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">

<!-- Beginn der eigentlichen Szenariobeschreibung -->
<vnuml>
  <!-- Festlegung globaler Einstellungen -->
  <global>
    <!-- Angabe der VNUML-Version -->
    <version>1.8</version>
    <!-- Frei wählbarer Name für die Simulation -->
    <simulation_name>test-szenario</simulation_name>
    <!-- MAC-Adressen der virtuellen Netzwerkschnittstellen werden automatisch generiert -->
    <automac/>
    <!-- Definiert die Einstellung zum Management-Netzwerk. "none" heisst, dass kein
     Management-Netz benutzt wird. Zugriff ist also nur über SSH möglich -->
    <vm_mgmt type="none" />
    <!-- Standard-Einstellungen für die virtuellen Maschinen -->
    <vm_defaults exec_mode="mconsole">
    <!-- Pfad zum Dateisystem für die virtuellen Maschinen -->
    <filesystem type="cow">/usr/share/vnuml/filesystems/root_fs_tutorial</filesystem>
    <!-- Pfad zum Kernel für die virtuellen Maschinen -->
    <kernel>/usr/share/vnuml/kernels/linux</kernel>
    <!-- Angabe der Konsole, mit der nach dem Start der Simulation auf die VMs zugegriffen
       werden kann -->
    <console id="0">xterm</console>
    </vm_defaults>
  </global>
  <!-- Ende der Festlegung globaler Einstellungen -->

  <!-- Definition der Netze -->
  <net name="Net0" mode="uml_switch" />
  <net name="Net1" mode="uml_switch" />
  <net name="Net2" mode="uml_switch" />

  <!-- Einstellungen für die 1. virtuelle Maschine "uml1" -->
  <vm name="uml1">
    <!-- Konfiguration der Netzwerkschnittstelle durch zugehöriges Netz und IP-Adresse -->
    <if id="1" net="Net0">
      <ipv4>10.0.0.1</ipv4>
    </if>
    <!-- Definition eines Default-Gateways innerhalb der VM -->
    <route type="ipv4" gw="10.0.0.3">default</route>
    <!-- Makro-Befehle, die über vnumlparser.pl -x [Startsequenz]@[Szenariodatei] initiiert
     werden können -->
    <!-- Beispiel: vnumlparser.pl -x start@test-szenario.xml -->
    <exec seq="start" type="verbatim">nohup /usr/bin/hello &lt;/dev/null &gt;/dev/null 2&gt;
     &amp;1 &amp;</exec>
    <exec seq="stop" type="verbatim">killall hello</exec>
  </vm>

  <!-- Einstellungen für die 2. virtuelle Maschine "uml2" -->
  <!-- Bedeutung ist analog zu "uml1" -->
  <vm name="uml2">
    <if id="1" net="Net0">
      <ipv4>10.0.0.2</ipv4>
    </if>
    <route type="ipv4" gw="10.0.0.3">default</route>
  </vm>

  <!-- Einstellungen für die 3. virtuelle Maschine "uml3" -->
```

```
62   <vm name="uml3">
63     <!-- Konfiguration mehrerer Netzwerkschnitstellen -->
64     <if id="1" net="Net0">
65       <ipv4>10.0.0.3</ipv4>
66     </if>
67     <if id="2" net="Net1">
68       <ipv4>10.0.1.1</ipv4>
69     </if>
70     <!-- Einrichten einer Route nach 10.0.2.0/24 -->
71     <route type="ipv4" gw="10.0.1.2">10.0.2.0/24</route>
72     <!-- Weiterleitung der IP-Pakete zwischen den Netzwerkschnittstellen aktivieren -->
73     <forwarding type="ip" />
74   </vm>
75
76   <!-- Einstellungen für die 4. virtuelle Maschine "uml4" -->
77   <vm name="uml4">
78     <if id="1" net="Net1">
79       <ipv4>10.0.1.2</ipv4>
80     </if>
81     <if id="2" net="Net2">
82       <ipv4>10.0.2.1</ipv4>
83     </if>
84     <route type="ipv4" gw="10.0.1.1">default</route>
85     <forwarding type="ip" />
86   </vm>
87
88   <!-- Einstellungen für die 5. virtuelle Maschine "uml5" -->
89   <vm name="uml5">
90     <if id="1" net="Net2">
91       <ipv4>10.0.2.2</ipv4>
92     </if>
93     <route type="ipv4" gw="10.0.2.1">default</route>
94   </vm>
95 </vnuml>
```

Abbildung 3.1: Beispiel einer VNUML-Szenariodatei

Eine ausführlichere Erklärung aller zur Verfügung stehender Tags findet sich bei [Mad]. Die Topologie des aus dieser XML-Datei erzeugten Szenarios zeigt die folgende Abbildung.

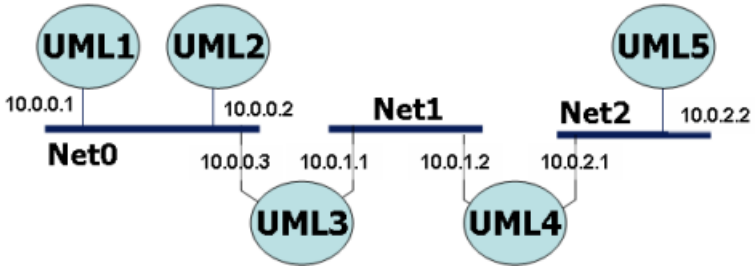

Abbildung 3.2: Topologie des VNUML-Beispielszenarios (VNUML-Tutorial in [Mad])

3.2.2 VNUML-Interpreter

Wurde mithilfe der VNUML-Sprache ein erstes Szenario erstellt, kommt der VNUML-Interpreter zum Einsatz. Er dient dazu, die jeweiligen Szenario-Dateien zu analysieren und aus den daraus gewonnenen Informationen über die Topologie des Szenarios, mithilfe von User-Mode Linux ein entsprechendes virtuelles Netzwerk aufzubauen.
Durch das Perl-Skript *vnumlparser.pl* können Szenarios gestartet, beendet oder verschiedene Aktionen während der Simulation ausgeführt werden. Alles was dazu in die Konsole eingegeben werden muss, ist *vnumlparser.pl* gefolgt von einem Parameter und dem Namen der Szenariodatei. Eine ausführliche Liste aller zur Verfügung stehenden Parameter, sowie eine kurze Erklärung ihrer Funktion, kann durch Eingabe von *vnumlparser.pl -H* aufgerufen werden.

Starten eines Szenarios

Vor dem erstmaligen Starten eines Szenarios sollte für den Fall, dass noch kein öffentlicher SSH-Schlüssel existiert, ein solcher erzeugt werden. Dadurch wird es später möglich, mittels SSH[7] auf die virtuellen Maschinen zuzugreifen, ohne jedes Mal nach einem Passwort gefragt zu werden. Erzeugen lässt sich ein solcher SSH-Schlüssel durch:

```
1 ssh-keygen -t rsa1
```

Die darauffolgenden Abfragen sollten alle mit [Enter] bestätigt werden.
Für den Fall, dass noch kein SSH-Client auf dem Rechner installiert war, muss dieser vorher erst noch installiert werden.

```
1 apt-get install openssh-client
```

Das Starten eines Szenarios erfolgt bei VNUML durch Eingabe von:

```
1 vnumlparser.pl -t Szenariodatei.xml -v
```

Dabei wird zunächst überprüft, ob das zu startende Szenario oder ein Szenario mit demselben Namen bereits aktiv ist. Sollte dies der Fall sein, bricht der VNUML-Interpreter mit einer entsprechenden Fehlermeldung ab.
Anderenfalls sorgt der Interpreter nun dafür, dass alle benötigten virtuellen Netzwerkschnittstellen eingerichtet und die einzelnen virtuellen Maschinen hochgefahren werden. Der Parameter *-t* ist dabei notwendig und für das Starten verantwortlich, wohingegen *-v* optional ist und den sogenannten Verbose-Modus aktiviert, der

[7] SSH (Secure Shell) ist ein Programm, das dazu dient, eine verschlüsselte Netzwerkverbindung mit einem entfernten Gerät herzustellen. Manpage: http://linux.die.net/man/1/ssh

dafür sorgt, dass die vom Interpreter ausgeführten Aktionen auf dem Bildschirm ausgegeben werden. Es ist daher empfehlenswert den Verbose-Modus immer zu aktivieren, um einen besseren Überblick zu haben, was gerade passiert oder wo eventuelle Fehler aufgetreten sind. Wurde das Szenario erfolgreich gestartet, öffnet sich für jede virtuelle Maschine ein xterm-Fenster, über das auf die VM zugegriffen werden kann. Der dafür benötigte Login-Name lautet *root* und das Passwort ist *xxxx*. Mit Root-Rechten kann alternativ auch per SSH auf die virtuellen Maschinen zugegriffen werden:

```
1 ssh VM-Name
```

Hierbei wird kein Passwort benötigt, da ja zuvor bereits ein öffentlicher SSH-Schlüssel angelegt wurde und ein entsprechender Eintrag in der Szenariodatei auf diesen verweist.
Um zu überprüfen, ob das Szenario richtig aufgebaut wurde und eine Verbindung zwischen den jeweiligen virtuellen Maschinen besteht, kann nun beispielsweise mit *ping*[8] oder *traceroute*[9] die Erreichbarkeit der anderen VMs getestet werden.

Kommandos während der Simulation

Um während der Simulation des Szenarios bestimmte vordefinierte Kommandos auszuführen, wie beispielsweise das Einrichten eines Routing-Protokolls, wird der Parameter *-x* verwendet.

```
1 vnumlparser.pl -x Kommando@Szenariodatei.xml -v
```

Das jeweilige Kommando muss zuvor für alle virtuellen Maschinen, auf denen es ausgeführt werden soll, in der Szenariodatei definiert werden. Dies geschieht durch spezielle <exec>-Tags, in denen die nötigen Befehle zur Ausführung des Kommandos stehen. Durch die Verwendung solcher Kommandos lässt sich oft viel Zeit und Arbeit sparen, da die nötigen Befehle sonst jedes Mal erneut von Hand eingeben werden müssten und zwar für jede virtuelle Maschine auf denen sie ausgeführt werden sollen.

Beenden eines Szenarios

Um Ressourcen zu sparen, sollte jedes Szenario auch wieder heruntergefahren werden, sobald es nicht mehr verwendet wird. Dies kann auf zwei Arten geschehen. Wird zum Beenden der Parameter *-d* verwendet, so werden alle virtuellen Maschinen des Szenarios heruntergefahren und die zuvor erstellten virtuellen Netzwerkschnittstellen wieder gelöscht. Änderungen am Dateisystem bleiben dabei jedoch erhalten, sodass sich der Benutzer bei einem erneuten Start des Szenarios wieder im selben Zustand wie zuvor befindet.

[8] Manpage: http://linux.die.net/man/8/ping
[9] Manpage: http://linux.die.net/man/8/traceroute

```
vnumlparser.pl -d Szenariodatei.xml -v
```

Eine radikalere Art ein Szenario zu beenden, lässt sich mithilfe des Parameters *-P* erreichen. Hierbei werden, neben dem Herunterfahren der virtuellen Maschinen und dem Löschen der virtuellen Netzwerkschnittstellen, auch alle Änderungen am Dateisystem, sowie sämtliche zu dem Szenario gehörende temporäre Dateien und Prozesse gelöscht. Wird das Szenario nun erneut gestartet, befindet sich dieses wieder im Anfangszustand.

