Bachelorarbeit

Philip Raddatz

Entwicklung einer Netzwerkvirtualisierung zur Konfiguration und zum automatisierten Testen von iptables-basierten Firewalls

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Philip Raddatz

Entwicklung einer Netzwerkvirtualisierung zur Konfiguration und zum automatisierten Testen von iptables-basierten Firewalls

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Heitmann
Zweitgutachter : Prof. Dr. Korf

Abgegeben am 6. Januar 2011

Philip Raddatz

Thema der Bachelorarbeit

Entwicklung einer Netzwerkvirtualisierung zur Konfiguration und zum automatisierten Testen von iptables-basierten Firewalls

Stichworte

iptables, Firewall, Netzwerk, Netzwerkvirtualisierung, Konfiguration, Testen, Generierung, User-Mode-Linux, Virtualisierung, Auswertung, GUI, Managementkonsole, Scripte, Python, Linux, Client/Server-Kommunikation

Kurzzusammenfassung

Diese Arbeit behandelt die Entwicklung einer Netzwerkvirtualisierung, basierend auf User-Mode-Linux, mit einer zentralen grafischen Oberfläche, über die die einzelnen Netzwerkclients verwaltet und administriert werden. Innerhalb dieser Managementkonsole können iptables-Scripte auf den Clients ausgeführt werden. Aus den iptables-Scripten werden Abfrage-Scripte generiert. Diese sind ausführbar, testen die Korrektheit und Funktionsfähigkeit der iptables-Regeln und werten sie aus.

Philip Raddatz

Title of the paper

Development of a network virtualization for the configuration and automatic testing of iptables-based firewalls

Keywords

Iptables, Firewall, Network, Networkvirtualization, Configuration, Testing, Generating, User-Mode-Linux, Virtualization, Evaluation, GUI, Management-GUI, Scripts, Python, Linux, Client/Server-Communication

Abstract

This thesis discusses the development of a network virtualization, based on User-Mode-Linux, with a central grafical interface, where each networkclient can be managed. The Management-GUI can execute and run iptables-scripts on the clients. It generates executable queryscripts from iptables-scripts to test their correctness and evaluate the results.

Inhaltsverzeichnis

Abbildungsverzeichnis **7**

1 Einleitung **8**
1.1 Problemstellung . . . 8
1.2 Motivation . . . 9
1.3 Allgemeine Grundlagen . . . 10
1.4 Gliederung der Arbeit . . . 10

2 Anforderung **11**

3 Stand der Technik **13**
3.1 Virtuelle Konzepte . . . 13
3.1.1 VMWare . . . 14
3.1.2 Virtual Box . . . 15
3.1.3 Microsoft Virtual PC . . . 15
3.1.4 UML - User Mode Linux . . . 15
3.2 Firewall . . . 16
3.3 iptables . . . 18
3.3.1 Konzepte . . . 18
3.3.2 Tabellen . . . 19
3.3.3 Paketfluss . . . 20
3.3.4 Regeln . . . 21
3.3.5 Targets . . . 21
3.4 Hilfsprogramme . . . 22
3.4.1 Allgemein . . . 22
3.4.2 Hilfsprogramme für iptables . . . 23
3.4.3 iptables-testing . . . 23
3.5 Netzwerkvirtualisierung . . . 23

4 Analyse **25**
4.1 Virtualisierungskonzept . . . 26
4.2 iptables . . . 28
4.3 iptables-Regeln konfigurieren . . . 30

5 Umsetzung 31
5.1 Allgemein . . . 31
5.1.1 Python . . . 31
5.1.2 Netzwerkvirtualisierung . . . 32
5.1.3 Iptables-Regeln . . . 33
5.2 Netzwerkvirtualisierung . . . 33
5.2.1 Die Distribution . . . 33
5.2.2 copy-on-write (COW) . . . 34
5.2.3 uml-switch . . . 34
5.3 Aufbau . . . 35
5.3.1 Allgemein . . . 35
5.3.2 Start-Script . . . 35
5.3.3 Kill-Script . . . 37
5.3.4 Topologie-Script . . . 37
5.3.5 Initialisierungs-Script . . . 38
5.3.6 GUI . . . 40
5.3.7 UML-Host . . . 40
5.3.8 UML-Client . . . 42
5.3.9 Client-Server-Kommunikation . . . 43
5.3.10 Zentrales Management . . . 45
5.4 Generieren und Auswerten . . . 47
5.4.1 Abfrage-Befehle generieren . . . 47
5.4.2 Überprüfung der iptables-Regeln . . . 49
5.4.3 Auswertung . . . 50
5.4.4 Grenzen der Überprüfbarkeit . . . 52
5.5 iptables-Konfigurator . . . 53
5.5.1 Realisierungsansatz A: Zusammenstellung mehrerer Regeln . . . 53
5.5.2 Realisierungsansatz B: Selbstgestaltung . . . 54
5.5.3 Entscheidung . . . 54
5.5.4 Funktionsweise des Konfigurators . . . 54
5.6 Installation und Ausführung . . . 55
5.6.1 Python . . . 56
5.6.2 GUI-Toolkit: wxWidgets - wxPython . . . 57

6 Zusammenfassung 58

7 Ausblick 60

Literaturverzeichnis 62

I Anhang **64**

A Hilfe **65**
A.1 Vorbereitung und Starten der GUI . . . 65
A.2 Arbeiten in der GUI . . . 66
A.2.1 Starten und Beenden des Netzwerks . . . 67
A.2.2 Die Befehlszeile . . . 67
A.2.3 Die Textfenster . . . 67
A.2.4 Scripte ausführen . . . 68
A.2.5 Abfragen generieren . . . 68
A.2.6 Auswertung . . . 69
A.2.7 iptables-Konfigurator . . . 69
A.3 Filesystem Update . . . 70

Abbildungsverzeichnis

3.1 Betriebssystemvirtualisierung mittels OS-Container 13
3.2 Systemvirtualisierung mittels Hypervisor 14
3.3 Beispiel des Paketflusses und Hooks der Filtertabelle 19
3.4 Weg eines Paketes durch die iptables-Regeln 20
3.5 Ein Paket wird durch die Regeln einer Kette geprüft. Diese Regeln werden nach und nach abgearbeitet. Trifft keine der Regeln zu, tritt die default Policy in Kraft 21

4.1 Benutzer interagiert über die zentrale Verwaltungseinheit mit den Knoten im Netzwerk 25

5.1 Beispiel eines Netzwerkes mit drei Knoten 38
5.2 Aufbau der grafischen Benutzeroberfläche (GUI) 41
5.3 Befehlsübertragung mit dem pickle-Modul 45
5.4 Kommunikation zwischen Host (GUI) und Knoten (UML) 46
5.5 Der Weg eines Befehls von der Eingabe bis zur Ausgabe auf der grafischen Oberfläche 47
5.6 Beispiel: Veranschaulichung der Funktion zum Abfragen von iptables-Scripten (hier mit falscher Bearbeitung der Aufgabenstellung des Benutzers) 48
5.7 Flussdiagramm: Auswertung 51
5.8 Sequenzdiagramm: iptables-Konfigurator 56

A1 Beispielaufbau eines Netzwerks mit vier Knoten 65
A2 Die grafische Oberfläche (GUI) 66
A3 Beispiel eines iptables-Scripts 69

1 Einleitung

1.1 Problemstellung

In der heutigen Zeit gibt es Computer wie Sand am Meer, gerade an Universitäten, an denen diese miteinander vernetzt sind. Doch was ist, wenn in einem Seminar mehrere Computer für einen Benutzer zur Verfügung stehen müssen? Bekommt dieser Benutzer dann gleich einen Pool von Computern? Und was ist mit den anderen Benutzern? Hat am Ende jeder zehn Rechner vor sich stehen oder gehen einige leer aus? Gibt es überhaupt genug Platz um zu bewerkstelligen, dass viele Benutzer oder Studenten mehrere Rechner auf einmal zur Verfügung haben?
Aus diesem Grund - natürlich nicht nur aus diesem - gibt es virtuelle Netzwerke die dem User an einem PC ein komplettes Netzwerk simulieren. Ein Vorteil dieser Virtualisierung ist, dass der User auf den einzelnen virtuellen Maschinen (im folgenden auch VM genannt) arbeiten kann, ohne Gefahr zu laufen, dass das Hostsystem beschädigt wird. Es wird in den virtuellen Maschinen gearbeitet, beim Starten gibt es eine Grundkonfiguration, die auf allen VMs gleich ist. Falls während der Arbeit ein gravierender Fehler in der VM auftritt, beendet man diese, und startet sie neu, ohne das Hostsystem in Mitleidenschaft zu ziehen.

Ein Teil dieser Arbeit hat den Aufbau eines eben solchen virtuellen Netzwerkes mit einem zentralen Management über eine grafische Oberfläche als Schwerpunkt. Weiterhin wird das Thema Firewall einen Teil dieser Bachelorarbeit einnehmen.

Anstatt - wie in vielen virtuellen Netzwerken - jeden Client bzw. jedes Betriebssystem einzeln über seine eigenen Console oder Oberfläche zu steuern, soll dem Benutzer ermöglicht werden, Einstellungen und Konfigurationen (den Client betreffend) über eine zentrale Verwaltung zu managen. Die Zentrale soll sich zudem dynamisch anpassen, je nachdem wieviele Clients gestartet werden. Das heißt, der Benutzer soll nicht zwischen den einzelnen Consolen der Clients wechseln müssen und dort Einstellungen vornehmen, sondern diese sollen zentral an einem Ort geschehen. Eingegebene Befehle sollen dann an den jeweiligen Client gesendet werden und die Ausgabe von Rückgabewerten oder auch Fehlermeldungen sollen wieder an diese zentrale Verwaltungsstelle gesendet werden.

Außerdem soll dem Benutzer ermöglicht werden, unter Einhaltung einiger Vorgaben, einfach gestaltete Firewall-Scripte und Einstellungen auf den virtuellen Maschinen zu verteilen, oh-

ne auf jedem Client selber die Befehle/Regeln eingeben zu müssen. Gerade in diesem Fall benötigt man mehrere Rechner, die miteinander vernetzt sind, um die verschiedenen Firewalleinstellungen vorzunehmen und zu testen. Um die Investitionskosten in viele Computer zu sparen, ist es praktisch hier eine Netzwerkvirtualisierung zu erstellen in der der User ausgiebig konfigurieren, ausprobieren und testen kann.

Auf den Punkt gebracht ist es das Ziel, die Netzwerkvirtualisierung als Spielwiese für das Studieren und Konfigurieren von Firewalls zu nutzen. Das Ganze soll an einem Arbeitsplatz möglich sein, also auf einem einzelnen Computer laufen. Die Installation soll einfach gehalten sein, das heißt es sollen wenige Schritte erforderlich sein um das Programm zu starten. Ein wichtiger Aspekt ist das Arbeiten ohne root-Privilegien.

Um das ganze praxisnah zu gestalten soll es auch möglich sein, zu einer bestimmten Aufgabe ein Firewall-Script zu schreiben, und die eigene Lösung beziehungsweise die Ausarbeitungen anderer Benutzer zu dieser gegebenen Aufgabe zu prüfen und die Ergebnisse automatisch auszuwerten und gegenüberzustellen.

Das heißt, der Aufgabensteller führt sein Script (das diese Aufgabe löst) aus und kann sich ein Abfragescript generieren lassen, welches alle Befehle und Regeln, die vorher ausgeführt wurden, abfragt. Dieses Script kann abgespeichert werden und die Aufgabe kann nun von anderen gelöst werden. Um dann nicht alle Lösungen einzeln Testen zu müssen, wird das Abfragescript ausgeführt, und eine Auswertung kann angezeigt werden, um zu überprüfen welche Punkte der Aufgabe gelöst/nicht gelöst wurden. Weiterhin kann dem Benutzer schon das richtige Lösungsscript gegeben werden und er bzw sie kann im Nachhinein damit die eigenen Einstellungen der Firewall testen und überprüfen.

Um Anfänger den Einstieg in iptables zu erleichtern gibt es zu guter Letzt noch einen Konfigurator, mit dem iptables-Regeln erstellt werden können. Der Benutzer kann sich einen korrekten iptables-Befehl selber zusammenstellen. Er muss also nicht die Regel selber eingeben und dann überprüfen ob diese semantisch korrekt ist. Da diese Bachelorarbeit eine Hilfe für das erlernen im Umgang mit Firewalls sein soll, bietet es sich an, dem Benutzer diese kleine Hilfe mit an die Hand zu geben, um Grundbefehle erstellen zu können.

1.2 Motivation

Diese Bachelorarbeit soll den Aufbau eines auf User-Mode-Linux basierten Netzwerkes zeigen, bei dem die einzelnen Netzwerkknoten (auch Router oder Clients genannt) über eine zentrale Management Konsole verwaltet werden. Es soll gezeigt werden wie diese zentrale Verwaltungseinheit funktioniert und wie die Kommunikation dieser Managementkonsole mit den einzelnen Netzwerk-Clients abläuft. In dieser Virtualisierung soll es möglich sein Firewalls zu konfigurieren und zu testen. Unter Linux ist der Befehl IPTABLES dafür zuständig

Pakete zu manipulieren und zu dirigieren. In Bezug auf die zentrale Verwaltung soll aufgezeigt werden wie iptables-Scripte von der Verwaltung auf den einzelnen Clients ausgeführt werden und Scripte zur Abfrage von iptables-Befehlen generiert werden. Um den Einstieg in iptables für Neulinge zu erleichtern wird den Benutzer noch ein Konfigurator zur Hand gegeben mit dem man einzelne iptables-Regeln zusammenstellen kann.

1.3 Allgemeine Grundlagen

Bezüglich dieser Bachelorarbeit sind Grundlagen wie das Verständnis und der Funktion eines Netzwerkes sowie Befehle zum Konfigurieren und Abfragen der Konfiguration Voraussetzung. Die Funktion einer Firewall und der Begriff 'iptables' sollte bekannt sein, jedoch wird im Verlauf der Bachelorarbeit darauf genauer eingegangen. Es wird der Aufbau einer iptables-Regel näher erläutert, aber es gibt keine Auflistung von Befehlen, da der Umfang von iptables so groß ist, das damit weitere Bachelorarbeiten gefüllt werden können.

1.4 Gliederung der Arbeit

Diese Arbeit ist im Wesentlichen in fünf Bereich gegliedert. In Kapitel 2 wird die Anforderung an das Programm beschrieben. Welche Teilbereiche gibt es, und welche Funktionalität sollen diese enthalten. Das nächste Kapitel bezieht sich dann auf den Stand der Technik. Was für Software, Projekte und Anwendungen gibt es bezüglich des Themas dieser Arbeit schon. Des weiteren werden Programme aufgelistet, die in diese Arbeit integriert werden können, um zum Beispiel dabei zu helfen, bestimmte Konfigurationen abzufragen. Kapitel 4 analysiert dann die in Kapitel 3 aufgezählten Programme und überprüft die Einsetzbarkeit dieser in diese Arbeit. Ausserdem sind ein paar Grundideen von Programminhalten dort schon erläutert. Das Kapitel 5 beschäftigt sich dann mit der Umsetzung. Hier wird detailliert gezeigt wie Teile des Programms umgesetzt und implementiert werden, wie das Netzwerk aufgebaut wird, wie die Kommunikation innerhalb des Netzwerkes verläuft und wie sich das Netzwerk administrieren lässt. Dazu sind kleine Quellcode-Listings aufgeführt, die Funktionsweisen einzelner Komponenten veranschaulichen. Die danach folgenden Kapitel fassen diese Arbeit noch einmal zusammen und bieten einen Ausblick darauf, wie mit dieser Arbeit weitergearbeitet werden kann. Eine Hilfe zu diesem Programm rundet diese Arbeit ab.

2 Anforderung

Das Programm soll im Wesentlichen drei Bereiche erfüllen:

Der erste Teil ist das Erstellen einer Netzwerkvirtualisierung und das zentrale Managen dieser aus einer grafischen Oberfläche heraus. Hier soll es möglich sein, mehrere Clients zu virtualisieren und diese miteinander zu vernetzen. Es soll für den Benutzer so aussehen, als würde er auf seinem Computer ein kleines Netzwerk managen und konfigurieren können. Gerade an Universitäten, an denen es bestimmte Seminare gibt, in welchen man mehrere Computer zu Testzwecken braucht, soll diese Netzwerkvirtualisierung einsetzbar sein. Jeder Benutzer kann ein eigenes virtuelles Netzwerk aufbauen und testen. So belegt er dafür nicht mehrere Computer, die möglicherweise von anderen Studenten genutzt werden könnten. Hauptaugenmerk wird bei der Virtualisierung wie oben beschrieben darauf gelegt, dass die Clients zentral verwaltet werden. Anstatt für jeden Client eine einzelne Console zu haben, soll es möglich sein, in einer GUI Zugriff auf alle Clients zu haben, von dort Befehle an sie zu übermitteln, oder Scripte auf den einzelnen Clients ausführen zu lassen. Kurz gesagt soll es möglich sein, das Netzwerk zentral in einer grafischen Benutzeroberfläche zu verwalten.

Weiterhin soll es möglich sein, dass der Benutzer selbst geschriebene Scripte mit Firewall-Konfigurationen oder sonstigen Befehlen nicht an jeden Client einzeln schicken muss. Unter Einhaltung einiger Vorgaben kann der Benutzer ein komplettes Script, das alle Clients des Netzwerkes betrifft, als Ganzes ausführen, und es dann automatisch an die jeweiligen Clients (Knoten im virtuellen Netz) verteilen.

Zu guter Letzt soll dem Benutzer ein Abfragetool an die Hand gegeben werden, mit welchem er bestimmte Firewall-Einstellungen überprüfen kann. Um dies genauer zu erklären, nehmen wir als Beispiel ein Seminar an der Universität oder Hochschule. Die Aufgabenstellung ist gegeben: Es ist ein Netzwerk einzurichten. Die einzelnen Clients sollen bestimmte Kriterien erfüllen. Ein Client soll als stateful Firewall dienen, ein anderer als Workstation etc. Die Lehrkraft lässt seine Lösung der Aufgabe auf den Clients ausführen und kann daraus ein Script generieren, welches die eingespeisten Regeln abfragt. Die Studenten, die diese Aufgabe nun bearbeiten, schreiben ihr Firewall-Script und führen dieses innerhalb der Netzwerkvirtualisierung aus. Die Lehrkraft muss nun nicht mehr jeden einzelnen Befehl und jede Einstellung des Scripts des Studenten überprüfen. Er kann sein Abfragescript, das vorher

erstellt wurde, ausführen und somit das Script des Studenten auf Korrektheit überprüfen. Dadurch wird Zeit gespart, da pro Bearbeiter der Aufgabe nur noch das Abfragescript ausgeführt und nicht mehr jedes einzelne Script analysiert werden muss.

Zusätzlich zu diesen drei Bereichen gibt es noch die Idee, dem Benutzer ein Tool an die Hand zu geben. Mit diesem Tool sollen Firewall-Einstellungen (zum Beispiel eine iptables-Regel) nicht mehr eingeben werden, sondern (durch Zusammenstellen bestimmter Teilbereiche eines Befehls) über eine GUI erstellt werden. Gleichzeitig soll sichergestellt werden, dass die Regel semantisch korrekt ist und beim ausführen der Regel auf einem Computer oder Netzwerkclient kein Fehler auftritt.

3 Stand der Technik

3.1 Virtuelle Konzepte

Die Virtualisierung bezeichnet Methoden, die es erlauben, Ressourcen eines Computers aufzuteilen, wohingegen Emulations-Programme Schritt für Schritt die Hardware emulieren. Bei einer Emulation müssen Systemaufrufe immer durch eine zusätzliche Ebene, den Emulator, laufen, so dass die Geschwindigkeit darunter leidet. Dafür ist es aber plattformunabhängig. Virtualisierungsmethoden sind in der Regel betriebssystemunabhängig, aber nur auf spezieller Hardware lauffähig. User-Mode-Linux und VMWare zum Beispiel sind nur auf x86-Architekturen einsetzbar. Weiterhin ist die Performance nicht optimal wenn mehrere virtuelle Maschinen laufen.

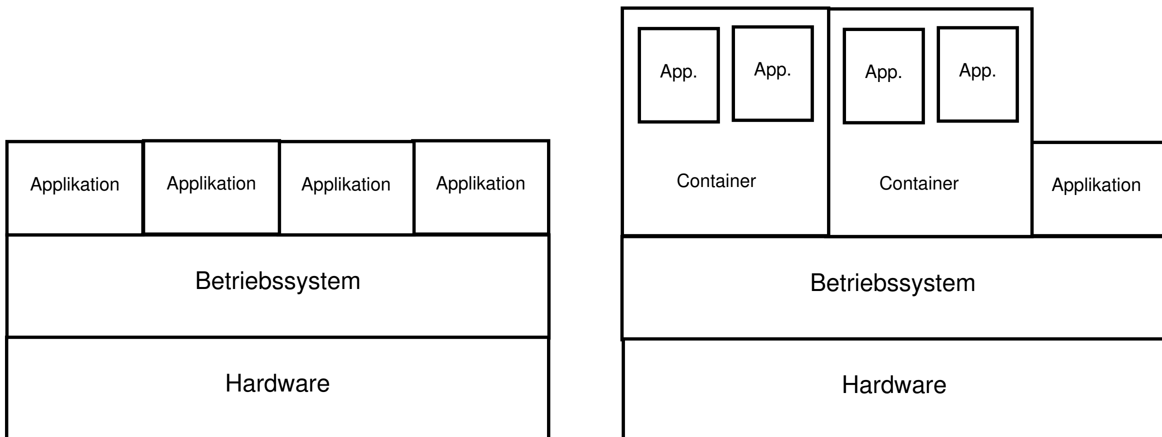

Abbildung 3.1: *Betriebssystemvirtualisierung mittels OS-Container*

Die Konzepte sind außerdem nach der Realisierung der Virtualisierung zu unterscheiden. Zum einen gibt es Betriebssystemvirtualisierungen, wo dem virtuellem Betriebssystem eine komplette Laufzeitumgebung innerhalb eines geschlossenen Container zugewiesen wird (siehe Abbildung 3.1). Es läuft hier nur ein Host-Kernel. *User-Mode-Linux* ist ein Beispiel hierfür.

