

# DIPLOMARBEIT

Eine auf Checklisten basierende Methode  
zur Realisierung hochverfügbarer, sicherer  
und performanter Open Source-Webportale  
am Beispiel des MNI-Portals  
der Fachhochschule Gießen-Friedberg



**zur Erlangung des akademischen Grades  
Diplominformatiker (FH)**

vorgelegt dem

Fachbereich Mathematik, Naturwissenschaften und Informatik  
der Fachhochschule Gießen-Friedberg

von

**Tim Pommerening**

im Juli 2004

Referent: Prof. Dr.-Ing. Klaus Quibeldey-Cirkel

Korreferent: Prof. Dr. Thomas Letschert

# Inhaltsverzeichnis

<b>1</b>	<b>Vorwort</b>	<b>5</b>
<b>2</b>	<b>Einführung</b>	<b>6</b>
2.1	Motivation und Aufgabenstellung . . . . .	6
2.2	Kapitelübersicht . . . . .	8
2.3	Typographische Konventionen . . . . .	10
<b>3</b>	<b>Grundlagen</b>	<b>11</b>
3.1	Rechnerlandschaft des Testportals . . . . .	11
3.2	Eigenschaften der verwendeten Rechner . . . . .	12
3.3	Aufbau einer Entwicklungsumgebung . . . . .	14
3.3.1	Hardware . . . . .	15
3.3.2	Software . . . . .	16
<b>4</b>	<b>Performance</b>	<b>18</b>
4.1	Lastverteilungsverfahren . . . . .	21
4.1.1	DNS-Round Robin . . . . .	22
4.1.2	Network Address Translation . . . . .	22
4.1.3	Direct Routing . . . . .	24
4.1.4	IP-Tunneling . . . . .	26
4.2	Scheduling-Algorithmen . . . . .	28
4.3	Einrichten der Lastverteiler . . . . .	29
4.3.1	Network Address Translation . . . . .	31
4.3.2	Direct Routing . . . . .	33
4.3.3	IP-Tunneling . . . . .	35
4.3.4	Konfigurationsbeispiel Fachbereich MNI . . . . .	36
4.4	Einrichten der Realserver . . . . .	37
4.4.1	Network Address Translation . . . . .	37
4.4.2	Direct Routing . . . . .	37
4.4.3	IP-Tunneling . . . . .	38
4.5	Das ARP-Problem . . . . .	40
4.6	Realserver-Konfigurationstool . . . . .	43
4.7	Lasttests . . . . .	44
4.7.1	Testkonfiguration . . . . .	44
4.7.2	Tests . . . . .	46
4.7.3	Fazit . . . . .	53

<b>5</b>	<b>Hochverfügbarkeit</b>	<b>54</b>
5.1	Heartbeat . . . . .	55
5.1.1	Installation . . . . .	55
5.1.2	Konfiguration . . . . .	56
5.1.3	Heartbeat und Webmin . . . . .	60
5.2	Mon . . . . .	61
5.2.1	Installation von Mon . . . . .	61
5.2.2	Ausfallsicherung der Realserver mit Mon . . . . .	64
5.3	LDIRECTORD . . . . .	65
5.4	IPVSADM Client/Server . . . . .	66
5.5	Datensynchronisation . . . . .	67
5.5.1	Datenbanksynchronisation . . . . .	68
5.5.2	Dateisynchronisation . . . . .	75
<b>6</b>	<b>Sicherheit</b>	<b>77</b>
6.1	Datenschutz . . . . .	78
6.1.1	Risikoanalyse . . . . .	78
6.1.2	Firewalls . . . . .	79
6.2	Datensicherung . . . . .	83
<b>7</b>	<b>Checklisten</b>	<b>86</b>
7.1	QA-Portal . . . . .	86
7.2	Evaluationschecklisten . . . . .	88
<b>8</b>	<b>Abschließende Betrachtung</b>	<b>90</b>
8.1	Zusammenfassung . . . . .	90
8.2	Ausblick . . . . .	90
<b>A</b>	<b>Quellcode</b>	<b>92</b>
A.1	Startskriptvorlage für Heartbeat . . . . .	92
A.2	Mon-Installationsskript . . . . .	94
A.3	Mon-Skript für Linux-Virtual-Server . . . . .	95
A.4	Rsync-Skript für Dateiabgleich . . . . .	95
A.4.1	/etc/globals . . . . .	95
A.4.2	/lib/common_functions . . . . .	96
A.4.3	Rsync-Skript . . . . .	97
A.5	Backupskripten . . . . .	99
A.5.1	Konfigurationsdatei (backupconst.cfg) . . . . .	99
A.5.2	Skript zum Löschen alter Backups (backupdel) . . . . .	99
A.5.3	Backupskript für MySQL Datenbanken . . . . .	102
A.5.4	Backupskriptvorlage für Verzeichnisse . . . . .	102
A.6	Firewall-Skript . . . . .	102
A.6.1	Konfigurationsdatei <i>ports.tcp</i> . . . . .	104
A.6.2	Konfigurationsdatei <i>ports.udp</i> . . . . .	104
<b>B</b>	<b>CD zur Diplomarbeit</b>	<b>106</b>

<b>Abbildungsverzeichnis</b>	<b>108</b>
<b>Tabellenverzeichnis</b>	<b>110</b>
<b>Literaturverzeichnis</b>	<b>111</b>
<b>Index</b>	<b>113</b>

# Kapitel 1

## Vorwort

Diese Arbeit hat mich auf ein interessantes Themengebiet der Systemtechnik geführt, welches in Zukunft sicher immer mehr an Bedeutung gewinnen wird. Ich konnte durch die Bearbeitung mein Wissen über einen breit gefächerten und bunten Aufgabenbereich erweitern und bin der Meinung, dass es sich gelohnt hat, dieses Thema aufzugreifen.

Damit es nicht in Vergessenheit gerät, möchte ich hier die Gelegenheit beim Schopfe packen und einigen Personen danken, die ihren Teil dazu beigetragen haben, das in dieser Arbeit besprochene Projekt durchzuführen.

Ich bedanke mich bei meinen Betreuern Prof. Dr.-Ing. Klaus Quibeldey-Cirkel und Prof. Dr. Thomas Letschert für das mir entgegengebrachte Vertrauen. Prof. Dr.-Ing. Klaus Quibeldey-Cirkel danke ich außerdem für die hilfreichen Diskussionen und Tipps, sowie die Unterstützung, die er mir im Laufe der gesamten Arbeit gewährte.

Weiterer Dank geht an Gerhard Franke, Administrator im Fachbereich MNI für die Hilfe bei sämtlichen technischen Fragen und Problemen im Bezug auf die verwendete Testumgebung.

Martin Müller danke ich für seinen umfassenden Korrekturdurchgang und die vielen daraus resultierenden virtuellen gelben Klebezettel mit Anmerkungen und Hinweisen. Das hat mir bei den Nacharbeiten der Arbeit sehr geholfen.

Für die Unterstützung und den Rückhalt während des Studiums und auch in der Zeit davor danke ich meinen Eltern Elke und Rüdiger Pommerening.

Tim Pommerening

am 28. Juli 2004

# Kapitel 2

## Einführung

„Aller Anfang ist schwer, am schwersten der Anfang der Wirtschaft.“

– Johann Wolfgang Goethe

Dieses Kapitel gibt die Motivation und Aufgabenstellung für die Diplomarbeit wieder. Anschließend wird eine kurze Vorschau über die folgenden Kapitel und somit die in der Diplomarbeit behandelten Themen gegeben. Den Abschluss des Kapitels bildet eine Übersicht der in dieser Arbeit verwendeten typographischen Konventionen.

### 2.1 Motivation und Aufgabenstellung

Webauftritte sollen *hochverfügbar*, *performant* und *sicher* sein. Dabei sollen die Antwortzeiten *skalieren*. Das alles soll mit möglichst geringen Kosten realisiert werden. Da aber die Software der Marktführer im Allgemeinen sehr teuer ist, wäre es wünschenswert Alternativen zu haben, die bei geringen Kosten mit Markenprodukten vergleichbare Leistungen anbieten. Im so genannten Open Source-Bereich<sup>1</sup> findet sich fast zu jeglicher Thematik Software, die sehr häufig den Marktführerprodukten um nichts nachsteht oder zumindest die gewünschten Mindestanforderungen erfüllen.

Im Rahmen der Lehrveranstaltung *Open Source Portale* des Fachbereichs Mathematik, Naturwissenschaften und Informatik (MNI) der Fachhochschule Gießen-Friedberg fand im Wintersemester 2003/2004 bei Herrn Prof. Dr. Quibeldey-Cirkel eine Evaluation von Webportalen statt. Bekannte und weniger bekannte Portale wurden daraufhin getestet, dem Anspruch an ein Fachbereichsportal für eine Hochschule zu genügen. Nachdem aus einer Vielzahl von Kandidaten der Sieger *Typo3*<sup>2</sup> gekürt worden war, wurde noch eine geeignete Plattform benötigt, auf der das Portal letztendlich laufen sollte. Ohne eine performante Basis ist jede noch so ergonomische und benutzerfreundliche Portalsoftware nutzlos. Die Statistik besagt, dass Besucher einer Website durchschnittlich bis maximal sieben Sekunden auf die Anzeige der Seite warten, bevor sie wegklicken und sich anderweitig umsehen<sup>3</sup>. Das mag bei einem

<sup>1</sup>Open Source <http://www.opensource.org/>

<sup>2</sup>Typo3 Homepage <http://www.typo3.org>

<sup>3</sup>„Eine noch so interessante und spektakuläre Website hat am Markt verloren, wenn lange Ladezeiten die Nutzer abschrecken. Untersuchungen haben ergeben, dass Wartezeiten von mehr als sieben Sekunden die Mehrheit der Nutzer bereits zum Verlassen der Site veranlassen.“ [Zsc04]

Fachbereichsportal kein Grund zur Beunruhigung sein, im *e-commerce* Bereich ist dieser Umstand jedoch mit Einnahmeverlusten gleichzusetzen.

Die nachfolgende Arbeit soll zeigen, inwiefern die oben genannten Ziele mit kostenlosen Open Source-Produkten erreicht werden können, welche Lösungen sich anbieten und worauf zu achten ist. Innerhalb des Projektes soll eine solide Basis für das neue Webportal des Fachbereichs MNI der Fachhochschule Gießen-Friedberg geschaffen werden. Die Plattform soll dabei aber möglichst nicht auf eine bestimmte Portalsoftware ausgelegt, sondern universell einsetzbar sein. Die angesprochene Hochverfügbarkeit, Performanz und Skalierbarkeit der Portalplattform soll dabei durch Lastverteilung im Zusammenhang mit Ausfalllösungen erreicht werden. Geplant ist ein System mit folgendem Aufbau. Ein Lastverteiler nimmt die Anfragen an das Portal aus dem Internet entgegen und verteilt diese an zwei Rechner, auf denen sich die Webserver und Datenbanken befinden. Muss einer dieser beiden Rechner gewartet werden oder fällt dieser aus, leitet der Loadbalancer die Anfragen automatisch nur an den anderen Rechner weiter. Fällt der Loadbalancer selbst aus, soll dies über eine Überwachungssoftware erkannt werden. In diesem Fall soll ein Backup-Loadbalancer an dessen Stelle treten, der mit den gleichen beiden Webserver-Rechnern verbunden ist. Obwohl das Testsystem nur aus zwei Loadbalancern und zwei realen Webservern besteht, sollte das Loadbalancer-System so ausgelegt sein, dass es für  $n$  Realserver verwendet werden kann. Das System soll auf einer oder unterschiedlichen Linux-Plattformen betrieben werden. Dabei sollen die verschiedenen Server-Rechner vollständig fernadministriert werden können.