```
vnumlparser.pl -P Szenariodatei.xml -v
```

Kapitel 4

EDIV

Bei dem Tool EDIV (spanisch: Escenarios Distribuidos con VNUML, deutsch: Verteilte Szenarios mit VNUML) handelt es sich um eine Sammlung von Skripten, die eine verteilte Simulation von VNUML-Szenarios auf mehreren Hostsystemen ermöglichen soll. Entwickelt wurde es an der Technischen Universität Madrid und stellt gewissermaßen eine Weiterentwicklung von VNUML dar.
Da die Ressourcen eines einzelnen Hostsystems zur Simulation eines großen und komplexen Szenarios oft nicht ausreichen, gab es bislang nur eine Möglichkeit, solch große Netze mit VNUML aufzubauen: Das Szenario musste aufgeteilt und jedes Teilszenario manuell administriert werden, was die Komplexität jedoch drastisch erhöhte. Um diese Aufgabe zu automatisieren und somit die problemlose Simulation nahezu unbeschränkt großer Netzwerke zu ermöglichen, wurde EDIV entwickelt. Aufgrund der Tatsache, dass es sich bei EDIV wie bereits erwähnt um eine Weiterentwicklung von VNUML handelt, ist der Umgang damit für Benutzer von VNUML sehr schnell zu erlernen.
Üblicherweise wird EDIV auf einem separaten Rechner installiert, der nicht zum Host-Cluster gehört und fortan als Kontrollrechner dient. Dies ist allerdings keine Pflicht, ebenso gut ist es möglich, EDIV auf einem der Hostsysteme zu installieren und das Szenario von dort aus zu steuern. Der Kontrollrechner hat generell die Aufgabe, die vom Benutzer erteilten Befehle entgegen zu nehmen und entsprechend umzusetzen. So ist er insbesondere dafür zuständig, die erstellten Szenarios mittels EDIV auf die Hosts zu verteilen und dort zu starten.
Die physischen Hostsysteme und der Kontrollrechner müssen stets, wie in Abbildung 4.1 dargestellt, über einen oder mehrere VLAN-fähige Switches verbunden sein, da EDIV VLANs verwendet um virtuelle Maschinen, die auf verschiedenen Hosts laufen, miteinander zu verbinden. [GF07]
Zum Überwachen derzeit laufender Simulationen stellt EDIV zudem mehrere Skripte zur Verfügung, die dem Nutzer das manuelle Auslesen von Informationen aus der Datenbank ersparen. Um während einer Simulation vorher definierte Befehle auf den Hosts auszuführen, sind auch die aus VNUML bereits bekannten Kommandosequenzen wieder nutzbar.

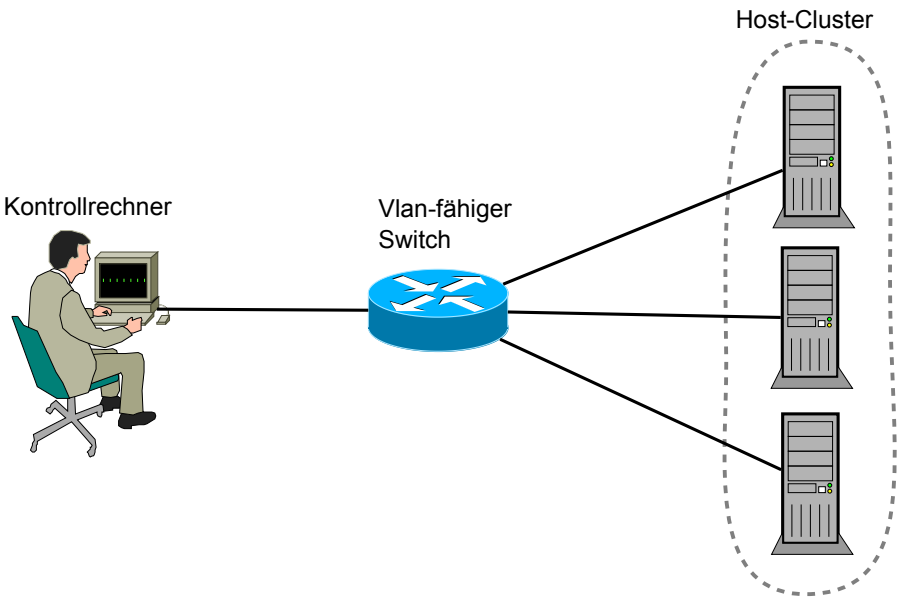

Abbildung 4.1: Hardware-Architektur für EDIV

4.1 Installation

Bevor nun die Installation von EDIV im Detail beschrieben wird, sollte zunächst sichergestellt werden, dass alle verwendeten Cluster-Hosts über eine vollständige und lauffähige VNUML-Installation verfügen. Zusätzlich muss auf jedem Host zuvor noch ein SSH-Server installiert werden, was unter Ubuntu beispielsweise mit dem Befehl

```
1 apt-get install openssh-server
```

erledigt wird. Bei anderen Distributionen funktioniert dies ähnlich, hier muss gegebenenfalls *apt-get* durch den entsprechenden distributionsspezifischen Befehl ersetzt werden. Um sicher zu gehen, dass SSH richtig funktioniert, sollte der SSH Server danach einmal neu gestartet werden.

```
1 /etc/init.d/ssh restart
```

Des Weiteren sollte auf jedem Host das Paket *vconfig* installiert werden. Dieses Paket ermöglicht es, die notwendigen VLAN-Interfaces für die verschiedenen VLANs zu erzeugen und zu löschen. Um zu guter Letzt sicherzustellen, dass der Kontrollrechner auch Kommandos mit Root-Rechten auf den Hostsystemen ausführen kann ohne jedes mal ein Passwort eingeben zu müssen, ist es nötig den öffentlichen SSH-Schlüssel des Kontrollrechners den Hosts bekannt zu machen. Dazu muss der Inhalt

der Datei */root/.ssh/id_rsa.pub* in die Datei */root/.ssh/authorized_keys* der jeweiligen Hosts kopiert werden. Bei Bedarf können auch die SSH-Schlüssel der anderen Hosts in diese Datei kopiert werden, falls der Benutzer von einem Host auf einen anderen zugreifen möchte, ohne dabei nach einem Passwort gefragt zu werden.
Wurden diese vorbereitenden Schritte ausgeführt, kann mit der eigentlichen Installation von EDIV begonnen werden. Wie bereits bei VNUML, so gibt es auch hier für Debian-Systeme die unkomplizierte Methode mittels .deb-Paket, sowie die etwas aufwendigere Variante durch ein Quellcode-Paket, die jedoch für jede Distribution geeignet ist.

4.1.1 Debianbasierte Linux-Distributionen

Wie schon bei der Installation von VNUML, ist auch die Installation von EDIV durch ein .deb-Paket in nur wenigen Schritten erledigt. Alles was dazu getan werden muss, ist auf dem Kontrollrechner die Zeile

```
1 deb http://jungla.dit.upm.es/~vnuml/debian binary/
```

in die Datei */etc/apt/sources.list* einzutragen und die beiden folgenden Befehle auszuführen:

```
1 apt-get update
2 apt-get install ediv
```

Dadurch wird zunächst die Paketquellenliste aktualisiert, um die neu hinzugefügte Paketquelle mit einzubeziehen. Danach wird EDIV mitsamt allen abhängigen Paketen installiert.
Wurde besagte Paketquelle bereits bei der Installation von VNUML in die Liste aufgenommen, so muss diese natürlich nicht erneut eingetragen werden. Hat alles funktioniert, so ist die Installation von EDIV damit bereits abgeschlossen.

4.1.2 Sonstige Linux-Distributionen

Sollte für Benutzer von Debian oder davon abgeleiteter Distributionen die Installation mittels .deb-Paket aus irgendwelchen Gründen nicht funktioniert haben, so kann EDIV auch manuell mithilfe eines Quellcode-Pakets installiert werden. Dies ist gleichzeitig auch die einzige Möglichkeit für Nutzer nicht-debianbasierter Distributionen, EDIV zu installieren. Dazu muss auf dem Kontrollrechner zunächst das benötigte Quellcode-Paket durch den Befehl

```
1 wget http://sourceforge.net/projects/vnuml/files/ediv/0.9.4/ediv_0.9.4.orig.tar.gz/download
      ?use_mirror=netcologne
```

heruntergeladen und in einen beliebigen Ordner entpackt werden.
Bevor EDIV nun installiert wird, sollte sichergestellt werden, dass sowohl Perl als auch bestimmte Perl-Module vorhanden sind. Perl lässt sich dabei einfach durch

```
apt-get install perl
```

installieren und der Befehl zum Installieren der benötigten Perl-Module und Bibliotheken lautet:

```
apt-get install libmath-round-perl libappconfig-perl libdbi-perl
```

Alternativ können die Perl-Module *AppConfig*[1] und *DBI*[2] auch manuell aus dem Comprehensive Perl Archive Network (CPAN) heruntergeladen und installiert werden, was jedoch umständlicher und daher nicht zu empfehlen ist.
Um die Installation von EDIV zu erleichtern, befindet sich in dem entpackten Quellcode-Paket ein Makefile, das die nötigen Schritte automatisiert, indem es die Dateien aus dem Quellcode-Paket an die richtigen Orte kopiert. Damit jedoch die im Quellcode-Paket enthaltenen Perl-Module an die richtige Stelle verschoben werden können, muss zunächst der Pfad für die lokalen Perl-Module herausgefunden und an der entsprechenden Stelle in das Makefile eingetragen werden. Dazu wird einfach der Befehl

```
perl -V
```

ausgeführt und einer der hinter der Variable *@INC* stehenden Pfade kopiert. Dieser wird nun in der ersten Zeile des Makefiles hinter *MODULES_INSTALL_DEST* eingefügt (z.B. *MODULES_INSTALL_DEST=/usr/local/share/perl/5.10.1*).
Wurde dies erledigt, müssen nur noch die Befehle

```
make install
make modules-install
```

ausgeführt werden und schon ist EDIV installiert.
Bei Bedarf kann EDIV auf ähnliche Weise auch wieder deinstalliert werden. Die Befehle hierzu lauten:

```
make uninstall
make modules-uninstall
```

All diese Befehle setzen jeweils voraus, dass das Paket *make* installiert ist. Sollte das nicht der Fall sein, so kann dies einfach mit

```
apt-get install make
```

[1]Download unter: http://search.cpan.org/dist/AppConfig/
[2]Download unter: http://search.cpan.org/dist/DBI/

nachgeholt werden. Neben der eigentlichen Installation von EDIV wird jetzt nur noch eine MySQL-Datenbank benötigt, in der EDIV wichtige Informationen über die jeweiligen Szenarios speichert. Die in Kapitel 4.4 erläuterten EDIV-Skripte greifen unter anderem auf diese Datenbank zurück, um entsprechende Informationen auszulesen. Die MySQL-Datenbank wird mit folgendem Befehl installiert:

```
1 apt-get install mysql-server
```

Bei einer Installation von EDIV durch das Quellcode-Paket muss nun noch das EDIV-Skript

```
1 ediv_db_manage.pl create
```

ausgeführt werden, welches dafür sorgt, dass eine spezielle Datenbank-Struktur für EDIV angelegt wird. Zuvor muss jedoch das zum Zugriff auf die Datenbank nötige Passwort an der dafür vorgesehenen Stelle in die Datei */etc/ediv/cluster.conf* eingetragen werden, da das Skript sonst fehlschlägt. Es handelt sich dabei um die sogenannte Cluster-Konfigurationsdatei, die nun im Folgenden näher erläutert wird.

4.1.3 Cluster-Konfigurationsdatei

Die Cluster-Konfigurationsdatei ist ein wichtiger Bestandteil von EDIV. In ihr werden wichtige Informationen über die verwendeten Cluster-Rechner in einer speziellen Syntax eingetragen, sodass diese später beim Starten eines Szenarios von EDIV ausgewertet werden können. Der Standardpfad für die Cluster-Konfigurationsdatei ist */etc/ediv/cluster.conf*, es kann jedoch auch an einem anderen Ort eine eigene Datei erzeugt werden, solange die Syntax eingehalten und der entsprechende Pfad zu der Datei beim Starten eines Szenarios mit angegeben wird. Abbildung 4.2 auf der nächsten Seite zeigt ein Beispiel einer solchen Datei.
Die Datei ist in mehrere Sektionen unterteilt, die durch eckige Klammern gekennzeichnet sind. Zeilen, die beim Starten eines Szenarios von EDIV nicht ausgewertet werden sollen, können mit # auskommentiert werden. Folgende Sektionen sind Bestandteil der Cluster-Konfigurationsdatei:

[db] Diese Sektion behandelt die Datenbank. Hier wird unter anderem die Art der SQL-Datenbank festgelegt, sowie verschiedene Daten, die zum Zugriff auf die Datenbank erforderlich sind, wie beispielsweise der Benutzername und das Passwort.

[vlan] Hier wird festgelegt, in welchem Zahlenbereich die verwendeten VLANs liegen werden und somit auch, wie viel VLANs maximal verwendet werden können. Standardmäßig sind hier 100 VLANs im Bereich von 100-199 vorgesehen, dies kann bei Bedarf jedoch beliebig verändert werden.

```
# Informationen zum Zugriff auf die Datenbank
[db]
type = mysql
name = ediv
host = localhost
port = 3306
user = root
pass = xxxx


# Verfügbare VLAN-Nummern
[vlan]
first = 100
last = 199


# Liste verwendeter Cluster-Hosts
[cluster]
host = ediv1
host = ediv2
host = ediv3
default_segmentation = RoundRobin
#default_segmentation = WeightedRoundRobin
mgmt_network = 192.168.0.0
mgmt_network_mask = 16


# Host-Eigenschaften
[ediv1]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1

[ediv2]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1

[ediv3]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1
```

Abbildung 4.2: Beispiel einer Cluster-Konfigurationsdatei

[cluster] In diesem Abschnitt werden die Hostrechner angegeben, auf denen später die Szenarios laufen sollen. Ob die Hosts durch ihren Namen oder ihre IP-Adresse beschrieben werden, ist dem Nutzer selbst überlassen. Wird eine frei gewählte Bezeichnung für einen Host verwendet, so muss diese jedoch zuvor in der Datei */etc/hosts* der Adresse des Hosts zugeordnet werden.
Auch der Algorithmus, mit dem die virtuellen Maschinen des zu startenden Szenarios auf die Hosts verteilt werden, wird in diesem Abschnitt festgelegt. Damit EDIV mit den virtuellen Maschinen auch kommunizieren kann, muss noch ein sogenanntes Management-Network angelegt werden. Hier wird standardmäßig das Netz 10.250.0.0/16 verwendet, aber auch dies kann bei Bedarf angepasst werden.

[Hostname] Diese Sektion muss für jeden in der Sektion **[cluster]** aufgelisteten Host erstellt werden, indem *Hostname* jeweils durch den Namen bzw. die IP-Adresse des Hosts ersetzt wird.
Für jeden Host werden hier gewisse Eigenschaften spezifiziert, wie die Größe des verwendeten Arbeitsspeichers oder die prozentuale Geschwindigkeit der CPU (verglichen mit dem schnellsten Host des Clusters). Zudem muss das zu verwendende Netzwerkinterface und die maximale Anzahl an virtuellen Maschinen pro Host angegeben werden, wobei 0 für eine unbegrenzte Anzahl steht.

4.2 Tutorial

Der Umgang mit EDIV ähnelt sehr stark dem mit VNUML, was nicht weiter verwunderlich ist, da EDIV eine Weiterentwicklung von VNUML darstellt. Wurde EDIV ordnungsgemäß auf dem Kontrollrechner installiert und die Cluster-Konfigurationsdatei entsprechend angepasst, so reicht fortan das Skript *ediv_ctl.pl* aus, um, abhängig von den gewählten Parametern und Argumenten, alle Funktionalitäten von EDIV zu steuern. Da unter anderem für die Konfiguration der Netzwerkschnittstellen Root-Rechte benötigt werden, ist es erforderlich sich für die Nutzung von EDIV als *root* anzumelden.

Starten eines Szenarios

Zum Starten eines Szenarios mittels EDIV ist, wie schon bei VNUML, der Parameter *-t* zuständig, gefolgt von mehreren teils optionalen Argumenten.