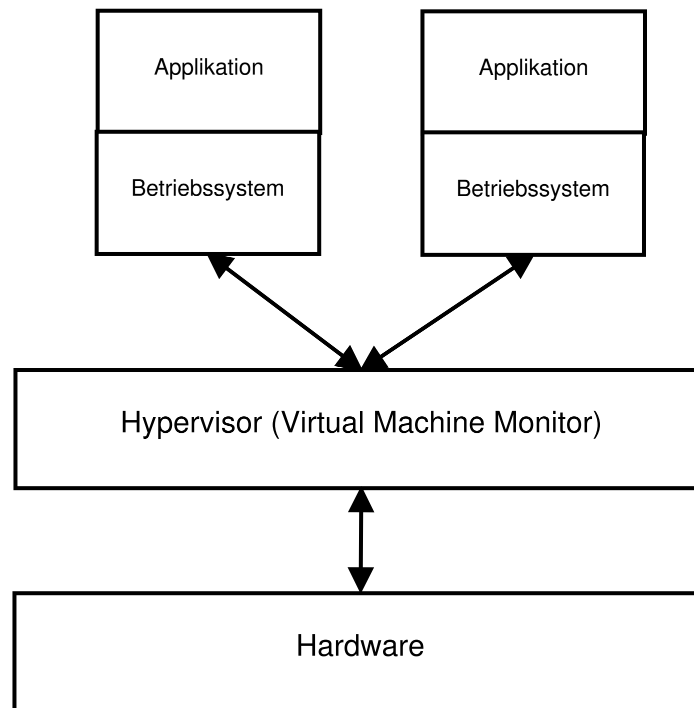

Abbildung 3.2: *Systemvirtualisierung mittels Hypervisor*

Zum anderen gibt es Systemvirtualisierungen (Abbildung 3.2), bei denen die bereitstehenden reellen Ressourcen mittels eines Hypervisors intelligent verteilt werden. Der Vorteil hier ist, dass die Gastsysteme alle einen eigenen Kernel laufen haben und es dadurch etwas flexibler ist als eine Betriebssystemvirtualisierung. Beispiele: *VMWare*, *MS Virtual PC* und *Virtual Box*.

3.1.1 VMWare

VMWare (VMWare) ist ein amerikanisches Unternehmen, das zahlreiche Software-Programme zum Thema Virtualisierung herausgegeben hat. Im Prinzip wird für die Virtualisierung nur eines dieser Programme (zum Beispiel VMWare Workstation, VMWare ESX etc.) benötigt. Nach der Installation kann innerhalb des Programms dann ein beliebiges Betriebssystem (oder mehrere) aufgesetzt werden. Diese virtuellen Systeme funktionieren genau wie physische Systeme. Man kann sie ins Netzwerk einbinden und an oder ausschalten. Dabei gehen geänderte Daten oder Systemeigenschaften gehen nicht verloren. Das erstellte virtuelle System bekommt auf dem physischen Hostsystem Festplattenspeicher in Form einer Datei zugesagt. Je nach Einstellung passt diese sich dynamisch an das Wachstum an oder ihre Größe wird beim erstellen festgelegt. Außerdem kann eingestellt werden,

wieviel Arbeitsspeicher das erstelle System besitzen soll. Dieses Gastsystem ist einfach auf andere Wirtsysteme portierbar, indem die vmdk-Datei (die Datei in der alle Einstellungen sowie das Betriebssystem gespeichert sind) kopiert oder verschoben wird. Voraussetzung ist, dass eines der VMWare Programme auf dem neuen Host installiert ist. Mit dem VMWare Konverter lassen sich zudem auch laufende Betriebssysteme virtualisieren. Weiterhin gibt es den VMWare Player, mit dem erstellte VMs geöffnet werden können. Allerdings dient dieser nicht als Server für mehrere virtuelle Betriebssysteme, sondern wie der Name schon sagt, als Player.

3.1.2 Virtual Box

Virtual Box (VirtualBox) ist eine Virtualisierungssoftware die einen x86-Prozessor, beziehungsweise einen x64-Prozessor, in einer virtuellen Umgebung simuliert. Daher sind auch nur x86-/x64-Architekturen sowohl als Gast- wie auch Wirtsystem einsetzbar. Die Simulation erstellt eine VM, in der der Prozessor dem im Wirt tatsächlich eingebauten Prozessor entspricht. Er wird also nicht vollständig simuliert.

3.1.3 Microsoft Virtual PC

Virtual PC (Microsoft, 2010) wurde eigentlich entwickelt, um x86-Betriebssyteme auf Mac-OS zu simulieren. Microsoft übernahm dieses Programm später vom eigentlichen Gründer Connectix. Seit Apple allerdings die x86-Architektur selber einsetzt, wird die Software für Mac-OS nicht mehr weiterentwickelt und ist derzeit kostenlos von Microsoft erhältlich. Mit Virtual PC wird eine Maschine virtualisiert, in der ein Standard-Betriebssystem ablaufen kann. Linux-Distributionen werden eigentlich nicht unterstützt, lassen sich meist aber ohne größere Probleme auch virtualisieren. Gastsysteme sind nur als 32-bit Systeme vorgesehen, 64-bit Systeme werden nicht unterstützt.

3.1.4 UML - User Mode Linux

User-Mode-Linux (Dike, Jeff, 2006) erlaubt es, Linux-Kernel als Anwendungsprozess laufen zu lassen. Das heißt, es setzt nicht direkt auf die Hardware auf, sondern arbeitet als Prozess in einem Wirtsystem. In Abbildung 3.1 ist dieser Ansatz der Betriebssystemvirtualisierung auf der rechten Seite dargestellt. UML wird eine komplette Laufzeitumgebung innerhalb eines geschlossenen Containers zur Verfügung gestellt. Auf der linken Seite sieht man ein normales Betriebsystem, das Applikationen als Prozesse startet und direkt auf der Hardware aufsetzt.

User-Mode-Linux hat bei der Betriebssystemvirtualisierung allerdings eine Sonderrolle. Wie in Abschnitt 3.1 erwähnt, läuft normalerweise immer nur ein Host-Kernel. Bei UML jedoch laufen spezielle User-Mode-Kernel unter der Kontrolle des Host-Kernels. Daher wird UML des öfteren auch der Paravirtualisierung zugeordnet, bei der ein zusätzliches Betriebssystem zwar neu gestartet, aber keine zusätzliche Hardware virtualisiert oder emuliert wird.

Vor allem für Testzwecke (Treiberentwicklung, Netzwerke etc) ist UML (nicht zu verwechseln mit Unified Modeling Language) von Vorteil, da bei einem Kernel Fehler des virtuellen Kernels das eigentlich Linuxsystem nicht betroffen ist Es kann einfach ein neuer virtueller Kernel gestartet werden. Ebenfalls können mehrere Kernels gleichzeitig gestartet werden, um z.B. ein Netzwerk aufzubauen und dort Sicherheitseinstellungen zu testen.

Zuerst sollte UML Linux-on-Linux genannt werden, dies wurde aber verworfen um Verwechslungen mit dem Akronym LOL vorzubeugen.

Der Vorteil von UML gegenüber anderen virtuellen Konzepten wie VMWare oder Virtual Box, ist die Unabhängigkeit der Grafik. Es basiert allein auf Consolen. Da das Hauptsystem nicht von Fehlern betroffen ist, die im UML passieren, wird es unter anderem dazu genutzt, neue Kernel-Versionen zu testen. Ein weiterer Vorteil ist, dass mehrere Distributionen gleichzeitig laufen können, der UML-Kernel ohne root-Rechte gestartet werden kann und das UML als 'Spielwiese' einsetzbar ist, in der beispielsweise neue Applikationen getestet werden.

Anwendungsszenarien für User-Mode-Linux (User-Mode-Linux):
'UML can run basically anything that the host can, so its possible to split physical systems into a bunch of independent virtual machines' (User-Mode-Linux, What are people using it for). Eigentlich wurde UML entwickelt, um Kernel und Kerneländerungen zu überprüfen und zu testen. Allerdings wurde schnell deutlich, dass dieses Konzept mit einigen Veränderungen noch viel mehr bieten kann. Heute wird es unter anderem zum Testen neuer Software/Applikationen benutzt. Auch für virtuelles Hosting ist UML geeignet, da es sich um einen vollwertigen Kernel handelt und es Hostingprovidern ermöglicht ein physisches System in mehrere unabhängige virtuelle Maschinen aufzuteilen. Es wird damit eine bessere Hardwareauslastung erzielt. Weiterhin wird es als Honeypot genutzt. Honeypot ist ein System, das bewusst angreifbar gemacht wird, um Auskunft über die Strategie der Angreifer zu bekommen. Es ist möglich, eine Vielzahl an Honeypots auf einem System laufen zu lassen und dabei noch ein virtuelles Netz zu simulieren. Zu guter Letzt dient UML natürlich auch der Netzwerksimulation. Es können Topologien nachgebildet und untersucht werden.

3.2 Firewall

'Das englische Wort Firewall lässt sich mit Brandschutzmauer übersetzen' (Barth, Wolfgang, 2001, S. 5)

Unter einer Firewall kann man sich - dem englischen Wort entsprechend - am besten eine Art Mauer vorstellen, die Feuer fernhält. Solang es nicht brennt, sind die Brandschutztüren geöffnet, werden bei Gefahr aber geschlossen. So ähnlich läuft es bei einer Firewall auch ab. Allerdings gibt es einen hundertprozentigen Schutz vor Gefahren aus dem Internet nur, wenn keine Verbindung besteht.

Das Ziel im Internet ist es zumeist, Daten miteinander auszutauschen. Es soll aber auch gleichzeitig sichergestellt werden, dass nur geforderte Anwendungen und der zugehörige Datenaustausch innerhalb dieser Anwendungen stattfindet. Andere Daten sollen nicht ins eigene Netzwerk gelangen oder dieses verlassen. Eine Firewall trennt den privaten vom öffentlichen Bereich ab, sorgt also für bedingte Durchlässigkeit. Basierend auf den Absender-/Zieladressen und genutzten Diensten dient sie dazu den Netzwerkzugriff zu beschränken. Der durch die Firewall laufende Datenverkehr wird überwacht und anhand festgelegter Regeln wird entschieden, ob bestimmte Netzwerkpakete durchgelassen werden oder nicht. Auf diese Weise versucht die Firewall unerlaubte Netzwerkzugriffe zu unterbinden. Dort wo der Zugang gestattet ist, überprüft die Firewall die ankommenden und ausgehenden Pakete. Ist kein Zugang notwendig, riegelt sie den Zugang komplett ab.

Eine Firewall ist also ein Konzept für die Verbindung zwischen einem öffentlichen und einem nicht-öffentlichen Bereich. Sie kann aus nur einem Stück Hardware bestehen oder aber auch aus einer ganzen Infrastruktur von Servern, Routern etc.

Die Firewall ist allerdings nicht dafür zuständig, Angriffe zu erkennen. Sie setzt Regeln für die Kommunikation um. Das Intrusion Detection System (IDS) dient zum erkennen von Angriffen. Dieses lässt sich zwar auf eine Firewall aufsetzen, gehört aber nicht zum eigentlichen Firewallmodul.

Es gibt zwei verschiedene Arten von Firewalls:

Personal Firewall Eine personal Firewall ist eine lokal auf dem Computer installierte Firewallsoftware. Ihre Aufgabe ist es, Zugriffe von außerhalb zu unterbinden. Je nach Produkt kann sie auch Anwendungen daran hindern, mit der Außenwelt zu kommunizieren, bevor der Benutzer nicht sein Einverständnis dafür gegeben hat. Auf Heimrechnern wird oft eine personal Firewall eingesetzt.

Externe Firewall Eine externe Firewall beschränkt die Verbindung zwischen zwei Netzen. Sie werden oft auch missverständlich als Hardwarefirewall bezeichnet. Der Begriff wird aber nur dazu verwendet deutlich zu machen, das es sich um eine separate Hardware handelt, auf der die Firewall läuft. Eine Firewall ist immer eine Software.

Ein Computer, der direkt am Internet angeschlossen ist und auf dem eine personal Firewall läuft, hat das Problem, das aus dem Internet heraus auf die Dienste des Computers zugegriffen werden kann, was einen Fernzugriff ermöglicht. Eine externe Firewall kann dies ausschließen, indem sie auf der einen Seite an das Internet angeschlossen wird und auf

der anderen Seite an das private Netz. Anfragen aus dem privaten Netz werden nun nicht direkt in das Internet geschickt, sondern an die externe Firewall, die diese weiterleitet. Antworten aus Internet werden von der externen Firewall dann wieder an den entsprechenden Computer im internen Netz weitergeleitet. Dadurch ist es auch möglich, Netzwerkpakete zu analysieren, zu filtern und zu überprüfen bevor sie das Ziel bzw. den Kommunikationspartner erreichen.

3.3 iptables

Wie in Abschnitt 3.2 bereits erläutert, ist eine Firewall zum einen ein Konzept um einen Computer oder ein Netzwerk vor Angriffen zu schützen. Zum anderen ist es eine Bezeichnung für eine Soft- oder Hardwarelösung, die einen Teil dieser Aufgaben durch Paketfilterung löst.

Iptables ist der Befehl, der das Linux-Kernel-Subsystem zur Verarbeitung von Netzwerkpaketen – genannt Netfilter - konfiguriert. Da iptables Systemprivilegien benötigt, muss es mit root-Rechten ausgeführt werden.

Diese Paketfilterung unter Linux erfolgt bereits im Kernel selbst. Firewallsoftware hingegen, welche als normales Programm oberhalb der Betriebssystemebene arbeitet, kann noch so sicher implementiert sein. Wenn das Betriebssystem bereits Fehler aufweist, bleibt es trotz dieser Software angreifbar. Daher hat die Paketfilterung unter Linux im Vergleich zu einer Firewallsoftware dort einen Vorteil. Der Nachteil ist aber, dass diese Lösung im Kernel schwer auf andere Betriebssysteme portierbar ist. D 'Die iptables-Architektur gruppiert die Regeln für die Verarbeitung der Netzwerkpaketen gemäß ihrer Funktion in Tabellen. Diese Tabellen enthalten Ketten von Verarbeitungsregeln. Verarbeitungsregeln bestehen aus Mustern und Targets.' (Purdy, Gregor N., 2005, S. 5)
Darauf wird im folgenden näher eingegangen.

3.3.1 Konzepte

Es gibt fünft Einstiegspunkte - die sogenannten Hooks - in der Paketverarbeitung des Kernels:

- PREROUTING: Bevor eine Routingentscheidung getroffen wird, landen die Pakete in dieser Kette.
- INPUT: Das Paket wird lokal zugestellt.
- FORWARD: Alle Pakete, die geroutet und nicht lokal zugestellt wurden, passieren diese Kette.

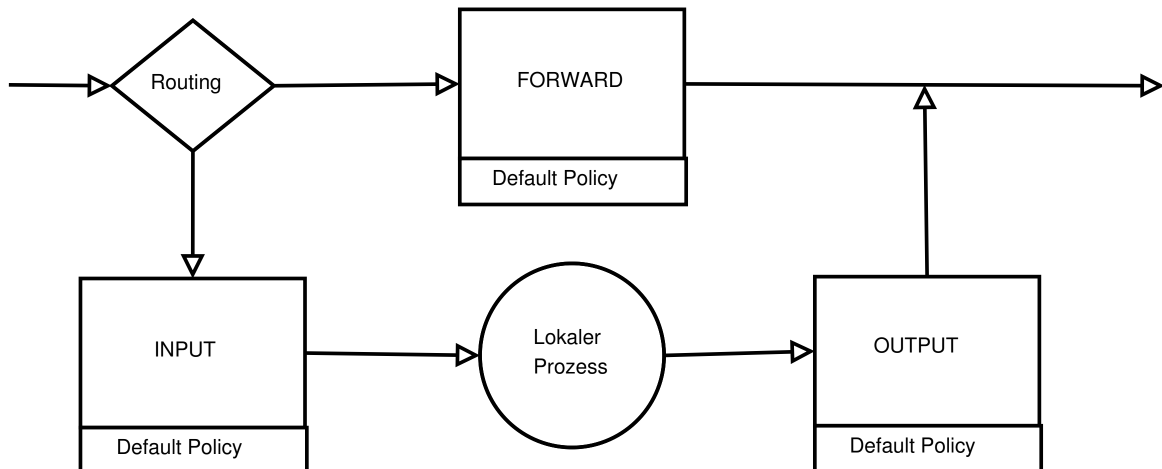

Abbildung 3.3: *Beispiel des Paketflusses und Hooks der Filtertabelle*

- OUTPUT: Pakete, die vom eigenen Computer erzeugt wurden, tauchen hier auf.
- POSTROUTING: Die Routingentscheidung wurde getroffen. Bevor die Pakete an die Hardware weitergegeben werden, durchlaufen sie noch einmal die Kette

Ketten werden in diese Hooks eingebunden und in jeden Hook kann man eine Folge von Regeln einfügen. Die Regeln bieten Möglichkeiten zur Beeinflussung und Überwachung des Paketflusses.

3.3.2 Tabellen

In iptables gibt es drei fest eingebaute Tabellen:

- filter: legt fest, welche Art von Traffic in, durch und aus dem Computer laufen darf. Wenn keine andere Tabelle angegeben ist, wird standardmäßig mit dieser Tabelle gearbeitet. Die fest eingebauten Ketten sind FORWARD, INPUT, OUTPUT. In Abbildung 3.3 ist als Beispiel der Paketfluss der Filtertabelle aufgeführt. Ankommende Pakete werden durch die Input-Kette an einen lokalen Prozess geleitet, sofern das Paket dafür bestimmt ist, ansonsten gelangt es durch die Forward-Kette und wird von dort weitergeleitet. Ein Paket verlässt einen lokalen Prozess durch die Output-Kette.
- nat: die nat-Tabelle wird zur Umleitung von Verbindungen bei NAT (Network Address Translation) genutzt. Die fest eingebauten Ketten sind OUTPUT, POSTROUTING, PREROUTING.

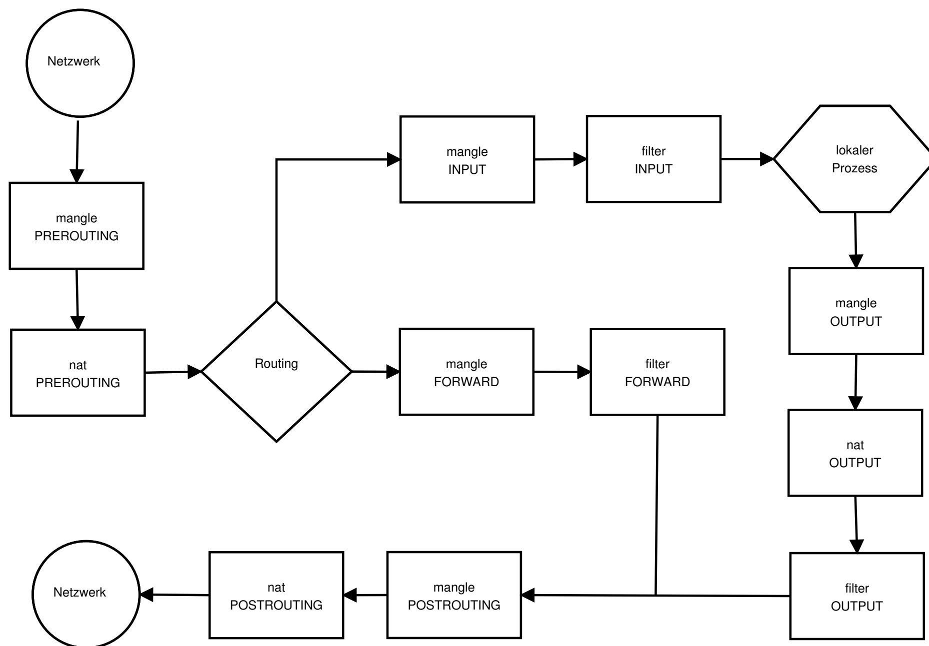

Abbildung 3.4: *Weg eines Paketes durch die iptables-Regeln*

- mangle: dient zur Paket-Manipulation. Die fest eingebauten Ketten sind INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING

3.3.3 Paketfluss

Wie in der Abbildung 3.4 zu erkennen, passiert ein Paket noch vor dem Routing die PREROUTING-Ketten, und zwar in der Reihenfolge, wie sie in der Abbildung ersichtlich ist. Danach wird vom Routing entschieden, ob das Paket für einen lokalen Dienst bestimmt ist (INPUT-, OUTPUT-Kette), oder weitergeleitet wird (FORWARD-Kette). Hier ist zu sehen, das ein weitergeleitetes Paket nicht die Regeln der INPUT- sowie OUTPUT-Kette durchläuft. Zu guter Letzt durchläuft das Paket die POSTROUTING-Kette. Regeln in der PREROUTING- und POSTROUTING-Kette werden auf alle Pakete angewandt. Für jede Kette kann es mehrere Regeln geben, die in einer Liste gespeichert sind und nacheinander durchlaufen werden. Ein Paket durchläuft jede Regel, bis eine davon auf das Paket zutrifft. Die zugehörige Aktion wird dann ausgeführt und bis auf wenige Ausnahmen wird die Abarbeitung weiterer Regeln abgebrochen. Trifft keine der Regeln auf das Paket zu, wird die Default Policy ausgeführt. Diese trifft auf alle Pakete zu, die nach Abarbeitung aller Regeln noch in der Kette sind. (siehe Abbildung 3.5). Die Default Policy erleichtert die Umsetzung der eigenen Sicherheitspolitik. In den meisten Fällen wird man nur wenige Verbindungen erlauben, die man mit den Filterregeln gezielt setzt. Den Rest verbietet man über die Default Policy.