Das Ziel dieser Arbeit besteht nicht darin, die Basis für ein bestimmtes Fachbereichsportal mit bestimmter Software und bestimmten Anforderungen zu legen. Ziel ist es vielmehr, eine Checkliste zu entwickeln, die auf jede Art von Webportalen angewendet werden kann. Betreiber von Portalen sollen diese Checkliste aktiv nutzen, um die Plattform auf der ihre Portale laufen zu bewerten und anhand von Tipps und Anleitungen zu verbessern. Die Portalplattform des Fachbereichs MNI wird hierbei für Beispiele und als praktische Vorlage verwendet.

Es folgt eine Auflistung der zu bearbeitenden Aufgaben:

### **Open Source**

Für die Umsetzung der folgenden Punkte sollen nur Komponenten verwendet werden, die aus dem Open Source-Bereich stammen. Ziel ist es, eine Plattform zu schaffen, die möglichst wenig Kosten verursacht.

### **Plattform**

Zu allererst ist es notwendig, die Grundplattform - also die benötigte Hard- und Software - zu planen und für seine Bedürfnisse anzupassen. Dabei soll die im Fachbereich vorhandene Infrastruktur genutzt werden.

### **Hochverfügbarkeit**

Der Hochverfügbarkeitsaspekt ist wichtig für jedes kommerzielle Webportal. Es werden Aspekte herausgearbeitet und Möglichkeiten aufgezeigt, wie eine möglichst hohe Verfügbarkeit der Dienste gewährleistet werden kann.

### Performance

Genauso wichtig ist die Performance des Systems. Skaliert ein System nicht bei steigender Benutzerzahl, ist es kommerziell nicht einsetzbar. Es werden Möglichkeiten herausgearbeitet, um die Skalierbarkeit zu steigern und die allgemeine Performanz zu verbessern.

### Sicherheit

Absolute Sicherheit gibt es in keinem System, auf das im Grunde jeder Mensch Zugriff hat. Heute wird davon ausgegangen, dass fehlerfreie Software nicht existiert. Daher muss immer mit neuen Sicherheitslöchern gerechnet werden. Aufgabe ist es allerdings, Methoden aufzuzeigen, wie die Portalbasis so abgesichert werden kann, dass die ohnehin verfügbaren Möglichkeiten ausgenutzt sind. Dabei geht es nicht nur um Sicherheit vor Angriffen aus dem Netz, sondern auch um die Datensicherheit.

### Lasttests

Lasttests sind der Hauptbestandteil des Performancemanagements. Eingesetzt werden sie um zu planen, wie lange ein System noch skaliert oder um herauszufinden, weswegen ein System nicht skaliert. Dabei können Flaschenhälse aufgedeckt und die allgemeine Performance verbessert werden.

### Checklisten

Um dem Betreiber eines Webportals zu helfen, seine vorhandene Plattform in puncto Sicherheit, Hochverfügbarkeit und Performanz zu verbessern, kann er sich über diese Checklisten eine Bewertungsgrundlage schaffen. Es soll eine Reihe von Fragen geben, die der Betreiber mit *Ja* oder *Nein* beantworten kann und anschließend eine Auswertung bekommt. Zu nicht erfüllten Punkten soll der Betreiber gleich die nötigen Informationen anhand von Beispielen und Quellen erhalten, damit Missstände abgestellt werden können.

Die aufgezeigten Aufgaben gehen natürlich teilweise ineinander über. So muss bspw. bei der Installation der Grundplattform auch auf Sicherheitsaspekte eingegangen werden. Es kann also nicht davon ausgegangen werden, dass alle genannten Aufgaben für sich abgegrenzte Bereiche darstellen. Die angesprochenen Checklisten finden sich nicht in diesem Dokument. Es handelt sich dabei um ein interaktives Qualitätssicherungs-Webportal, das der Arbeit beiliegt und einen Großteil dieser ausmacht.

## 2.2 Kapitelübersicht

Die Diplomarbeit besteht aus sieben aufeinander aufbauenden Kapiteln. Da spätere Kapitel Begriffe und Inhalte aus den vorangegangenen Kapiteln voraussetzen wird empfohlen, die Kapitel in der richtigen Reihenfolge zu lesen.

### Kapitel 2, Einführung

Behandelt die Motivation sowie Aufgabenstellung für die Diplomarbeit, diese

Übersicht und die Festlegung typographischer Konventionen für die gesamte Ausarbeitung.

### **Kapitel 3, Grundlagen**

Zeigt welche Hardware für dem Testbetrieb verwendet wurde. Enthält eine Auflistung der genauen Rechnerdaten und gibt Richtlinien und Tipps für eine sinnvolle Zusammensetzung der Rechnerkonstellationen einfacher und komplexer Systeme.

### **Kapitel 4, Performance**

Erklärt die Notwendigkeit von performanten Webauftritten und mit welchen Problemen diese zu kämpfen haben. Anschließend wird als erster Schwerpunkt dieser Arbeit eine mögliche Lösung für performante und skalierende Webauftritte aufgestellt und der Weg dorthin mit Hilfen und Konfigurationsbeispielen beschrieben. Dabei wird ein einfacher Grundstein für ein performantes Webportal herausgearbeitet und aufgestellt. Ebenso wird auf besondere Schwierigkeiten und Probleme eingegangen. Zum Abschluss des Kapitels wird gezeigt, wie mittels Performance- und Lasttests der Ist-Zustand eines Webauftritts analysiert und die dabei ermittelten Ergebnisse zum Aufdecken von Flaschenhälsen verwendet werden können.

### **Kapitel 5, Hochverfügbarkeit**

Zeigt als zweiter Schwerpunkt die heutige Notwendigkeit, die Verfügbarkeit einer Website so hoch wie nur irgend möglich anzulegen. Anschließend werden Möglichkeiten aufgezeigt wie Hochverfügbarkeit erreicht werden kann und was dabei unbedingt beachtet werden muss. Durch Konfigurationsbeispiele werden auch hier wieder Hilfestellungen zu ausgewählten Werkzeugen gegeben. Der performante Unterbau aus dem vorherigen Kapitel erhält hier nun zusätzlich seine Hochverfügbarkeitskomponente. Am Ende des Kapitels werden Probleme aufgezeigt, die beim Erreichen von Hochverfügbarkeit auftreten. Deren Lösungswege werden diskutiert.

### **Kapitel 6, Sicherheit**

Ist ein wesentlicher Aspekt jedes öffentlich zugänglichen Netzdienstes. Der Hauptteil dieses Kapitels liegt als Checkliste vor. Im diesem Dokument wird ein kurzer Überblick über Datenschutz und Datensicherheit gegeben.

### **Kapitel 7, Checklisten**

In diesem Kapitel werden die Checklisten erläutert, die im Laufe der Arbeit entstanden und deren Bedienung erklärt. Die Checklisten liegen in Form einer Webanwendung vor.

### **Kapitel 8, Abschließende Betrachtung**

Dieses letzte Kapitel geht auf Probleme und Grenzen bei der Realisierung der Hochverfügbarkeit, Performance und Sicherheit von Webportalen ein. Es stellt den *Ist*-Zustand dar und gibt einen Ausblick auf neue Technologien.

## 2.3 Typographische Konventionen

Damit die Einheitlichkeit und damit die Übersichtlichkeit gewährleistet ist, werden die folgenden typographischen Konventionen in der Arbeit verwendet:

### *Kursiv*

Wird für zum ersten Mal aufgetretene Fachbegriffe, URLs, Dateinamen, Verzeichnisse, Rechner- und Programmnamen benutzt.

### Nichtproportional

Wird für Quellcode und Programmausgaben benutzt.

### **Nichtproportional Fett**

Wird für Kommandos oder Eingaben genutzt, die der Leser selbst tätigen muss.

### *Nichtproportional Kursiv*

Wird für Variablen innerhalb von Quellcode verwendet, die der Leser seinen Bedürfnissen anpassen muss.

Um deutlich zu machen, ob bestimmte Befehle als normaler Benutzer eingegeben werden können, oder ob *root*-Rechte benötigt werden, finden die üblichen Unix Shell-Prompts Verwendung.

\$	Benutzerprompt
#	Rootprompt

# Kapitel 3

## Grundlagen

„Erkannte und eingestandene Irrtümer waren schon immer die beste Grundlage für neue Einsichten.“

– Günther Grass

Dieses Kapitel stellt die Grundlagen für ein System vor, mit dem alle Voraussetzungen erfüllt werden können, die an Hochverfügbarkeit, Performanz und Sicherheit gestellt werden. Es zeigt welche Hardware überhaupt benötigt wird und schlägt Alternativen für den Fall vor, dass es an ausreichender Hardware mangelt. Grundvoraussetzungen für das gesamte Vorhaben sind natürlich gute Kenntnisse in Linux/Unix und im Bereich Netzwerkkonfiguration.

### 3.1 Rechnerlandschaft des Testportals

Um ein performantes und hochverfügbares Webportal realisieren zu können, wird entweder eine kleine Anzahl an sehr zuverlässiger und schneller Hardware oder aber eine große Anzahl an *normaler* Hardware benötigt. Da der Fokus auf dem Open Source-Bereich liegt, ist kaum damit zu rechnen, dass Mainframes eingesetzt werden. Ziel ist es daher, die Portalplattform mit vorwiegend herkömmlichen PC-Rechnern zu realisieren. Das ist nichts ungewöhnliches. Bekannte Suchmaschinenbetreiber, wie *Yahoo* oder *Google* nutzen ebenso Webcluster bestehend aus ganz normalen PCs. Im Grunde ähneln deren Methoden den hier vorgestellten in vielen Dingen und stimmen sehr häufig vollkommen überein.

Abbildung 3.1 zeigt, welche Rechner im Fachbereich MNI der Fachhochschule Gießen-Friedberg für das Projekt zur Verfügung standen. Die in dieser Abbildung gestrichelten Linien stellen den redundanten Zweig des Systems dar. Die Rechner *PC-KQC* und *Galatea* müssen vom Internet her erreichbar sein. Die Webserver der übrigen Rechner müssen und sollen auch nicht von außerhalb des Netzes erreicht werden können. Dazu müssen geeignete Maßnahmen, wie etwa der Einsatz von Paketfiltern oder die Zugriffskontrolle auf Dienste berücksichtigt werden. *PC-KQC* ist der *Loadbalancer* des Systems. Er verteilt von außen ankommende Anfragen an die *Realserver Jupiter* und *Naiade*. Für die nahe Zukunft ist der Einsatz weiterer Realserver geplant. Als Minimalkonfiguration sind diese beiden jedoch ausreichend. *Galatea* ist

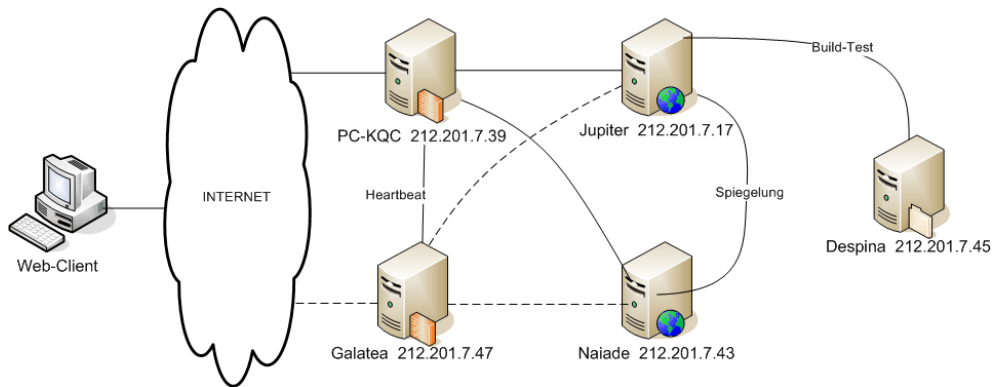


Abbildung 3.1: Portalaufbau des Fachbereichs MNI

eine exakte Kopie von *PC-KQC* und dient damit als Redundanz des Loadbalancers. Der Rechner *Despina* hat mit dem laufenden Webportal nichts zu tun. Er stellt jedoch eine Testplattform für neue Komponenten des späteren Portals dar, die dort entwickelt und nach Fertigstellung in das Produktionssystem eingespielt werden können.