```
1 ediv_ctl.pl -t -s Szenariodatei.xml
```

Die Angabe der zu startenden Szenariodatei ist dabei zwingend erforderlich und folgt hinter dem Parameter *-s*. Alle weiteren Argumente sind optional und werden, falls nicht angegeben, durch ihre festgelegten Default-Werte ersetzt. Die möglichen optionalen Argumente mit ihrem jeweiligen Parameter sind:

-a Segmentierungalgorithmus Durch Angabe des Segmentierungsalgorithmus wird festgelegt, wie die virtuellen Maschinen auf die vorhandenen Cluster-Hosts verteilt werden. Standardmäßig stehen bei EDIV zwei Segmentierungsalgorithmen zur Verfügung, wobei bei Bedarf und entsprechenden Kenntnissen zusätzlich auch eigene Algorithmen entworfen und verwendet werden können.
Der Segmentierungsalgorithmus *RoundRobin* sorgt dafür, dass die in der Szenariodatei definierten virtuellen Maschinen der Reihe nach auf die Cluster-Hosts verteilt werden, um so für eine gleichmäßige Lastverteilung zu sorgen. Die Reihenfolge, in der den Hosts eine virtuelle Maschine zugeteilt wird, wäre somit bei 3 verfügbaren Hosts: *Host1, Host2, Host3, Host1, Host2...*
Der zweite Algorithmus ist *WeightedRoundRobin* und stellt, wie sich anhand des Namens schon erahnen lässt, eine Weiterentwicklung von *RoundRobin* dar. Hierbei wird zusätzlich die relative CPU-Auslastung zwischen den Cluster-Hosts berücksichtigt, sodass Hosts mit geringer Auslastung automatisch mehr virtuelle Maschinen zugeteilt bekommen.
Wird beim Start des Szenarios kein Segmentierungsalgorithmus als Argument angegeben, so wird der in der Cluster-Konfigurationsdatei angegebene Algorithmus verwendet. Standardmäßig ist dies *RoundRobin*.

-r Restriktionsdatei Durch Angabe einer Restriktionsdatei als optionales Argument ist es möglich, die Verteilung der virtuellen Maschinen auf die Cluster-Hosts explizit festzulegen. Die verschiedenen Restriktionsmöglichkeiten, sowie die Syntax dieser in XML geschriebenen Datei, werden in Kapitel 4.3 ausführlich beschrieben.

-c Cluster-Konfigurationsdatei Für den Fall, dass mehr als eine Cluster-Konfigurationsdatei existiert, kann hier angegeben werden, welche der Dateien zum Starten des Szenarios verwendet werden soll. Wird nichts angegeben, so wird automatisch die Datei *cluster.conf* aus dem EDIV-Ordner verwendet.

Beim Starten eines Szenarios mit EDIV wird dieses zunächst in mehrere kleinere Szenarios zerlegt. Wie die einzelnen Teilszenarios aussehen, hängt davon ab, welcher Segmentierungsalgorithmus beim Start gewählt wurde und ob zusätzliche Zuweisungsregeln in Form einer Restriktionsdatei existieren. Im Detail sehen die von EDIV dabei ausgeführten Schritte folgendermaßen aus:

1. Zunächst werden vom sogenannten „Segmentierer“ die virtuellen Maschinen des zu startenden Szenarios, unter Beachtung des gewählten Segmentierungsalgorithmus und der Restriktionsdatei, den vorhandenen Cluster-Hosts zugeordnet.

2. Gemäß dieser Zuordnung wird nun für jeden Host ein entsprechendes Teilszenario erzeugt, das die jeweiligen virtuellen Maschinen und Netze enthält. Bei den virtuellen Netzen wird dabei zwischen Netzen unterschieden, die VMs

auf demselben Host miteinander verbinden (Intra-Host Netze) und Netzen, die VMs auf unterschiedlichen Hosts verbinden (Inter-Host Netze). Dies ist wichtig, da den Inter-Host Netzen in den jeweiligen Teilszenarios ein zusätzliches Attribut angehängt wird, das später zur Einrichtung dieser Netze benötigt wird.

3. Als nächstes wird jedem Inter-Host Netz eine eindeutige VLAN-ID zugeordet. Welche VLAN-IDs dabei zur Verfügung stehen, ist in der Cluster-Konfigurationsdatei festgelegt.

4. Basierend auf dieser VLAN-ID werden nun auf jedem Host für jedes VLAN, an dem dieser Host beteiligt ist, zusätzliche virtuelle Interfaces erzeugt. Die Bezeichnung dieser zusätzlichen Interfaces setzt sich dabei aus dem Namen des ursprünglichen Netzwerkinterfaces und der VLAN-ID zusammen. Ist beispielsweise das verwendete Interface *eth1* und die VLAN-ID *101*, so lautet das neue Interface *eth1.101* .

5. Mithilfe des zusätzlichen Attributs, das in den Teilszenarios den Inter-Host Netzen hinzugefügt wurde, werden nun die jeweiligen Interfaces durch Virtual Bridges angebunden.

6. Nachdem all diese vorbereitenden Maßnahmen getroffen wurden, sendet EDIV die erzeugten Teilszenarios an die jeweiligen Hosts, wo diese dann mittels VNUML gestartet werden.

EDIV informiert währenddessen den Benutzer über den aktuellen Status des Startvorgangs. Dabei wird zunächst gezeigt, wie der Segmentierer die virtuellen Maschinen auf die zur Verfügung stehenden Cluster-Hosts verteilt.

```
**** Calling segmentator... ****

Segmentator: Using RoundRobin
Segmentator: Cluster physical machines -> 3
Segmentator: Virtual machine VM1 to physical host Host1
Segmentator: Virtual machine VM2 to physical host Host2
Segmentator: Virtual machine VM3 to physical host Host3
Segmentator: Virtual machine VM4 to physical host Host1
Segmentator: Virtual machine VM5 to physical host Host2
```

Sobald die Teilszenarios mit den jeweiligen virtuellen Maschinen auf die Hosts verteilt wurden, wird mittels VNUML versucht, diese VMs zu starten. Auch dies wird dem Benutzer von EDIV entsprechend angezeigt.

```
**** Checking simulation status ****

Checking VM1 status
   VM1 still booting, waiting...
   VM1 running
Checking VM4 status
   VM4 running
```

```
Checking VM2 status
   VM2 running
Checking VM5 status
   VM5 running
Checking VM3 status
   VM3 running
```

Sofern es in der Cluster-Konfigurationsdatei unter dem Tag *<console>* angegeben wurde, öffnet sich nun für jede virtuelle Maschine ein Konsolenfenster, über das auf die jeweilige VM zugegriffen werden kann. Gerade bei großen Szenarios mit vielen virtuellen Maschinen kann dies jedoch sehr schnell extrem unübersichtlich werden, sodass dort lieber darauf verzichtet werden sollte.
Dies ist kein Problem, da es neben den Konsolenfenster auch andere Möglichkeiten gibt, auf die virtuellen Maschinen zuzugreifen. So besteht entweder die Möglichkeit direkt mittels SSH und Portnummer der entsprechenden VM auf diese zuzugreifen oder aber das von EDIV bereitgestellte Skript *ediv_ console.pl console* zu verwenden. Die Syntax beider Methoden, sowie die Portnummern der virtuellen Maschinen, werden von EDIV am Ende des Startvorgangs bereitgestellt.

```
**** Creating tunnels to access VM ****

  To access VM VM1 at Host1 use local port 64000
  To access VM VM2 at Host2 use local port 64001
  To access VM VM3 at Host3 use local port 64002
  To access VM VM4 at Host1 use local port 64003
  To access VM VM5 at Host2 use local port 64004

  Use command ssh -2 root@localhost -p <port> to access VMs
  Or ediv_console.pl console <simulation_name> <vm_name>
  Where <port> is a port number of the previous list
  The port list can be found running ediv_console.pl info

****** Succesfully finished ******
```

Kommandos während der Simulation

Auch das Ausführen vordefinierter Kommandosequenzen ist bei EDIV wieder möglich und baut auf der in Kapitel 3.2.2 beschriebenen Funktionsweise der VNUML-Kommandosequenzen auf. Lediglich der einzugebende Befehl ist etwas anders, da das Ganze nun von EDIV gesteuert wird. Statt *vnumlparser.pl* muss nun *ediv_ ctl.pl* verwendet werden und statt einem @ wird nun mit einem vorangestellten *-s* das Szenario von dem darin auszuführenden Kommando getrennt.

```
ediv_ctl.pl -x Kommando -s Szenariodatei.xml
```

Wird ein solcher Befehl ausgeführt, leitet EDIV dies an die betroffenen Teilszenarios auf den Cluster-Hosts weiter und ruft dort den zuständigen VNUML-Befehl zum Ausführen des Kommandos auf.

Beenden eines Szenarios

Beendet wird ein verteiltes Szenario, wie bei VNUML, mit dem Parameter -P.

```
1 ediv_ctl.pl -P -s Szenariodatei.xml
```

Dabei wird zuerst überprüft, ob ein Szenario mit diesem Namen überhaupt läuft. Ist dies nicht der Fall, weist EDIV den Benutzer mit einer entsprechenden Fehlermeldung darauf hin.
Anderenfalls schickt EDIV nun an jeden Cluster-Host den Befehl, das jeweilige Teilszenario herunterzufahren. Dazu werden zunächst alle virtuellen Maschinen des Teilszenarios beendet, bevor dann sämtliche, beim Start des Szenarios eingerichtete, virtuelle Netzwerkschnittstellen entfernt und alle zum Szenario gehörenden temporären Dateien gelöscht werden.

4.3 Restriktionsdatei

Die Verteilung der virtuellen Maschinen auf bestimmte Hosts wird bei EDIV standardmäßig durch den gewählten Segmentierungsalgorithmus bestimmt. Dieser stellt jedoch nur ein generelles Verfahren dar, die virtuellen Maschinen zu verteilen. Der Algorithmus nimmt dabei keine Rücksicht auf die jeweilige Topologie des Szenarios oder bestimmte Wünsche des Nutzers, die die Verteilung der VMs betreffen.
Um explizit zu bestimmen, welche virtuelle Maschine auf welchen Host verteilt wird oder welche VMs nicht gemeinsam auf demselben Hosts laufen sollen, gibt es die sogenannten Restriktionsdateien. Dies sind einfache XML-Dateien, die beim Start eines Szenarios als Argument mit angegeben werden können. Die genaue Syntax und die fünf zur Verfügung stehenden Zuweisungsregeln werden nun im Folgenden näher erläutert.

<net_deploy_together>

Die durch dieses Tag beschriebene Zuweisungsregel sorgt dafür, dass alle virtuellen Maschinen eines Szenarios, die an ein bestimmtes Netz angeschlossen sind, demselben Host zugewiesen werden. Dies ist üblicherweise der Host, der als erstes verfügbar ist, sofern keine weiteren Zuweisungsregeln etwas anderes fordern. *Netzname* steht dabei im folgenden Syntaxbeispiel für den Namen des betroffenen Netzes, der in der Szenariodatei innerhalb des <net>-Tags festgelegt wurde.

```
1 <net_deploy_together net="Netzname"/>
```

<net_deploy_at>

Hierbei handelt es sich um eine Erweiterung der zuvor vorgestellten Regel. Neben dem betroffenen Netz kann nun auch der Host bestimmt werden, auf den die virtuellen Maschinen, die an dieses Netz angeschlossen sind, verteilt werden sollen. *Netzname* muss dabei wieder durch den Namen des betroffenen Netzes ersetzt werden und *Hostname* durch den Namen des Hosts, so wie er auch in der Cluster-Konfigurationsdatei angegeben wurde.

```
<net_deploy_at net="Netzname" host="Hostname"/>
```

<vm_deploy_at>

Diese Zuweisungsregel ermöglicht es, einzelne virtuelle Maschinen explizit einem bestimmten Host zuzuweisen. Hierbei steht *VM-Name* für den Namen der virtuellen Maschine, wie er im <vm>-Tag innerhalb der Szenariodatei angegeben wurde und *Hostname* steht wieder für den Namen des Zielhosts.

```
<vm_deploy_at vm="VM-Name" host="Hostname"/>
```

<affinity>

Bei dieser Regel können diejenigen virtuellen Maschinen angegeben werden, die auf jeden Fall auf demselben Host laufen sollen. Diese werden dann dem ersten verfügbaren Host des Clusters zugewiesen, sofern keine weiteren Zuweisungsregel etwas anderes fordern. *VM-Name* steht wieder für die in den <vm>-Tags der Szenariodatei angegebenen Namen der betroffenen VMs.