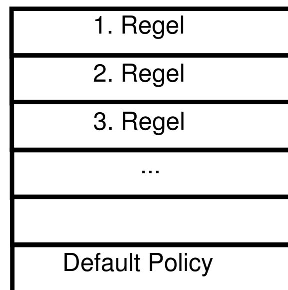

Abbildung 3.5: *Ein Paket wird durch die Regeln einer Kette geprüft. Diese Regeln werden nach und nach abgearbeitet. Trifft keine der Regeln zu, tritt die default Policy in Kraft*

3.3.4 Regeln

Eine iptables-Regel besteht im Grunde aus mehreren Match-Kriterien und einem Target (Ziel). Durch sie wird bestimmt, was mit den betroffenen Paketen geschieht. Alle Match-Kriterien müssen erfüllt sein, damit die Regel auf das Paket zutrifft. Um zu bestimmen, wie das Paket beeinflusst wird, muss das Target festgelegt sein. Es gibt einen Paket- und Bytezähler und sofern eine Regel zutrifft, wird der Paketzähler inkrementiert und der Bytezähler um die Größe des Pakets erhöht. Beide Teile (Match und Target) sind optional. Alle Pakete sind zutreffend, sofern keine Match-Option vorhanden ist. Ohne Target passiert nichts mit den Paketen. Es geht weiter, als gäbe es diese Regel nicht. Nur die beiden Zähler werden hochgezählt.

3.3.5 Targets

- ACCEPT: Beendet die Verarbeitung der aktuellen Kette und lässt das Paket durch.
- DROP: Beendet die Verarbeitung des Pakets und kümmert sich nicht um weitere Regeln oder Ketten. Es wird keine Rückmeldung gesendet (man sollte REJECT nutzen um eine Rückmeldung an den Sender zu geben).
- QUEUE: Sendet das Paket an den Userspace.
- RETURN: Die Verarbeitung des Pakets wird an die aufrufende Kette weitergegeben.

3.4 Hilfsprogramme

3.4.1 Allgemein

Es gibt zahlreiche Hilfsprogramme um getätigte Einstellungen bezüglich der iptables-Regeln zu testen und zu überprüfen. Einige davon sind im folgenden näher erläutert.

nmap Nmap (Network Mapper) (vgl. NMAP, 1997) wird in erster Linie für das Portscanning genutzt. Es dient zum Scannen und Auswerten von Hosts in einem Computernetzwerk. Mit 'nmap 192.168.1.3' lassen sich zum Beispiel alle Ports des Computers mit dieser IP scannen. Nachdem dieser Scan durchgeführt wurde, wird eine Liste der Ports angezeigt die derzeit offen sind. Mit 'nmap 192.168.1.3 -p 21' lässt sich ein Port gezielt scannen (hier Port 21 auf dem Computer mit der IP-Adresse 192.168.1.3).

ping Mit Ping wird überprüft, ob ein bestimmter Host in einem IP-Netzwerk erreichbar ist. Zudem gibt Ping die Zeitspanne zwischen dem Senden eines Paketes zu einem Zielrechner und dem Empfang des Antwortpaketes an. Dies nennt man RTT (Round-Trip-Time), also die 'Rundreise' eines Paketes vom Host zum Ziel und wieder zurück. Genauer gesagt versendet Ping einen icmp echo-request (ping) an die Adresse des zu überprüfenden Hosts. Dieser wird dann als Antwort einen icmp echo-reply (pong) zurücksenden, sofern er erreichbar ist. Ist dies nicht der Fall, kommt als Antwort 'Network unreachable' oder 'Host unreachable' zurück. Anstatt der IP-Adresse des Zielhosts kann auch FQDN (Fully Qualified Domain Name, z.B. www.beispiel.de) angegeben werden. Das DNS (Domain Name System) kann diesen Namen dann in eine IP-Adresse auflösen.

traceroute Mit Traceroute kann ermittelt werden, über welche IP-Router Datenpakete zu einem Host vermittelt werden. Das Programm schickt mehrere Pakete mit einem veränderten und jeweils um 1 erhöhten Time-To-Live (TTL) Wert an einen bestimmten Host im Netz. Die Router reduzieren diesen Wert jeweils um 1 und der Router, der den Wert am Ende von 1 auf 0 reduziert, schickt eine Antwort zurück. So wird nach und nach eine Sequenz aufgebaut, an der zu erkennen ist, über welche HOPs (Übergang von einem zu einem anderen Netzknoten) der Hostrechner mit dem Zielrechner kommuniziert.

tcpdump tcpdump (vgl. tcpdump, 2010) liest Paketdaten ein, die über ein Netzwerk gesendet wurden und stellt diese auf dem Bildschirm dar. Haupteinsatzgebiete sind die Aufzeichnung/Darstellung der Kommunikation zwischen verschiedenen Teilnehmern eines Netzwerkes, sowie die Fehlersuche und -analyse von Programmen die über das Netzwerk kommunizieren.

3.4.2 Hilfsprogramme für iptables

Es gibt derzeit vielerlei Hilfsprogramme für iptables, die dem User allerdings zumeist eine vorgegebene Auswahl an Befehlen geben und per Checkbox nur noch angehakt werden müssen (vgl. (Bauer, 2009) oder (Mandt, 2007)). Der eigentliche Befehl dahinter bleibt dem User zuerst einmal unklar. Es werden nur Haken in einer Checkbox, eine Auswahl in einer Combobox etc. gesetzt und das Programm generiert daraus ein Firewall-Script. Gerade wenn es um Portsperren geht, gibt es aber nur eine beschränkte Auswahl vordefinierter Ports die anzuklicken sind. Zumeist sind es die bekannteren Standartports (http/80, https/443, ftp/21, ssh/22 etc).

- meist kleiner Überblick über eine überschaubare Menge von Befehlen
- diese stehen per Checkbox zur Auswahl
- aus den zusammengeklickten Befehlen wird ein Script generiert

3.4.3 iptables-testing

Es gibt einige Versuche, die sich mit dem Problem der fehlerhaften Konfiguration einer Firewall beschäftigen und es lösen wollen. Dabei gibt es zwei unterschiedliche Ansätze: das aktive und das passive Testen. Das passive Testen besteht aus offline Analysen der Firewallkonfiguration. Das aktive Testen wird mit Tools wie Nessus (Nessus, 2010) und den in Abschnitt 3.4.1 aufgeführten Hilfsprogrammen wie *nmap*, *traceroute* und *ping* realisiert. Diese Tools sind nicht direkt zur Prüfung von iptables-Regeln entwickelt worden, sondern eher um Informationen über das Netzwerk zu sammeln und auszuwerten. Bestimmte Funktionen dieser Tools helfen aber dabei, iptables-Regeln zu überprüfen. Umfangreiche Tools, wie zum Beispiel Nessus, sind aber nicht Open-Source. Mit ITVal (Marmorstein und Kearns, 2005) wurde ein Open-Source-Tool entwickelt, das Firewalls analysiert. Als Input nimmt es die gesetzten iptables-Regeln (mit Hilfe des Befehls 'iptables -L') und ein Query-File. Daraus werden Pakete generiert, die die Firewall akzeptiert. Diese werden danach mit den Querys abgefragt. Das heisst, es überprüft alle vorhandenen iptables-Regeln, die zum Zeitpunkt in der Liste stehen.

3.5 Netzwerkvirtualisierung

Das wohl bekannteste Programm einer Netzwerkvirtualisierung in Verbindung mit User-Mode-Linux ist das VNUML - Virtual Network User-Mode-Linux (Telematics Engineering Department, 2002). Seit 2002 (development) dient VNUML zum schnellen Definieren und

Testen einer Netzwerkumgebung. Das Netzwerkszenario wird in einem extra entwickelten XML-Format, der VNUML-Language, definiert. Diese XML-Datei wird von einem Perl-Script geparst. Parser sind Programme zur Umwandlung einer beliebigen Eingabe in ein, zur Weiterverarbeitung brauchbares, Format. Nacheinander werden dann die virtuellen Maschinen mit User-Mode-Linux gestartet. Jede dieser virtuellen Maschinen lässt sich dann einzeln konfigurieren und administrieren.
Es wurde auch eine Bachelorarbeit zum Thema "Java-basiertes Managementsystem zur Konfiguration und Steuerung von Routingalgorithmen in virtuellen Netzen"(Assouroko, 2010) geschrieben. In dieser Arbeit wurde ein Werkzeug entwickelt, um Routingalgorithmen in einem virtuellen Netz zu steuern und zu konfigurieren. Die Kommunikation wurde in diesem Projekt mit Java-RMI realisiert.

4 Analyse

Dieses Kapitel erklärt noch einmal grundlegend den Aufbau sowie die Funktionen des Programms. Danach vergleicht es die in Kapitel 3. Stand der Technik aufgelisteten virtuellen Konzepte im Hinblick auf die Einsatzfähigkeit innerhalb der zu erstellenden Anwendung.

Wie schon in Kapitel 2. Anforderung beschrieben, soll ein Netzwerk aufgebaut werden. Dieses Netzwerk kann aus einer beliebigen Anzahl an Knoten bestehen. Diese Knoten können verschiedene Rollen haben. Sie können beispielsweise ein Endrechner, ein Server, ein Router, oder eine Firewall sein. Diese Netzwerkknoten sind untereinander vernetzt. Wie sie vernetzt sind kann der Benutzer in einem Script festlegen. Die Knoten sollen also nicht einzeln per Hand gestartet werden müssen, sondern das komplette virtuelle Netzwerk soll im Hintergrund„ aus einer grafischen Oberfläche heraus erstellt werden. Diese GUI soll als zentrale Verwaltungsstelle dienen, um Benutzern die Interaktion mit dem Netzwerk zu erleichtern. Sie soll Zugriff auf die Knoten erleichtern, indem Einstellungen und Konfigurationen eines oder mehrerer Knoten nicht auf den Consolen Knoten speziell, sondern in der GUI getätigt werden.

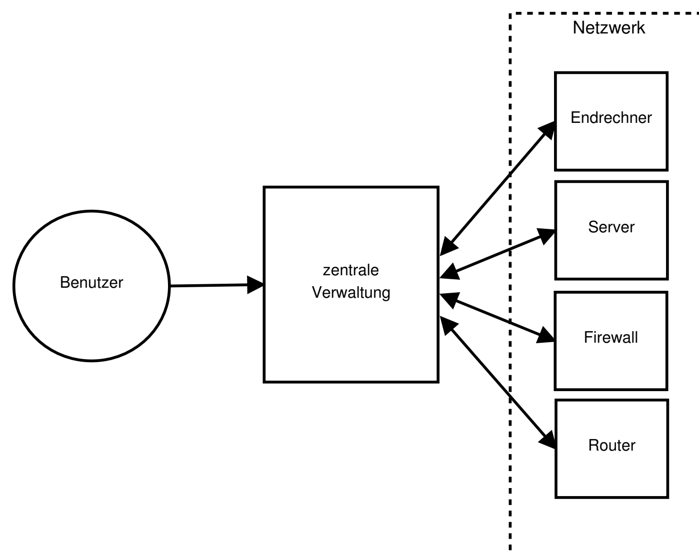

Abbildung 4.1: *Benutzer interagiert über die zentrale Verwaltungseinheit mit den Knoten im Netzwerk*

In Abbildung 4.1 ist die Interaktion zwischen Benutzer und den Netzwerkknoten vereinfacht dargestellt. Der Benutzer soll nicht direkt auf die einzelnen Knoten zugreifen müssen, um sie zu administrieren. Die zentrale Verwaltungseinheit (die grafische Oberfläche) bildet die einzelnen Knoten des Netzes ab, so dass der Benutzer über die GUI mit den Knoten interagiert. Einstellungen, Konfigurationen oder Befehlsaufrufe an das System sollen in der GUI eingestellt, eingeben und ausgeführt werden. Die Eingaben werden dann von der GUI an die einzelnen Knoten des Netzes weitergeleitet. Eventuelle Antworten oder Rückgaben werden wiederum von den Knoten an die GUI geleitet, welche sie dem Benutzer dann darstellt.

Die zentrale Verwaltungseinheit ist dabei kein Teil des Netzwerkes, also kein Knoten. Es soll auf dem Hostsystem laufen und nicht in das Netzwerk mit eingebunden sein. Daher wird das Hostsystem zum Beispiel auch nicht die einzelnen Knoten des virtuellen Netzes pingen können oder sie selber mit den Hilfsprogrammen aus Abschnitt 3.4.1 überprüfen können. Das virtuelle Netz soll völlig losgelöst vom reellen System laufen.

4.1 Virtualisierungskonzept

Die Auswahl des Konzeptes der Virtualisierung wurde unter zwei wichtigen Aspekten vorgenommen: den root-Privilegien und der Darstellung (basierend auf grafischer Aufmachung oder Console). Unter diesen Aspekten wurden unter anderem auch die virtuellen Konzepte aus Kapitel 3.1 betrachtet.

Die meisten bisherigen Konzepte der Virtualisierung haben den Nachteil, dass sie nicht zentral verwaltet werden. Das heißt, es fehlt der Ansatz, die virtuellen Maschinen auf einen Blick zu haben, ohne zwischen ihnen hin und her zu wechseln. Implementiert ist zumeist eine Auflistung aller derzeit vorhandenen virtuellen Maschinen. Es kann sich aber nur eine VM zur Zeit anschauen. Das Ganze macht natürlich Sinn bei grafischer Darstellung. Handelt es sich dabei um mehrere Windows-VMs, ist es durchaus sinnvoll nicht alle VMs auf einmal im Blick zu haben. Darunter würde die Darstellung leiden. Entweder jede einzelne Oberfläche hätte Scrollleisten, so dass nur ein Teil der Oberfläche sichtbar ist, oder - falls die komplette Oberfläche sichtbar sein soll - müsste sie gestaucht werden. Das würde alle Icons extrem verkleinern,sie wären kaum mehr erkennbar. Basiert die VM nur auf Consolen, ist es zwar auch wünschenswert eine möglichst große Console zu haben, aber da nur Text dargestellt wird, ist dies nicht erforderlich. Es lassen sich als Beispiel vier Consolen deutlich besser darstellen, als es vier Windows-Oberflächen täten.

Da die einzelnen Netzwerkknoten wie erwähnt als Firewall, Computer, Server oder Router dienen sollen, und an ihnen nur Konfigurationen bezüglich des Netzwerkes (wie zum Beispiel Routing, Firewalleinstellungen, IP-Adressen) getätigt werden, kann auf grafische Aufarbeitung verzichtet werden. Es ist nur wichtig, das man schnell und unkompliziert diese

Einstellungen vornehmen kann und sie vor allem von einer zentralen Einheit steuerbar und zu tätigen sind.

Sowohl mit VMWare als auch mit Virtual Box lassen sich unterschiedlichste Betriebssystem emulieren. Von Linux, über OS2 bis Windows ist alles dabei. Daher wird in diesen Virtualisierungssoftware auch die Benutzeroberfläche grafisch 1zu1 wie bei einem original Betriebssystem dargestellt, wobei Virtual Box auch ein Konsolenprogramm (VBoxManage) als Frontend zur Bedienung beinhaltet. Bei VMWare lässt sich unter anderem auch der Arbeitsspeicher einer VM festlegen. Um zum Beispiel neuere Windows-Systeme in einer VM einwandfrei laufen zu lassen, sollten schon 2 Gigabyte (GB) RAM dafür bereitgestellt werden. Es kann also zu Problemen kommen, sobald mehr als eine virtuelle Maschine gestartet wird. Der Arbeitsspeicher wird von dem Host-/Wirtsystem gestellt, wenn dann zwei VMs schon 4GB RAM nutzen erkennt man, dass das Hostsystem schon völlig ausgereizt ist und die virtuellen Maschinen nicht mehr flüssig zu bedienen sind (eine durchschnittliche Home-Workstation hat derzeit 4GB Arbeitsspeicher). User-Mode-Linux (UML) kann zwar seit Kernel-Version 2.4 mit einer grafischen Oberfläche gestartet werden, allerdings ist dieses nicht zwingend nötig. Da - wie im Namen schon zu erkennen - UML nur Linux-Kernel als Anwendungsprozesse ausführen kann und nicht wie die beiden anderen Methoden unter anderem Windows, ist es mit UML auch möglich, nur consolenbasiert zu Arbeiten, ohne eine grafische Darstellung zu erhalten.

Da VirtualBox sowie VMWare ein komplettes Betriebssystem virtualisieren und bereitstellen sind root-Rechte erforderlich. Ein User-Mode-Kernel kann ohne root-Rechte gestartet werden. Und auch innerhalb von User-Mode-Linux können das System betreffende Befehle eingegeben werden, die im normalen Linux root-Rechte voraussetzen. Da in Linux alle Einstellungen bezüglich der Netzwerkknoten mit Systembefehlen zu tätigen sind und grafische Anwendungen wegfallen, eignet sich User-Mode-Linux für die Arbeit beziehungsweise für das Erstellen der Anwendung am besten.

Zudem ist es wichtig, dass ein neu gestartetes System funktionsfähig ist. Das heißt, dass es sofort, ohne Fehler, nutzbar ist. Daher sollte ein Netzwerkclient eine Grundkonfiguration haben, die während der Betriebsphase veränderlich ist, aber diese Änderungen beim Beenden der Simulation nicht abspeichert. Damit wird garantiert, dass man beim Starten der Simulation weiß, wie das Grundsystem konfiguriert ist. Mit dem Testen und Konfigurieren kann dann begonnen werden ohne dass die Sorge besteht, dass eventuelle Fehler früherer Einstellungen noch aktiv sind und Teile der Simulation nicht funktionieren. Daher wurde diese Speicherung während des laufenden Betriebs außer Acht gelassen. Möchte man bestimmte Einstellungen dauerhaft verfügbar machen, so dass sie bei einem Neustart weiterhin aktiv sind, muss man diese im Initialisierungsskript eintragen (darauf wird später näher eingegangen).

Wie in Abschnitt 3.5 bereits angedeutet, gab es eine Bachelorarbeit die auch das Thema

Netzwerk und Managementkonsole beinhaltete. Die Kommunikation zwischen den Knoten im virtuellen Netz und dem Server (Wirtrechner) lief über die Java-RMI. Der Wirtrechner hatte direkten Zugang auf das Netzwerk. Das wurde über ein TAP-Device realisiert. Jeder UML-Rechner im virtuellen Netzwerk wurde mit einer seiner simulierten Netzwerkschnittstellen über das TAP-Device an den Wirtrechner angeschlossen. Das heißt der Wirtrecher hatte dadurch Zugriff auf jeden einzelnen simulierten UML-Rechner (Knoten) im Netz. Das Problem dort war die Portierung der Software auf andere Rechner. Bevor die Software voll einsatzfähig war, mussten die einzelnen Adressen der Clients in die hosts-Datei eingetragen werden, ansonsten war RMI (Remote Method Invocation) nicht in der Lage die Kommunikation zu bewerkstelligen. Der Aufwand der RMI-Konfiguration war dort sehr hoch. Ein wichtiger Aspekt sollte jedoch genau die Möglichkeit sein, die Software ohne große Umstände auf einem anderen Linux-System laufen zu lassen, ohne dass der User noch Voreinstellungen tätigen muss.

4.2 iptables

Wie in der Anforderung beschrieben, sollen Abfragebefehle aus Firewall-Einstellungen generiert werden. In Linux gibt es für diese Einstellungen den Befehl *iptables* (siehe Abschnitt 3.3). Iptables konfiguriert die Verarbeitung von Netzwerkpaketen. Diese Abfragebefehle sollen zudem später auch ausgewertet werden. Es gibt vielerlei Hilfsprogramme, die dabei helfen, bestimmte iptables-Regeln zu überprüfen. Ein Großteil davon ist im Abschnitt 3.4.1 aufgelistet und erläutert. Um iptables-Regeln auf ihre Funktionalität testen zu können, braucht man diese Hilfsprogramme. Zwar ist eine Auflistung der Regeln mit dem Befehl 'iptables -L' möglich, allerdings zählt diese Auflistung nur für die derzeit aktiven Regeln. Das Programm soll aber so erweitert werden, dass es auch möglich ist, zu einer gegebenen Aufgabestellung ein Abfragescript der ausgeführten iptables-Regeln zu erstellen, dieses abzuspeichern und es später dann als Abfragescript für ein - von einem anderen Benutzer - erstelltes iptables-Script zu nutzen.

Die Idee hinter dem Generieren von Abfragebefehlen ist, dass es zu einer iptables-Regel einen Befehl gibt, der diese Regel überprüft. Problematisch ist nur, dass es eine Vielzahl von iptables-Regeln gibt und diese sich nicht zusammenfassend überprüfen lassen. Das heißt, dass für jede iptables-Regel ein Abfragebefehl gefunden werden muss. Damit wird eine Liste der Abfragebefehle genau so groß wie die Liste aller iptables-Regeln.

Die Funktion wird im Folgenden an einem Beispiel verdeutlicht. Es wird gezeigt, wie die iptables-Befehle in Abfragebefehle umgewandelt und ausgewertet werden können und welche Voraussetzung gegeben sein müssen.