### 3.2 Eigenschaften der verwendeten Rechner

Zu allererst wurde der *Ist*-Zustand detailliert festgehalten. Es folgt eine tabellarische Auflistung der verwendeten Hardware. Dabei wurden jeweils technische Details wie CPU oder Größe des Arbeitsspeichers erfasst, die für die spätere Konfiguration wichtig sind.

#### PC-KQC

Interface IP/Mask Name	eth0:0 212.201.7.39/24 pc-kqc.mni.fh-giessen.de
	eth0 212.201.7.38/24 pc-kqc-1.mni.fh-giessen.de
Interface HW-Adr.	eth0 Linksys 10/100, 00:30:05:56:4D:7E
CPU	Intel Celeron 2,4 GHz
Arbeitsspeicher	1,256 GB
Betriebssystem	Linux, Suse 9.0, 2.4.21-215-default
Standort	F 210
Ansprechpartner	Klaus Quibeldey-Cirkel
Dienste	ssh, http(80,10080), https(443, 10443), webmin, mysql, cvs, rsync(10837)
Bemerkungen	./.

**Neptun**

Interface IP/Mask Name	eth0 212.201.7.47/24 Neptun.mni.fh-giessen.de
Interface HW-Adr.	eth0 00:30:48:27:6F:3A
CPU	Dual Xeon 2,4 GHz
Arbeitsspeicher	4 GB
Betriebssystem	Linux, Mandrake 9, 2.4.20gf_UMLskas
Standort	F 210
Ansprechpartner	Gerhard Franke
Dienste	ssh, Virtuelle Plattformen
Bemerkungen	Keine direkte Verwendung. Dient als Host für virtuelle Maschinen <i>Galatea</i> , <i>Despina</i> und <i>Naiade</i>

**Galatea**

Interface IP/Mask Name	eth0 212.201.7.47/24 Galatea.mni.fh-giessen.de
Interface HW-Adr.	eth0 FE:FD:D4:C9:07:2F
CPU	siehe Neptun
Arbeitsspeicher	192 MB
Betriebssystem	Linux, Debian, 2.4.26gf1-1um
Standort	siehe Neptun
Ansprechpartner	Gerhard Franke
Dienste	ssh, http(80, 10080), https(443, 10443), mysql, webmin, rsync(10837)
Bemerkungen	Ist ein virtueller Rechner, der auf Neptun läuft.

**Despina**

Interface IP/Mask Name	eth0 212.201.7.45/24 Despina.mni.fh-giessen.de
Interface / HW-Adr.	eth0 FE:FD:D4:C9:07:2D
CPU	siehe Neptun
Arbeitsspeicher	192 MB
Betriebssystem	Linux, Debian, 2.4.26gf1-1um
Standort	siehe Neptun
Ansprechpartner	Gerhard Franke
Dienste	ssh, http(80, 10080), https(443, 10443), mysql, smtp, ftp, webmin, rsync(10837)
Bemerkungen	Ist ein virtueller Rechner, der auf Neptun läuft.

**Naiade**

Interface IP/Mask Name	eth0 212.201.7.45/24 Naiade.mni.fh-giessen.de
Interface HW-Adr.	eth0 FE:FD:D4:C9:07:2B
CPU	siehe Neptun
Arbeitsspeicher	192 MB
Betriebssystem	Linux, Debian, 2.4.26gf1-1um
Standort	siehe Neptun
Ansprechpartner	Gerhard Franke
Dienste	ssh, http(80, 10080), https(443, 10443), ftp, cvs, mysql, webmin, rsync(10837)
Bemerkungen	Ist ein virtueller Rechner, der auf Neptun läuft.

**Jupiter**

Interface IP/Mask Name	eth0 212.201.7.17/24 Jupiter.mni.fh-giessen.de
Interface HW-Adr.	eth0 08:00:20:FC:A9:2A
CPU	2 x 400 MHz
Arbeitsspeicher	1 GB
Betriebssystem	Solaris 5.8
Standort	F 210
Ansprechpartner	Gerhard Franke
Dienste	ssh, http(8080, 10080), https(8443, 10443), mysql, rsync(10837)
Bemerkungen	keine Rootrechte, Portalroot: /misc/mni-portal

Die aufgelistete Hardware verwendet eine 100 MBit/s Netzwerkanbindung. Der Rechner *www.fh-giessen.de*, auf dem die aktuelle MNI-Fachbereichsseite liegt, sollte ursprünglich ebenfalls als Realserver eingesetzt werden. Jedoch stellte sich heraus, dass dieser Rechner nur sehr schlecht ausgestattet und für dynamische Inhalte gänzlich ungeeignet ist.

**3.3 Aufbau einer Entwicklungsumgebung**

Während des Aufbaus und der Testphase der neuen Produktionslandschaft darf natürlich das bestehende System nicht beeinträchtigt werden. Es wäre nicht sonderlich vorteilhaft, an einer Hochverfügbarkeitslösung zu arbeiten und gleichzeitig dadurch die Verfügbarkeit des bestehenden Systems zu beeinträchtigen. Um alle gewünschten Dienste einrichten und ungestört testen zu können, wird verschiedene Hard- und Software benötigt. Damit eine funktionierende Testumgebung sichergestellt werden kann, werden eine Reihe von Mindestanforderungen an diese Hard- und Software gestellt. Diese werden im Folgeabschnitt erläutert.

### 3.3.1 Hardware

Wie an der neuen Portalumgebung des MNI-Portals der Fachhochschule Gießen (Abbildung 3.1 auf Seite 12) zu sehen ist, wird eine ganze Reihe Hardware für ein solches System benötigt. Am einfachsten ist es natürlich, eine identische Plattform auch für Konfiguration und Tests zu nutzen. Es kann allerdings davon ausgegangen werden, dass ein solcher Aufwand an Material und Kosten nur in sehr großen Firmen getätigt wird.

Was wird als Folge davon an Hardwareausstattung benötigt, um den Ansprüchen einer minimalen Testumgebung gerecht zu werden?

**1 Webclient** Dieser Rechner dient lediglich dazu, die Konfigurationen und Verfügbarkeit der einzurichtenden Netzdienste zu prüfen. Der Webclient kann auf jedem beliebigen Betriebssystem laufen und benötigt nur einen Webbrowser für den Zugriff auf das Portal. Dabei sollte der Browser sämtliche Angebote, die das Portal bietet testen können.

**2 Lastverteiler** Einer dieser Rechner wird später die Anlaufstelle für den Webclient sein. Für den Client wird es so aussehen, als laufe das gesamte Portal auf diesem einen Rechner. Der zweite Lastverteiler stellt die Hochverfügbarkeitskomponente dar. Er wird den Haupt-Loadbalancer ersetzen, falls dieser aus irgendwelchen Gründen ausfallen sollte. Für die Einrichtung und den Testbetrieb ist es selbstverständlich nicht notwendig, dass die Rechner räumlich getrennt aufgestellt werden. Im Gegenteil ist lokale Verfügbarkeit – speziell des An/Aus-Schalters – hier nur von Vorteil.

**2 Realserver** Diese beiden Rechner stellen die eigentliche Plattform des Portals dar. Im späteren Produktionssystem bedeutet bei entsprechender Auslastung der bereits vorhandenen Realserver, jeder zusätzliche Realserver ein Performancegewinn. Für den Client treten sie jedoch nie in Erscheinung. Für ihn ist das gesamte System transparent. Die Realserver beherbergen die benötigten Dienstprogramme, wie einen HTTP<sup>1</sup>-Server, einen SQL<sup>2</sup>-Datenbank-Server, FTP<sup>3</sup>-Server, etc.

Das ergibt eine Mindestanzahl von 5 Rechnern. Kann über diese Anzahl an Maschinen nicht verfügt werden, gibt es Möglichkeiten dennoch eine praktikable Testlandschaft zu errichten:

1. Um zu testen, ob der Loadbalancer korrekt arbeitet, werden grundsätzlich mindestens zwei Realserver mit den entsprechenden Diensten benötigt. In Wirklichkeit verhält es sich jedoch etwas anders. Benötigt werden nicht unbedingt

---

<sup>1</sup>Hypertext Transfer Protocol

<sup>2</sup>Structured Query Language

<sup>3</sup>File Transfer Protocol

zwei Rechner. Im Grunde genügt es auch, die entsprechenden Dienste auf einem Rechner mehrmals zu starten.

Beispiel:

Im Falle des Apache Webserver, ist es für grundlegende Tests ausreichend, diesen so zu konfigurieren, dass er auf unterschiedlichen Ports lauscht. Um anschließend beide Server über den Browser des Client-Rechners unterscheiden zu können, wird der jeweilige Port explizit mit angegeben.

Angenommen der Realserver besitzt die IP-Adresse: 192.168.1.10

- `http://192.168.1.10` oder `192.168.1.10:80` erreicht Webserver 1
  - `http://192.168.1.10:8080` erreicht Webserver 2
2. Besitzt ein ohnehin verwendeter Rechner genügend Systemressourcen, so können auch virtuelle Maschinen eine Lösung darstellen. Werden auf einem mit ausreichend Arbeitsspeicher und CPU-Leistung ausgestatteten Rechner einige virtuelle Maschinen betrieben, können reale Testrechner eingespart werden. Eine Konzept für virtuelle Maschinen im Open Source-Bereich bietet „User-mode Linux“<sup>4</sup> an, welches auch im Fachbereich MNI eingesetzt wird. Die Maschinen *Naiade*, *Galatea* und *Despina* sind alles virtuelle Maschinen auf dem Rechner *Neptun*.

Es ist also auch möglich mit knapper Hardware eine Testumgebung aufzubauen. Was die Hardwareausstattung angeht, so ist zum Entwickeln und Testen kein besonders schneller Prozessor notwendig. Jedes System, welches mit aktuellen Linux- oder Windows-Versionen umgehen kann, ist vollkommen ausreichend. Fällt die Entscheidung auf eine Variante mit virtuellen Maschinen, sollte der Hostrechner unbedingt über genügend Arbeitsspeicher verfügen.

### 3.3.2 Software

Was die Software in der Testumgebung angeht, so gelten folgende Einschränkungen:

- Bei dem zu verwendenden Webbrowser auf dem Webclient sollte im Vorhinein klar sein, ob er alle für die zu testende Portalsoftware benötigten optionalen Merkmale, wie *Javascript* oder *Java-Applets* unterstützt. Da Portale ohnehin darauf ausgerichtet sind, eine Vielzahl von Besuchern zu erreichen, sollten die bekannten aktuellen Browser keine Probleme mit der Darstellung haben.
- Was die Software der Loadbalancer angeht, so ist hier der Einsatz von Linux zwingend erforderlich. Das Loadbalancer-System, welches für diese Arbeit verwendet wird, ist Bestandteil des Linux-Kerns, womit sich gleich noch

---

<sup>4</sup>User-mode Linux <http://user-mode-linux.sourceforge.net/>

eine weitere Bedingung ergibt. Erst ab der Kernelversion 2.4.21 ist es standardmäßig im Kernel enthalten. Wird noch ein älterer Kern eingesetzt, so muss dieser entsprechend *gepatcht* oder besser komplett aktualisiert werden.