```
<affinity>
  <vm>VM1-Name</vm>
  <vm>VM2-Name</vm>
  <vm>VM3-Name</vm>
</affinity>
```

<antiaffinity>

Im Gegensatz zur zuvor beschriebenen Affinity-Regel werden bei der Antiaffinity-Regel diejenigen virtuellen Maschinen angegeben, die auf keinen Fall auf demselben Host laufen sollen. Jede der aufgelisteten VMs wird demnach auf einen anderen Host verteilt, der jedoch nicht vom Benutzer bestimmt werden kann.

```
<antiaffinity>
  <vm>VM1-Name</vm>
  <vm>VM2-Name</vm>
  <vm>VM3-Name</vm>
</antiaffinity>
```

Mithilfe dieser fünf Zuweisungsregeln können nun eigene Restriktionsdateien entworfen werden, die auf das zu startende Szenario abgestimmt sind. Konflikte zwischen einzelnen Zuweisungsregeln werden dabei automatisch erkannt und der Benutzer durch eine Fehlermeldung darauf hingewiesen. Abbildung 4.3 zeigt ein Beispiel einer solchen Restriktionsdatei.
Bei diesem Beispiel sorgt die Restriktionsdatei für folgende Verteilung der virtuellen Maschinen: VM1 wird auf Host3, VM2 auf Host1 und VM3 auf Host2 gestartet. VM5 wird ebenfalls auf Host3 gestartet, da dort bereits VM1 läuft und für die beiden VMs die Affinity-Regel gilt. Aufgrund der Antiaffinity-Regel wird nun noch VM4 auf Host2 gestartet, da VM1 bereits dem Host3 und VM2 dem Host1 zugewiesen wurde und alle drei VMs auf unterschiedlichen Hosts laufen sollen, womit nur noch Host2 als möglicher Host übrig blieb.

```
<deployment_restrictions>

<vm_deploy_at vm="VM1" host="Host3"/>
<vm_deploy_at vm="VM2" host="Host1"/>
<vm_deploy_at vm="VM3" host="Host2"/>

<affinity>
  <vm>VM1</vm>
  <vm>VM5</vm>
</affinity>

<antiaffinity>
  <vm>VM1</vm>
  <vm>VM2</vm>
  <vm>VM4</vm>
</antiaffinity>

</deployment_restrictions>
```

Abbildung 4.3: Beispiel einer Restriktionsdatei

4.4 EDIV-Skripte

EDIV stellt dem Benutzer einige nützliche Skripte zur Verfügung, die verschiedene Informationen über die Szenarios liefern, sobald diese auf die Hostsysteme verteilt wurden. Die Skripte können dabei einfach durch Angabe des Skriptnamens und der entsprechenden Argumente ausgeführt werden. Folgende Skripte werden von EDIV zur Verfügung gestellt:

ediv_segmentation_info.pl

Durch dieses Skript erfährt der Benutzer, wie EDIV das Szenario, abhängig von den gewählten Argumenten, auf die Hostsysteme verteilen würde. Als Argument

muss dazu das zu simulierende Szenario angegeben werden. Für den Fall, dass der zu verwendende Segmentierungsalgorithmus ein anderer sein soll, als der in der Cluster-Konfigurationsdatei angegebene, muss dieser ebenfalls als Argument angegeben werden. Wurde explizit ein Segmentierungsalgorithmus angegeben, so muss danach auch eine Restriktionsdatei (siehe Kapitel 4.3) angegeben werden, die genauere Regeln für die Verteilung des Szenarios enthält.

```
1 ediv_segmentation_info.pl szenario.xml Roundrobin restriktion.xml
```

ediv_monitor.pl

Dieses Skript zeigt den aktuellen Status der verwendeten Hostsysteme.
Soll der Status nicht nur einmalig ausgegeben werden, sondern sich nach einem bestimmten Zeitintervall aktualisieren, so muss als Argument die Aktualisierungsrate in Sekunden angegeben werden. Für eine einmalige Ausgabe muss hier eine 0 stehen. Sollen lediglich Informationen zu einem bestimmten Hostsystem und nicht zu allen ausgegeben werden, so ist der Name dieses Hosts explizit als Argument anzugeben.

```
1 ediv_monitor.pl 0 host
```

ediv_query_status.pl

Mithilfe dieses Skripts können Informationen über die derzeit laufenden Szenarios abgerufen werden.
Um zu erfahren, auf welchen Hostsystemen welche Szenarios laufen, kann das Skript einfach ohne Argument ausgeführt werden. Wird als Argument hingegen das jeweilige Szenario angegeben, über das Informationen ausgegeben werden sollen, so werden alle virtuellen Maschinen des Szenarios und die Hostsysteme, auf denen sie laufen, aufgelistet.

```
1 ediv_query_status.pl szenario.xml
```

ediv_query_vlan.pl

Auch diesem Skript kann als optionales Argument ein Szenarioname mitgegeben werden. Wird das Skript ohne Argument ausgeführt, listet es die erstellten virtuellen Netzwerke zwischen den Hostsystemen auf und die Szenarios, die diese VLANs

benutzen. Wird als Argument jedoch ein laufendes Szenario angegeben, so zeigt das Skript, welche virtuellen Netzwerke das Szenario verwendet und welche Hostsysteme diese VLANs benutzen.

```
1 ediv_query_vlan.pl szenario.xml
```

ediv_locate_vm.pl

Um zu erfahren, wo sich bestimmte virtuelle Maschinen befinden, bietet sich dieses Skript an.
Wird als Argument der Name einer virtuellen Maschine angegeben, so zeigt das Skript auf welchem Host sich diese befindet. Ohne entsprechende Angabe eines Arguments werden alle laufenden VMs mitsamt dem Hostsystem, auf dem sie befinden, aufgelistet.

```
1 ediv_locate_vm.pl VM
```

ediv_console.pl info

Dieses Skript hilft dabei, den SSH Port herauszufinden, über den auf die jeweilige virtuelle Maschine zugegriffen werden kann.
Ohne Argumente ausgeführt, zeigt das Skript sämtliche virtuelle Maschinen von allen laufenden Szenarios mit ihren jeweiligen SSH Ports. Sind nur die virtuellen Maschinen eines bestimmen Szenarios von Interesse, so muss dieses Szenario als Argument mit angegeben werden. Folgt nach dem Szenarionamen als zweites Argument auch noch der Name einer virtuellen Maschinen, so wird einzig der Port zu dieser VM ausgegeben.

```
1 ediv_console.pl info szenario.xml VM
```

ediv_console.pl console

Hierbei handelt es sich streng genommen um dasselbe Skript wie zuvor, allerdings in einem anderen Modus.
Dieser Modus dient dazu, nach Angabe des Szenarios und der virtuellen Maschine, durch Errichten eines SSH Tunnels auf die angegebene VM zuzugreifen.

```
1 ediv_console.pl console szenario.xml VM
```

ediv_db_manage.pl

Um wichtige Informationen über laufende Szenarios speichern zu können, greift EDIV auf eine MySQL Datenbank zurück. Dort gespeicherte Informationen sind beispielsweise die verwendeten VLANs, die Namen der virtuellen Maschinen oder die jeweiligen Ports, um auf diese zuzugreifen. Durch die Argumente *create* bzw. *destroy* kann die Struktur dieser Datenbank erzeugt bzw. zerstört werden.
Wurde EDIV über das .deb-Packet installiert, so sollte dieses Skript nicht verwendet werden, da bei der Installation bereits alle nötigen Schritte zum Erzeugen einer geeigneten Datenbankstruktur für EDIV durchgeführt wurden.

```
1 ediv_db_manage.pl create
```

ediv_db_reset.pl

Für den Fall, dass beim Starten oder Herunterfahren eines Szenarios Fehler auftreten, kann dieses Skript eingesetzt werden, um alle in der Datenbank enthaltenen Informationen zurückzusetzen, wobei die Struktur der Datenbank erhalten bleibt. Die Angabe eines Szenarios als Argument führt dazu, dass nur die zu diesem Szenario gehörenden Informationen gelöscht werden.

```
1 ediv_db_reset.pl szenario.xml
```

ediv_cluster_cleanup.pl

Dieses Skript benötigt keine zusätzlichen Argumente. Es ist ein sehr radikales Skript und sollte nur verwendet werden, wenn es wirklich notwendig ist, da es den gesamten Cluster in den Initialzustand zurücksetzt.

Dazu werden folgende Schritte ausgeführt:

- Sämtliche Informationen aus der Datenbank werden gelöscht.
- Alle zu VNUML bzw. EDIV gehörenden Prozesse werden beendet.
- Der VNUML-Ordner mit den Statusinformationen wird gelöscht. (*/root/.vnuml/*)
- Die SSH Tunnel zu den virtuellen Maschinen werden entfernt.
- Alle in der Cluster-Konfigurationsdatei angegebenen VLAN-Schnittstellen werden entfernt.

```
1 ediv_cluster_cleanup.pl
```

Kapitel 5

Praktische Tests

Nachdem in den vorangegangenen Kapiteln verschiedene Funktionen von VNUML und EDIV erläutert wurden, soll nun in diesem Kapitel der praktische Einsatz dieser Tools vorgeführt und beschrieben werden. Um die vorgestellten Möglichkeiten von EDIV hinsichtlich der verteilten Simulation von Netzwerken testen und untersuchen zu können, benötigt es, wie bereits erwähnt, mehrere Hostsysteme inklusive eines Kontrollrechners.
Für die im Folgenden beschriebenen Tests übernahm ein Laptop die Aufgabe des Kontrollrechners. Als Hostsysteme dienten vier durch QEMU[1] simulierte virtuelle Hosts, die auf einem Server gestartet wurden und für EDIV nicht von „echten“ Hostsystemen zu unterscheiden sind. Um die vier virtuellen Hosts miteinander zu vernetzen, musste zudem ein VDE-Switch[2] installiert und entsprechend konfiguriert werden. Die Konfiguration bestand darin, alle später verwendeten VLANs zuvor manuell im VDE-Switch einzurichten, damit dieser später die Netzwerkpakete zwischen den virtuellen Maschinen den richtigen VLANs zuordnen und korrekt weiterleiten kann. Weiterführende Informationen zu QEMU und VDE finden sich in [WR10] und [VDE].
Würde der Host-Cluster statt virtuellen Host komplett aus physischen Hosts bestehen, dann entfiele die etwas umständliche Installation und Konfiguration des VDE-Switches, da für die Vernetzung der Hosts nun ein physischer VLAN-fähiger Switch eingesetzt werden müsste, der diese Konfiguration nicht erfordert. Diese Variante wurde im Zuge dieser Arbeit ebenfalls getestet und funktionierte ohne Probleme.
Für die nun folgende Demonstration der Möglichkeiten von EDIV wurden im Laufe

[1] QEMU steht für „Quick Emulator“ und ist ein Open-Source Prozessoremulator, der die komplette Hardware eines Rechners emuliert und es ermöglicht verschiedene Betriebssysteme innerhalb eines anderen Betriebssystems zu starten. QEMU ist dabei auf nahezu allen Betriebssysteme lauffähig und zeichnet sich durch eine hohe Ausführungsgeschwindigkeit aus.

[2] Ein VDE-Switch (Virtual Distributed Ethernet) ist das virtuelle Äquivalent zu einem physischen Switch und dient dazu, virtuelle Maschinen miteinander zu vernetzen.

der Arbeit mehrere Szenarios entwickelt, von denen ein Szenario mit 10 virtuellen Maschinen ausgewählt wurde, um hier als Beispielszenario zu dienen. Es sind jedoch problemlos auch Szenarios mit deutlich mehr virtuellen Maschinen möglich, sofern die verfügbaren Ressourcen der Hostsysteme dies erlauben.

5.1 10-Router-Szenario

Anhand eines Test-Szenarios mit 10 virtuellen Maschinen, soll nun die praktische Anwendung von EDIV demonstriert werden. Neben dem Start des Szenarios mit entsprechender Konfiguration der Restriktionsdatei und Auswahl des Segmentierungsalgorithmus, soll im Folgenden auch die Repräsentation innerhalb der SQL-Datenbank, sowie die Konnektivität der einzelnen virtuellen Maschinen untereinander näher untersucht werden. Die Topologie des verwendeten Szenarios ist in Abbildung 5.1 zu sehen. Die komplette XML-Repräsentation des Szenarios kann dem Anhang entnommen werden.

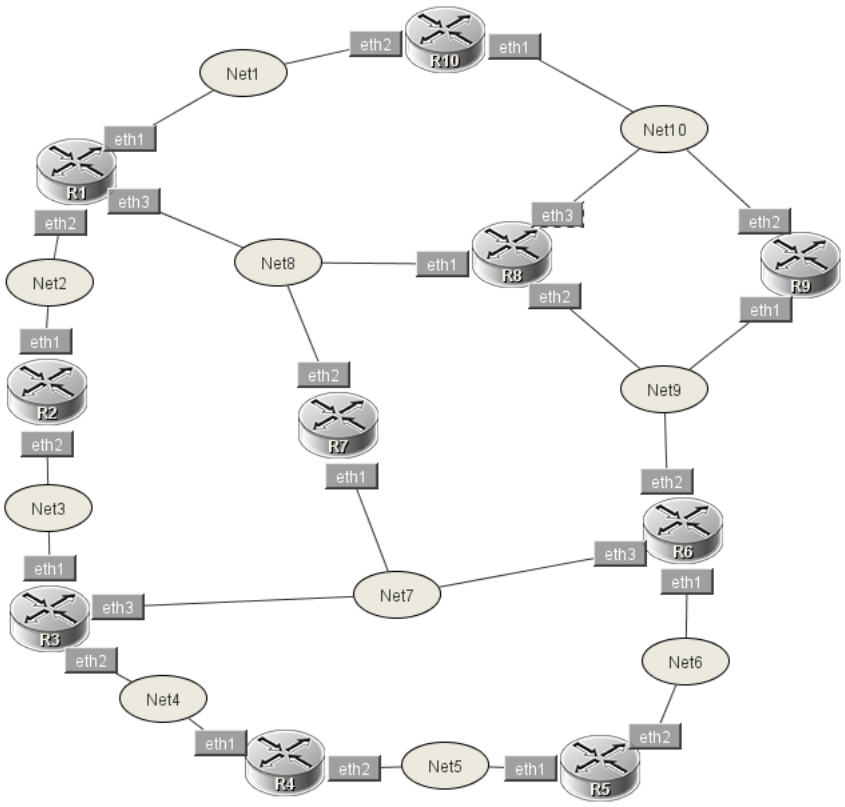

Abbildung 5.1: Topologie des 10-Router-Szenarios

5.1.1 Verteilte Simulation des Szenarios

Bevor die Simulation gestartet werden kann, muss zunächst der verwendete Host-Cluster eingerichtet werden. Was dazu auf den jeweiligen Hosts zu installieren ist, wurde bereits in Kapitel 4.1 beschrieben.
Der in dieser Arbeit verwendete Cluster besteht aus vier (virtuellen) Rechnern: *ediv1...ediv4*. Um EDIV die Eigenschaften dieser Rechner bekannt zu machen, müssen diese, wie in Kapitel 4.1.3 beschrieben, in die Cluster-Konfigurationsdatei eingetragen werden. Sollen hierbei für eine bessere Übersicht nicht die IP-Adressen der Hosts, sondern frei gewählte Bezeichnungen benutzt werden, so ist dies ebenfalls möglich. Dann sind jedoch zuvor die IP-Adressen der Hosts diesen Bezeichnungen zuzuordnen. Dies geschieht in der Datei */etc/hosts*. Es müssen dazu lediglich die IP-Adresse des Hosts und die gewählte Bezeichnung, durch ein Leerzeichen voneinander getrennt, eingefügt werden. Die Cluster-Konfigurationsdatei für den hier verwendeten Host-Cluster ist in Abbildung 5.2 auf der nächsten Seite zu sehen.
Nachdem die Hosts eingerichtet wurden, kann mit der Simulation des Szenarios begonnen werden. Im Folgenden sind mehrere Varianten zur Verteilung des Szenarios gezeigt, die sich jeweils durch die Angabe bestimmter Argumente beim Start des Szenarios unterscheiden.

Variante 1: Ohne optionale Argumente

Bei dieser Variante wird das Szenario lediglich mit der Angabe der Szenariodatei als notwendigem Argument gestartet. Optionale Argumente wie der Segmentierungsalgorithmus, die Restriktionsdatei oder eine andere als die übliche Cluster-Konfigurationsdatei werden nicht verwendet, wodurch automatisch die im ediv-Ordner liegende *cluster.conf* und der darin angegebene Default-Segmentierungsalgorithmus verwendet werden. Der Befehl zum Start des Szenarios lautet somit:

```
1 ediv_ctl.pl -t -s szenario10.xml
```

Wird dieser Befehl ausgeführt, beginnt EDIV damit ein Netzwerk zwischen den vier in der Cluster-Konfigurationsdatei angegebenen Hosts aufzubauen und anschließend die 10 virtuellen Maschinen des Szenarios auf diese Hosts zu verteilen. Da RoundRobin als Default-Segmentierungsalgorithmus in der Datei *cluster.conf* angegeben wurde, erfolgt die Zuteilung der virtuellen Maschinen zu den jeweiligen Hosts gleichmäßig. Die virtuellen Maschinen werden dabei der Reihe nach den einzelnen Rechnern zugeteilt, bis schließlich alle vergeben wurden.
Entsprechend der Zuteilung zu den Hosts teilt EDIV das ursprüngliche Szenario in vier kleinere Szenarios auf. Diese Teilszenarios enthalten die, dem jeweiligen Host zugeordneten, virtuellen Maschinen und werden von EDIV an die entsprechenden Hosts gesendet, um dort dann mittels VNUML gestartet zu werden. Dies ist auch der Grund, warum auf den Cluster-Hosts zuvor VNUML installiert werden musste. Damit der Benutzer direkt sieht, welche virtuellen Maschinen auf welche Hosts verteilt werden, gibt EDIV dies entsprechend aus:

```
[db]
type = mysql
name = ediv
host = localhost
port = 3306
user = root
pass = xxxx

[vlan]
first = 100
last = 199

[cluster]
host = ediv1
host = ediv2
host = ediv3
host = ediv4
default_segmentation = RoundRobin
mgmt_network = 192.168.0.0
mgmt_network_mask = 16


[ediv1]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1

[ediv2]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1

[ediv3]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1

[ediv4]
mem = 2048
cpu = 100
max_vhost = 0
ifname = eth1
```

Abbildung 5.2: Cluster-Konfigurationsdatei für die Rechner *ediv1...ediv4*

```
**** Calling segmentator... ****

Segmentator: Using RoundRobin
Segmentator: Cluster physical machines -> 4
Segmentator: Virtual machine R1 to physical host ediv1
Segmentator: Virtual machine R2 to physical host ediv2
Segmentator: Virtual machine R3 to physical host ediv3
Segmentator: Virtual machine R4 to physical host ediv4
Segmentator: Virtual machine R5 to physical host ediv1
Segmentator: Virtual machine R6 to physical host ediv2
Segmentator: Virtual machine R7 to physical host ediv3
Segmentator: Virtual machine R8 to physical host ediv4
Segmentator: Virtual machine R9 to physical host ediv1
Segmentator: Virtual machine R10 to physical host ediv2
```

Während dem Start der virtuellen Maschinen teilt EDIV dem Benutzer fortlaufend mit, welche VMs momentan gestartet werden und welche bereits laufen.
Dabei sind gerade bei großen Szenarios oft deutliche Geschwindigkeitsvorteile einer verteilten Simulation mittels EDIV gegenüber einer normalen Simulation durch VNUML zu erkennen. Dies liegt vor allem daran, dass bei der verteilten Simulation die jeweiligen Cluster-Hosts parallel die ihnen zugeteilten VMs starten können, während bei einer Simulation auf nur einem Host immer auf jede einzelne VM gewartet werden muss, bevor die nächste gestartet werden kann.
Sobald alle virtuellen Maschinen des Szenarios hochgefahren wurden, stellt EDIV entsprechende Informationen bereit, wie darauf zugegriffen werden kann. Dazu wird jeder virtuellen Maschine ein Port zugewiesen, über den mittels SSH auf die VM zugegriffen werden kann. Als zusätzliche Möglichkeit steht auch ein Skript zur Verfügung, das unter Angabe des Szenarionamens und der gewünschten virtuellen Maschine ebenfalls Zugriff auf die VMs ermöglicht.

```
**** Creating tunnels to access VM ****

  To access VM R1 at ediv1 use local port 64000
  To access VM R2 at ediv2 use local port 64001
  To access VM R3 at ediv3 use local port 64002
  To access VM R4 at ediv4 use local port 64003
  To access VM R5 at ediv1 use local port 64004
  To access VM R6 at ediv2 use local port 64005
  To access VM R7 at ediv3 use local port 64006
  To access VM R8 at ediv4 use local port 64007
  To access VM R9 at ediv1 use local port 64008
  To access VM R10 at ediv2 use local port 64009

  Use command ssh -2 root@localhost -p <port> to access VMs
  Or ediv_console.pl console <simulation_name> <vm_name>
  Where <port> is a port number of the previous list
  The port list can be found running ediv_console.pl info

****** Succesfully finished ******
```

Möchte der Benutzer jetzt beispielsweise vom Kontrollrechner aus auf die virtuelle Maschine R3 zugreifen, so stehen ihm dazu zwei Möglichkeiten zur Verfügung.

Möglichkeit 1: Zugriff mittels SSH und entsprechender Portnummer der VM.

```
ssh -2 root@localhost -p 64002
```

Möglichkeit 2: Zugriff durch das von EDIV bereitgestellte Skript.

```
ediv_console.pl console szenario10 R3
```

Variante 2: Segmentierungsalgorithmus Weighted-RoundRobin

Anders als bei der zuvor beschriebenen Variante, kann beim Start eines Szenarios der Segmentierungsalgorithmus auch explizit angegeben werden. Um die CPU-Last der einzelnen Cluster-Hosts bei der Verteilung der virtuellen Maschinen zu berücksichtigen, wird im Folgenden der Segmentierungsalgorithmus *Weighted-RoundRobin* anstelle des als Default-Algorithmus angegebenen *RoundRobin* verwendet. Alternativ könnte auch ein selbstentwickelter Segmentierungsalgorithmus verwendet werden, standardmäßig sind bei EDIV jedoch nur die Algorithmen *RoundRobin* und *Weighted-RoundRobin* enthalten. Der Befehl zum Start des verwendeten 10-Router-Szenarios lautet somit:

```
ediv_ctl.pl -t -s szenario10.xml -a WeightedRoundRobin
```

EDIV versucht nun beim Start des Szenarios die virtuellen Maschinen so zu verteilen, dass alle zur Verfügung stehenden Cluster-Hosts optimal ausgelastet werden. Dazu wird zunächst von jedem Host die momentane CPU-Auslastung ermittelt und auch entsprechend angezeigt. Abhängig von dieser CPU-Auslastung bestimmt EDIV, wie viel Prozent der virtuellen Maschinen den jeweiligen Hosts zugeteilt werden. Je höher die Auslastung, desto geringer die Prozentzahl der zugeteilten VMs. Mithilfe dieser Prozentzahlen wird nun für jeden Host die effektive Anzahl an virtuellen Maschinen berechnet. EDIV zeigt dem Benutzer all diese Schritte detailliert an:

```
**** Parsing scenario ****

WeightedRoundRobin segmentation mode selected

**** Calling segmentator... ****

Segmentator: Dynamic CPU load of ediv1 is 0.00
Segmentator: Dynamic CPU load of ediv2 is 0.00
Segmentator: Dynamic CPU load of ediv3 is 0.20
Segmentator: Dynamic CPU load of ediv4 is 0.00
Segmentator: Assigned 29.6296296296296\% to ediv1
Segmentator: Assigned 29.6296296296296\% to ediv2
Segmentator: Assigned 11.1111111111111\% to ediv3
Segmentator: Assigned 29.6296296296296\% to ediv4
Segmentator: 3 VMs assigned to ediv1
Segmentator: 3 VMs assigned to ediv2
```

```
17 Segmentator: 1 VMs assigned to ediv3
18 Segmentator: 3 VMs assigned to ediv4
19 Segmentator: Virtual machine R1 goes to physical host ediv1
20 Segmentator: Virtual machine R2 goes to physical host ediv1
21 Segmentator: Virtual machine R3 goes to physical host ediv1
22 Segmentator: Virtual machine R4 goes to physical host ediv2
23 Segmentator: Virtual machine R5 goes to physical host ediv2
24 Segmentator: Virtual machine R6 goes to physical host ediv2
25 Segmentator: Virtual machine R7 goes to physical host ediv3
26 Segmentator: Virtual machine R8 goes to physical host ediv4
27 Segmentator: Virtual machine R9 goes to physical host ediv4
28 Segmentator: Virtual machine R10 goes to physical host ediv4
```

Zum Zeitpunkt des Szenariostarts besaß nur *ediv3* eine etwas erhöhte CPU-Auslastung, weshalb dieser Host lediglich eine virtuelle Maschine zugewiesen bekam und die übrigen VMs gleichermaßen auf die verbliebenen Hosts aufgeteilt wurden. Sollte der Weighted-RoundRobin-Algorithmus hingegen feststellen, dass alle verwendeten Cluster-Hosts über keine oder nur eine sehr geringe Auslastung verfügen, so stoppt dieser automatisch und es wird stattdessen der normale RoundRobin-Algorithmus verwendet.

Variante 3: Restriktionsdatei + Roundrobin

Zu guter Letzt soll noch eine Variante vorgestellt werden, bei der der Benutzer direkten Einfluss auf die Verteilung der virtuellen Maschinen ausüben kann, indem er explizit bestimmt, welche VMs welchen Hosts zugeordnet werden. Dazu muss zunächst eine sogenannte Restriktionsdatei angelegt werden, in der die Regeln festgelegt werden, die EDIV bei der Verteilung der virtuellen Maschinen zu beachten hat. Die Syntax und Semantik einer solchen Restriktionsdatei wurden bereits in Kapitel 4.3 ausführlich beschrieben.
Um die Erstellung einer Restriktionsdatei und deren Auswirkung beim Start des Szenarios an einem Beispiel zu zeigen, seien folgenden Bedingungen für die Verteilung der virtuellen Maschinen des 10-Router-Szenarios gegeben:

1. Alle virtuellen Maschinen, die an *Net3* angeschlossen sind, sollen dem Host *ediv1* zugeteilt werden
2. Die virtuelle Maschine *R1* soll dem Host *ediv2* zugeteilt werden
3. Die virtuelle Maschine *R4* soll dem Host *ediv3* zugeteilt werden
4. Die virtuellen Maschinen *R4*, *R5* und *R6* sollen demselben Host zugeteilt werden
5. Die virtuellen Maschinen *R1*, *R7* und *R8* sollen unterschiedlichen Hosts zugeteilt werden

Diese Bedingungen werden nun, unter Verwendung der entsprechenden syntaktischen Konstrukte, als Regeln in eine XML-Datei geschrieben. Damit ist die Erzeugung der Restriktionsdatei auch bereits abgeschlossen. Die vollständige Restriktionsdatei für unser 10-Router-Szenario sieht somit wie folgt aus:

```
<deployment_restrictions>

  <!-- 1. Bedingung -->
  <net_deploy_at net="Net3" host="ediv1" />

  <!-- 2. Bedingung -->
  <vm_deploy_at vm="R1" host="ediv2" />

  <!-- 3. Bedingung -->
  <vm_deploy_at vm="R4" host="ediv3" />

  <!-- 4. Bedingung -->
  <affinity>
    <vm>R4</vm>
    <vm>R5</vm>
    <vm>R6</vm>
  </affinity>

  <!-- 5. Bedingung -->
  <antiaffinity>
    <vm>R1</vm>
    <vm>R7</vm>
    <vm>R8</vm>
  </antiaffinity>

</deployment_restrictions>
```

Abbildung 5.1: Restriktionsdatei für das 10-Router-Szenario

Um diese Restriktionsdatei nun auf unser Szenario mit den 10 virtuellen Maschinen anzuwenden, muss sie einfach als Argument beim Start des Szenarios mit angegeben werden. EDIV wendet die Regeln dann auf das zu startende Szenario an und sorgt für eine entsprechende Verteilung der virtuellen Maschinen. Der dazu nötige Befehl lautet somit

```
ediv_ctl.pl -t -s szenario10.xml -a RoundRobin -r restriktion10.xml
```

wobei *restriktion10.xml* die zuvor entworfene Restriktionsdatei ist.
Der Segmentierungsalgorithmus wird dabei nur auf die virtuellen Maschinen angewendet, die noch keinem Host durch die Restriktionsdatei explizit zugeordnet wurden. Daraus ergibt sich die im Folgenden dargestellte Verteilung der virtuellen Maschinen.

```
**** Parsing scenario ****

RoundRobin segmentation mode selected

**** Calling static processor... ****

```

```
Static assignment: Virtual machine R1 to physical host ediv2
Static assignment: Virtual machine R3 to physical host ediv1
Static assignment: Virtual machine R7 to physical host ediv4
Static assignment: Virtual machine R6 to physical host ediv3
Static assignment: Virtual machine R4 to physical host ediv3
Static assignment: Virtual machine R5 to physical host ediv3
Static assignment: Virtual machine R2 to physical host ediv1
Static assignment: Virtual machine R8 to physical host ediv3

**** Calling segmentator... ****

Segmentator: Cluster physical machines -> 4
Segmentator: Virtual machine R9 goes to physical host ediv2
Segmentator: Virtual machine R10 goes to physical host ediv4
```

An dieser Ausgabe lässt sich erkennen, wie EDIV zunächst die Regeln aus der Restriktionsdatei abarbeitet. Da keine der vorhandenen Regeln einen Widerspruch zu einer anderen Regel dargestellt hat, war es für EDIV ohne Probleme möglich, die virtuellen Maschinen gemäß der vorhandenen Regeln zu verteilen. Die zwei verbliebenen, nicht durch eine Regel betroffenen VMs, wurden im Anschluss mithilfe des Segmentierungsalgorithmus RoundRobin auf diejenigen Hosts verteilt, denen bislang die wenigsten virtuellen Maschinen zugeordnet wurden. Abbildung 5.4 auf der folgenden Seite veranschaulicht noch einmal die Verteilung der virtuellen Maschinen auf die vier Hosts.
Sollte EDIV jedoch beim Anwenden der Regeln aus der Restriktionsdatei feststellen, dass ein Widerspruch zwischen zwei Regeln besteht, so wird der Start des Szenarios abgebrochen und dem Benutzer eine entsprechende Fehlermeldung angezeigt, die das Problem näher beschreibt.

5.1.2 Repräsentation des Szenarios in der Datenbank

Bei der Installation von EDIV in Kapitel 4.1.2 wurde bereits erwähnt, dass EDIV zum Speichern von Informationen über die Szenarios eine MySQL-Datenbank benötigt. Wurde diese Datenbank bei der Installation ordnungsgemäß eingerichtet, so kann nun mithilfe der üblichen SQL-Befehle auf die darin gespeicherten Daten zugegriffen werden.
Um dies zu demonstrieren, wird im Folgenden davon ausgegangen, dass aktuell nur ein einziges Szenario aktiv ist. Dieses Szenario sei unser im vorigen Kapitel vorgestelltes 10-Router-Szenario mit einer Verteilung der virtuellen Maschinen nach Variante 3. Um nun nähere Informationen über dieses Szenario zu erhalten, kann vom Kontrollrechner aus mit dem Befehl

```
mysql -u root -p ediv
```

auf die entsprechende Datenbank zugegriffen werden. Dabei steht *root* für den Benutzernamen und *ediv* für den Namen der Datenbank. Der Parameter *-p* sorgt für die Abfrage des Benutzerpassworts nach Ausführung des Befehls. Sowohl der

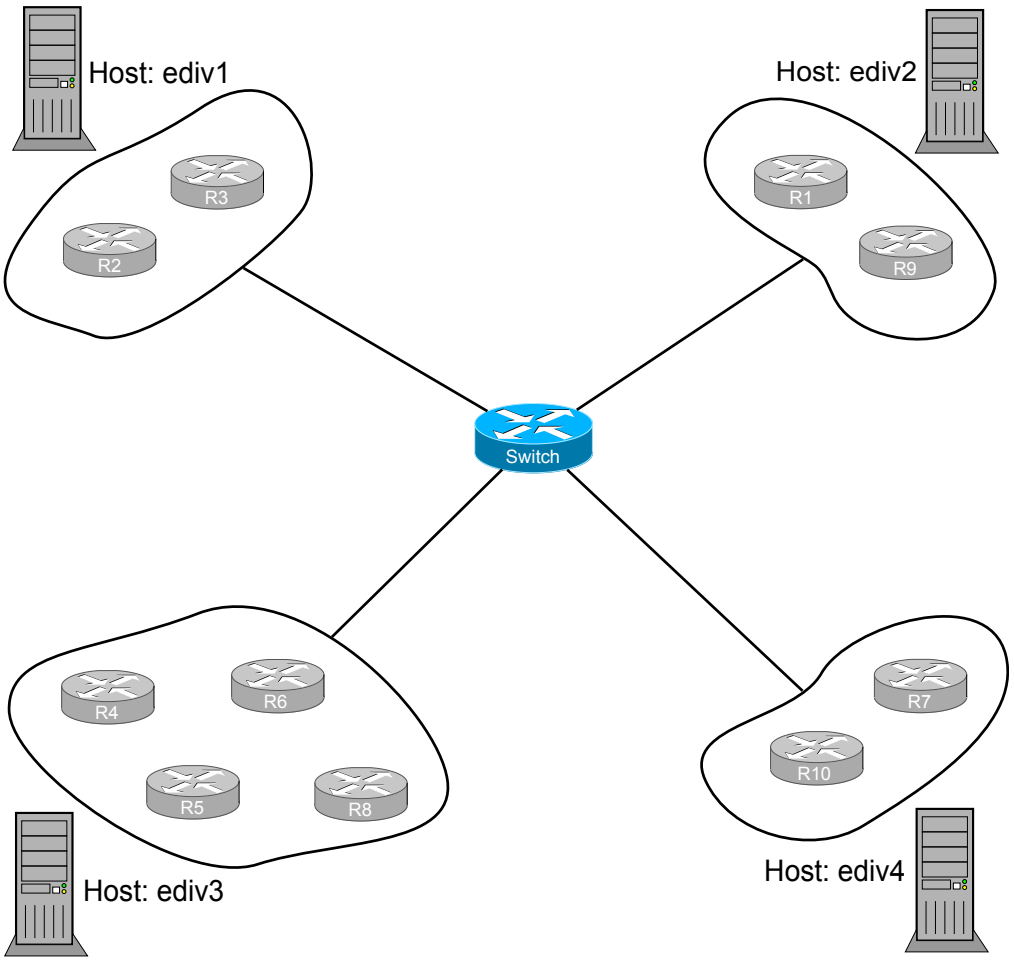

Abbildung 5.4: Verteilung des 10-Router-Szenarios auf die Cluster-Hosts

Name des Benutzers und der Datenbank, als auch das Passwort müssen beim Start des Szenarios schon in der verwendeten Cluster-Konfigurationsdatei eingetragen gewesen sein. Um sich nun den Inhalt der Datenbank anzeigen zu lassen, können zunächst mit dem Befehl

```
show tables;
```

die Namen sämtlicher Tabellen angezeigt werden, die EDIV in der Datenbank angelegt hat. Standardmäßig legt EDIV immer folgende fünf Tabellen an:

```
+----------------+
| Tables_in_ediv |
+----------------+
| hosts          |
| nets           |
| simulations    |
| vlans          |
| vms            |
+----------------+
5 rows in set (0.00 sec)
```

Der Inhalt dieser Tabellen kann nun mithilfe des SQL-Befehls *select* näher betrachtet werden. Der Befehl, um sich beispielsweise den kompletten Inhalt der Tabelle *vms* anzeigen zu lassen, lautet:

```
select * from vms;
```

EDIV gibt nun die entsprechende Tabelle aus. Zu sehen sind dort sämtliche gestarteten virtuellen Maschinen, sowie das jeweilige Szenario, dem sie angehören. Zusätzlich ist zu jeder VM der Host angegeben, auf dem sie läuft, sowie der SSH-Port, mit dem auf die virtuelle Maschine zugegriffen werden kann.

```
+------+------------+-------+----------+
| name | simulation |  host | ssh_port |
+------+------------+-------+----------+
|  R10 | szenario10 | ediv4 |  64000   |
|  R9  | szenario10 | ediv2 |  64007   |
|  R8  | szenario10 | ediv3 |  64008   |
|  R7  | szenario10 | ediv4 |  64003   |
|  R6  | szenario10 | ediv3 |  64004   |
|  R5  | szenario10 | ediv3 |  64006   |
|  R1  | szenario10 | ediv2 |  64001   |
|  R2  | szenario10 | ediv1 |  64009   |
|  R3  | szenario10 | ediv1 |  64002   |
|  R4  | szenario10 | ediv3 |  64005   |
+------+------------+-------+----------+
10 rows in set (0.00 sec)
```

Auf dieselbe Weise können mit dem select-Befehl auch die Tabellen *hosts*, *vlans*, *simulations* und *nets* ausgegeben werden. So werden beispielsweise in der Tabelle *vlans* für jedes angelegte VLAN die entsprechenden Hosts angezeigt, die daran angeschlossen sind, sowie die verwendete Netzwerkschnittstelle.

```
+--------+------------+-------+-------------+
| number | simulation | host  |  external_if |
+--------+------------+-------+-------------+
|   106  | szenario10 | ediv2 |    eth1     |
|   106  | szenario10 | ediv3 |    eth1     |
|   106  | szenario10 | ediv4 |    eth1     |
|   105  | szenario10 | ediv3 |    eth1     |
|   105  | szenario10 | ediv4 |    eth1     |
|   104  | szenario10 | ediv2 |    eth1     |
|   104  | szenario10 | ediv1 |    eth1     |
|   103  | szenario10 | ediv3 |    eth1     |
|   103  | szenario10 | ediv4 |    eth1     |
|   103  | szenario10 | ediv1 |    eth1     |
|   102  | szenario10 | ediv2 |    eth1     |
|   102  | szenario10 | ediv3 |    eth1     |
|   101  | szenario10 | ediv3 |    eth1     |
|   101  | szenario10 | ediv1 |    eth1     |
|   100  | szenario10 | ediv2 |    eth1     |
|   100  | szenario10 | ediv3 |    eth1     |
|   100  | szenario10 | ediv4 |    eth1     |
+--------+------------+-------+-------------+
17 rows in set (0.00 sec)
```

Eine alternative Möglichkeit, Informationen über die aktiven Szenarios zu erhalten, stellen die von EDIV bereitgestellten und in Kapitel 4.4 bereits beschriebenen EDIV-Skripte dar. Die Informationen, die durch die Skripte abgefragt werden können, decken sich größtenteils mit den in der Datenbank gespeicherten Daten.
So liefert beispielsweise der Aufruf des Skripts *ediv_ query_ status.pl* Informationen darüber, welche Szenarios derzeit auf welchen Hosts laufen.

```
The simulation szenario10 is running at host ediv1
The simulation szenario10 is running at host ediv2
The simulation szenario10 is running at host ediv3
The simulation szenario10 is running at host ediv4
```

Wird hingegen das Skript *ediv_ console.pl info* aufgerufen, so liefert dies dieselben Daten, die auch die Ausgabe der Tabelle *vms* ergeben hätte, lediglich in einer anderen Darstellungsform.