Knoten: *r1* mit der IP: *172.16.101.1*

Befehl: *iptables -A INPUT -p ICMP -j DROP*

Diese Regel wird an die Inputkette angehängt, daher nur auf eingehende Pakete angewandt. Falls das Paket ein icmp-type Paket ist (trifft zu, da –protocol ICMP), wird dies ignoriert bzw. verworfen. Diese Regel bzw. dieser Befehl wird über die GUI direkt auf einem Knoten ausgeführt. Entweder kann das direkt über das Textfenster des Knotens geschehen (über die simulierte Befehlszeile) oder im iptables-Script-Textfenster.

Generieren: *r1* ist durch die oben genannte Regel nicht pingbar. Es wird ein Knoten gesucht, der mit dem *r1* verbunden ist und von diesem wird der Ping-Befehl ausgeführt. Die Reihenfolge um diese Regel zu überprüfen läuft dann wie folgt ab:

1. Aus dem Netzwerk werden die am Knoten *r1* angeschlossenen Teilnetze herausgefiltert.
2. Diese Teilnetze werden nach einem weiteren angeschlossenen Knoten durchsucht.
3. Zurückgegeben wird eine Liste von Knoten, die über ein Subnetz mit *r1* verbunden sind. (als Beispiel nehmen wir den Knoten *r2* der mit *r1* verbunden ist)
4. Generierter Befehl: *r2 ping -c4 172.16.101.1*

Der ping auf die IP *172.16.101.1* wird also auf dem Knoten *r2*, der direkt mit *r1* verbunden ist ausgeführt. Unter Windows wurde der Standard gesetzt, ping viermal auszuführen. Ping -c4 ist das äquivalent unter Linux dazu, da ohne den Counter sonst unendlich oft eine icmp-Anfrage versendet wird (dieses wiederum ist äquivalent zum Windows Befehl ping -t [IP-Adresse]). Da r1 dank der Regel nicht erreichbar ist, ist es nicht möglich diesen Knoten zu pingen.
Hinzu kommt noch, das bei Abfragebefehlen zusätzlich eine Erwartung mitgeliefert wird. Soll eine Regel ping sperren, also ein Zielrechner nicht pingbar sein, ist die Erwartung, dass der ping fehlschlägt ('fail'). Soll er erreichbar sein wird 'ok' als Erwartung mitgeliefert. Der generierte Befehl lautet also eigentlich *r2 fail ping -c4 172.16.101.1*. Wird der Befehl ausgeführt, wird von Knoten *r2* der Befehl *ping -c4 172.16.101.1* gesendet und es wird erwartet das keine Antwort zurückkommt. Der Knotenname *r2* taucht in diesem Befehl mit auf, da es eine Reihe von generierten Befehlen geben kann. Damit nachher auch die Befehle an die richtigen Knoten gesendet werden, stehen ihre Namen mit dabei. So wird nachher nur noch die Zeile ausgelesen und der Befehl an den jeweiligen, in der Zeile angegebenen, Knoten gesendet und dort ausgeführt.

Auswertung: Laut Regel wird erwartet, das *r1* nicht pingbar ist. Es findet nun eine Überprüfung des Ping-Befehls statt, die auswertet, ob der Ping auf *r1* erfolgreich war oder nicht. Dann wird die Erwartung (im obigen Beispiel soll ping nicht erfolgreich sein, also 'fail') mit dem tatsächlichen Ergebnis überprüft. Stimmt die Erwartung mit dem Ergebnis überein, war

der Test erfolgreich, ansonsten erscheint eine Fehlermeldung.

Dieses Beispiel dient als grundlegende Erklärung worauf dieser Teil des Programms abzielt, nämlich auf die Generierung, Auswertung und Überprüfung von iptables-Regeln.

Das obige Beispiel ist an einem Befehl festgemacht. Diese Funktion wäre noch zu erweitern, um komplette Scripte zu übersetzen. Dann würde die Aufgabenstellung zum Beispiel lauten, ein iptables-Script zu entwickeln, welches bestimmte Ports sperrt, Ping für einige Knoten erlaubt, etc. Der Aufgabensteller hat für diese Aufgabe die richtige Lösung parat, also sein eigenes Script, das er in ein Abfragescript umwandeln und abspeichern kann. Nun kann die Aufgabe von anderen Benutzern bearbeitet werden. Die Benutzer erstellen ihrerseits ein iptables-Scripts mit ihrer Lösung zur Aufgabe. Um die Richtigkeit der Benutzerscripte zu überprüfen, kann der Aufgabensteller nun die iptables-Scripte der Benutzer in die GUI laden und sie im virtuellen Netz ausführen. Danach läd er sein erstelltes Abfragescript und führt es ebenfalls über die GUI aus. Zum Schluss kann er sich dann die Auswertung anschauen. Dort ist dann zu erkennen ob die Bearbeitung der Aufgabe von dem jeweiligen Benutzer korrekt oder fehlerhaft ist.

Da die iptables-Regeln des Users nicht selber überprüft werden sollen - also keine allgemeine Regelüberprüfung der unter 'iptables -L' aufgelisteten Regeln - sondern eigentlich ein Vergleich zwischen zwei iptables-Scripten passieren soll, sind vorhandene Tools zum Testen nicht nutzbar. Diese überprüfen die zum Zeitpunkt aktiven iptables-Regeln eines Computers. Ansich eine gute Idee, allerdings in diesem Fall nicht hilfreich, da die aktiven iptables-Regeln mit den Regeln verglichen werden sollen, die ein anderer vorher zur selben Aufgabe geschrieben hat.

4.3 iptables-Regeln konfigurieren

Da das Thema iptables ein Teil dieser Bachelorarbeit ist und nicht jeder Benutzer automatisch damit vertraut ist, ist auch kleines Konfigurationstool für iptables enthalten. Im Gegensatz zu den gängigen Tools soll hier aber der Schwerpunkt nicht darauf liegen, das schon aufgeführte Befehle und Regeln per Häkchen setzen ausgewählt werden, sondern das sie Schritt für Schritt - von der Tabelle, über Ketten, Protokolle bis zum Target - entwickelt werden. Keinem Benutzer der sich bisher noch nicht mit der Materie beschäftigt hat zieht einen Nutzen daraus, Häkchen zu setzen, um danach automatisch ein Script zu erhalten. Es dient eher Nutzern, die mit iptables schon einmal gearbeitet haben, und schnell ohne grossen Aufwand ein Script erstellen lassen wollen. Das ist aber nicht der Ansatz der hier verfolgt wird. Der Benutzer soll selbstständig eine iptables-Regel nach und nach aufbauen können.

5 Umsetzung

5.1 Allgemein

5.1.1 Python

Python (vgl Kaiser, Peter und Ernesti, Johannes, 2008) ist eine einfache und übersichtliche Programmiersprache. Sie kommt mit wenigen Schlüsselwörtern aus und die Syntax ist optimiert auf Übersichtlichkeit. Diese Übersicht gewährt Python durch Einrückung als Strukturelement. In anderen Sprachen dient die Einrückung nur der Übersicht, hat aber keinen Einfluss auf den eigentlichen Code. Das ist in Python anders. Hier werden Blöcke nicht wie in anderen Programmiersprachen durch Klammern und Schlüsselwörter markiert. Das Endkennzeichen ist der Zeilenwechsel. Daher entspricht eine logische Zeile einer physischen. Ausnahmen sind hierbei sind zum einen der Backslash am Zeilenende, der eine logische Zeile auf der nächsten physischen Zeile fortsetzt. Zum anderen geklammerte Konstrukte, bei deren Nutzung über mehr als eine Zeile der Backslash entfallen kann. Programmierer werden dadurch zu einer gewissen Ordnung bei der Code-Notation gezwungen. Der Sprachkern und die Grammatik ist klein und überschaubar, daher ist sie relativ schnell zu erlernen.

In Python ist der Datentyp nicht an eine Variable gebunden, sondern an das Objekt (den Wert). Sie werden also dynamisch vergeben. Generell enthalten Variablen hier nur Referenzen auf ein Objekt. Python ist eine Skriptsprache, mögliche Fehler erkennt man daher erst zur Laufzeit, da der Code nicht vor dem Starten kompiliert wird. Python besitzt eine große Standard-Bibliothek sowie eine Vielzahl freier Zusatz-Module. Daher hat es ein großes Einsatz-Spektrum. Viele große Firmen setzen auch Python als Programmiersprache ein, die wohl bekannteste ist Google (Google Inc., 1998). Zum Beispiel wurde YouTube (YouTube, LLC, 2005) fast komplett in Python geschrieben.

Das Zen of Python (Peters, 2004) beschreibt sehr schön die Python-Philosophie:

'Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.

Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
*Although never is often better than *right* now.*
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!'

5.1.2 Netzwerkvirtualisierung

Die Kommunikation zwischen der GUI und dem virtuellen Netzwerkknoten wird - wie später erläutert - mit Scripten realisiert. Auf dem Host läuft ein Script das über serielle Schnittstellen mit einem Script auf einem Knoten kommuniziert. Das Script auf der Knotenseite soll direkt ausführbar sein. Da das Programm die Möglichkeit bieten soll, später noch erweitert zu werden, wurde mit Python eine Programmiersprache gewählt, die schnell verständlich und einfach zu bedienen ist. In Python werden mögliche Fehler erst zur Laufzeit erkannt. Ein Script wird ausgeführt und eventuelle Fehler werden dann erst sichtbar. In Python wird das Programm durch den Benutzer selber getestet, treten Fehler auf, behebt man diese im Code und startet das Script erneut.

Host Als Host wird in dieser Arbeit die GUI mit dem darin enthaltenen Host-Script bezeichnet. Befehle lassen sich von der GUI über das Host-Script an das Client-Script, also den jeweiligen Knoten des Netzwerks schicken.

Knoten Als Knoten werden die einzelnen Knoten des Netzwerks bezeichnet. Sie können wie schon beschrieben mehrere Rollen einnehmen. Auf jedem Knoten läuft ein Client-Script, welches sich mit dem Server-Script des Hosts verbindet und kommuniziert.

5.1.3 Iptables-Regeln

Da es eine Vielzahl von iptables-Regeln gibt, wurde in dieser Arbeit nur ein kleiner Teil davon berücksichtigt sobald es um die Generierung von Abfragen und deren Auswertung geht. Da die Knoten alle mit einem Debian-Linux Filesystem arbeiten, kennen diese natürlich alle Regeln, so dass ein erstelltes iptables-Script auch die komplette Spannweite dieser Befehle beinhaltet und an die Knoten geschickt werden kann. Die Abfrage erstellt aus einer Regel einen Überprüfungsbefehl, so dass zu jeder Regel auch eine Abfrage geschrieben werden muss. Dafür ist der Umfang von iptables aber zu groß.

5.2 Netzwerkvirtualisierung

5.2.1 Die Distribution

Als Distribution wird in dieser Arbeit die Linux-Distribution Debian 5.0 'lenny' stable verwendet. Debian enthält nur freie Software und jede Version hat einen Codenamen, der von Charakteren des Films Toy Story stammt. Unstable Versionen werden immer Sid genannt. Sid ist in Toy Story der Junge, der immer die Spielzeuge kaputt macht.
Die verschiedenen Varianten der Distribution sind:

- Experimental: Vorstufe zur unstable Version, hier werden Änderungen getestet die Auswirkungen auf das gesamte System haben.
- Unstable: Die Hauptarbeit der Entwicklung wird in diese Distribution hochgeladen. Sie wird nie freigegeben. Stattdessen verbreiten sich die Pakete davon ins 'Testing' und danach in ein offizielles Release.
- Stable: Die aktuelle Version die sich mit Ausnahme von Sicherheitsupdates nicht mehr ändert.
- Testing: Sobald eine neue stable Version veröffentlicht wird, sind Testing und Stable identisch. In die Testing Versionen werden allerdings nach und nach Updates und neue Anwendungspakete eingebunden.
- Oldstable: Vorgänger der jeweils aktuellen Stable-Version. Ein Jahr lang gibt es hierfür weiter Sicherheitsupdates.

In der vorliegenden Linux-Distribution sind Programme und Tools schon implementiert, die dabei helfen, die Netzwerkkonfiguration abzufragen und zu testen. Alle im Abschnitt 3.4.1

aufgeführten Tools sind also schon standardmäßig enthalten, beziehungsweise nachinstalliert worden. Wie weitere Programme installiert werden können, ist in der Hilfe in Anhang A.3 erläutert.

5.2.2 copy-on-write (COW)

Um das Programm im Ursprungszustand möglichst klein zu halten, ist nur ein Betriebssystem vorhanden. Wenn aber mehrere Knoten mit dem selben System gebootet werden, werden Änderungen innerhalb einer virtuellen Maschine auch auf allen anderen VMs übernommen. Da der Initialzustand jeder VM gleich sein soll, aber danach alle Änderungen einer bestimmten VM auch nur diese betreffen sollen, wird ein COW-File für jede VM erstellt. Dieses COW-File ist eine Kopie des original Systems, Änderungen innerhalb der VM werden nun nur in dem zu der VM gehörigen COW-File gespeichert. Dadurch entstehen aus einem System mehrere COW-Files, die unabhängig voneinander konfigurierbar sind, ohne gegenseitig Einfluss aufeinander zu haben. Diese COW-Files sind vor dem Starten gelöscht, daher sind alle VMs zunächst einmal gleich (es gibt nur das original Betriebssystem). Beim Starten des virtuellen Netzwerkes werden Kommandozeilenparameter mit übergeben um die Knoten unterscheidbar zu machen. Der Knotenname ist dabei der wichtigste Parameter. Die Idee dahinter ist, dass Einstellungen in einem init-Script in Form einer switch/case-Anweisung getätigt werden. Anhand der Knotennamen kann sich jede VM dann die für sie vorgesehen Konfiguration aus diesem init-Script herausfiltern. Diese werden in das dann erstellte COW-File übernommen. Dadurch hat jede virtuelle Maschine ein COW-File, dass sich durch die vorgenommenen Einstellungen vom original System sowie den anderen COW-Files unterscheidet.

5.2.3 uml-switch

Der uml-switch ist ein Daemon der es ermöglicht, virtuelle Netzwerke aufzubauen. Es ist die softwaremäßige Nachbildung eines Switches. Für jeden Teil innerhalb des virtuellen Ethernet-Netzwerkes ist ein uml-switch zu starten. Dieser läuft als Prozess. Die Prozess-IDs der gestarteten uml-switches werden in einer Datei abgespeichert, um sie später (beim Beenden des Netzes) auch wieder schließen zu können. Ein uml-switch wird wie folgt gestartet:

```
1 uml_switch -unix tmp/networkx
```

Das networkx steht in diesem Fall für den jeweiligen Teil des virtuellen Netzes. Sie werden aus dem Topologie-Script ausgelesen. Gibt es zum Beispiel drei Knoten die drei Endrechner

simulieren (A, B und C), wobei A mit B und B mit C verbunden ist, so würden zwei uml-switches für die zwei Netze (A-B, B-C) gestartet werden. Diese simulieren dort einen Switch, der die Knoten miteinander verbindet, so dass diese untereinander erreichbar sind. Der Daemon hört die Ports der an ihm angeschlossenen Knoten ab und leitet Pakete zwischen den Knoten weiter, welche mit ihm verbunden sind.

5.3 Aufbau

5.3.1 Allgemein

Vorweg wird zuerst das Schema des Aufbaus grob erläutert. Später wird dann auf die einzelnen Komponenten genauer eingegangen. Beim Starten des Programms öffnet sich eine grafische Oberfläche, mit Textfenstern für die Knoten und Scripte. Über das Menü lässt sich das virtuelle Netzwerk starten. Dabei wird ein Start-Script ausgeführt. Es greift auf zwei weitere Scripte zu und filtert aus diesen Scripten Informationen.

Topologiescript Hier enthalten ist die Topologie des zu erstellenden Netzwerks. Wieviele Knoten hat das Netzwerk und an welche Teilnetze sind diese angeschlossen.

Initialisierungsscript Das Initialisierungsscript ist über einen Blockdevice mit dem Knoten verbunden und enthält Grundinformationen über die Konfiguration der Knoten.

Anhand dieser Informationen wird das Netzwerk gestartet. Nun kann von der GUI aus auf die einzelnen Knoten im virtuellen Netz zugegriffen werden. Die Kommunikation läuft über ein Pseudoterminal, an dem der Host und der Knoten angebunden sind. Eingaben und Befehle werden darüber von der GUI aus gezielt an den jeweiligen Knoten im Netz gesendet. Dieser verarbeitet den Befehl und gibt die Antwort oder Rückgabe zurück. Dargestellt werden diese Rückgaben dann wieder in der GUI. Im folgenden werden die einzelnen Komponenten nun genauer erläutert und ihre Funktionen deutlich gemacht.

5.3.2 Start-Script

Das Start-Script greift sowohl auf das Initialisierungsscript als auf das Topologie-Script zu. Sobald aus der GUI das virtuelle Netzwerk gestartet wird, wird das Startscript ausgeführt. Es beinhaltet das Bereitstellen des Knotennamens sowie die Zuordnung der Netzwerke, die an diesen Knoten gebunden sind. Die Informationen hierzu befinden sich im Topologie-Script, und werden dort heraus extrahiert.

```
def evalNetworks():
  l = {}
  for r in topo.ifcs:
    for nets in r:
      if nets[1] not in l:
        l[nets[1]] = 0
      l[nets[1]] += 1
  return l.keys()
```

Für jedes Teilnetz wird ein uml-switch (siehe 5.2.3) erstellt. Die Prozess-IDs dieser uml-switchs werden in einer Datei gespeichert, um sie später auch wieder beenden zu können. Auch die einzelnen Knoten des virtuellen Netzes liest das Startscript aus und startet dann für jeden Knoten ein User-Mode-Linux und mit mehreren Kommandozeilenparametern.

```
args=[progdir + KERNEL, 'ubd0=tmp/cowr%d,%s' % (nr,progdir + FS), 'ubd1=init.sh',
      'mem=60M', 'umid=r%d' % nr, 'name=r%d' % nr, 'umlpath=%s' % umlguipath ]
```

Durch diese Codezeile wird beim Start eines Knotens (zum Beispiel Knoten 1) ein User-Mode-Kernel gestartet, das am Blockdevice ubd0 mit einem COW-File (eine Kopie des Filesystems lenny)und an ubd1 mit dem Initialisierungsscript verbunden ist. Der Kernel erhält 60 MB Memory und hat den Namen r1. Im Umlpath ist der Pfad zum umlgui-Verzeichnis des Rechners enthalten. Dieser sorgt dafür, dass das Programm auf jedem Linux-Rechner läuft. Der Knoten besitzt dadurch eine auslesbare Information darüber, wo sich das Verzeichnis des Programms auf dem Host befindet. Auf diesen Parameter wird beim mounten des umlgui-Verzeichnis in der init.sh zugegriffen. Somit weiß der vom Hostsystem eigentlich getrennte UML-Knoten welches Verzeichnis des Hosts er mittels hostfs mounten kann. Jede virtuelle Maschine benötigt als System eine Image-Datei auf dem Host, welche eine Linux-Distribution erhält. Um das mehrfache Anlegen des Dateisystems zu sparen, wird beim Starten einer UML-Instanz (also eines Knotens) wie beschrieben das COW-File (siehe 5.2.2) angelegt. Dies ermöglicht also den einzelnen Instanzen vom selben Dateisystem zu booten, aber erstellt für jede Instanz ein Kopie. Es hat allerdings den Nachteil das im Initialzustand alle Systeme identisch sind. Dafür wird das Programm im ausgelieferten Zustand schlank gehalten und nicht durch viele Image-Dateien vergrößert. Außerdem lässt es sich einfach pflegen, da die Konfiguration aller VMs an einer zentralen Stelle liegen.

5.3.3 Kill-Script

Wenn in der GUI das virtuelle Netz oder das Programm beendet wird, wird das Kill-Script ausgeführt. Aus dem Topologie-Script (welches im nächsten Kapitel genauer erläutert wird) wird die Anzahl der Knoten ausgelesen und diese werden dann heruntergefahren. Die Prozess-ID der uml-switches werden aus der uml-switch-pids-Datei ausgelesen und nacheinander 'gekillt'. Alle angelegten Links, COW-files, Ressources und Networks werden danach geschlossen und gelöscht. Dadurch wird sichergestellt, dass bei einem Neustart des Programms nicht alte Links oder Files noch vorhanden sind. Das vermeidet Fehler in der Kommunikation, wenn sich der neu gestartete Host mit veralteten Knoten unterhalten will.

Der Code zum Ausführen der erwähnten Löschung wird wie folgt realisiert:

```
1  fs=os.listdir("tmp")
   for f in fs:
3    if f.find( "cow" ) == 0 or f.find( "log" ) == 0 or f.find( "resources" ) == 0
         or f.find( "network" ) == 0 or f.find( "console") == 0:
         os.unlink( "tmp/"+f )
5        print "Delete: tmp/" + f
```

Der Tmp-Ordner (in ihm werden alle Files erstellt) wird eingelesen (Zeile 1) und nach den Dateien durchsucht, die im Code gezeigte Wörter beinhalten (Zeile 3). Falls sie noch vorhanden sind, werde sie gelöscht (Zeile 4).

5.3.4 Topologie-Script

In diesem Script wird die Anzahl der Knoten festgelegt, auf die später über die GUI zugegriffen werden kann. Weiterhin enthält das Script Informationen über die Topologie. Wie ist das Netzwerk aufgebaut und welcher Knoten ist über welchen Teil des Netzes mit anderen Knoten verbunden.