Die Kernelversion eines installierten Linux-Systems wird folgendermaßen in Erfahrung gebracht:

```
$ uname -r  
2.6.3-13mdk
```

Verwendet wird hier ein Kern der 2.6er Reihe. Dieser hat die benötigte Loadbalancer-Funktionalität bereits fest eingebaut.

- Ein Realserver ist im Grunde nicht an ein bestimmtes Betriebssystem gebunden. Dieses muss lediglich einen TCP/IP-Stack besitzen. Außerdem müssen die für das verwendete Portal benötigten Dienste auf dem System verfügbar sein.  
Wie sich im weiteren Verlauf noch herausstellen wird, kommen für einige Verfahren nur bestimmte Betriebssysteme in Frage. Daher sollten in der Testumgebung möglichst Linux-Maschinen eingesetzt werden.

Vorteilhaft wäre es, jede Testmaschine mit dem gleichen Linux-System auszustatten. Dadurch wird gewährleistet, dass während des Betriebs ein fehlerhafter Rechner schnell ausgetauscht werden kann. Außerdem bieten die meisten Distributionen verteilte Softwareupdates an, die sich aber zwischen den Distributionen unterscheiden. Mit diesen verteilten Installationen können bei Festlegung auf eine einzige Distribution die Programmpakete schnell auf sämtlichen Maschinen aktualisiert werden, was eine erhebliche Zeitersparnis darstellt. Wenn das System später einmal läuft bietet es sich an, Rechner mit anderen Betriebssystemen oder Linux-Distributionen auszustatten und sie zur Realserverfarm hinzuzufügen. Auf diese Weise zeichnet sich bei Tests ein allgemeineres Bild ab.

# Kapitel 4

## Performance

„Es gibt wichtigeres im Leben, als beständig dessen Geschwindigkeit zu erhöhen.“

– Mahatma Gandhi

Welchen Nutzen hat eine Website, die zwar 24 Stunden am Tag und an sieben Tagen in der Woche verfügbar ist, deren Seitenaufbau jedoch so lange dauert, dass die gesuchten Informationen bis sie geladen wurden bereits wieder veraltet sind. Diese Anmerkung klingt im ersten Augenblick scherzhaft. Handelt es sich jedoch um Aktienkäufe oder auch nur um eine ablaufende Auktion bei einer Online-Versteigerung, so wird die Notwendigkeit offensichtlich, Methoden für die Verkleinerung von Ladezeiten und zur Erhöhung der Skalierbarkeit zu finden. Der Begriff *Skalierbarkeit* bedeutet in diesem Zusammenhang, dass die Ladezeit einer angeforderten Seite nicht proportional zur Anzahl der Besucher ansteigt, sondern in etwa gleich bleibt. Dabei handelt es sich nicht um einen absoluten Begriff. Die absolute Aussage: „Eine Website skaliert“ ist nicht korrekt. Richtiger wäre es zu sagen, dass eine Website bis  $x$  gleichzeitige Besucher skaliert.

In der Praxis verhält es sich so, dass Website-Besuche sich nicht gleichmäßig über den gesamten Tag verteilen. Es gibt Zeitspannen in denen die Seite praktisch unbesucht ist während auch Stoßzeiten existieren, in denen die Besucherzahl über dem Doppelten des Durchschnitts und höher liegt. Abbildung 4.1 zeigt die Tagesleistungskurve, die mit einigen Einschränkungen für jeden Menschen Gültigkeit besitzt. Ein

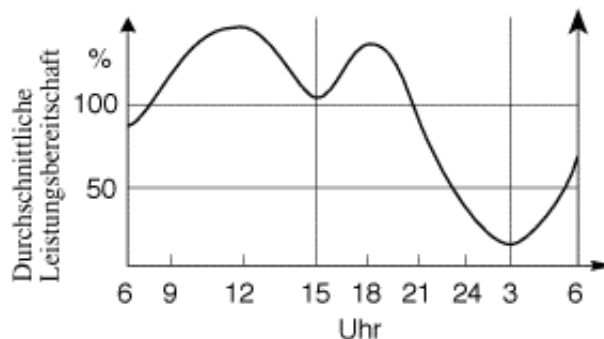
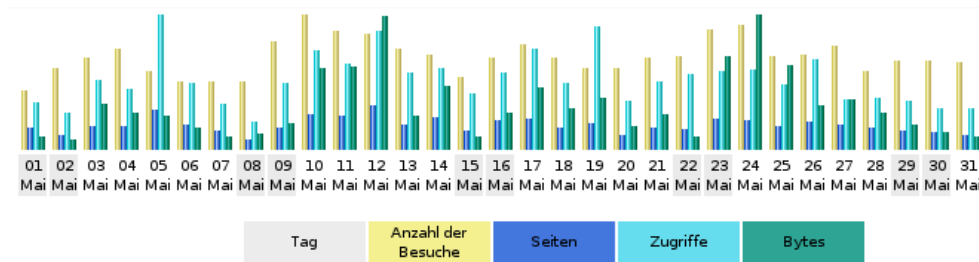
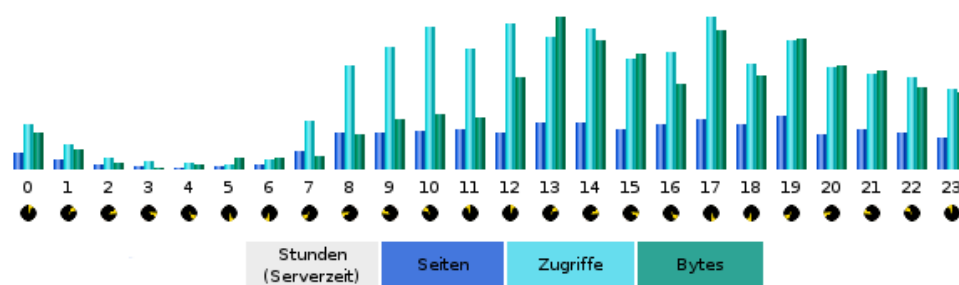


Abbildung 4.1: Tagesleistungskurve



(a) Monatsstatistik



(b) Tagesdurchschnitt

Abbildung 4.2: Webserverstatistik von pc-kqc.mni.fh-giessen.de im Mai 2004

Maximum an Leistungsbereitschaft besitzt der Mensch etwa zwischen 10 Uhr und Mittag. Danach fällt die Kurve rapide ab und erreicht etwa gegen 15 Uhr im so genannten Mittagsloch ein Minimum. Gegen Abend hin nimmt die Leistungsbereitschaft noch einmal zu, fällt dann gegen 21 Uhr rapide ab, um am Morgen gegen 3 Uhr ihr absolutes Minimum zu erreichen.

Was diese Kurve mit der Performance von Webanwendungen zu tun hat wird deutlich, wenn sie mit einer Webserverstatistik wie in den Abbildungen auf dieser Seite dargestellt verglichen wird. Abbildung 4.2(a) zeigt die Zugriffe auf das Lehrportal eStudy an der Fachhochschule Gießen. Deutlich erkannt werden kann ein starker Anstieg der Zugriffe von Montag bis Mittwoch einer Woche<sup>1</sup> und eine daraufhin folgende Abnahme bis zu einem Minimum am Wochenende. Das ist nicht verwunderlich, denn studiert wird nicht am Wochenende. Die Notwendigkeit für Forenbesuche nimmt allerdings ab Montag rapide zu, da durch die Vorlesungen neuer Diskussionsstoff existiert. Deutlich zu sehen an diesen Statistiken ist auch, dass die Anzahl der Zugriffe an einigen Tagen bis 3 Mal so hoch ist, als an anderen. Am 5., 12. und 19. Mai wurden besonders viele Zugriffe registriert. Immens hohes Datenaufkommen gab es am 12. und 24. des Monats.

<sup>1</sup>Die Wochenenden sind grau gefärbt.

Abbildung 4.2(b) zeigt die durchschnittlichen Zugriffe auf das Portal im Laufe eines Tages, gemessen über einen ganzen Monat. Die Abbildung verglichen mit der Tagesleistungskurve von Abbildung 4.1 lässt bei einem Durchschnitt von nur 31 Tagen bereits eine große Ähnlichkeit erkennen. Das Mittagsloch ist in der Zugriffsstatistik zwar weit weniger ausgeprägt, aber immer noch deutlich zu erkennen. Da auf dieses Portal vorwiegend Studenten zugreifen, ist die Kurve ein wenig nach rechts verschoben.

```

top - 00:01:46 up 29 days, 11:06, 2 users, load average: 23.92, 17.60, 9.27
Tasks: 134 total, 17 running, 116 sleeping, 0 stopped, 1 zombie
Cpu(s): 86.3% user, 13.7% system, 0.0% nice, 0.0% idle
Mem: 1283468k total, 1219608k used, 63860k free, 172840k buffers
Swap: 1204864k total, 1024k used, 1203840k free, 674412k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM     TIME+  COMMAND
 22376 wwwrun    17   0     0     0     0  Z   4.0   0.0   0:01.83 httpd <defunct>
 22318 wwwrun    16   0 32396  31m  29m  S   3.1   2.5   0:00.28 httpd
 22654 wwwrun    15   0 33048  32m  29m  S   2.8   2.6   0:01.20 httpd
 22670 wwwrun    15   0 33264  32m  29m  S   2.8   2.6   0:01.24 httpd
 23117 wwwrun    15   0 33280  32m  29m  S   2.8   2.6   0:00.54 httpd
 23319 wwwrun    16   0 33132  32m  29m  S   2.8   2.6   0:00.35 httpd
 22624 wwwrun    15   0 33060  32m  29m  S   1.2   2.6   0:01.15 httpd
 22672 wwwrun    16   0 33064  32m  29m  S   1.2   2.6   0:01.17 httpd
 22967 wwwrun    16   0 33064  32m  29m  S   1.2   2.6   0:00.77 httpd
 23302 wwwrun    16   0 32428  31m  29m  S   1.2   2.5   0:00.30 httpd
 22642 wwwrun    16   0 33032  32m  29m  S   0.9   2.6   0:01.11 httpd
 22655 wwwrun    16   0 33064  32m  29m  S   0.9   2.6   0:01.24 httpd
 23288 wwwrun    16   0 32332  31m  29m  S   0.9   2.5   0:00.36 httpd
 23451 mysql     15   0  9408  9404 3072  S   0.9   0.7   0:00.03 mysqld
 23440 mysql     16   0  9408  9404 3072  S   0.6   0.7   0:00.03 mysqld
 23452 mysql     15   0  9408  9404 3072  S   0.6   0.7   0:00.02 mysqld
 23489 mysql     18   0  9408  9404 3072  R   0.6   0.7   0:00.02 mysqld
 23493 mysql     16   0  9408  9404 3072  S   0.6   0.7   0:00.02 mysqld

```

Abbildung 4.3: Überlasteter Webserver

Nun wird dieses Portal nur für eine geringe Anzahl von Studenten eingesetzt und absolute Zahlen wären nicht repräsentativ. Die Grafik zeigt jedoch, wie die für einen Website-Betreiber so wichtigen Leistungsspitzen früh erkannt werden und im Vorhinein in die Planung einer neuen Portalumgebung einfließen können. Wichtig ist, dass ein Webaufttritt auch in solchen Stoßzeiten skaliert. Probleme entstehen hier vor allem durch einen zu geringen Arbeitsspeicher im Produktionswebserver. Ein einziger Prozess des Apache Webservers benötigt zwischen 10 und 30 MB Arbeitsspeicher. Das macht bei 1GB Arbeitsspeicher einen Anteil von 1,6% aus. Kommt dazu noch ein MySQL- oder anderer Datenbankverwaltungsprozess in der gleichen Größenordnung, was bei allen gängigen Open Source-Portalen der Fall sein dürfte, so liegt die Systemauslastung für nur einem Benutzer bei etwa 3% des verfügbaren Arbeitsspeichers.