```
Simulation szenario10: To access VM R7 at ediv4 use local port 64003
Simulation szenario10: To access VM R8 at ediv3 use local port 64008
Simulation szenario10: To access VM R9 at ediv2 use local port 64007
Simulation szenario10: To access VM R10 at ediv4 use local port 64000
Simulation szenario10: To access VM R6 at ediv3 use local port 64004
Simulation szenario10: To access VM R5 at ediv3 use local port 64006
Simulation szenario10: To access VM R4 at ediv3 use local port 64005
Simulation szenario10: To access VM R3 at ediv1 use local port 64002
Simulation szenario10: To access VM R1 at ediv2 use local port 64001
Simulation szenario10: To access VM R2 at ediv1 use local port 64009
```

Es liegt somit beim Benutzer, ob er den direkten Zugriff auf die Datenbank oder die Nutzung der bereitgestellten Skripte bevorzugt, um Informationen über das Szenario abzurufen.

5.1.3 Konnektivität der virtuellen Maschinen

Nachdem in den vorangegangenen Unterkapiteln auf verschiedene Varianten bei der Verteilung von Szenarios, sowie auf die Repräsentation dieser Szenarios in der Datenbank eingegangen wurde, soll nun noch ein Blick auf die Erreichbarkeit der virtuellen Maschinen untereinander geworfen werden. Auch hierfür wird wieder unser 10-Router-Szenario verwendet, das ohne zusätzliche Argumente nach dem RoundRobin-Algorithmus verteilt wurde. Die Topologie des Szenarios ist bereits in Abbildung 5.1 zu sehen gewesen.
Zum Testen der gegenseitigen Erreichbarkeit der virtuellen Maschinen eignen sich insbesondere die Befehle *ping* und *traceroute*. Da in dem Szenario bislang noch kein Routing-Protokoll eingerichtet wurde, kennt jede virtuelle Maschine ausschließlich ihre direkten Nachbarn. So ist beispielsweise ein Ping von der virtuellen Maschine R1 nach R10 erfolgreich, wohingegen ein Ping von R1 nach R5 fehlschlägt.

```
1 R1:~# ping 10.0.1.2
2 PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
3 64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.385 ms
4 64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.447 ms
5
6 --- 10.0.1.2 ping statistics ---
7 2 packets transmitted, 2 received, 0% packet loss, time 999ms
8 rtt min/avg/max/mdev = 0.385/0.416/0.447/0.031 ms
```

```
1 R1:~# ping 10.0.5.2
2 connect: Network is unreachable
```

Ausschlaggebend für einen erfolgreichen Ping ist in diesem Fall also lediglich, welche virtuellen Maschinen an dasselbe Netz wie R1 angeschlossen und somit direkte Nachbarn von R1 sind. Dabei ist es völlig gleichgültig auf welchem Host sich diese virtuellen Maschinen befinden, da sie hostübergreifend durch VLANs miteinander vernetzt sind. Um die jeweiligen IP-Adressen von den Interfaces der virtuellen Maschinen herauszufinden, empfiehlt sich ein Blick in die XML-Repräsentation des Szenarios, die im Anhang A zu finden ist.
Damit nun jedoch nicht nur die Konnektivität der direkt miteinander verbundenen VMs, sondern die aller im Szenario vorhandenen virtuellen Maschinen getestet werden kann, ist es nötig für das Szenario ein Routing-Protokoll einzurichten. Dadurch wird für jede virtuelle Maschine jede andere VM des Szenarios erreichbar, egal ob sie ein direkter Nachbar ist oder nicht. Für diese Aufgabe wurde das Routing-Protokoll OSPF[3] gewählt.
Um später das Routing-Protokoll während der Simulation starten und stoppen zu können, müssen zunächst einige Anpassungen an der Szenariodatei vorgenommen

[3]OSPF steht für Open Shortest Path First und ist ein, innerhalb autonomer Systeme eingesetztes, dynamisches Routing-Protokoll. Ausführliche Informationen zur Funktionsweise des Protokolls finden sich unter: http://tools.ietf.org/pdf/rfc2328.pdf

werden. So müssen zu jeder virtuellen Maschine in der Szenariodatei die Befehle zum Starten und Beenden des OSPF-Diensts in <exec>-Tags geschrieben werden, damit diese während der Simulation durch die Kommandosequenzen *start* und *stop* ausgeführt werden können. Wie die benötigten Befehle innerhalb der <exec>-Tags genau aussehen, kann der entsprechend modifizierten Szenariodatei in Anhang B entnommen werden. Wird nun während der Simulation durch den Befehl

```
ediv_ctl.pl -x start -s szenario10.xml
```

die Kommandosequenz *start* ausgeführt, so wird auf jeder virtuellen Maschine ein OSPF-Router gestartet, der mit den anderen OSPF-Routern Nachbarschaftsinformationen austauscht, sodass sich schon nach kurzer Zeit alle virtuellen Maschinen des Szenarios gegenseitig erreichen können. Ein erneuter Ping von R1 nach R5 wird nun korrekt durch das Netzwerk zu seinem Ziel geleitet.

```
R1:~# ping 10.0.5.2
PING 10.0.5.2 (10.0.5.2) 56(84) bytes of data.
64 bytes from 10.0.5.2: icmp_seq=1 ttl=62 time=1.27 ms
64 bytes from 10.0.5.2: icmp_seq=2 ttl=62 time=1.27 ms

--- 10.0.5.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 1.270/1.272/1.275/0.035 ms
```

Um zu erfahren, welchen Weg der Ping von R1 nach R5 dabei innerhalb des Netzwerks genommen hat, kann sich der Benutzer dies mit dem Befehl *traceroute* anzeigen lassen.

```
R1:~# traceroute -n 10.0.5.2
traceroute to 10.0.5.2 (10.0.5.2), 30 hops max, 60 byte packets
 1  10.0.2.2 0.449 ms 0.315 ms 0.226 ms
 2  10.0.3.2 0.648 ms 0.510 ms 0.540 ms
 3  10.0.4.2 1.228 ms 0.839 ms 1.857 ms
 4  10.0.5.2 2.184 ms 2.063 ms 1.903 ms
```

Kapitel 6

Fazit und Ausblick

Ziel dieser Arbeit war es, dem Leser eine Möglichkeit an die Hand zu geben, auf einfache Weise selbst große Netzwerk-Szenarios simulieren zu können. Dazu wurde ausführlich auf die Installation, Konfiguration und Anwendung der beiden Netzwerk-Simulationstools VNUML und EDIV eingegangen und deren Funktionsweise erläutert. Insbesondere die verschiedenen Möglichkeiten, die EDIV bei der Verteilung und Steuerung der Szenarios bietet, wurden dabei umfassend erklärt und anhand eines selbst entwickelten Szenarios veranschaulicht. Dieses Szenario bestand der Übersichtlichkeit halber nur aus zehn virtuellen Maschinen, kann jedoch ohne Probleme beliebig vergrößert werden, solange die Ressourcen der zur Verfügung stehenden Hosts dies zulassen.
EDIV bietet durch die Verteilung der Szenarios auf mehrere Hosts nicht nur die Möglichkeit, wesentlich größere Szenarios als zuvor zu simulieren, es bietet zudem den Vorteil, den Start der Szenarios erheblich zu beschleunigen, indem die virtuellen Maschinen gleichmäßig auf die vorhandenen Hosts verteilt werden und sich jeder Host nur um den Start der ihm zugeteilten VMs kümmern muss. Üblicherweise sind dabei die Cluster-Hosts echte physische Rechner, die über einen VLAN-fähigen Switch miteinander verbunden sind. Ebenso gut ist es aber auch möglich noch einen Schritt weiter zu gehen und gewissermaßen eine doppelte Virtualisierung zu erzeugen, indem die Szenarios auf virtuelle statt auf physische Hosts verteilt und dort simuliert werden.

Was zukünftige Arbeiten zu diesem Thema betrifft, so gibt es einige Möglichkeiten, wie der Funktionsumfang von EDIV erweitert oder angepasst werden könnte.
Eine Idee wäre es beispielsweise, die bisher notwendige räumliche Nähe der Hosts zueinander aufzugeben. Bislang müssen sich die verwendeten Cluster-Hosts nämlich noch in einem lokalen Netz befinden, in welchem sie durch einen oder mehrere Switches miteinander verbunden sind, was zwangsweise voraussetzt, dass sich die Hosts in geographischer Nähe befinden. Viel praktischer wäre es doch, wenn nicht alle Hosts an demselben Ort stehen müssten, sondern über das Internet miteinander verbunden werden könnten. Neben dem Einrichten von Tunnel zur Vernetzung

der Hosts über das Internet, wäre auch die Sicherheit ein wichtiger Aspekt, den es in diesem Zusammenhang zu beachten gelte.
Ein weiterer Punkt, an dem der Funktionsumfang von EDIV noch erweitert werden könnte, sind die zur Verfügung stehenden Segmentierungsalgorithmen. Hier sind bislang erst zwei Algorithmen in EDIV integriert, die beide recht simpel gehalten sind und die Topologie der Szenarios bei der Verteilung der virtuellen Maschinen komplett ignorieren. Hier könnten weitere Segmentierungsalgorithmen entwickelt werden, die beispielsweise unter Zuhilfenahme von Metriken jeweils diejenigen virtuellen Maschinen ermitteln, die innerhalb des Szenarios nah beieinander liegen, um diese dann bei der Verteilung der VMs auf demselben Host zu platzieren.
Auch wäre es denkbar, EDIV so zu erweitern oder abzuändern, dass es neben User-Mode Linux und dem darauf aufbauenden VNUML auch mit anderer zugrunde liegender Virtualisierungssoftware umgehen kann, da VNUML bereits nicht mehr weiterentwickelt wird.
Dies sind nur ein paar Ideen, wie die weitere Entwicklung aussehen könnte. Sicherlich lassen sich zu diesem Thema noch viele weitere interessante Möglichkeiten zur Weiterentwicklung finden.

Anhang A

10-Router-Szenario ohne Routing-Protokoll

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">
<vnuml>
  <global>
    <version>1.8</version>
    <simulation_name>szenario10</simulation_name>
    <ssh_version>2</ssh_version>
    <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
    <automac/>
    <vm_mgmt type="private" network="192.168.0.0" mask="24" offset="100">
    <host_mapping/>
    </vm_mgmt>
    <vm_defaults exec_mode="mconsole">
      <filesystem type="cow">/usr/share/vnuml/filesystems/root_fs_tutorial</filesystem>
      <kernel>/usr/share/vnuml/kernels/linux</kernel>
      <console id="0">xterm</console>
      <forwarding type="ipv4"/>
    </vm_defaults>
  </global>

  <net name="Net1" mode="uml_switch" type="lan" />
  <net name="Net2" mode="uml_switch" type="lan" />
  <net name="Net3" mode="uml_switch" type="lan" />
  <net name="Net4" mode="uml_switch" type="lan" />
  <net name="Net5" mode="uml_switch" type="lan" />
  <net name="Net6" mode="uml_switch" type="lan" />
  <net name="Net7" mode="uml_switch" type="lan" />
  <net name="Net8" mode="uml_switch" type="lan" />
  <net name="Net9" mode="uml_switch" type="lan" />
  <net name="Net10" mode="uml_switch" type="lan" />

  <vm name="R1">
    <if id="1" net="Net1">
      <ipv4>10.0.1.1</ipv4>
    </if>
    <if id="2" net="Net2">
      <ipv4>10.0.2.1</ipv4>
    </if>
```

```
39      <if id="3" net="Net8">
40        <ipv4>10.0.8.1</ipv4>
41      </if>
42    </vm>
43
44    <vm name="R2">
45      <if id="1" net="Net2">
46        <ipv4>10.0.2.2</ipv4>
47      </if>
48      <if id="2" net="Net3">
49        <ipv4>10.0.3.1</ipv4>
50      </if>
51    </vm>
52
53    <vm name="R3">
54      <if id="1" net="Net3">
55        <ipv4>10.0.3.2</ipv4>
56      </if>
57        <if id="2" net="Net4">
58        <ipv4>10.0.4.1</ipv4>
59      </if>
60        <if id="3" net="Net7">
61        <ipv4>10.0.7.1</ipv4>
62      </if>
63    </vm>
64
65    <vm name="R4">
66      <if id="1" net="Net4">
67        <ipv4>10.0.4.2</ipv4>
68      </if>
69        <if id="2" net="Net5">
70        <ipv4>10.0.5.1</ipv4>
71      </if>
72    </vm>
73
74    <vm name="R5">
75      <if id="1" net="Net5">
76        <ipv4>10.0.5.2</ipv4>
77      </if>
78        <if id="2" net="Net6">
79        <ipv4>10.0.6.1</ipv4>
80      </if>
81    </vm>
82
83    <vm name="R6">
84      <if id="1" net="Net6">
85        <ipv4>10.0.6.2</ipv4>
86      </if>
87        <if id="2" net="Net9">
88        <ipv4>10.0.9.1</ipv4>
89      </if>
90        <if id="3" net="Net7">
91        <ipv4>10.0.7.2</ipv4>
92      </if>
93    </vm>
94
95    <vm name="R7">
96      <if id="1" net="Net7">
97        <ipv4>10.0.7.3</ipv4>
98      </if>
99        <if id="2" net="Net8">
100       <ipv4>10.0.8.2</ipv4>
```

```
101     </if>
102   </vm>
103
104   <vm name="R8">
105     <if id="1" net="Net8">
106       <ipv4>10.0.8.3</ipv4>
107     </if>
108       <if id="2" net="Net9">
109       <ipv4>10.0.9.2</ipv4>
110     </if>
111       <if id="3" net="Net10">
112       <ipv4>10.0.10.1</ipv4>
113     </if>
114   </vm>
115
116   <vm name="R9">
117     <if id="1" net="Net9">
118       <ipv4>10.0.9.3</ipv4>
119     </if>
120       <if id="2" net="Net10">
121       <ipv4>10.0.10.2</ipv4>
122     </if>
123   </vm>
124
125   <vm name="R10">
126     <if id="1" net="Net10">
127       <ipv4>10.0.10.3</ipv4>
128     </if>
129       <if id="2" net="Net1">
130       <ipv4>10.0.1.2</ipv4>
131     </if>
132   </vm>
133
134 </vnuml>
```