Beim Starten des virtuellen Netzwerkes werden aus dem Topologie-Script die Informationen über die Anzahl der Knoten und Netzwerke ausgelesen.

```
1  r1 = [ ['eth0','network1'],['eth1','network2'] ]
   r2 = [ ['eth0','network1'],['eth1','network3'] ]
3  r3 = [ ['eth0','network2'],['eth1','network3'] ]
   ifcs = [ r1 , r2 , r3 ]
```

So würde das Script aussehen, wenn es drei Knoten gäbe, die alle untereinander vernetzt sind (siehe Abbildung 5.1). Pro Zeile wird hier ein Knoten beschrieben, und definiert an welchem seiner Netzwerkschnittstellen welches Subnetz ist. Der Knoten *r1* hat das *network1* an seiner Netzwerkschnittstelle *eth0* und das *network2* an *eth1* angeschlossen. Die letzte Zeile gibt alle Knoten an, die aktiv im Netzwerk eingebunden werden sollen. Das heißt, nur die angegebenen Knoten in dieser Zeile werden auch später mit dem Netzwerk zusammen gestartet. Es ist also möglich, ein Netz aus mehreren Knoten zu erstellen, aber nicht alle davon zu starten. Wenn man wie im Beispiel nur drei Knoten braucht um bestimmte Einstellungen zu testen, man hat aber mehrere Knoten (z.B. r1-r8) erstellt, kann man wie in Zeile 4 zu sehen nur die Knoten angeben, die auch gestartet werden sollen.

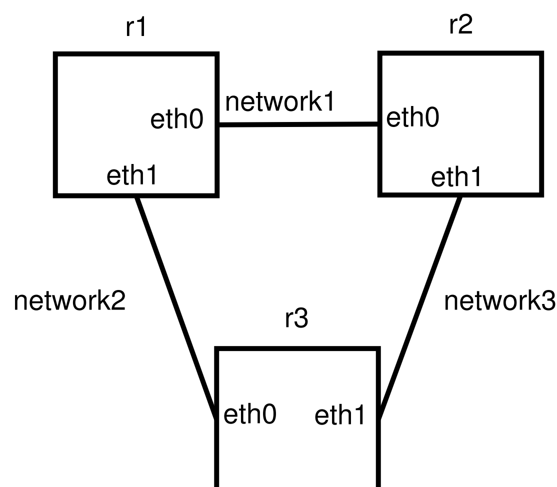

Abbildung 5.1: *Beispiel eines Netzwerkes mit drei Knoten*

Auf das Topologie-Script wird aus dem Start- und Kill-Script zugegriffen, um zum Beispiel die Anzahl der am virtuellen Netz beteiligten Knoten auszulesen oder aber um die Anzahl der zu startenden uml-switches (siehe 5.2.3) herauszufiltern.

5.3.5 Initialisierungs-Script

Das Initialisierungsscript init.sh enthält die Grundinformationen für die Knoten. Hier werden die IP Adressen der Knoten festgelegt und es können schon iptables-Regeln mit übergeben werden. Da das Initialisierungs-Script im Start-Script (siehe Punkt 5.3.2) implementiert ist (über den Blockdevice ubd1 in jedem UML enthalten), wird es beim Starten des Netzwerks auf den einzelnen Knoten mit ausgeführt. Die einzelnen Knoten bekommen durch das Script also eine Grundkonfiguration zugewiesen:

- Mounten des Arbeitsverzeichnisses: Mit hostfs wird das umlgui-Verzeichnis des Hostsystems gemountet/eingebunden. Jeder Knoten hat dadurch zugriff darauf. Das ist wichtig, da in dem gemounteten Verzeichnis die Kommunikationsschnittstelle der Netzwerkknoten (umlclient.py) liegt. Da das Script auf einem Knoten läuft, muss es beim mounten wissen, wo das umlgui-Verzeichnis auf dem Hostsystem zu finden ist. Beim Starten der Knoten wird daher der Pfad dazu als Parameter mit übergeben (siehe 5.3.2). Dieser kann in der */proc/cmdline* ausgelesen werden und dann mittels hostfs gemountet.

- hosts-Datei: Die einzelnen IP-Adressen der Knoten und deren Interfaces werden in die hosts-Datei geschrieben. Dadurch kann nicht nur über die IP, sondern auch über den Namen auf die Knoten des Netzes zugegriffen werden (Beispiel: r1_0 ist das Interface eth0 auf dem Knoten r1).

- switch/case: In einem switch/case-Konstrukt können für jeden Knoten im virtuellen Netzwerk schon Grundkonfigurationen vorgenommen werden. Zum Beispiel IP-Adressen der Netzwerkschnittstellen, Routen, iptables-Regeln etc. Hier sollte das Topologie-Script (5.3.4) mit einbezogen werden. Dort wird die Topologie festgelegt, und im Initialisierungsscript wird diese umgangssprachlich 'mit Leben gefüllt'. Beispiel:

```
case name in
  r1)
    ifconfig eth0 172.16.101.1/24 up
  ;;
  r2)
    ifconfig eth0 172.16.101.2/24 up
  ;;
```

  Wenn im Topologie-Script also zwei Knoten *r1* und *r2* über das Netz *network1* verbunden sind, muss im Init-Script für die Knoten auch angegeben werden, welche IP-Adressen sie haben. Da sie in einem Teilnetz liegen wären zum Beispiel 172.16.101.1/24 und 172.16.101.2/24 möglich. /24 ist die abgekürzte Schreibweise für das Subnetzmaske (eigentlich 255.255.255.0). Die Netzadresse wäre demnach 172.16.101.0.

- umlclient-Script: Starten des umlclient-Scripts auf den Knoten im Netzwerk, das im gemounteten Verzeichnis enthalten ist.

5.3.6 GUI

Die grafische Benutzeroberfläche passt sich dynamisch an die Anzahl der Knoten an. Jedem Knoten ist ein Textfenster mit Befehlszeile zugeordnet. Somit lassen sich alle Knoten über die grafische Oberfläche steuern und verwalten. In Abbildung 5.2 ist der Aufbau der GUI skizziert. Eingaben in die Befehlszeilen lassen sich per Button an die Knoten schicken und dort ausführen. Die Kommunikation zwischen den Knoten und dem Host wird in Abschnitt 5.3.9 näher erläutert. Rückgabewerte und Antworten der Knoten werden automatisch in den Textfenstern angezeigt. Gibt man einen Befehl in der Befehlszeile des Knoten 1 innerhalb der GUI ein, wird dieser Befehl an den Knoten 1 gesendet, dort verarbeitet und die zurückgeschickte Antwort wird im Textfenster des Knoten 1 angezeigt. Das Textfenster und die Befehlsleiste bilden die virtuellen Consolen der Knoten nach. Allerdings ist zu beachten, dass Aufrufe die eine längere Bearbeitungszeit haben, nicht sofort in den Knoten-Textfenstern angezeigt werden. Der Befehl wird gesendet und die komplette Rückgabe des Befehls wird zusammen zurückgegeben. Wird einer der Knoten gepingt, sieht man nicht jede Antwortzeile sofort in der GUI. Sie tauchen gesammelt auf, sobald der Befehl auf dem Knoten fertig abgearbeitet wurde. Je länger also ein Befehl zur Bearbeitung braucht, desto länger dauert es, bis im Knoten-Textfenster die Ausgabe erscheint. Ist ein Knoten durch einen Ping beispielsweise nicht erreichbar, kommt die Anzeigeverzögerung nur zustande, weil der Knoten noch nicht fertig mit dem Ping ist. Um die Ausgabe auf den Knoten abzufangen und damit weiterarbeiten zu können dient eine Pipe (siehe Abschnitt 5.3.9 sowie Abbildung 5.5).

5.3.7 UML-Host

Der UML-Host ist ein Pythonscript und ist die Kommunikationsschnittstelle des Servers, die in der GUI implementiert auf dem Hostsystem ist. Die Aufrufe und Befehle werden aus der GUI heraus an die Knoten mittels des pickle-Moduls gesendet. Soll ein Befehl vom Host an den Knoten gesendet werden, geschieht folgendes:

```
1 def sendRouter(param, router):
      flag = False
3     for r in umls:
          if router == r.name:
5             print "sendRouter() aufgerufen, Routername = %s" % r.name
              r.send(param)
```

In *umls* sind alle Namen der Knoten enthalten, die vorher aus dem Topologie-Script ausgelesen wurden. Der Methode werden der auszuführende Befehl und der Name des Knotens auf dem dieser Befehl ausgeführt werden soll übergeben. Wenn der Name eines Knoten dem

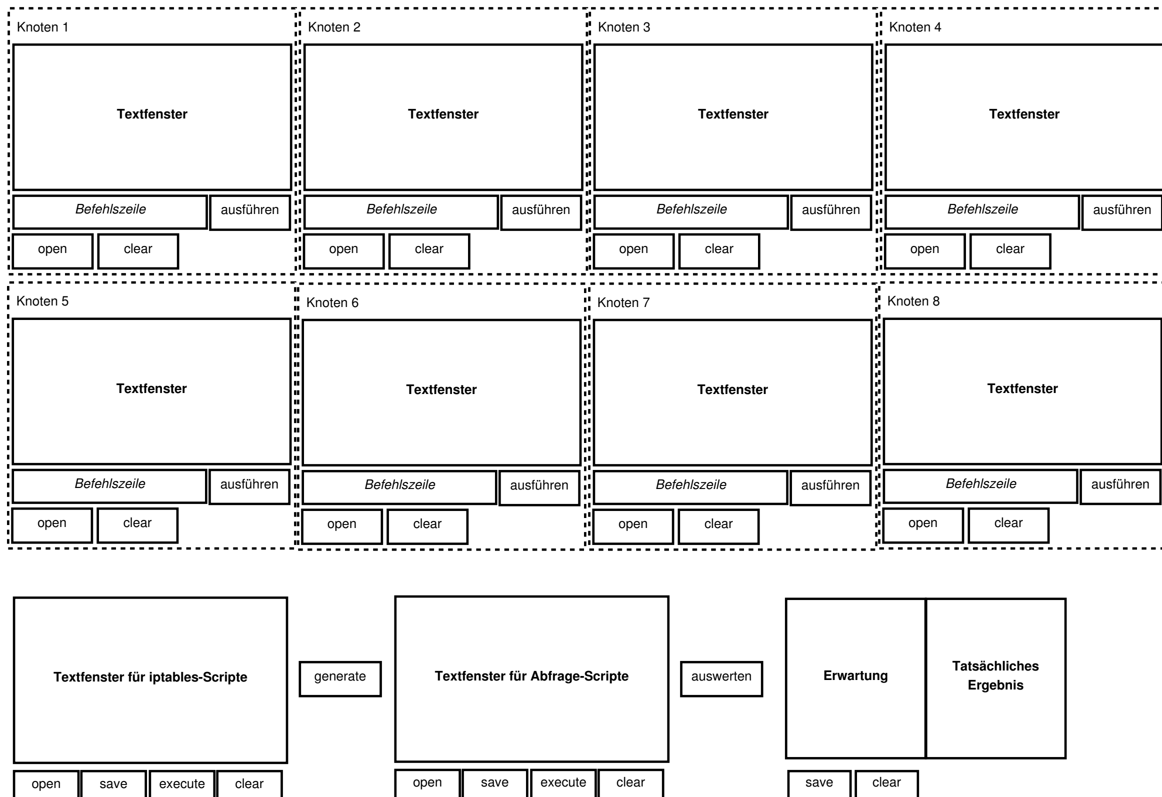

Abbildung 5.2: *Aufbau der grafischen Benutzeroberfläche (GUI)*

übergebenen Knotennamen (hier die Variable *router*) entspricht, wird die Methode *send()* aufgerufen und der Befehl übergeben:

```
def send (self, param):
    pickle.dump( {"cmd": "run", "param":param}, self.masterPty, 1)
    print "== "+param+" send"
    pkt=pickle.load( self.masterPty )
    if "Type" in pkt and pkt["Type"] == "Response":
        MainWindow.setText(frame, self.name, pkt["Param"])
    else:
        print "error"
```

In der Methode *send()* wird dann der Befehl (hier *param*) an den ausgewählten Knoten (siehe Code der Methode *sendRouter()*) gesendet. Der vom Knoten zurückgegebene Wert wird dann in der GUI dargestellt. Auf die Funktion von pickle wird später noch genauer eingegangen (Abschnitt 5.3.9).

5.3.8 UML-Client

UML-Client ist ein Pythonscript und dient als Kommunikationsschnittstelle des Netzwerkknotens. Aufrufe und Befehle des Hosts kommen hier an und werden auf dem Knoten ausgeführt. Es ist Zustandslos, dass heißt der Knoten merkt sich seinen Zustand nicht. Ein Verzeichniswechsel mit 'cd xxx' lässt sich zwar ausführen, allerdings wird der neue Zustand nicht abgespeichert. Beim Starten des Netzwerkes wird das Script auf den Knoten automatisch gestartet. Das geschieht durch die Anbindung des Initialisierungsscript an einen Blockdevice. Dieses Init-Script wird automatisch ausgeführt und die enthaltenen Einstellungen (den Knoten betreffend) ausgeführt. Darin ist auch der Befehl *python /mnt/umlclient.py* enthalten, der das Script startet.

Falls man einen UML-Client oder seinen Prozess beendet, kann man diesen wieder starten, indem man auf seine virtuelle Konsole wechselt und dort das Script neu startet, welches im gemounteten Verzeichnis /mnt liegt (Befehl: *python /mnt/umlclient.py*).

Der UML-Client wartet auf eine Nachricht des UML-Hosts. Sobald eine Nachricht eintrifft, wird diese verarbeitet (siehe dazu auch Abschnitt 5.3.7).

```
   ...
 2 o = pickle.load( ser )
   if "cmd" in o and o["cmd"]=="run":
 4     if ...
       ...
 6     else:
           p = Popen(o["param"], shell = True, stdin = PIPE, stdout = PIPE,
               stderr = STDOUT, close_fds= True)
 8         p = Popen("PATH=\"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
               sbin:/bin\";"+o["param"], shell = True, stdin = PIPE, stdout =
               PIPE, stderr = STDOUT, close_fds= True)
           rc = p.stdout.read()
10         pickle.dump( {"Type":"Response", "Param":rc}, ser, 1 )
           ...
```

Das ganze wird am Beispiel Ping erläutert: In Zeile 2 wird das Objekt deserialisiert. Übertragen wird hier ein dictionary (key:value Paar). Im value steht der ping-Befehl (zum Beispiel *ping 192.168.1.1*). Dieser wird ausgeführt und die Ausgabe die ping zurückliefert (Zeile 9) wird wieder an den Host gesendet (Zeile 10). Damit der Host weiss, das es sich um eine Antwort eines Knoten handelt, steht im key des dictionary *response* und im value die Auswertung des ping-Befehls. Hier nochmal die Serverseite, die die Antwort ausliest und verarbeitet:

```
pkt=pickle.load( self.masterPty )
if "Type" in pkt and pkt["Type"] == "Response":
    MainWindow.setText(frame, self.name, pkt["Param"])
```

Es wird die Rückgabe des ping-Befehls in dem Textfenster des Knotens dargestellt, von dem der Befehl gesendet wurde.

5.3.9 Client-Server-Kommunikation

Um die Kommunikation zwischen den Knoten (5.3.8) und dem Host (5.3.7) zu realisieren gibt es verschiedene Ansätze.

tun/tap-Device Zum einen kann man tun/tap-Devices verwenden. Tun und tap sind virtuelle Netzwerk-Kerneltreiber, die Netzwerkgeräte simulieren. Normalerweise befindet sich hinter Netzwerkgeräten, wie z.B. eth0 oder eth1. direkt eine Hardware in Form einer Netzwerkkarte. Pakete die an ein tun/tap-Device gesendet werden, werden an ein Programm im Userspace weitergeleitet. Damit andersrum auch ein Userspace-Programm auf die tun/tap-Geräte zugreifen kann, besitzen diese eine Gerätedatei. Mit einem TAP-Device kann man vom Hostsystem direkt auf die Knoten im virtuellen Netz zugreifen. Eine Schnittstelle wird über das TAP-Device mit dem Host verbunden. Allerdings erfordert das Einrichten root-Rechte.

serielle Schnittstellen Die Kommunikation kann auch über serielle Schnittstellen hergestellt werden und zwar über eine simulierte Hardware. Es wird eine virtuelle serielle Verbindung zwischen zwei UML-Instanzen kreiert. Dies geschieht am Pseudoterminal des Hosts. Die zuerst geöffnete Seite ist das PTY, die Master-Seite. Diese existiert erst sobald sie geöffnet wird. Die erste UML-Instanz öffnet also die Master-Seite des Device und danach kann die zweite UML-Instanz die Slave-Seite (tty) öffnen.

Das virtuelle Netzwerk soll getrennt vom Wirtsystem laufen, das heißt, ein direkter Zugriff auf die einzelnen Knoten im Netzwerk soll nicht möglich sein. Die Erstellung und Bedienung des Programms soll ohne root-Privilegien möglich sein. Daher fiel die Entscheidung auf serielle Schnittstellen.

Die Funktion in Bezug auf die Kommunikation wird im Folgenden erläutert:

Pseudoterminal Bei einem Pseudoterminal sind, im Gegensatz zu einem Terminal, beide Seiten mit einem Prozess verbunden. Daher ist das Pseudoterminal eine Art der Interprozesskommunikation. Es hat eine Master und eine Slave Seite. Die Slave-Seite ist

mit einem Prozess verbunden der eine Gerätedatei geöffnet hat (z.B. /dev/ttyS0), also genau wie bei einem normalen Terminalgerät. Die Master-Seite ist (im Kernel) direkt mit der Slave-Seite verbunden, ist also das was normalerweise die serielle Schnittstelle ist. Nur tauscht die Master-Seite hier Daten mit einem Prozess anstatt einer seriellen Schnittstelle aus.

Das Pseudoterminal wird beim Starten des UML-Netzes erstellt. Die Masterschnittstelle wird mit dem Host (5.3.7), die Clientschnittstelle mit dem Knoten (5.3.8) verbunden (siehe Abbildung 5.4). Damit das Verbinden funktioniert, wird auf dem Host ein symbolischer Link angelegt.

```
1 self.path = os.path.abspath( "tmp/console" + name )
os.symlink( os.ttyname(self.slave), self.path    )
```

Nun kann sich ein python-Script auf dem Host direkt mit einem python-Script auf dem Knoten (umlclient.py) über eine serielle Verbindung in beide Richtungen unterhalten. Zum Austausch von Informationen wird das Serialisierungsmodul von Python - pickle - genutzt. Damit können komplexe Datenstrukturen wie z.B. Dictionaries, Listen etc. serialisiert und über die Leitung übertragen werden.

Das pickle Modul erzeugt beim Serialisieren eines Objekts einen String, der alle Informationen des Objektes speichert. Bei der Deserialiserung kann das Objekt so später rekonstruiert werden. Zu Beginn dieser Bachelorarbeit wurden Dictionarys serialisiert. Dictionarys enthalten einen key (Schlüssel, z.B Strings oder Nummern) und einen value (Wert). Der Schlüssel bestand anfangs zum Beispiel aus einem Kommando wie *run* oder *echo*, und der Wert aus einem Parameter - zumeist ein Befehls-String. Wurde als Kommando *echo* übergeben und als Wert *'Hello World'* führte das zu einer einfachen *'Hello World'*-Ausgabe auf der Knotenseite. Das Kommando *run* hingegen enthielt als Wert einen Befehl, welcher auf dem Knoten als Systembefehl ausgeführt wurde (z.B. *cmd = run, paramter = ping 192.168.1.1*). Dies diente der Unterteilung von Systemaufrufen und normalen Ausgaben. Gerade zu Anfang der Arbeit wurden mit dem echo-Befehl die ordnungsgemäße Kommunikation zwischen Knoten und Host getestet. Später allerdings wurden alle Befehle, die an den Knoten gesendet wurden, nur mit dem key *run* übergeben, und auf dem Knoten selber wurde ausgewertet, was mit dem String des values passieren soll. Der Befehl zum Serialisieren ist *pickle.dump()*, zum deserialiseren wird *pickle.load()* genutzt.

Die Funktionsweise des pickle-Moduls ist in Abbildung 5.3 dargestellt. Zur Verdeutlichung der Funktion nehmen wir den Befehl 'ping 192.168.1.1', der auf Knoten X ausgeführt werden soll, als Beispiel. Der Befehl wird in der Befehlsleiste des Knoten X eingegeben und dann ausgeführt. Der Host kennt alle seine Knoten, weiß also, dass der Befehl an Knoten X zu senden ist. *ping 192.168.1.1* wird nun mit *pickle.dump()* serialisiert und gesendet. der ping-Befehl

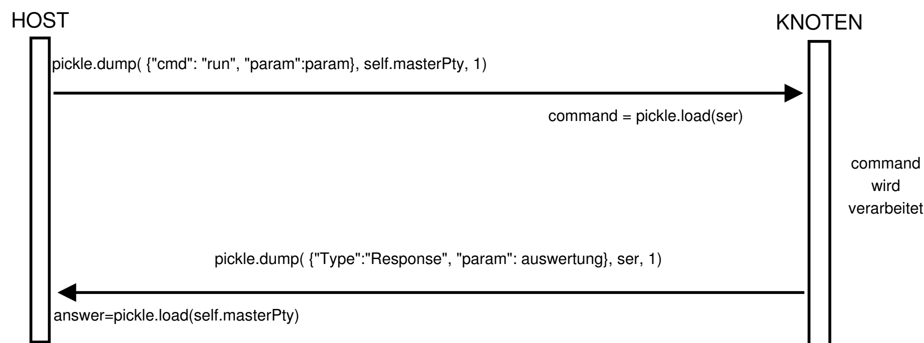

Abbildung 5.3: *Befehlsübertragung mit dem pickle-Modul*

ist in diesem Fall der Paramater (param). Auf der Knotenseite wird mit *pickle.load(object)* darauf gewartet das ein Befehl eintrifft. Sobald dies geschehen ist, wird der Befehl deserialisert und ausgewertet. Systembefehle, die ohne Vorarbeit oder Bearbeitung (Erklärung dazu folgt nach diesem Beispiel) direkt auf dem Knoten ausgeführt werden können - wie auch ping - benötigen keine weitere Auswertung sondern werden direkt auf dem Knoten aufgerufen. Der Knoten pingt nun also die IP *192.168.1.1*. Die Ausgabe von ping wird dann mittels *pickle.dump()* wieder serialisert an den Host gesendet. In diesem Fall ist der Parameter die Auswertung des ping-Befehls. Auf dem Host wird deserialisert und der im *param* stehende Text (die Auswertung) auf der GUI im Textfenster des Knoten X dargestellt. Andere Befehle, wie zum Beispiel das Abfragen eines Ports, der durch eine iptables-Regel gesperrt oder geöffnet ist, können nicht sofort abgefragt werden. Ein Port, auf dem kein aktiver Dienst läuft, ist standardmäßig geschlossen. Um also zu Überprüfen, ob ein Port offen/gesperrt ist, muss noch eine Vorarbeit vom Programm getätigt werden (siehe dazu 5.4.2).