Bei Besucherspitzen kommt es daher sehr häufig vor, dass Daten im Arbeitsspeicher auf die Festplatte ausgelagert werden müssen. Die Ladezeiten dauern dadurch immer länger. Dies eskaliert bei weiterem Besucherzuwachs schließlich darin, dass der Rechner mit nichts anderem mehr beschäftigt ist, als Seiten aus dem Arbeitsspeicher auf die Festplatte zu schreiben und andere von dieser in den Arbeitsspeicher zu laden. Das System kann in diesem Falle keine Anfragen mehr bearbeiten, da es sich nur noch selbst verwaltet. Der Website-Provider ist einer ungewollten verteilten

DOS<sup>2</sup>-Angriffe zum Opfer gefallen. Abbildung 4.3 zeigt genau dieses Problem. Der Ausschnitt des Systemmonitors *top* zeigt die Prozesse eines Webservers während 30 simulierte Benutzer gleichzeitige Anfragen starteten, wobei nicht alle offenen Prozesse dargestellt werden. Die CPU läuft unter Vollast und kann keine weiteren Anfragen mehr bearbeiten.

Um auch während solcher Lastspitzen ein skalierendes System aufweisen zu können und die allgemeine Performance zu verbessern, gibt es Verfahren, um die durch Anfragen entstehende Last von einem einzelnen Rechner auf mehrere zu verteilen. Die folgenden Abschnitte beschreiben eine Möglichkeit, diese Lastverteilung mit einer Softwarelösung auf einem Linux-Rechner durchzuführen.

## 4.1 Lastverteilungsverfahren

Um hohe Skalierbarkeit des Web-Portals zu erreichen, muss bei besonders hohen Lastaufkommen, welche etwa durch eine gute Werbestrategie oder bei immer wiederkehrenden Besucherstoßzeiten auftreten könnten, die Last auf mehrere Rechner verteilt werden. Das führt dazu, dass die Belastung pro Rechner herabgesetzt wird. Es gibt Hardware-Loadbalancer, welche die eingehenden Anfragen auf mehrere Rechner verteilen. Da jedoch eine Lösung mit minimierten Kosten angestrebt ist und sich in diesem Bereich eine Open Source-Lösung anbietet, wird für die Umsetzung ein Software-Loadbalancer verwendet.

Das Open Source-Projekt *Linux Virtual Server Project* [Zha04a] (im Folgenden LVS-Projekt genannt) bietet einen solchen Software-Loadbalancer an. Es handelt sich dabei in der Terminologie des OSI<sup>3</sup>-Referenzmodells um einen Schicht-4<sup>4</sup> Loadbalancer, der anhand von Zieladresse und Zielport eines empfangenen Paketes Entscheidungen über dessen weiteres Ziel trifft. Der Loadbalancer ist im Grunde ein auf die entsprechenden Bedürfnisse angepasster Linux-Kern. Ab Kernen der Version 2.4.23 ist die LVS-Funktionalität standardmäßig enthalten. Wird ein älterer Kernel verwendet, kann dieser mit einem Software-Patch von der LVS-Website mit der Loadbalancer-Funktionalität aufgerüstet werden.

Die Lastverteilung läuft so ab, dass ein Rechner sämtliche Anfragen an einen bestimmten Domain-Namen oder eine IP-Adresse aus dem Internet oder einem Intranet entgegennimmt. Auch wenn hier im Besonderen auf HTTP-Anfragen eingegangen wird, ist die Lastverteilung nicht auf ein besonderes Protokoll beschränkt, sondern unterscheidet die Anfragen einfach anhand ihres Zielports. Für den Benutzer, der z.B. eine HTTP-Anfrage abschickt sieht es so aus, als sei dieser Rechner der wirkliche und einzige Webserver. In Wahrheit ist er jedoch der Loadbalancer und besitzt

<sup>2</sup>Denial Of Service – Belasten eines Dienstes mit Anfragen, bis dieser schließlich nicht mehr antworten kann und seinen Dienst verweigert.

<sup>3</sup>Open Systems Interconnection – Offenes Schichtenmodell, das als Grundlage für herstellerunabhängige Netzprotokolle dient. Es umfasst sieben Schichten: *Physikalische Schicht*, *Sicherungsschicht*, *Netzwerkschicht*, *Transportschicht*, *Sitzungsschicht*, *Darstellungsschicht* und *Anwendungsschicht*.

<sup>4</sup>Die Schicht 4 des OSI-Modells ist die Transportschicht. Im Internet werden die Transportdienste *TCP* und *UDP* verwendet.

nur eine einzige Aufgabe: Die Verteilung von Anfragen an die echten Server. Diese werden im Folgenden als *Realserver* bezeichnet. Der Webclient erkennt dabei nicht, auf welchem Realserver er sich befindet. Für ihn ist das gesamte Verfahren transparent.

Erreicht ein Paket den Loadbalancer, so wird in einer Hashtabelle anhand von Kriterien, wie *Quelladresse*, *Quell- und Zielport* sowie *vorheriger Realserver* nachgesehen, ob der Dienst vor kurzem bereits in Anspruch genommen wurde. Wird ein Eintrag in der Tabelle gefunden, wird die Anfrage an den gleichen Realserver weitergeleitet. Dies hat den Sinn, dass bei einem sitzunglosen Protokoll – wie HTTP – noch bestehende *Cookies* und anderweitige Daten, die sich nur auf dem jeweiligen Realserver befinden weiterverwendet werden und so eine simulierte Sitzung aufrecht erhalten werden kann. Bei sitzungsorientierten Protokollen, wie etwa SSH<sup>5</sup> ist die Weiterleitung an den gleichen Realserver innerhalb einer Sitzung sogar zwingend erforderlich, damit diese aufrecht erhalten werden kann. Wird bei einer Anfrage kein entsprechender Eintrag in der Hashtabelle gefunden, so wird nach einem *Scheduling-Algorithmus* (siehe Kapitel 4.2) ein Realserver ausgewählt und ein neuer Eintrag geschrieben. Anschließend wird die Anfrage an den Realserver weitergeleitet.

#### 4.1.1 DNS-Round Robin

Dieses Verfahren gehört nicht zum LVS-Projekt und wird hier nur der Vollständigkeit halber erwähnt. Ein Rechner, der als DNS<sup>6</sup>-Server dient, kann bereits ein einfacher Lastverteiler sein. Es besteht die Möglichkeit, einem Namen im DNS mehrere IP-Adressen zuzuweisen. Bei einer Anfrage an einen Webserver über den DNS-Namen wird dieser jedesmal in eine andere IP-Adresse aufgelöst. Am Ende der vorhandenen IP-Adressliste wird erneut bei der ersten Adresse begonnen. Dieses Verfahren hat allerdings einige Nachteile gegenüber den folgenden Verfahren. *Round Robin* an sich ist in diesem Zusammenhang zum einen ein eher schlechtes Verfahren, da es sämtliche Realserver gleich behandelt. Außerdem funktioniert es nur, so lange auch wirklich der Name verwendet wird und nicht etwa direkt die IP-Adresse. Das größte Problem beim DNS-Round Robin ist allerdings, dass beim Ausfall eines Realservers wegen des statischen Aufbaus des DNS-Systems nicht flexibel darauf reagiert werden kann. Der Hochverfügbarkeitsaspekt würde hier verletzt, da ein Realserver in der Liste nicht mehr antworten würde.

#### 4.1.2 Network Address Translation

Network Address Translation (im Folgenden als NAT bezeichnet) ist ein Verfahren, bei dem Quell- und/oder Zieladressen von IP-Paketen manipuliert werden. Loadbalancer verwenden das *Full-NAT*-Verfahren, bei dem IP-Adressen aus einem Netzsegment in gültige IP-Adressen eines anderen Netzsegments umgesetzt werden. Voraus-

---

<sup>5</sup>Secure Shell

<sup>6</sup>Domain Name System – Namensdienst für die Umsetzung von Namen in IP-Adressen und umgekehrt.

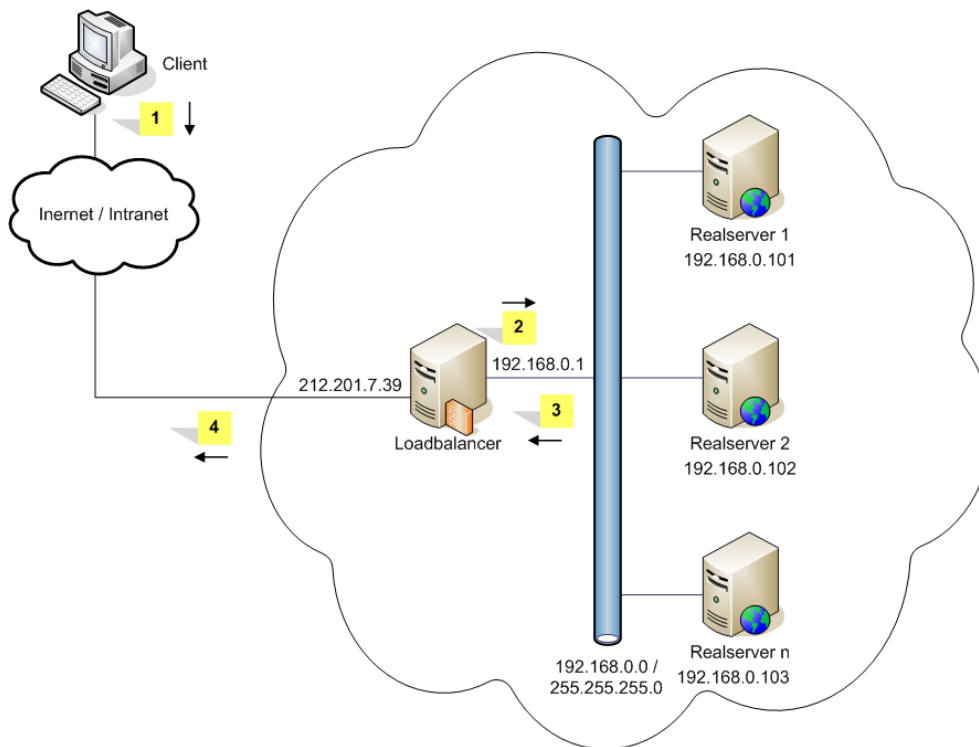


Abbildung 4.4: Loadbalancingverfahren: NAT

setzung hierfür ist, dass der Loadbalancer zwei Netzwerkschnittstellen besitzt und ein lokales Netz existiert. Bei den Schnittstellen kann es sich zum Testen auch um virtuelle Schnittstellen<sup>7</sup> handeln, sofern die Netze nicht durch Hardware getrennt sind. Für den Realbetrieb stellt die Verwendung von nur einer realen Schnittstelle jedoch einen Flaschenhals dar, zumal ein nicht durch Hardware getrenntes Netz den Einsatz des NAT-Verfahrens für das Loadbalancing in Frage stellen würde.