Abbildung A.1: Szenariodatei des 10-Router-Szenarios

Anhang B

10-Router-Szenario mit OSPF

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">
<vnuml>
  <global>
    <version>1.8</version>
    <simulation_name>szenario10</simulation_name>
    <ssh_version>2</ssh_version>
    <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
    <automac/>
    <vm_mgmt type="private" network="192.168.0.0" mask="24" offset="100">
      <host_mapping/>
    </vm_mgmt>
    <vm_defaults exec_mode="mconsole">
      <filesystem type="cow">/usr/share/vnuml/filesystems/root_fs_tutorial</filesystem>
      <kernel>/usr/share/vnuml/kernels/linux</kernel>
      <console id="0">xterm</console>
      <forwarding type="ipv4"/>
    </vm_defaults>
  </global>

  <net name="Net1" mode="uml_switch" type="lan" />
  <net name="Net2" mode="uml_switch" type="lan" />
  <net name="Net3" mode="uml_switch" type="lan" />
  <net name="Net4" mode="uml_switch" type="lan" />
  <net name="Net5" mode="uml_switch" type="lan" />
  <net name="Net6" mode="uml_switch" type="lan" />
  <net name="Net7" mode="uml_switch" type="lan" />
  <net name="Net8" mode="uml_switch" type="lan" />
  <net name="Net9" mode="uml_switch" type="lan" />
  <net name="Net10" mode="uml_switch" type="lan" />

  <vm name="R1">
    <if id="1" net="Net1">
      <ipv4>10.0.1.1</ipv4>
    </if>
    <if id="2" net="Net2">
      <ipv4>10.0.2.1</ipv4>
    </if>
    <if id="3" net="Net8">
      <ipv4>10.0.8.1</ipv4>
    </if>
    <filetree root="/etc/quagga" seq="start">conf</filetree>
```

```
43      <exec seq="start" type="verbatim">hostname</exec>
44      <exec seq="start" type="verbatim">ip route flush scope global</exec>
45      <exec seq="start" type="verbatim">
46      /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
47      </exec>
48      <exec seq="start" type="verbatim">
49      /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
50      </exec>
51      <exec seq="stop" type="verbatim">hostname</exec>
52      <exec seq="stop" type="verbatim">killall zebra</exec>
53      <exec seq="stop" type="verbatim">killall ospfd</exec>
54    </vm>
55
56    <vm name="R2">
57      <if id="1" net="Net2">
58        <ipv4>10.0.2.2</ipv4>
59      </if>
60      <if id="2" net="Net3">
61        <ipv4>10.0.3.1</ipv4>
62      </if>
63      <filetree root="/etc/quagga" seq="start">conf</filetree>
64      <exec seq="start" type="verbatim">hostname</exec>
65      <exec seq="start" type="verbatim">ip route flush scope global</exec>
66      <exec seq="start" type="verbatim">
67      /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
68      </exec>
69      <exec seq="start" type="verbatim">
70      /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
71      </exec>
72      <exec seq="stop" type="verbatim">hostname</exec>
73      <exec seq="stop" type="verbatim">killall zebra</exec>
74      <exec seq="stop" type="verbatim">killall ospfd</exec>
75    </vm>
76
77    <vm name="R3">
78      <if id="1" net="Net3">
79        <ipv4>10.0.3.2</ipv4>
80      </if>
81        <if id="2" net="Net4">
82        <ipv4>10.0.4.1</ipv4>
83      </if>
84        <if id="3" net="Net7">
85        <ipv4>10.0.7.1</ipv4>
86      </if>
87      <filetree root="/etc/quagga" seq="start">conf</filetree>
88      <exec seq="start" type="verbatim">hostname</exec>
89      <exec seq="start" type="verbatim">ip route flush scope global</exec>
90      <exec seq="start" type="verbatim">
91      /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
92      </exec>
93      <exec seq="start" type="verbatim">
94      /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
95      </exec>
96      <exec seq="stop" type="verbatim">hostname</exec>
97      <exec seq="stop" type="verbatim">killall zebra</exec>
98      <exec seq="stop" type="verbatim">killall ospfd</exec>
99    </vm>
100
101   <vm name="R4">
102     <if id="1" net="Net4">
103       <ipv4>10.0.4.2</ipv4>
104     </if>
```

```
105       <if id="2" net="Net5">
106       <ipv4>10.0.5.1</ipv4>
107     </if>
108     <filetree root="/etc/quagga" seq="start">conf</filetree>
109     <exec seq="start" type="verbatim">hostname</exec>
110     <exec seq="start" type="verbatim">ip route flush scope global</exec>
111     <exec seq="start" type="verbatim">
112     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
113     </exec>
114     <exec seq="start" type="verbatim">
115     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
116     </exec>
117     <exec seq="stop" type="verbatim">hostname</exec>
118     <exec seq="stop" type="verbatim">killall zebra</exec>
119     <exec seq="stop" type="verbatim">killall ospfd</exec>
120   </vm>
121
122   <vm name="R5">
123     <if id="1" net="Net5">
124       <ipv4>10.0.5.2</ipv4>
125     </if>
126       <if id="2" net="Net6">
127       <ipv4>10.0.6.1</ipv4>
128     </if>
129     <filetree root="/etc/quagga" seq="start">conf</filetree>
130     <exec seq="start" type="verbatim">hostname</exec>
131     <exec seq="start" type="verbatim">ip route flush scope global</exec>
132     <exec seq="start" type="verbatim">
133     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
134     </exec>
135     <exec seq="start" type="verbatim">
136     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
137     </exec>
138     <exec seq="stop" type="verbatim">hostname</exec>
139     <exec seq="stop" type="verbatim">killall zebra</exec>
140     <exec seq="stop" type="verbatim">killall ospfd</exec>
141   </vm>
142
143   <vm name="R6">
144     <if id="1" net="Net6">
145       <ipv4>10.0.6.2</ipv4>
146     </if>
147       <if id="2" net="Net9">
148       <ipv4>10.0.9.1</ipv4>
149     </if>
150       <if id="3" net="Net7">
151       <ipv4>10.0.7.2</ipv4>
152     </if>
153     <filetree root="/etc/quagga" seq="start">conf</filetree>
154     <exec seq="start" type="verbatim">hostname</exec>
155     <exec seq="start" type="verbatim">ip route flush scope global</exec>
156     <exec seq="start" type="verbatim">
157     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
158     </exec>
159     <exec seq="start" type="verbatim">
160     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
161     </exec>
162     <exec seq="stop" type="verbatim">hostname</exec>
163     <exec seq="stop" type="verbatim">killall zebra</exec>
164     <exec seq="stop" type="verbatim">killall ospfd</exec>
165   </vm>
166
```

```
167   <vm name="R7">
168     <if id="1" net="Net7">
169       <ipv4>10.0.7.3</ipv4>
170     </if>
171       <if id="2" net="Net8">
172       <ipv4>10.0.8.2</ipv4>
173     </if>
174     <filetree root="/etc/quagga" seq="start">conf</filetree>
175     <exec seq="start" type="verbatim">hostname</exec>
176     <exec seq="start" type="verbatim">ip route flush scope global</exec>
177     <exec seq="start" type="verbatim">
178     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
179     </exec>
180     <exec seq="start" type="verbatim">
181     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
182     </exec>
183     <exec seq="stop" type="verbatim">hostname</exec>
184     <exec seq="stop" type="verbatim">killall zebra</exec>
185     <exec seq="stop" type="verbatim">killall ospfd</exec>
186   </vm>
187
188   <vm name="R8">
189     <if id="1" net="Net8">
190       <ipv4>10.0.8.3</ipv4>
191     </if>
192       <if id="2" net="Net9">
193       <ipv4>10.0.9.2</ipv4>
194     </if>
195       <if id="3" net="Net10">
196       <ipv4>10.0.10.1</ipv4>
197     </if>
198     <filetree root="/etc/quagga" seq="start">conf</filetree>
199     <exec seq="start" type="verbatim">hostname</exec>
200     <exec seq="start" type="verbatim">ip route flush scope global</exec>
201     <exec seq="start" type="verbatim">
202     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
203     </exec>
204     <exec seq="start" type="verbatim">
205     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
206     </exec>
207     <exec seq="stop" type="verbatim">hostname</exec>
208     <exec seq="stop" type="verbatim">killall zebra</exec>
209     <exec seq="stop" type="verbatim">killall ospfd</exec>
210   </vm>
211
212   <vm name="R9">
213     <if id="1" net="Net9">
214       <ipv4>10.0.9.3</ipv4>
215     </if>
216       <if id="2" net="Net10">
217       <ipv4>10.0.10.2</ipv4>
218     </if>
219     <filetree root="/etc/quagga" seq="start">conf</filetree>
220     <exec seq="start" type="verbatim">hostname</exec>
221     <exec seq="start" type="verbatim">ip route flush scope global</exec>
222     <exec seq="start" type="verbatim">
223     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
224     </exec>
225     <exec seq="start" type="verbatim">
226     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
227     </exec>
228     <exec seq="stop" type="verbatim">hostname</exec>
```

```
229     <exec seq="stop" type="verbatim">killall zebra</exec>
230     <exec seq="stop" type="verbatim">killall ospfd</exec>
231   </vm>
232
233   <vm name="R10">
234     <if id="1" net="Net10">
235       <ipv4>10.0.10.3</ipv4>
236     </if>
237       <if id="2" net="Net1">
238       <ipv4>10.0.1.2</ipv4>
239     </if>
240     <filetree root="/etc/quagga" seq="start">conf</filetree>
241     <exec seq="start" type="verbatim">hostname</exec>
242     <exec seq="start" type="verbatim">ip route flush scope global</exec>
243     <exec seq="start" type="verbatim">
244     /usr/lib/quagga/zebra -f /etc/quagga/zebra.conf -d
245     </exec>
246     <exec seq="start" type="verbatim">
247     /usr/lib/quagga/ospfd -f /etc/quagga/ospfd.conf -d -P 2604
248     </exec>
249     <exec seq="stop" type="verbatim">hostname</exec>
250     <exec seq="stop" type="verbatim">killall zebra</exec>
251     <exec seq="stop" type="verbatim">killall ospfd</exec>
252   </vm>
253
254 </vnuml>
```

Abbildung B.1: Szenariodatei des 10-Router-Szenarios mit OSPF

Literaturverzeichnis

[Dik06] Jeff Dike: *User Mode Linux.* Prentice Hall, 2006. S. 1-14.

[Gar10] Andreas Garbe: *Simulation großer Netzwerke in der VNUML-Umgebung.* Diplomarbeit, Universität Koblenz-Landau: Campus Koblenz, September 2010.

[GF07] Fermín Galán und David Fernández: *Distributed Virtualization Scenarios Using VNUML.* In: *First System and Virtualization Management Workshop (SVM 2007)*, Toulouse (Frankreich), Oktober 2007.

[Keu05] Tim Keupen: *User-Mode Linux.* Seminararbeit, Universität Koblenz-Landau: Campus Koblenz, November 2005.

[Mad] Technische Universität Madrid: *VNUML-Wiki.* `http://neweb.dit.upm.es/vnumlwiki/index.php/Main_Page`, Letzter Zugriff: 27.11.2011.

[oEE98] Institute of Electrical und Electronics Engineers: *IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks*, 1998. `http://standards.ieee.org/getieee802/download/802.1Q-1998.pdf`, Letzter Zugriff: 27.11.2011.

[PD03] Larry L. Peterson und Bruce S. Davie: *Computer Networks: A Systems Approach.* Kaufmann, 3. Auflage, 2003. S. 680-684.

[Sch04] Jens Schäfer: *VLAN Switching.* Diplomarbeit, Universität Koblenz-Landau: Campus Koblenz, November 2004.

[Sys02] Cisco Systems: *Cisco Networking Academy Program: Lehrbuch 3. und 4. Semester.* Markt+Technik, 2002. S. 67-82.

[Tan03] Andrew S. Tanenbaum: *Computernetzwerke.* Pearson Studium, 4. überarbeitete Auflage, 2003. S. 365-372.

[VDE] *VDE-Wiki.* `http://wiki.virtualsquare.org/wiki/index.php/Main_Page`, Letzter Zugriff: 27.11.2011.

[WR10] Robert Warnke und Thomas Ritzau: *qemu-kvm & libvirt.* Books on Demand, 4. Auflage, 2010.