5.3.10 Zentrales Management

Wie erwähnt, geschieht die Kommunikation zwischen dem Host und dem Knoten über serielle Schnittestellen. Somit kann der Host mit den Knoten über die dafür zuständigen Scripte kommunizieren (siehe UML-Host in Abschnitt 5.3.7 und UML-Client in Abschnitt 5.3.8). Um nun das ganze zentral zu managen und nicht die Befehle auf den Knoten selber auszuführen und dort deren Auswertung abzulesen, werden die Ausgaben auf der zentralen Verwaltungsstelle - der GUI - ausgegeben. Der Weg eines Befehl von seiner Eingabe, über die Verarbeitung, bis hin zu seiner Ausgabe ist in Abbildung 5.5 veranschaulicht. Das Ganze läuft nach folgendem Prinzip ab: Ein Befehl wird in der Befehlszeile eines Knotens in GUI eingegeben und gesendet. Auf dem Knoten selber wird dieser dann ausgeführt und seine Ausgabe wieder an die grafische Oberfläche zurückgeschickt, und dort im Textfenster des Knoten ausgegeben.

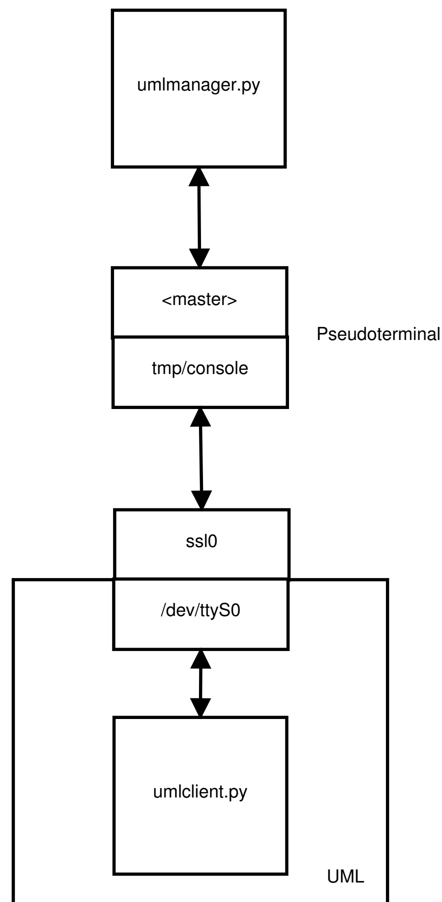

Abbildung 5.4: *Kommunikation zwischen Host (GUI) und Knoten (UML)*

Das Prinzip wird mittels PIPE realisiert. Die Pipe kann man sich wie einen Tunnel vorstellen, durch den ein Aufruf geschickt wird. Auf der anderen Seite des Tunnels wird dieser verarbeitet und die Antwort durch den Tunnel wieder zurück zum Initiator gesendet. Dieser kann die Antwort dann weiter verarbeiten.

```
...
p = Popen(o["param"], shell = True, stdin = PIPE, stdout = PIPE, stderr = STDOUT,
    close_fds= True)
rc = p.stdout.read()
pickle.dump( {"Type":"Response", "Param":rc}, ser, 1 )
...
```

Ein Befehl wird auf der Befehlszeile, im Textfenster eines Knotens, innerhalb der GUI eingegeben und abgeschickt (*pickle.dump()*). Auf dem Knoten wird der ankommende Befehl durch

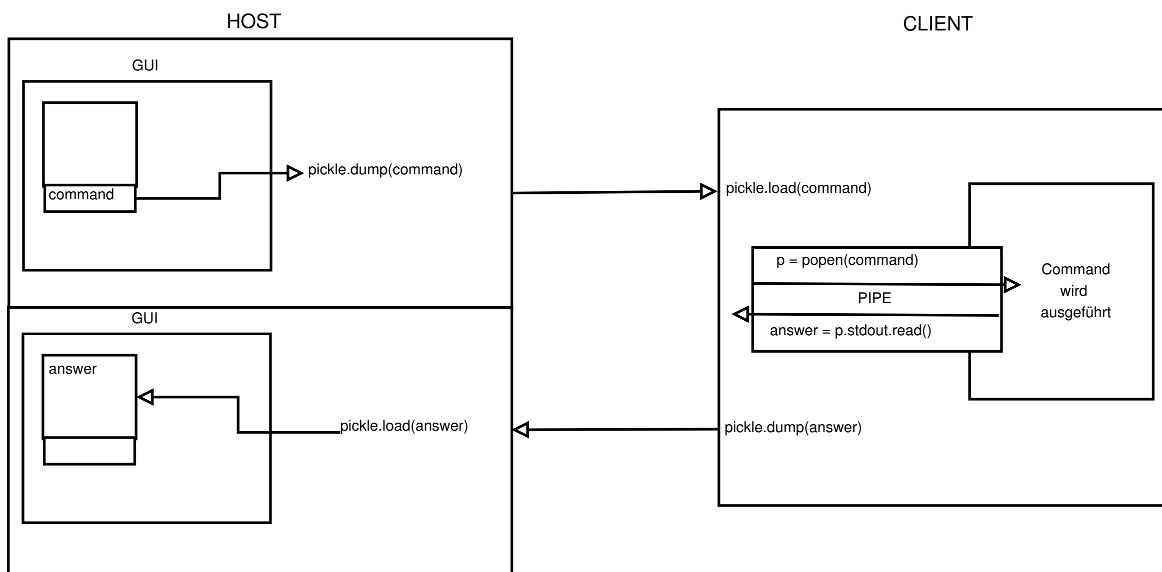

Abbildung 5.5: *Der Weg eines Befehls von der Eingabe bis zur Ausgabe auf der grafischen Oberfläche*

pickle.load() ausgelesen. Der Knoten initiiert nun eine Pipe. Die Standard-Datenströme (sdtin, stdout) werden an die Pipe angeschlossen. Der Befehl wird in einer Shell ausgeführt. Die Rückgabe dieses Befehls liegt an stdout an, kann ausgelesen und in eine Variable gespeichert werden. Diese Variable wird dann mit pickle.dump() vom Knoten an den Host gesendet, der diese ausliest. Sie wird dann im einem Textfenster in der GUI dargestellt/ausgegeben.

5.4 Generieren und Auswerten

Die Idee hinter der Generierung und der Auswertung von iptables-Scripten ist in der Analyse schon an einem Beispiel verdeutlicht worden (siehe 4.2). In diesem Abschnitt wird nochmal näher auf die Umsetzung eingegangen.

5.4.1 Abfrage-Befehle generieren

Beim Generieren der Abfrage-Befehle wird die jeweilige iptables-Regel nach mehreren Kriterien durchsucht. Am einfachsten zu erklären ist dieses beim Betrachten des jump-Targets (was passiert mit einem Paket, das auf eine Regel zutrifft). Wenn ein Paket ohne Rückmeldung verworfen wird (DROP) müssen die generierten Befehle in ihrer Ausführung fehlschlagen. Bei einem ICMP-Match mit dem Target DROP, ist die Erwartung, dass der Ping Befehl

fehlschlägt. Das heißt, in dem Abfrage-Befehl wird nicht nur ein Befehl generiert, der die iptables-Regel überprüft, sondern es ist gleichermaßen eine Erwartung integriert: Was wird erwartet von dieser iptables-Regel. Genau das Selbe ist bei einer Portsperre der Fall. Auch hier muss der generierte Abfrage-Befehl den Port des Hosts überprüfen. Dieser sollte dann nicht offen sein. Der Abfrage-Befehl beinhaltet also eine Portabfrage sowie die Erwartung, welche Antwort bei dieser Portabfrage zurückkommen sollte.

Diese Erwartung wird später dann bei der Auswertung mit einfließen. Wird von einem anderen User ein Script eingespeist, das die gleichen Vorgaben erfüllen soll und dieses danach mit einem Abfrage-Script (zum Beispiel von einer Lehrkraft, die die Aufgabe gestellt und selber schon gelöst hat) verglichen, wird eigentlich die Erwartung des Abfragescripts der Lehrkraft mit dem tatsächlichen Ergebnis des Abfrage-Script (welches die iptables-Regeln des Users abfragt) gegenübergestellt.

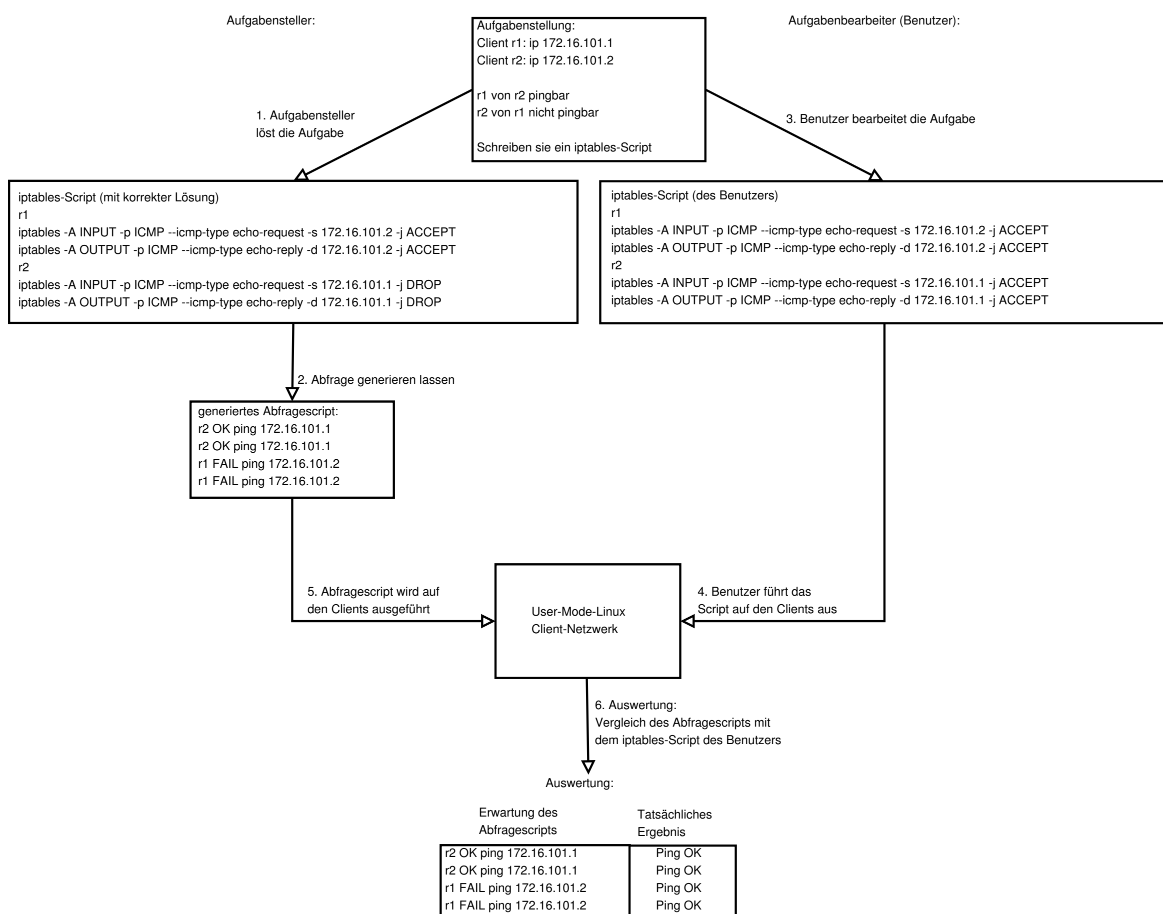

Abbildung 5.6: *Beispiel: Veranschaulichung der Funktion zum Abfragen von iptables-Scripten (hier mit falscher Bearbeitung der Aufgabenstellung des Benutzers)*

5.4.2 Überprüfung der iptables-Regeln

Um iptables-Regeln zu überprüfen gibt es eine Reihe von Hilfsprogrammen, die in Abschnitt 3.4.1 aufgeführt sind. Sobald nicht sicher ist, dass eine iptables-Regel auch wirklich die erhoffte Funktion hat, kann die Regel mit diesen Programmen überprüft werden.
Als Beispiel dient hier das Öffnen eines Ports. Standardmäßig sind zuerst einmal alle Ports durch die Policy DROP gesperrt. Mit einer iptables-Regel können aber Ports geöffnet werden. Um zu überprüfen, ob der Port nach ausführen der Regel auch wirklich offen ist, wird mit nmap ein Portscan durchgeführt. Wird auf einem Computer (Rechner A) mit der ip *172.16.101.1* mit Hilfe einer iptables-Regel der Port *5000* geöffnet (zum Beispiel: *iptables -A INPUT eth0 -p TCP –dport 5000 -j ACCEPT*), kann dieser mit nmap überprüft werden (*nmap 172.16.101.1 -p 5000*). Nmap liefert den Status des Ports zurück, ob er geöffnet, geschlossen oder gefiltert ist.
Wichtig zu wissen ist aber, dass jeder Port - auch wenn geöffnet - immer geschlossen ist, solang kein Dienst auf diesem Port läuft. Nmap wird also, sofern nicht zufällig auf Port *5000* gerade ein Dienst läuft, als Portstatus '*CLOSED*' zurückgeben. Das hilft aber bei der Überprüfung nicht, da der Port mit einer iptables-Regel geöffnet wurde. Es wäre also schön, wenn nmap auch '*OPEN*' zurückliefert.
Daher hält das Programm zwei Möglichkeiten bereit, diesen Port zu überprüfen.

Dienst-Simulierung Wie beschrieben ist der Port immer geschlossen, solange kein aktiver Dienst darauf läuft. Daher kann mit dieser Option auf einem bestimmten Port ein Dienst simuliert werden. Es wird dem Knoten dafür der Befehl *openSocket*, die IP des Knotens, sowie der Port übergeben. Zum Beispiel soll auf Knoten 1 (IP: 172.16.101.1) Port 70 überprüft werden. In der Befehlszeile des Knotens innerhalb der GUI wird dann *openSocket 172.16.101.1 70* eingegeben und gesendet. Dort wird jetzt auf Port 70 ein Dienst simuliert. Nun kann mit *nmap 172.16.101.1 -p 70* der Portscan durchgeführt werden. Danach ist dieser Port durch den Befehl *closeSocket 70* wieder zu schließen. Diese Option kann genutzt werden um den nmap Befehl auszuwerten. Wenn nun aber der Port durch eine iptables-Regel beeinflusst ist, wird als Ergebnis immer '*FILTERED*' zurückkommen, und nicht '*OPEN*' oder '*CLOSED*'. Diese Option ist dafür da, um zu schauen ob der Port von irgendwelchen Einstellungen betroffen ist oder ein Hindernis im Netzwerk ihn blockt.

Client/Server-Erstellung Um zu erkennen, ob der Port offen oder geschlossen ist, obwohl eine iptables-Regel ihn beeinflusst, gibt es die Möglichkeit, das genauer zu überprüfen. Dafür wird auf dem Knoten des zu überprüfenden Ports ein Server und auf einem anderem Knoten, der mit diesem verbunden ist, ein Client erstellt. Der Client sendet nun eine Nachricht an den Server, welcher auf dem zu überprüfenden Port läuft. Wenn der Port nun durch eine iptables-Regel wirklich geschlossen ist, bekommt der Client keine Antwort auf seine Anfrage und gibt zurück, dass der Port gesperrt ist. Ist der

Port jedoch offen, bekommt der Client eine Antwort, und gibt 'Port offen' zurück. Danach wird der Server sofort beendet. In diesem Fall wird also kein nmap verwendet, sondern lediglich versucht, eine Verbindung und ein Nachrichtenaustausch zwischen zwei Knoten herzustellen. Die Befehle für die Option der Überprüfung sind:

- auf dem Knoten X, dessen Port zu überprüfen ist: *connectServer [IP] [Port]* (z.B. *connectServer 172.16.101.1 2000*)
- auf dem Knoten Y, der mit Knoten X verbunden ist: *connectClient [IP des Server-Knotens]* [Port] (z.B. *connectClient 172.16.101.1 2000*)

Nach diesem Schema können die iptables-Regeln überprüft werden. Wird durch eine Regel etwas gesperrt, geöffnet, zugelassen oder abgelehnt, muss man diese entweder mit den zur Verfügung stehenden Hilfen abfragen oder Überlegungen anstellen, wie man die Regel überprüfbar macht.

Ähnlich ist es bei einem icmp-match, also der Aufstellung einer Regel bezüglich des icmp-Protokolls. Wird auf einem Knoten ein icmp-request gesperrt, muss ein Weg gefunden werden, diese Regel auch zu testen. Für diesen Fall hilft der Befehl ping weiter. Zu finden ist ein Knoten, der eine Verbindung zu dem von der Regel betroffenen Knoten hat. Ist dieser gefunden, kann ein ping-Befehl gesendet werden um die Regel zu überprüfen. Da die Regel besagt, das ein request abgewiesen wird, sollte kein reply zurückkommen.

Diese Ideen und Überlegungen zielen darauf ab, zwei iptables-Scripte miteinander zu vergleichen und auszuwerten. Das heißt, ein Benutzer erstellt ein Script und lässt sich daraus ein Abfragescript generieren. Ein weiterer Benutzer erstellt ebenfalls ein Script mit den selben Vorgaben. Nun können diese zwei Scripte miteinander verglichen werden. Um zu überprüfen ob eine eigene Regel, die gerade erstellt und ausgeführt wurde, aktiv ist, gibt es den Befehl 'iptables -L', der alle aktiven Regeln auflistet. Darunter sollte auch die eigene zu finden sein.

5.4.3 Auswertung

Wie schon im Abschnitt 5.4.1 angedeutet, werden bei der Auswertung zwei unterschiedliche Scripte gegenüber gestellt. Zum einen das iptables-Regel-Script, und zum anderen das Abfrage-Script zur selben Aufgabenstellung. Im Optimalfall sollte das Abfrage-Script aus einem iptables-Regel-Script erstellt werden das die Aufgabe richtig löst. So ist gegeben, dass die Auswertung auch auf die Lösung der Aufgabe zutrifft und alle Vorgaben abdeckt.

Für die Auswertung ist es wichtig, dass die in Abbildung 5.7 aufgeführten Schritte der Reihe nach durchgeführt werden, bevor der Auswertungsbutton betätigt wird. Das Diagramm geht davon aus, dass das iptables-Script eines Benutzers schon auf den Knoten, also im virtuellen

Netz, ausgeführt wurde und derjenige, der eine Auswertung starten möchte, ein iptables-Script zur richtigen Lösung der Aufgabenstellung vorliegen hat.

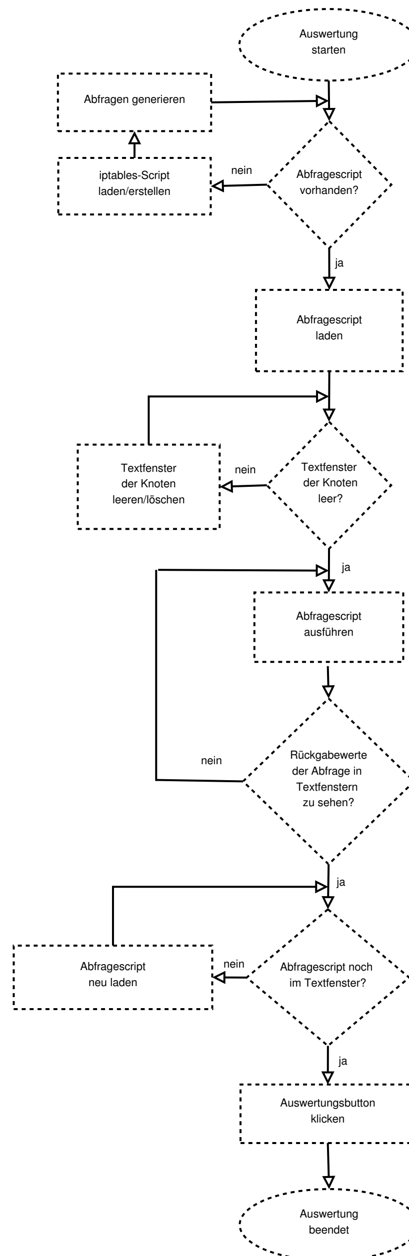

Abbildung 5.7: *Flussdiagramm: Auswertung*

Bei der Auswertung wird die Erwartung eines (aus einem iptables-Regel-Script generierten) Abfragescripts mit der Auswertung des dieses Abfragescripts gegenübergestellt. So lässt sich auf einen Blick erkennen, ob alle Vorgaben erfüllt wurden. Genauer gesagt werden bei

der Auswertung zwei Strings gegenüber gestellt. Auf der einen Seite das generierte Abfragescript. Dieses enthält die erwarteten Ergebnisse. Auf der anderen Seite das, auf den einzelnen Knoten, ausgeführte Abfragescript. Die Rückgaben und Antworten des Scripts sind dann in den einzelnen Textfenstern für die Knoten zu sehen. Diese werden ausgelesen, verarbeitet und in deutlich verkürzter Form dargestellt. Als Beispiel dient wieder der Befehl Ping. Besagt eine iptables-Regel, dass ping auf einem Knoten gedroppt wird, also ein request abgewiesen wird, ist die Erwartung, dass ein ping zu diesem Zielknoten nicht möglich ist, also keine Antwort zurückkommt (Erwartung: ping schlägt fehlt - 'fail'). Wird das Abfrage-Script nun ausgeführt, wird von einem mit diesem Knoten verbundenen weiteren Knoten ein ping-Befehl gesendet. Die Auswertung überprüft nun genau was ping zurückgibt. Die im Textfenster des Knotens ausgegebene Rückgabe des Ping-Befehls wird eingelesen und durchsucht. Gibt es hier eine Antwort des Zielknoten, ist der ping-Befehl erfolgreich ('ok') und dies wird in die Auswertung übernommen. Ist der Zielknoten nicht erreichbar, also der ping schlug fehlt ('fail'), steht dies in der Auswertung. Wenn man nun die Erwartung (ping schlägt fehl) mit der Auswertung der ping-Aktion (ping schlägt ebenfalls fehl) vergleicht, stimmen diese beiden überein. Das heißt, die vom User entwickelte iptables-Regel stimmt mit der Vorgabe eines anderen Users oder einer Lehrkraft überein.