In Abbildung 4.4 ist zu erkennen, dass der Loadbalancer zwei Netzwerkschnittstellen mit den IP-Adressen *212.201.7.39* als öffentliches und *192.168.0.1* als internes Interface besitzt. Es ist davon auszugehen, dass der Loadbalancer so eingerichtet wurde, dass er auf Anfragen an den Socket<sup>8</sup> *212.201.7.39:80* reagiert und diese Pakete an die *Realserver 1* bis *n* verteilt. Erhält der Loadbalancer nun eine neue Anfrage von einem Client über Internet oder ein Intranet (1), schreibt er die Zieladresse des IP-Pakets auf die des ausgewählten Realservers um und schickt das Paket weiter über die interne Schnittstelle *192.168.0.1* (2). Das Paket erreicht den eingetragenen Realserver im lokalen Netz *192.168.0.0/255.255.255.0*, welcher die Anfrage bearbeitet. Das Antwortpaket (3) enthält nun als Zieladresse die IP-Adresse des Client und als Quel-

<sup>7</sup>Werden einer Netzwerkschnittstelle mehr als eine IP-Adresse zugewiesen, so wird von virtuellen Schnittstellen gesprochen.

<sup>8</sup>Als Socket wird ein Tupel bestehend aus IP-Adresse und Port bezeichnet. Jeder Dienst kann über einen Socket eindeutig angesprochen werden. Sockets selbst werden noch nach dem verwendeten Protokoll (TCP oder UDP) unterschieden.

Tabelle 4.1: Beispiel: LVS-NAT

Pos. in Abb. 4.4	SRC-IP	SRC-PORT	DST-IP	DST-PORT
(1)	<b>80.133.160.24</b>	<b>2345</b>	212.201.7.39	80
(2)	80.133.160.24	2345	192.168.0.3	8080
(3)	192.168.0.3	8080	80.133.160.24	2345
(4)	212.201.7.39	80	<b>80.133.160.24</b>	<b>2345</b>

adresse die IP-Adresse des Realservers. Für die neue Zieladresse gibt es im internen Netz allerdings keine festgelegte Route, der das Paket folgen könnte. Es muss daher ein Standard-Gateway<sup>9</sup> festgelegt werden. Es kann jedoch kein beliebiges Gateway in das Internet oder Intranet gewählt werden, da die Quelladresse des Antwort-Pakets nicht mit der Zieladresse des Anfragepakets übereinstimmt und der Client das Paket daher verwerfen würde. Es ist deshalb beim NAT-Verfahren zwingend erforderlich, dass bei jedem Realserver als Standard-Gateway die interne IP-Adresse des Loadbalancers (hier: 192.168.0.1) eingetragen wird. Dieser überschreibt die Quelladresse des Antwortpakets mit seiner eigenen öffentlichen IP-Adresse (hier: 212.201.7.39) und leitet es an den Client zurück (4).

Dieses Verfahren wird an einem Beispiel in Tabelle 4.1 veranschaulicht. Man erkennt, dass Quell-IP und Quell-Port, sowie Ziel-IP und Zielport in Schritt (1) und (4) vertauscht sind. Damit ist gewährleistet, dass der Client das Antwortpaket nicht verwirft. Für den Client sieht es so aus, als würde das Paket direkt vom Loadbalancer abgeschickt.

Das Problem bei diesem Verfahren ist, dass es nur bei einer geringen Zahl an Realservern die Performance erhöhen bzw. das System skalieren lassen kann. Dadurch, dass jedes Antwort-Paket den Loadbalancer erneut passieren muss, kann bei mehr als etwa 25 Realservern eine Flaschenhalssituation entstehen. Aufgrund nicht vorhandener Hardware konnte diese Zahl nicht selbst ermittelt werden, sondern stützt sich auf Aussagen des LVS-Projektes [Hor04a] und einer Performance-Studie zu den drei unterschiedlichen LVS-Verfahren von Patrick O'Rourke und Mike Keefe aus dem Jahr 2001 [OK04]. Zudem benötigt NAT bereits an sich mehr Systemressourcen, als die Folgeverfahren. Wenn also damit gerechnet werden kann, dass eine Realserverfarm einmal mehr als 25 Realserver umfassen wird oder diese Anzahl bereits überschritten ist, so sollte lieber ein anderes Verfahren in Erwägung gezogen werden.

### 4.1.3 Direct Routing

Befinden sich alle Realserver mit dem Loadbalancer im gleichen Netzsegment, so ist das *Direct Routing* Verfahren vorzuziehen. Beim Direct Routing wird die Quell- und Zieladresse eines IP-Pakets nicht wie beim NAT umgeschrieben. Direct Routing findet eine Ebene tiefer auf Schicht 2 im OSI-Modell nur anhand der Hardwareadresse (MAC-Adresse<sup>10</sup>) statt.

<sup>9</sup>Ein Rechner, an den alle Pakete weitergeleitet werden, für die keine Route festgelegt wurde.

<sup>10</sup>Media Access Control Address – Bezeichnung für die Hardwareadresse eines Netzwerkgerätes.

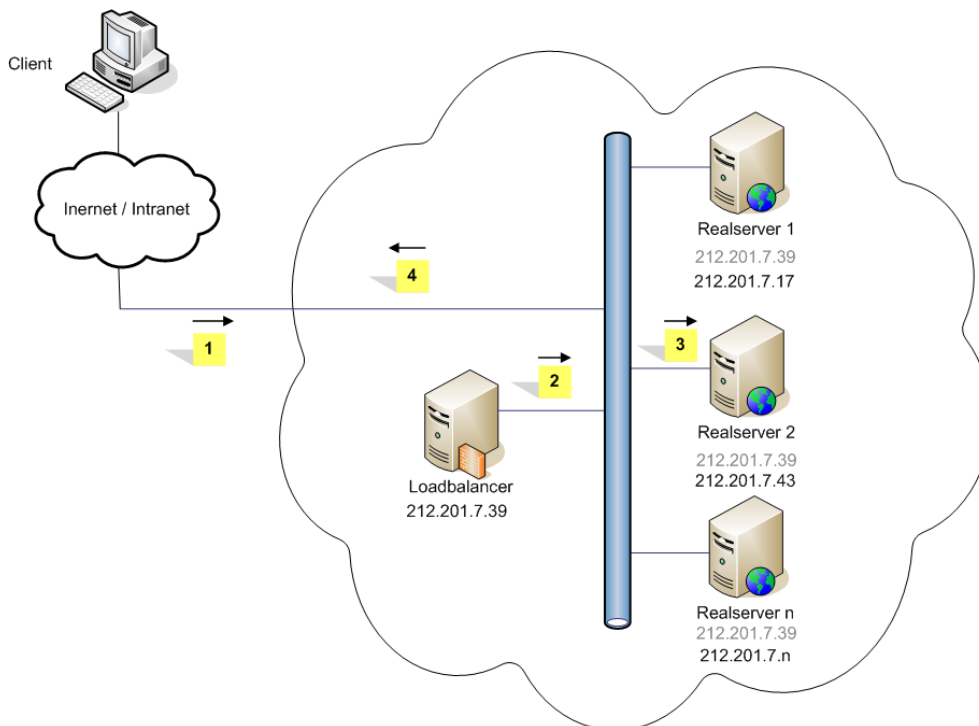


Abbildung 4.5: Loadbalancingverfahren: Direct Routing

Abbildung 4.5 zeigt das Direct Routing Verfahren. Bedingung hierbei ist, dass jeder Realserver zusätzlich zu seiner öffentlichen Netzwerkschnittstelle noch ein lokales Interface besitzt, welches mit der gleichen IP-Adresse wie der Loadbalancer konfiguriert ist (hier: 212.201.7.39). Dies führt zu einer weiteren Bedingung: Dieses Interface darf keine ARP<sup>11</sup>-Anfragen beantworten, da ansonsten bei einer Anfrage an den Loadbalancer ein Realserver die ARP-Anfrage beantworten könnte und somit der Loadbalancer umgangen würde. Die Realserver-Schnittstelle, welche die gleiche IP-Adresse wie der Loadbalancer besitzt, darf also weder durch die IP-Adresse selbst noch durch eine ARP-Anfrage von außen her zugänglich sein. Kapitel 4.5 auf Seite 40 beschäftigt sich näher mit dem ARP-Problem und gibt Lösungsansätze, wie dieses Problem umgangen werden kann.

Wie in der Abbildung zu erkennen ist, trennt der Loadbalancer beim Direct Routing nicht zwischen zwei Netzen. Daher wird in der Abbildung die Verbindungslinie auch nicht vom Internet bzw. Intranet zum Loadbalancer gezogen, wie etwa beim NAT-Verfahren. Stellt ein Client eine Anfrage an den Loadbalancer (1), verarbeitet dieser – so lange die Realserver und ihr Zusatzinterface korrekt konfiguriert sind – die Anfrage. Jedes ankommende IP-Paket ist in einem *Datenrahmen* der OSI-Schicht 2 verpackt, dessen Header die Hardwareadresse des Loadbalancers enthält. Der Loadbalancer entpackt nun diesen Rahmen nicht etwa, um an das IP-Paket zu gelangen, wie beim NAT-Verfahren, sondern schreibt direkt die Hardwareadresse im Header

<sup>11</sup>Address Resolution Protocol – Ordnet IP-Adressen Hardwareadressen zu.

des Rahmens auf die Hardwareadresse eines Realservers um. Anschließend wird der Rahmen an das Netz weitergereicht (2).

Anhand der Hardwareadresse erkennt der Realserver (z.B. Realserver 2), ob ein Rahmen an ihn gerichtet ist. Er empfängt und entfernt den Datenrahmen um das IP-Paket (3). Nun ist es nicht allzu schwer zu verstehen, warum der Realserver eine Schnittstelle mit gleicher IP-Adresse wie der Loadbalancer besitzen muss: Das soeben entpackte IP-Paket enthält als Zieladresse die Adresse des Loadbalancers (212.201.7.39). Könnte der Realserver keine Netzwerkschnittstelle mit dieser Adresse aufweisen, würde er das Paket verwerfen. Da sie vorhanden ist, erkennt der Realserver jedoch, dass das IP-Paket an ihn gerichtet ist und leitet es an den entsprechenden Port weiter.

Das Antwortpaket kann der Realserver über jedes beliebige Standard-Gateway an den Client zurücksenden (4), da er als Absenderadresse ja noch immer die IP-Adresse des Loadbalancers verwendet. Für den Client ist auch dieses Verfahren völlig transparent. Einziger Unterschied zum NAT-Verfahren besteht hier darin, dass der Client ohne entsprechende Vorkehrungen den Loadbalancer bewusst umgehen könnte, falls er die IP-Adresse oder den Namen des Realservers kennt. Beim Direct Routing tritt das Problem des NAT-Verfahrens, Antwortpakete auf dem Rückweg erneut über den Loadbalancer senden zu müssen nicht auf. Es ist daher aus Sicht der Performance im Gegensatz zum vorherigen Verfahren mit Direct Routing möglich, eine Serverfarm mit uneingeschränkt vielen Realservern zu betreiben.

#### 4.1.4 IP-Tunneling

IP-Tunneling funktioniert vom Prinzip her ähnlich, wie Direct Routing. Vor allem aus der Sicht eines Realservers. Der Vorteil beim IP-Tunneling besteht darin, dass die Realserver über beliebige Netze verteilt sein können. IP-Tunneling ist somit das flexibelste Verfahren zur Lastverteilung, setzt jedoch voraus, dass das Betriebssystem des jeweiligen Realservers einen IP-Tunnel unterstützt. Unix-Systeme unterstützen dies im Normalfall, aber auch für Windows ab *Windows 2000* kann ein IP-Tunnel aufgebaut werden. „Ein IP-Tunnel ist eine virtuelle Punkt-zu-Punkt-Leitung zwischen zwei Knoten, die eigentlich durch eine beliebige Anzahl von Netzwerken voneinander getrennt sind.“<sup>12</sup> Dabei werden IP-Pakete, die getunnelt werden sollen einfach in anderen IP-Pakete mit Zieladresse des Realservers gekapselt. Das neue so genannte *IPIP-Paket* wird gesendet und an seinem Ziel das enthaltene Paket wieder ausgepackt. In Abbildung 4.6 ist die Lastverteilung über IP-Tunnel schematisch dargestellt. Ähnlich dem lokalen Interface beim Direct-Routing, besitzen die Realserver auch beim IP-Tunneling ein zweites Interface mit der IP-Adresse des Loadbalancers. Dies ist allerdings hier keine lokale Schnittstelle, sondern ein Tunnel-Interface. Wie beim Direct Routing gilt für das IP-Tunneling-Verfahren aber auch: Befindet sich der Realserver im gleichen Netz, wie der Loadbalancer, so muss das ARP-Problem gelöst werden. Da der Realserver auch hier eine Schnittstelle mit der IP-Adresse des Loadbalancers besitzt, kann es ansonsten dazu führen, dass bei einer Anfrage an den

<sup>12</sup>aus „Computernetze“ von Larry L.Peterson und Bruce S.Davie, S.274 [PD00]

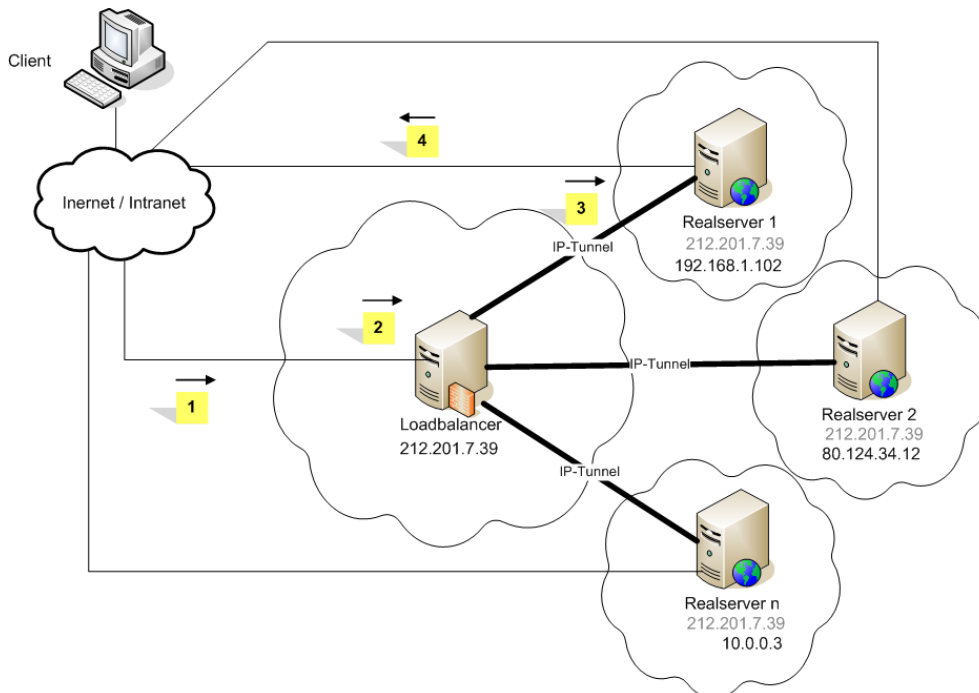


Abbildung 4.6: Loadbalancingverfahren: IP-Tunneling

Loadbalancer der Realserver statt dessen antworten und somit der Lastverteilung entgegen könnte. Antworten zum ARP-Problem und dessen Lösung ist Gegenstand von Kapitel 4.5 auf Seite 40.

Wie in Abbildung 4.6 aufgezeigt, verläuft das Lastverteilungsverfahren beim IP-Tunneling so, dass – genau wie bei den anderen Verfahren auch – Clientanfragen an die IP-Adresse des Loadbalancer (hier: 212.201.7.39) gerichtet werden (1). Der Loadbalancer wählt bei neuen Anfragen über einen Scheduling-Algorithmus einen Realserver aus und verpackt das IP-Paket des Client in ein neues IP-Paket mit Zieladresse des Realservers (2). Von einer Verteilung des Paketes an Realserver 1 ausgehend, entpackt dieser das innere IP-Paket und besitzt nun das Anfragepaket des Clients (3). Das Tunnel-Interface mit der IP-Adresse des Loadbalancers nimmt das Paket des Clients an und verarbeitet es weiter. Bei einer Webanfrage etwa generiert der Webserver die Antwortpakete und schickt diese über ein beliebiges Standard-Gateway über das Internet oder Intranet zurück an den Client (4).

Das IP-Tunneling-Verfahren ist durch das Verpacken von IP-Paketen geringfügig aufwändiger, als Direct Routing. Befinden sich alle Realserver im gleichen Netz, wie der Loadbalancer, so ist es unnötig das IP-Tunneling-Verfahren zu verwenden. Auch mit diesem Verfahren können im Grunde uneingeschränkt viele Realserver betrieben werden. Ein Nachteil von IP-Tunneling im Vergleich zu den Vorgängerverfahren ist das nicht stattfindende *Port remapping*<sup>13</sup>.

<sup>13</sup>Beim IP-Tunneling-Verfahren muss der Port an dem ein Dienst auf dem Realserver lauscht, iden-

## 4.2 Scheduling-Algorithmen

Dieser Abschnitt enthält eine Auflistung und Erklärung über die verschiedenen Scheduling-Algorithmen, mit der die Loadbalancer des LVS-Projektes ihre Realserver auswählen können. Bei dieser Lösung werden die Scheduling-Algorithmen als Kernel-Module geladen. Die Beschreibungen der Algorithmen findet sich in ähnlicher Form in der Manpage des Administrationswerkzeugs des LVS-Projektes [WZH04]. In der Version 1.24 des *ipvsadm* Konfigurationswerkzeugs, kann zwischen den folgenden zehn Verfahren zur Lastverteilung ausgewählt werden:

### Round Robin (rr)

Dieses einfache Verfahren, welches bereits auf Seite 22 erwähnt wurde, behandelt jeden Realserver gleich. Jeder kommt abwechselnd in gleicher Reihenfolge immer wieder an die Reihe.

### Weighted Round Robin (wrr)

Beim Einrichten des Loadbalancers kann bei diesem Verfahren jedem Realserver eine Gewichtung zugeteilt werden. Dabei werden höhergewichtete Rechner bevorzugt. Ansonsten verhält sich dieses Verfahren wie Round Robin. Erhält bspw. Realserver 1 eine Gewichtung von 1 und Realserver 2 eine Gewichtung von 2, so leitet der Loadbalancer immer eine Anfrage an Realserver 1 und zwei Anfragen an Realserver 2 weiter.

### Least-Connection (lc)

Der Lastverteiler kennt durch ankommende Pakete von Clients die Anzahl aktiver Verbindungen zu den Realservern. Bei diesem Verfahren teilt er eine neue Anfrage *dem* Realservern zu, der derzeit am wenigsten offene Verbindungen bearbeiten muss. Das Verfahren gleicht nur direkt nach dem Start des Loadbalancers dem Round Robin Verfahren. Später ergeben sich auf den Realservern unterschiedlich viele Verbindungen, da jede Verbindung unterschiedlich lang aufrechtgehalten wird.

### Weighted Least-Connection (wlc)

Dieses Verfahren ist analog zum Weighted Round Robin zu sehen. Den Realservern wird eine Gewichtung zugewiesen. Höher gewichteten Realservern werden mehr neue Verbindungsanfragen zugeteilt, als niedriger gewichteten. Besitzen bei einer ankommenden Anfrage bspw. zwei Realserver die gleiche Anzahl an offenen Verbindungen, jedoch einer eine höhere Gewichtung, so wird dieser vom Loadbalancer bevorzugt.

### Locality-Based Least-Connection (lbc)

Solche Verbindungen, welche an den gleichen Ziel-Socket gerichtet sind, werden dem selben Realserver zugewiesen, sofern dieser erreichbar und nicht zu

---

tisch mit dem Zielport des Anfragepaketes sein. Bei den beiden anderen Verfahren können sich die Ports auf denen der Dienst auf dem Realserver lauscht von denen, die der Loadbalancer annimmt unterscheiden.

überladen ist. Ist dies der Fall, wird das Least-Connection Verfahren angewendet.<sup>14</sup>

### Locality-Based Least-Connection with Replication (lblr)

Bei diesem Verfahren werden Realserver zu Gruppen zusammengefasst. Verbindungen, die den gleichen Ziel-Socket (also Dienst) verwenden, werden dem Realserver der Dienstgruppe zugewiesen, der am wenigsten aktive Verbindungen besitzt. Wenn alle Realserver dieser Gruppe überladen sind, wird für kurze Zeit ein Realserver aus einer anderen Gruppe entliehen.

### Destination Hashing (dh)

Bei diesem Verfahren wird der Realserver mit Hilfe des Ziel-Sockets und einer statischen Hashtabelle zugewiesen. Dabei findet keinerlei Verbindungsüberwachung, wie etwa bei den vorherigen Verfahren statt.

### Source Hashing (sh)

Dieses Verfahren entspricht dem Destination Hashing, nur dass der Realserver nicht anhand des Ziel-Sockets ermittelt wird, sondern durch den Quell-Socket des ankommenden Pakets.

### Shortest Expected Delay (sed)

Eine ankommende Verbindungsanfrage wird an den Realserver mit der derzeit kürzesten Verzögerungszeit zugewiesen. Diese berechnet sich wie folgt:

$$d_i = \frac{C_i + 1}{U_i}$$

$d_i$  := Erwartete Verzögerung auf Realserver  $i$

$C_i$  := Anzahl der existierenden Verbindungen zum Realserver  $i$

$U_i$  := Gewichtung von Realserver  $i$

### Never Queue (nq)

Hier wird eine eingehende Verbindung ohne Prüfung an einen derzeit ungenutzten Realserver weitergeleitet. Ist kein ungenutzter Server verfügbar, wird das Shortest Expected Delay Verfahren angewendet.

## 4.3 Einrichten der Lastverteiler

Die Grundlagen zur Lastverteilung mittels des LVS-Projektes wurden behandelt. Es sollte nun möglich sein, Lastverteiler mit den in Kapitel 4.1 beschriebenen Verfahren zu installieren. Dazu wird ein Linux-Kern mit der Version 2.4.23 oder höher benötigt. Alternativ gibt es für frühere Versionen Kernel-Patches. Zusätzlich wird das

<sup>14</sup>In der Manpage zu *ipvsadm* [WZH04] ist nicht die Rede von Ziel-Sockets sondern von Ziel-IP-Adressen. Da allerdings sämtliche Anfragepakete an den Loadbalancer die gleiche Zieladresse besitzen – nämlich die des Loadbalancers selbst – wäre dieses Verfahren hinfällig. Schicht-4-Loadbalancer unterscheiden Anfragen an Dienste anhand des Ports und nicht anhand der IP-Adresse.