Auf diese Weise wird auch der Befehl nmap ausgewertet. Die Rückgabe eines Befehls wie *nmap 172.16.101.2 -p 40* (es wird der Port 40 auf dem Knoten mit der IP 172.16.101.2 abgefragt) liefert eine mehrzeilige Analyse über den Port 40 des Zielrechners wieder. Diese wird durchlaufen und nach den für die Portabfrage wichtigen Strings wie OPEN, CLOSED und FILTERED durchsucht. Je nachdem, ob der Port offen, gesperrt oder gefiltert wird, wird der String ausgelesen und in der Auswertung dargestellt. Für die Generierung ist derzeit nur die in Kapitel 5.4.2 beschriebene Dienst-Simulierung implementiert. Die Client/Server-Erstellung ist aber trotzdem ausführbar.

Die Auswertung überprüft vor der Gegenüberstellung alle vom Abfrage-Script ausgewerteten Befehle.

5.4.4 Grenzen der Überprüfbarkeit

Es lässt sich nicht sagen ob es Grenzen gibt. Überprüfungen von Regeln lassen zwar realisieren, aber der Aufwand um einige iptables-Regeln zu überprüfen ist dermaßen groß, dass man Hinterfragen muss, ob dieser Aufwand gerechtfertigt ist. Um den Aufwand zu erkennen, dient als Beispiel ein Netzwerk aus vier UML-Knoten (A, B, C, D). Die Verbindungen sind schematisch kurz dargestellt: A – B – C – D. Knoten D simuliert das Internet, Knoten A ist eine Workstation. Knoten B und C liegen zwischen A und D und dienen als stateful Firewall. Zu großen Aufwänden kann es kommen wenn es eine Portweiterleitung gibt. Das heißt, wenn

ankommende TCP-Pakete an einem bestimmten Port über die stateful Firewalls weitergeleitet werden sollen (zum Beispiel an Knoten A). Solche TCP-Pakete kann man im Header noch darauf überprüfen ob sie ein SYN-Flag haben (also eine neue Verbindung erstellt werden soll) oder ob die Verbindung schon erstellt wurde. Um diese Regeln zu testen müssten eigens dafür TCP-Pakete erstellt werden um im Header die richtigen Flags zu setzen. Noch dazu müsste überprüft werden ob diese Pakete auch richtig weitergeleitet werden und ob sie bei A ankommen.

Weiterhin kann es zu Schwierigkeiten kommen, wenn die Regel als Target DROP besitzt, also verworfen wird. Denn dadurch bekommt ein Knoten der etwas sendet keine Rückmeldung darüber was mit dem Paket passiert, es wird einfach verworfen ohne eine Antwort zu senden. Das heißt, der Sender des Paketes erhält keine Information darüber, was mit dem Paket passiert ist. Er nimmt automatisch an, das es gesendet und empfangen wurde. Da in einigen Fällen eine Rückmeldung wünschenswert ist, gibt es das Target REJECT. Es funktioniert wie DROP, liefert allerdings eine Rückmeldung, sobald das Paket irgendwo abgewiesen wurde.

5.5 iptables-Konfigurator

Für die Erstellung des iptables-Konfigurators gab es zwei grundlegende Ansätze die nachfolgend erklärt werden.

5.5.1 Realisierungsansatz A: Zusammenstellung mehrerer Regeln

Der erste Ansatz beinhaltet die Bereitstellung verschiedener iptables-Regeln, dargestellt in einer GUI. Die Regeln sind komplett aufgelistet, so dass die gewünschten Befehle nur per Klick ausgewählt werden müssen und in einem Textfeld gelistet, beziehungsweise gleich als Script abgespeichert werden. Es wäre hier also möglich, ein ganzes Script nur mit Mausklicks zu erstellen, und dieses später auf den einzelnen Knoten auszuführen. Aber allein eine Portmanipulation - also das Öffnen oder Schließen eines Ports - würde die grafische Oberfläche schon ausreizen. Um es komfortabel zu halten, müsste jede iptables-Regel - die in diesem Fall allesamt, bis auf die Portnummer, identisch wären - aufgelistet sein. Bei der Anzahl an existierenden Ports (65535) müssten also entweder nur die wichtigsten aufgelistet sein oder die GUI wäre allein mit diesen Regeln schon überladen. Dieser Ansatz bietet allerdings keinerlei Selbstgestaltung, da die Befehle alle vorgegeben sind.

5.5.2 Realisierungsansatz B: Selbstgestaltung

In diesem Ansatz kann immer nur ein iptables-Befehl zur Zeit generiert werden. Diesen Befehl muss der User von Grund auf selber zusammenstellen. Angefangen vom Table über die Chain bis hin zur Parameterauswahl und den Targets wird hier die Zusammenstellung dem User überlassen. Hilfen werden dem User mit an die Hand gegeben. So reduziert sich zum Beispiel die Auswahl der Chains (INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING), nachdem man den table ausgewählt hat. Die Filter-Tabelle hat als fest eingebaute Ketten INPUT, OUTPUT sowie FORWARD. Das heißt die zwei übrigen Ketten (PREROUTING, POSTROUTING) werden ausgeblendet, sobald der User die Filter-Tabelle ausgewählt hat. Der User kann den Befehl selber zusammenstellen, es wird aber automatisch darauf geachtet, dass dieser auch gültig ist. Durch den Ablauf der Erstellung des Befehls lernt der unerfahrene Benutzer nach und nach dem Aufbau eines iptables-Befehls kennen und beschäftigt sich mehr mit der Thematik als in Ansatz A, in der das Script im Hintergrund erstellt wird. Selbst definierte Ketten werden allerdings nicht berücksichtigt beziehungsweise können nicht berücksichtigt werden. Diese sind vergleichbar mit Funktionen in Programmen. Damit lassen sich zum Beispiel mehrere Fälle notwendige Abläufe in einer Regel zusammenstellen.

5.5.3 Entscheidung

Entscheidend für die Auswahl war das Gesamtkonzept dieser Bachelorarbeit. Mit diesem Programm soll man die ersten Schritte im Bereich iptables/Firewall kennenlernen. Daher wurde Ansatz B - die Selbstgestaltung von iptables-Regeln - realisiert, um dem Benutzer nicht eine Fülle von Befehlen zum Anklicken an die Hand zu geben, sondern durch das gezielte Zusammensetzen eines Befehls dabei auch noch die Syntax zu verinnerlichen. Dadurch entsteht gewollt ein Lerneffekt, da der Benutzer sich mit der eingegebenen Regel auch beschäftigt und die genaue Syntax sieht. Im anderen Ansatz würde die ausgewählte Regel automatisch im Hintergrund gleich in ein Script geschrieben oder in einem Textfenster zusammen mit den anderen Regeln aufgelistet werden. Die Frage ist, ob der Benutzer sich das Script noch einmal genau durchliest und versucht es zu verstehen, oder ob das Script gleich übernommen und an die Knoten geschickt wird.

5.5.4 Funktionsweise des Konfigurators

Der iptables-Konfigurator beinhaltet eine Reihe von iptables-Regeln, die per Combobox zusammengestellt werden können. Dabei wird automatisch darauf geachtet, dass die Regel

auch korrekt ist und keine Fehlermeldung zurück kommt, wenn die Regel angewendet werden. Iptables baut auf Tabellen und Ketten innerhalb dieser Tabellen auf. Für jede Tabelle gibt es bestimmte Ketten die zulässig sind. Die Filter-Tabelle legt zum Beispiel fest, welche Art von Traffic durch und aus dem Computer laufen darf. Die fest eingebauten Ketten hier sind INPUT, OUTPUT und FORWARD (PREROUTING und POSTROUTING gehören nicht zur Filter-Tabelle). Sobald diese Tabelle ausgewählt wird, werden automatisch die Ketten, mit denen die Filter-Tabelle nicht arbeitet, ausgeblendet. Das selbe gilt für alle anderen Optionen wie beim icmp-Protokoll (um ein weiteres zu nennen). Den icmp-type *echo-request* kann nur der Computer erkennen der gepingt wird. Für den entfernten PC ist es also eine eingehende Anfrage, und daher über die INPUT-Kette zu erkennen. Genau anders herum ist es beim echo-reply: Ein Paket mit dem icmp-type *echo-reply* wird von einem lokalen Prozess erzeugt und daher von der OUTPUT-Hook verarbeitet. Soll ein entfernter Computer gepingt werden, ist der request (ping) die Ping-Anfrage an diesen, die Antwort vom entfernten Computer heißt reply (pong). Es wird also nach und nach eine komplette iptables-Regel erstellt. Sobald man der Ansicht ist, die Regel ist fertig, wird dieses per Button bestätigt und die Regel erscheint in einem Textfenster. Von dort aus, kann man die Regel dann kopieren und zum Beispiel in der Netzwerkvirtualisierung direkt auf einem der Knoten ausführen.

Wie in 5.8 zu sehen ist, baut man nach und nach die iptables-Regel auf. Beim Starten des Konfigurators kann zuerst nur die Tabelle angegeben werden. Ist dies geschehen, kann man die Aktion (soll die Regel an eine Kette angehängt oder soll sie gelöscht werden) auswählen. So geht es weiter bis man die komplette Regel aufgestellt hat. Um die fehlerfreie Funktion zu gewährleisten, ist der Konfigurator so ausgelegt, dass bei einer Auswahl immer alle nachfolgenden Optionen, die ausgewählt werden können, gelöscht werden. Hat man also mal eine komplette Regel erstellt und wählt eine neue Tabelle aus, wird alles zurückgesetzt. Das heißt, alle Felder (action, chain, rules) werden gelöscht, damit dort nicht noch Angaben aus der vorherigen Regel stehen, die eventuell für die neue Tabelle nicht zulässig sind. Kurz gesagt: Wählt man in irgendeiner Combobox etwas aus, werden alle nachfolgenden Auswahlen und Comboboxes geleert. Möchte man weiter in der selben Tabelle arbeiten, und auch die Aktion soll beim neuen Befehl die selbe sein, ändert man einfach die Kette ab (wie in Abbilung 5.8 zu sehen, dritter Punkt). Die ersten beiden Auswahlen (Tabelle, Aktion) bleiben bestehen, und alles nachfolgende wird zurückgesetzt.

5.6 Installation und Ausführung

Vor Inbetriebnahme des Programms ist es wichtig, zwei Komponenten auf dem Wirtsystem zu installieren, ohne die das Programm nicht ausführbar ist. Zum einen ist es Python, und zum anderen wxWidgets. Was diese zwei Komponenten/Programme machen und warum sie installiert werden müssen wird im Folgenden erklärt.

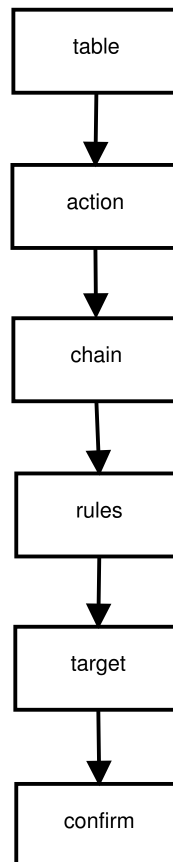

Abbildung 5.8: *Sequenzdiagramm: iptables-Konfigurator*

Nach der Installation der Pakete ist das Programm mit *python umlmanager.py* zu starten. Weitere Anweisungen und Hinweise zu Funktionen des Programms finden sie im Anhang (A).

5.6.1 Python

Da der Quellcode in Python 2.6 geschrieben wurde und Python nicht standardmäßig in einer Linuxdistribution enthalten ist, muss das Paket installiert werden, ansonsten ist das Programm nicht ausführbar. Wichtig ist auch, dass eine Python-Version vor 3.0 installiert wird, da es viele Neuerungen in Python 3.0 gibt und es möglich ist, dass nicht alles einwandfrei funktioniert.

5.6.2 GUI-Toolkit: wxWidgets - wxPython

wxWidget ist eine Klassenbibliothek zur Entwicklung grafischer Benutzeroberflächen, wie zum Beispiel auch Swing oder AWT in Java. Sie wurde in C++ entwickelt. Es gibt eine Vielzahl von Anbindungen an weitere Programmiersprachen, wie unter anderem auch für Python. wxPython ist der Wrapper dieses wxWidgets-Toolkits für Python. Es ist eine Alternative zu Tkinter, das mittlerweile zum Pythonpaket dazugehört. Mit wxPython lassen sich innerhalb der Programmiersprache grafische Benutzeroberflächen gestalten. Zu Beginn der Bachelorarbeit wurde die grafische Oberfläche noch mit wxGlade (GUI-Erstellungstool basierend auf wxWidgets) erstellt. Mit wxGlade ist es möglich eine komplette GUI per Drag & Drop zu erstellen. Weiterhin ist es möglich, die erstellte GUI direkt in Quellcode umzuwandeln und in das eigentliche Programm mit einzubauen. Es sind also nur noch Events zu kreieren, wie zum Beispiel: 'was soll passieren wenn der Button A geklickt wird'. Leider ist es sehr umständlich, damit bestimmte Komponenten der GUI genau an einem Ort zu platzieren, so dass nach zahlreichen Testen wxGlade wieder raus genommen und mit wxPython weitergearbeitet wurde. Die Buttons, Textfelder, Comboboxes etc. waren danach zwar selber zu erstellen, aber man konnte durch Eingabe der Koordinaten selber bestimmen, wo in der Oberfläche welches Item auftaucht. Die GUI wirkt dadurch optisch ansprechender und aufgeräumter. wxPython/wxWidgets ist also verwendet worden, um die grafische Benutzeroberfläche zu erstellen, daher muss dieses Paket vorher installiert werden.

6 Zusammenfassung

Die vorliegende Software realisiert die Erstellung eines virtuellen Netzwerkes mit einer zentralen grafischen Oberfläche. Die einzelnen Knoten des Netzes können mehrere Rollen einnehmen: Server, Endrechner, Firewall, Router etc. Der Benutzer kann vor dem Starten den Aufbau des Netzwerkes in einem Topologie-Script festlegen. Außerdem kann den einzelnen Knoten des virtuellen Netzes eine Grundkonfiguration geben (zum Beispiel IP-Adresse, Routing-Tabellen und Subnetze) werden. Diese sind in einem Initialisierungs-Script zusammengefasst. Beim Starten des virtuellen Netzes über die grafische Oberfläche wird das Initalisierungs-Script mit gestartet, liest aus dem Script die Konfigurationen der einzelnen Netzwerkknoten ein und konfiguriert diese dementsprechend. Die einzelnen Knoten des Netzes sind in der grafischen Oberfläche als Textfenster dargestellt (für jeden Knoten ein Textfenster) und sie können über die Fenster verwaltet werden. Es können Systembefehle an die Knoten gesendet werden sowie Einstellungen und Konfigurationen vorgenommen werden. Rückgabewerte und Antworten der Knoten werden wieder in dem jeweiligen Textfenster des Knotens angezeigt. Diese Textfenster simulieren die Consolen der einzelnen Knoten wie man sie aus Linux kennt. Die Kommunikation zwischen den Netzwerkknoten und dem - vom virtuellen Netz getrennten - Host (der GUI) wird über serielle Schnittstellen realisiert. Die beiden Seiten werden über ein Pseudoterminal miteinander verbunden.

Mit der vorliegenden Software lassen sich iptables-Scripte auf den Knoten ausführen. Es muss nicht für jeden einzelnen Knoten ein Script geschrieben werden, sondern es ist möglich, ein iptables-Script für alle Knoten zu erstellen und dieses dann über die zentrale Verwaltungseinheit an die Knoten zu schicken. Um das einwandfreie Ausführen dieser Scripte zu realisieren, gibt es einige Vorgaben, die einzuhalten sind, so dass die Regeln innerhalb der Scripte auch an die richtigen Knoten gesendet werden.

Aus den iptables-Scripten lassen sich Abfrage-Scripte generieren. Diese Abfrage-Scripte helfen, sobald es um die Lösung einer bestimmten Aufgabe geht. Ein Aufgabensteller kann ein iptables-Script zu einer von ihm erstellten Aufgabe schreiben (welches diese Aufgabe löst), und ein Abfrage-Script daraus generieren lassen und dieses abspeichern. Das Abfrage-Script hat dann folgenden Nutzen: Wenn ein Benutzer die Aufgabe löst und sein eigenes iptables-Script erstellt, kann der Aufgabensteller das iptables-Script des Nutzers mithilfe seines Abfrage-Script schnell und bequem testen. Ob die Aufgabe richtig gelöst wurde, ist dann in einer Auswertungsliste ablesbar. Falls nicht alle Teilbereiche gelöst wurden (es

wurde zum Beispiel eine Regel bei der Bearbeitung der Aufgabe vergessen), ist es ebenfalls in der Auswertung zu erkennen.

Als Zusatz hat das Programm noch einen kleinen iptables-Konfigurator. Damit lässt sich eine iptables-Regel einfach per Klick zusammenstellen.

Damit wurden die in der Einleitung (Kapitel 1) und der Anforderung (Kapitel 2) beschriebenen Themenbereiche erarbeitet und erfüllt. Im nächsten Kapitel folgt nun einen Ausblick auf die Erweiterungsmöglichkeiten dieses Programmes.

7 Ausblick

Mit der vorliegenden Software lassen sich natürlich nicht nur Auswirkungen von iptables-Befehlen/Regeln simulieren. Durch die Netzwerkvirtualisierung kann man jegliche Einstellungen, das Netzwerk betreffend, testen. Alles, was in einem reellen Netzwerk stattfindet, kann auch hier simuliert werden, da die Software das Grundgerüst hierzu bildet.

Zum Beispiel lässt sich das Thema Sicherheit in der Virtualisierung gut testen. Da User-Mode-Linux auch dazu dient, Sicherheitsaspekte zu überprüfen, ist es natürlich auch möglich, Honeypots zu installieren. Honeypots sind bewusst in Kauf genommene Schwachstellen im Sicherheitssystem, um so potentielle Angreifer abzulenken. Ein Honeypot kann zum Beispiel das Netzwerk, Dienste oder das Verhalten eines Anwenders simulieren. Angriffe auf den Honeypot werden protokolliert, um die Vorgehensweise und Verhaltensmuster der Angreifer herauszufiltern. Das reale Netzwerk bleibt dadurch verschont, da es besser gesichert ist.

Da iptables eine Vielfalt von Regeln beinhaltet, kann das vorliegende Programm beliebig erweitert werden. Es können neue, automatisch generierte, Abfragen implementiert werden. Zu schauen ist dabei nur, wie man eine bestimmte iptables-Regel richtig abfragt. Dieses kann danach im Quellcode hinzugefügt werden. Möglich wäre auch, eine Applikation zu schreiben, die von Benutzer erstellte Abfrage von iptables-Regeln automatisch in das Programm einpflegt, um es so nach und nach zu erweitern. So sind zum Beispiel nur die iptables-Regel sowie die Abfrage einzugeben, die Applikation erstellt daraus automatisch Code und fügt diesen in das Programm ein.

Der Konfigurator kann ebenfalls beliebig durch Hinzufügen weiterer iptables-Regeln erweitert werden. Der Quellcode ist, dank python, schnell zu verstehen. Das Hauptaugenmerk muss nur auf den Aufbau der iptables-Regel gelegt werden. Was folgt an welcher Stelle innerhalb einer iptables-Regel.

Ebenfalls könnte die grafische Oberfläche noch so erweitert werden, dass sich auch das Netzwerk grafisch zusammenstellen lässt. Es gibt eine Liste mit Symbolen für Clients und Verbindungen und diese können per Drag & Drop verteilt werden, so dass das Netzwerk gezeichnet und später daraus erstellt wird. Sozusagen als Ersatz für das derzeitige Topologie-Script. Es würde die Arbeit beim Zusammenstellen des Netzwerkes erleichtern, wenn nicht Codezeilen zu ändern sind, sondern das Netzwerk immer gezeichnet wird. Das Netzwerk

wird mit einfachen Formen visuell dargestellt, beim Starten wird das Diagramm eingelesen und in Code umgewandelt. Daraus entsteht dann das virtuelle Netz. Die Knoten können als Quadrat und die Verbindungen der Knoten als Linien dargestellt werden. In diesem Diagramm könnten noch viele weitere Einstellungen untergebracht werden. Man könnte zum Beispiel die Knoten anwählen und ihnen IP-Adressen geben. Die ganze Vorkonfiguration könnte man also statt mit Scripten mit Diagrammen machen, in denen Konfigurationen etc. festgelegt werden.

Der Bereich Auswertung lässt sich auch erweitern. Vorstellbar wäre zum Beispiel eine grafische Ausarbeitung, in der auf einen Blick zu erkennen ist, wo es Fehler bei der Bearbeitung gab (sofern welche auftraten). Gelöste Vorgaben könnten als Prozentzahl ausgegeben werden, so dass gleich erkennbar ist, wieviel Prozent der Ausarbeitung mit der Aufgabenstellung übereinstimmen.

Einige dieser Erweiterungen sind mit großem, andere mit geringerem Aufwand zu lösen. In jedem Fall aber steht durch die vorliegende Arbeit das Grundgerüst dafür.

Literaturverzeichnis

[Assouroko 2010] ASSOUROKO, Enoch: *Java-basiertes Managementsystem zur Konfiguration und Steuerung von Routingalgorithmen in virtuellen Netzen*. 2010

[Barrett, Daniel J. 2004] BARRETT, DANIEL J.: *Linux kurz und gut*. Köln : O'Reilly Verlag, 2004. – ISBN 978-3-8972-1501-6

[Barth, Wolfgang 2001] BARTH, WOLFGANG: *Das Firewall Buch*. Nürnberg : SuSE PRESS, 2001. – ISBN 3-934678-40-8

[Bauer 2009] BAUER, Tobias: *IPTables-Packetfilter-Script*. 2009. – URL http://www.tobias-bauer.de/computer/iptables/index.html. – Zugriffsdatum: 28. November 2010

[Dike, Jeff 2006] DIKE, JEFF: *User-Mode-Linux*. New Jersey : Prentice Hall, 2006. – ISBN 978-0-13-186505-1

[Google Inc. 1998] GOOGLE INC.: *Google*. 1998. – URL http://www.google.de. – Zugriffsdatum: 05. Januar 2011

[Kaiser, Peter und Ernesti, Johannes 2008] KAISER, PETER UND ERNESTI, JOHANNES: *Python*. Bonn : Galileo Computing, 2008. – ISBN 978-3-8362-1110-9

[Lutz, Mark und Schulten, Lars 2010] LUTZ, MARK UND SCHULTEN, LARS: *Python kurz und gut*. Köln : O'Reilly Verlag, 2010. – ISBN 978-3-8972-1556-6

[Mandt 2007] MANDT, Tarjei: *Simple Iptables Script Generator*. 2007. – URL http://www.mista.nu/iptables/. – Zugriffsdatum: 29. November 2010

[Marmorstein und Kearns 2005] MARMORSTEIN, Robert ; KEARNS, Phil: *A Tool for Automated iptables Firewall Analysis*. 2005. – URL http://www.usenix.org/event/usenix05/tech/freenix/full_papers/marmorstein/marmorstein_html/. – Zugriffsdatum: 17. Dezember 2010

[Microsoft 2010] MICROSOFT: *Virtual PC*. 2010. – URL http://www.microsoft.com/windows/virtual-pc/. – Zugriffsdatum: 15. Dezember 2010

[Nessus 2010] NESSUS: *Nessus*. 2010. – URL http://www.nessus.org. – Zugriffsdatum: 20. Dezember 2010

[NMAP 1997] NMAP: *Nmap Security Scanner*. 1997. – URL http://www.nmap.org. – Zugriffsdatum: 21. Dezember 2010

[Peters 2004] PETERS, Tim: *The Zen of Python*. 2004. – URL http://www.python.org/dev/peps/pep-0020/. – Zugriffsdatum: 26. Dezember 2010

[Purdy, Gregor N. 2005] PURDY, GREGOR N.: *LINUX iptables kurz und gut*. Köln : O'Reilly Verlag, 2005. – ISBN 3-89721-506-3

[Spenneberg, Ralf 2006] SPENNEBERG, RALF: *Linux-Firewalls mit iptables und Co*. München : Addison-Wesley, 2006. – ISBN 978-3-8273-2136-7

[tcpdump 2010] TCPDUMP: *tcpdump*. 2010. – URL http://www.tcpdump.org. – Zugriffsdatum: 21. Dezember 2010

[Telematics Engineering Department 2002] TELEMATICS ENGINEERING DEPARTMENT: *VNUML - Virtual Network User Mode Linux*. 2002. – URL http://www.dit.upm.es/vnumlwiki/index.php/Main_Page. – Zugriffsdatum: 04. Dezember 2010

[Thorns, Fabian 2008] THORNS, FABIAN: *Das Virtualisierungs-Buch*. Böblingen: Computer- und Literaturverlag, 2008. – ISBN 978-3-9365-4656-9

[User-Mode-Linux] USER-MODE-LINUX: *User-Mode-Linux*. – URL http://user-mode-linux.sourceforge.net. – Zugriffsdatum: 28. Dezember 2010

[User-Mode-Linux, What are people using it for] USER-MODE-LINUX, WHAT ARE PEOPLE USING IT FOR: *What are people using it for*. – URL http://user-mode-linux.sourceforge.net/old/uses.html. – Zugriffsdatum: 19. Dezember 2010

[VirtualBox] VIRTUALBOX: *VirtualBox*. – URL http://virtualbox.org. – Zugriffsdatum: 15. Dezember 2010

[VMWare] VMWARE: *VMWare*. – URL http://vmware.com. – Zugriffsdatum: 15. Dezember 2010

[YouTube, LLC 2005] YOUTUBE, LLC: *YouTube*. 2005. – URL http://www.youtube.com. – Zugriffsdatum: 05. Januar 2011

Teil I

Anhang

A Hilfe

A.1 Vorbereitung und Starten der GUI

Bevor die GUI gestartet wird, ist zuerst der Aufbau des Netzwerkes zu überlegen und eventuell zu skizzieren. In der Datei topo.py kann das Netzwerk eingepflegt werden. Beim Starten des Netzwerks wird es automatisch aus dieser Datei ausgelesen. In der Zeile ifcs = [x , xx, xxx] werden die einzelnen Knoten angegeben. Soll ein Netzwerk aus 4 Knoten bestehen, ist die Zeile zu ändern auf: ifcs = [r1 , r2 , r3 , r4]. Wie diese vier Knoten untereinander verbunden sind, ist darüber festzulegen. Für jeden Knoten wird eine Zeile angelegt uns es muss angegeben werden, welches Interface an welches Netz angebunden ist. In Abbildung A1 sehen sie ein Beispielnetzwerk mit vier Knoten.

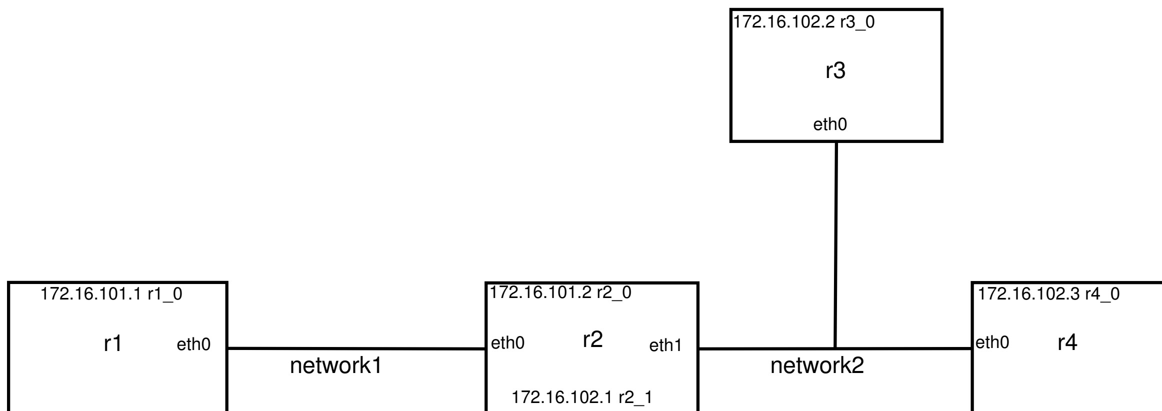

Abbildung A1: *Beispielaufbau eines Netzwerks mit vier Knoten*

Die zu diesem Netzwerk gehörige topo.py muss dann wie folgt aussehen:

```
1 r1 = [ ['eth0','network1']                                  ]
  r2 = [ ['eth0','network1'],['eth1','network2']       ]
3 r3 = [ ['eth0','network2']                                  ]
  r4 = [ ['eth0','network2']                 ]
5 ifcs = [ r1 , r2 , r3, r4 ]
```

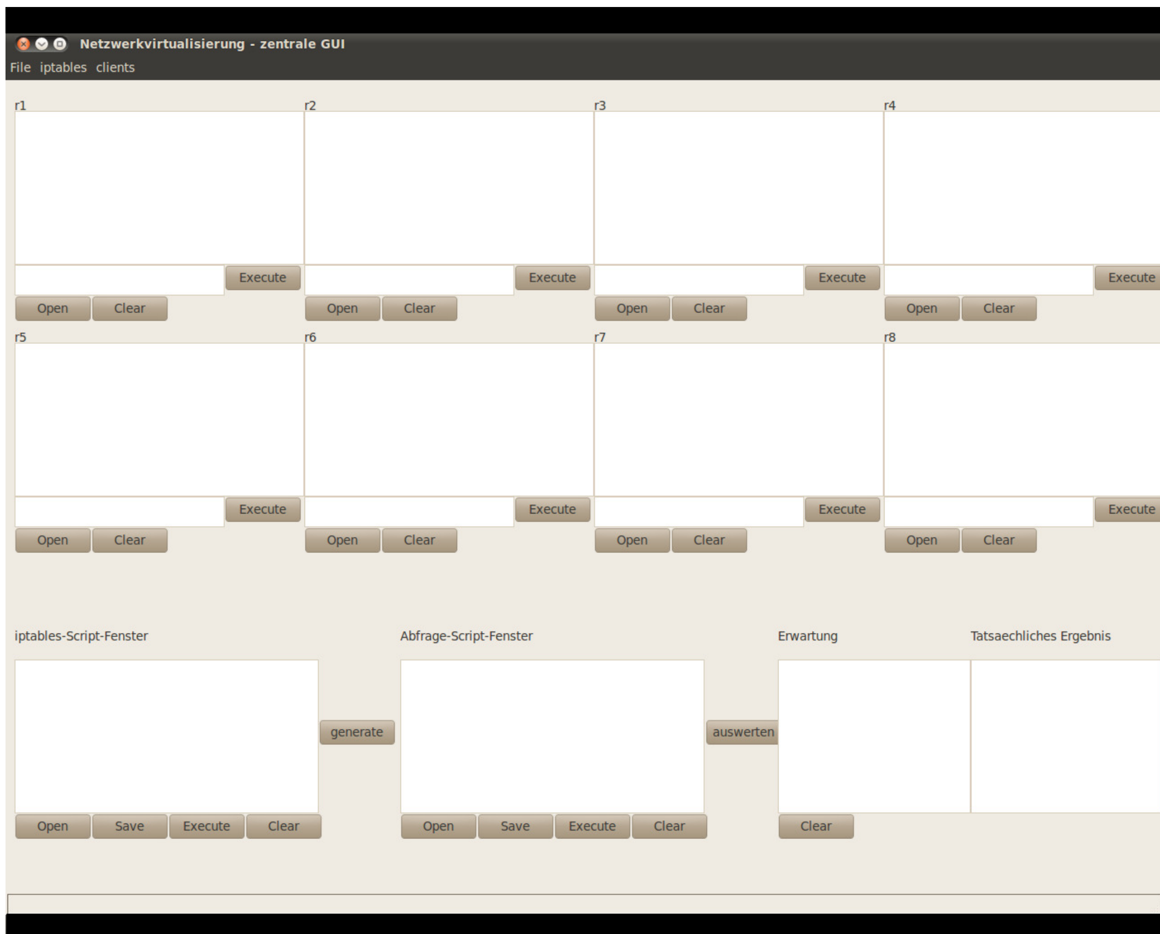

Abbildung A2: *Die grafische Oberfläche (GUI)*

Weiterhin können die IP-Adressen für die Knoten im Initialisierungsscript festgelegt werden, damit das nicht später in der Virtualisierung noch zu tun ist. Um Zugriff über die Namen der Knoten zu gewährleisten sollte die hosts-Datei editiert werden. Im vorliegenden Init-Script ist dies aber schon gemacht worden. Es kann also gleich losgelegt werden kann.

Gestartet wird das Programm danach per 'python umlmanager.py'

A.2 Arbeiten in der GUI

Im Folgenden wird gezeigt, wie mit und innerhalb der grafischen Oberfläche zu arbeiten ist. Die GUI (siehe Screenshot in Abbildung A2) besteht aus Textfenstern für die Knoten die jeweils eine Befehlszeile haben (siehe Konzept der GUI in Abbildung 5.2), sowie einem

Fenster für iptables-Scripte, einem Fenster für Abfragescripte und einem geteilten Fenster für die Auswertung. Dazu kommen eine Menge von Buttons zum Löschen der einzelnen Fenster, zum Ausführen von Befehlen oder Sripten etc.

A.2.1 Starten und Beenden des Netzwerks

Nachdem wie vorher beschrieben die Einstellungen für das Netzwerk vorgenommen sind, kann nun das Netzwerk gestartet werden, indem in der Menüleiste auf Clients und dort 'start Clients' angeklickt wird. Die einzelnen UMLs werden nun geladen. Je mehr Knoten eingestellt sind, desto länger dauert der Initialisierungsvorgang. Es öffnen sich pro Knoten dann noch extra eine virtuelles Console, mit denen man arbeiten kann, aber nicht muss, da die GUI auch die zentrale Verwaltung bietet. Allerdings lassen sich einige Befehle aufgrund des Designs nicht ausführen. Weiterhin kann über diese virtuellen Consolen das umlclient-Script neugestartet werden, falls es beendet wurde (Befehl: python /mnt/umlclient.py)

Um das Netzwerk zu beenden, gehen Sie bitte im Menü auf Clients -> close Clients.

A.2.2 Die Befehlszeile

Über die Befehlszeile der einzelnen Knoten innerhalb der GUI lassen sich Aufrufe, Befehle oder Abfragen an den Knoten schicken. Einfach in die Befehlszeile eingeben und danach auf den danebenliegenden Execute-Button klicken. Danach wird der Befehl abgeschickt. Wichtig zu beachten ist, dass die Eingaben nach einer endlichen Zeit beendet sind oder einen Wert zurückgeben, ansonsten ist der Knoten nicht mehr über die GUI erreichbar, da er noch arbeitet. Zum Beispiel ist ein ping wie 'ping 172.16.101.1' unter Linux eine unendliche Abfrage. Der Knoten ist nicht zu erreichen bis eine Rückgabe kommt. Es ist also nicht möglich über die GUI diesen Befehl abzubrechen. Daher beim ping immer darauf achten mit angeben, wie oft gepingt werden soll. 'ping -c4 172.16.101.1' oder 'ping -c2 172.16.101.1' pingt das jeweilige Ziel 4 bzw 2 mal. Danach kann man weitere Befehle an den Knoten senden. Da bestimmte Befehle einige Zeit in Anspruch nehmen bevor sie etwas zurückgeben, ist in dieser Zeit das Programm für weitere Interaktion gesperrt. Gerade wenn man später ein Abfragescript mit vielen ping Befehlen an die Knoten sendet, kann es unter Umständen zu längeren Wartezeiten kommen.

A.2.3 Die Textfenster

In den einzelnen Textfenstern werden die Rückgaben, Ausgaben und Antworten der Knoten dargestellt. Über die Befehlszeile gesendete Abfragen werden, sobald es einen Rückgabe-

wert gibt, in diesem Textfenster angezeigt. Um die Lesbarkeit zu garantieren ist die Schrift groß dargestellt. Daher passt in ein Textfenster nicht soviel Text, da es daneben noch viele weitere Fenster geben kann, je nachdem, wie viele Knoten das Netzwerk hat. Daher kann man die einzelnen Textfenster per Clear-Button auch leeren.

A.2.4 Scripte ausführen

Scripte mit Befehlen haben in diesem Programm immer folgenden Aufbau:

r1
Befehl a
Befehl b
r2
Befehl c
Befehl d
Befehl e
r3
Befehl f
Befehl g
usw...

Es wird immer zuerst der Knoten angegeben an den die nächsten Befehle geschickt werden sollen (hier als erstes 'r1'). Danach folgen die Befehle (einer pro Zeile) die an 'r1' gesendet werden sollen. Danach wird im selben Prinzip mit den weiteren Knoten fortgefahren. In Abbildung A3 sehen sie ein Beispiel eines kurzen Scripts.

A.2.5 Abfragen generieren

Um eine Abfrage aus einem erstellten iptables-Script zu generieren, öffnen sie im iptables-Script-Fenster entweder ein eventuell zuvor abgespeichertes Script oder schreiben sie ein neues. (es muss den Vorgaben aus Kapitel A.2.4 folgen). Mit einem Klick auf *generieren* wird automatisch das Abfrage-Script im Textfenster daneben erstellt. Dieses kann nun abgespeichert oder aber direkt auf den Knoten ausgeführt werden.

```
r1
iptables -A INPUT  -p icmp --icmp-type echo-request -s 172.16.101.2 -j ACCEPT
r2
iptables -A OUTPUT  -p icmp --icmp-type echo-reply -d 172.16.101.1 -j ACCEPT
iptables -A FORWARD -p tcp --dport 22 -s 172.16.101.0/24 -d r3_0 -j ACCEPT
iptables -A FORWARD -p tcp --sport 22 -d 172.16.101.0/24 -s r3_0  -j ACCEPT
iptables -A FORWARD -p tcp --dport 22 -s 172.16.104.0/24 -d r3_0 -j ACCEPT
iptables -A FORWARD -p tcp --sport 22 -d 172.16.104.0/24 -s r3_0  -j ACCEPT
r3
iptables -A INPUT  -p icmp --icmp-type echo-request -s 172.16.102.3 -j ACCEPT
iptables -A OUTPUT  -p icmp --icmp-type echo-reply -d 172.16.102.3 -j ACCEPT
r4
iptables -A OUTPUT  -p icmp --icmp-type echo-reply -d 172.16.102.2 -j DROP
iptables -A INPUT  -p icmp --icmp-type echo-request -s 172.16.102.2 -j DROP
```

Abbildung A3: *Beispiel eines iptables-Scripts*

A.2.6 Auswertung

Um eine Auswertung zu starten, sind drei Vorgaben, beziehungsweise Schritte, zu erfüllen und nacheinander abzuarbeiten. Ohne diese ist eine Auswertung nicht möglich (siehe Kapitel 5.4.3 für eine detaillierte Auflistung).

- 1. Es muss ein Abfragescript im dazugehörigen Fenster vorhanden sein (entweder ein geladenes oder ein gerade erstelltes Script)
- 2. Alle Textfenster der Knoten müssen einmal geleert werden.
- 3. Das Abfragescript muss ausgeführt werden.

Nun kann über den 'Auswerten-Button' die Auswertung beginnen. Auf der linken Seite steht das erwartete Ergebnis und auf der rechten Seite das tatsächliche Ergebnis. Weicht das tatsächliche vom erwarteten Ergebnis ab, ist das zugehörige iptables-Script an dieser Stelle fehlerhaft in Bezug auf eine bestimmte Aufgabenstellung.

A.2.7 iptables-Konfigurator

Der iptables-Konfigurator kann im Menü unter iptables -> Konfigurator gestartet werden. Es erscheint hierfür eine neue GUI, auf der nach und nach die iptables-Regel ausgewählt werden kann. Es beginnt bei der Auswahl der Tabelle, über das Kommando, die Kette, den Parametern bis hin zum Target. Hat man die komplette iptables-Regel entworfen, einmal den

Ausführen-Button betätigen und die Regel wird in der Textzeile unten aufgeführt. Diese kann man nun kopieren und zum Beispiel auf einem der Knoten im Netzwerk ausführen lassen.

A.3 Filesystem Update

Um ein Update aufzuspielen, zum Beispiel um neue Programme und Anwendungen zu installieren, gehen sie bitte wie folgt vor: Auf dem Hostrechner sind einige Einstellung die im Folgenden gelistet sind, vorzunehmen. Öffnen sie eine Console und tippen sie folgende Befehle als root ein:

- modprobe tun
- mkdir /dev/net
- mknod /dev/net/tun c 10 200
- chmod 777 /dev/net/tun
- tunctl -u <username> -t t0
- ifconfig t0 10.0.1.1
- iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
- echo 1 ›/proc/sys/net/ipv4/ip_forward

Danach als *<username>*:

- bin/linux_2.6.34 ubd0=bin/fs.lenny mem=60M name=r8 eth7=tuntap,t0„10.0.1.1 root=98:0

eingeben.

Das virtuelle Linux wird gestartet, indem sie auf der Console bitte folgendes eingeben:

- ifconfig eth7 10.0.1.2
- route add default gw 10.0.1.1
- echo nameserver xxx.xxx.xxx.xxx ›/etc/resolve.conf

Hier ersetzen sie bitte xxx.xxx.xxx.xxx mit der IP ihres Namensservers. Danach sollte ein ping auf eine Internetadresse wie beispielsweise google.de möglich sein. Nun können sie im virtullen Linux updates und upgrades machen, mit den üblichen Kommandos wie

- apt-get update
- apt-get upgrade

- apt-get install <paket>
- apt search

Versicherung über Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. Januar 2011	
Ort, Datum	Unterschrift