Kontrollprogramm *ipvsadm* benötigt. Mit diesem wird der Loadbalancer eingerichtet und kontrolliert. Es legt alle Regeln und Einstellungen fest, die in diesem Kapitel bisher erwähnt wurden. Die neueste Version des Kontrollprogramms befindet sich im Downloadbereich des LVS-Projektes<sup>15</sup> als *.src.rpm*- und *.tar.gz*-Archiv.

Das Einrichten eines Lastverteilers besteht aus Sicht von *ipvsadm* aus zwei Schritten:

1. Festlegen des weiterzuleitenden Dienstes und des Scheduling-Algorithmus.
2. Hinzufügen der Realserver, die diesen Dienst zur Verfügung stellen.

Dieser Vorgang wird an dem folgenden Beispiel einer SSH-Verbindung zum Loadbalancer (192.168.0.10) und den Realservern SSH1 mit der IP-Adresse 192.168.0.101 und SSH2 mit der IP-Adresse 192.168.0.102 gezeigt. Verwendet wird das Verfahren Direct Routing mit dem Scheduling-Algorithmus *Weighted Round Robin*:

```
(1) # ipvsadm -A -t 192.168.0.10:22 -s wrr -p
(2) # ipvsadm -a -t 192.168.0.10:22 \
    -r 192.168.0.101:22 -g -w 1
(3) # ipvsadm -a -t 192.168.0.10:ssh \
    -r 192.168.0.102:ssh -g -w 2
```

Zeile (1) fügt dem Loadbalancer mit dem Schalter *-A* einen neuen weiterzuleitenden Dienst hinzu. Die Option *-t* weist den folgenden Socket als einen TCP-Socket aus. Der *-s* Schalter legt danach den Scheduling-Algorithmus fest, der im Beispiel mit dem Kürzel *wrr* als *Weighted Round Robin* angegeben wurde. Zu guter Letzt sollte bei einem sitzungsorinetierten Protokoll, wie dem SSH-Protokoll noch das *-p* (=persistent) Flag zugewiesen werden. Dadurch werden sämtliche Pakete, die zu einer Verbindung gehören immer an den gleichen Realserver weitergeleitet.

Die Zeilen (2) und (3) legen daraufhin die Weiterleitungsregeln für den SSH-Dienst zu den Realservern fest. Der Schalter *-a* fügt eine Realserverregel hinzu. Auch hier bewirkt *-t*, dass für den nachfolgenden Socket das TCP-Protokoll gilt. Die Weiterleitungsmethode wird in diesem Beispiel mit dem Schalter *-g* (=gatewaying) auf Direct Routing gesetzt. Anschließend kann mit *-w* noch eine Gewichtung für jeden Realserver vorgenommen werden. Es wird beispielhaft davon ausgegangen, dass der Realserver SSH2 mehr Systemressourcen besitzt. Daher wurde dieser doppelt so hoch gewichtet, als Realserver SSH1.

Abbildung 4.7 zeigt die Ausgabe des Befehls *ipvsadm* direkt nach dem Anlegen der oben angegebenen Regeln. Jede am Anfang der Zeile beginnende Ausgabe gibt die Weiterleitung für einen Dienst an. Mit einem „->“ beginnende Zeilen zeigen je einen Realserver an. In den Realserverzeilen werden verschiedene Informationen zum aktuellen Status ausgegeben. Nach der IP-Adresse des Realservers wird zuerst der Weiterleitungsmechanismus angezeigt. Bei Direct Routing steht dort das Wort *Route*. In

<sup>15</sup>LVS Downloads: <http://www.linuxvirtualserver.org/software/ipvs.html>

```

tim@kamino: /home/tim
[root@kamino tim]# ipvsadm
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  192.168.0.10:ssh wrr persistent 360
  -> 192.168.0.12:ssh             Route   2      0         0
  -> 192.168.0.11:ssh             Route   1      0         0
[root@kamino tim]#

```

Abbildung 4.7: ipvsadm: Ausgabe nach der Konfiguration

der nächsten Spalte steht die Gewichtung des Realservers. Die letzten beiden Spalten zeigen die derzeit aktiven und inaktiven Verbindungen zum Realserver an.

Um den Vorgang zu kontrollieren oder über einen längeren Zeitraum die Verbindungen zu den Realservern anzuzeigen, kann folgendes einfache Skript verwendet werden:

```

while [ 1 ]; do
    clear
    ipvsadm
    echo ""
    echo "Drücken Sie Strg+C zum Abbrechen"
    sleep 5
done

```

Dadurch wird die Ausgabe alle fünf Sekunden aktualisiert. Veränderungen der Verbindungen können so direkt am Monitor beobachtet werden.

### 4.3.1 Network Address Translation

Für das folgende Beispiel wird davon ausgegangen, dass die einzige Netzwerk-Schnittstelle des Loadbalancers *eth0* ist. Da der Loadbalancer im Fachbereich MNI nur ein Interface besitzt, wird die zweite Schnittstelle durch ein virtuelles Interface mit der Bezeichnung *eth0:0* simuliert. Bei zwei vorhandenen realen Schnittstellen sind die Namen entsprechend zu ersetzen.

Um einen Rechner als NAT-Loadbalancer einzurichten, muss zuerst die IP-Weiterleitung aktiviert werden. In Linux geschieht dies über einen Schreibzugriff auf ein Kernel-Flag:

```
# echo "1" >/proc/sys/net/ipv4/ip_forward
```

Außerdem müssen *ICMP*<sup>16</sup>-Redirects abgeschaltet werden. Ein ICMP-Redirect wird in folgender Situation gesendet:

<sup>16</sup>Internet Control Message Protocol – Dient als Teil von TCP/IP dem Austausch von Fehler- und Informationsmeldungen.

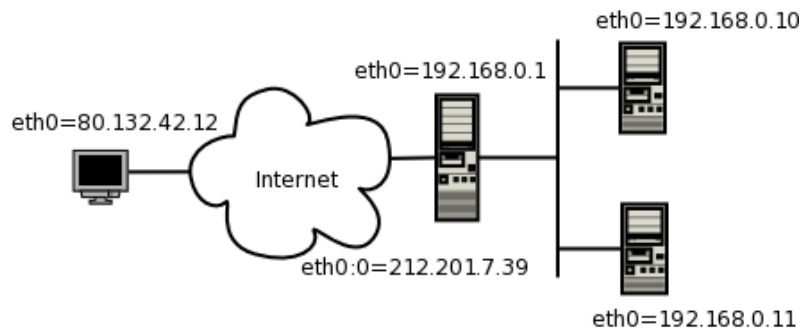


Abbildung 4.8: LVS NAT-Konfiguration

- Host A sendet ein Paket über seinen Standard-Router R1 an Host B, welcher sich in einem anderen Netz befindet.
- Router R1 prüft seine Routing-Tabelle und stellt fest, dass der nächste Router für den Weg zu Host B der Router R2 ist.
- Sind Host A, Router R1 und Router R2 an das gleiche Netz angeschlossen, wird von Router R1 ein ICMP-Redirect Paket an Host A zurückgesendet, welches diesen informiert, dass der Weg über Router R2 direkter ist.
- Router R1 sendet dennoch das IP-Paket an Router R2.
- Host A fügt eine neue Host-Route in seine Routingtabelle ein und sendet zukünftige Pakete direkt über Router R2.

Da aber der Loadbalancer beim NAT-Verfahren unbedingt als Router auf dem Rückweg dienen muss (siehe Tabelle 4.1 auf Seite 24), müssen die ICMP-Redirects in diesem Fall für diesen Rechner deaktiviert werden. Auch das geschieht über ein Kernel-Flag:

```
# echo "0" >/proc/sys/net/ipv4/conf/all/ \
  send_redirects
# echo "0" >/proc/sys/net/ipv4/conf/default/ \
  send_redirects
# echo "0" >/proc/sys/net/ipv4/conf/eth0/ \
  send_redirects
```

Anschließend wird das Interface zur Außenwelt mit der IP-Adresse konfiguriert, die nach außen hin gültig ist und mit der das Webportal erreicht werden soll:

```
# ifconfig eth0:0 212.201.7.39 netmask 255.255.255.0
```

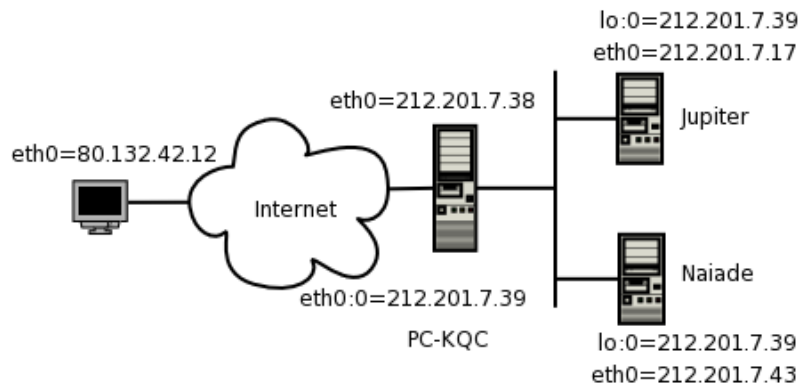


Abbildung 4.9: LVS Direct Routing-Konfiguration

Als Default-Gateway wird ein Router ins Internet oder Intranet gesetzt. Damit sind die Vorbereitungen abgeschlossen:

```
# route add default gw 212.201.7.1
```

Nun werden die Regeln für die NAT-Lastverteilung nach dem Muster gesetzt, wie es bereits auf Seite 29 behandelt wurde.

```
# ipvsadm -C
# ipvsadm -A -t 212.201.7.39:http -s rr
# ipvsadm -a -t 212.201.7:39:http \
-r 192.168.0.10:http -m
# ipvsadm -a -t 212.201.7:39:http \
-r 192.168.0.11:8080 -m
```

Anstelle der Portnummern können auch die Aliasnamen aus der Datei */etc/services* verwendet werden. Es gibt noch einige andere Unterschiede zu dem vorherigen Beispiel. Zum einen wird *HTTP*, also einen verbindungsloser Dienst verwendet. Daher wird auf den Schalter *-p* verzichtet. Außerdem wird mit der Option *-m* festgelegt, dass NAT verwendet werden soll. Beim NAT-Verfahren ist es möglich, dass die Realserver auf anderen Ports lauschen, als der Loadbalancer. Für diese Konfiguration wird davon ausgegangen, dass der Realserver mit der IP-Adresse 192.168.0.11 auf Port 8080 lauscht.

### 4.3.2 Direct Routing

Beim Direct Routing-Verfahren müssen sich bekanntlich alle Realserver und der Loadbalancer im gleichen Netz befinden. Verwendet wird auch hier die gleiche Rechnerkonfiguration wie zuvor. Die Testrechner im Fachbereich MNI befinden sich alle

im gleichen Netz. Deren Adresskonfigurationen kann in Abbildung 4.9 eingesehen werden.

Da hier wie bereits auf Seite 24 beschrieben im Grunde auf Basis der Hardwareadressen geroutet wird, kann IP-Forwarding abgeschaltet werden:

```
# echo "0" >/proc/sys/net/ipv4/ip_forward
```

Da der Loadbalancer bei diesem Verfahren nicht als Standard-Gateway für die Antwortpakete dient, können ICMP-Redirects keine Probleme verursachen. Diese können wieder aktiviert werden:

```
# echo "1" >/proc/sys/net/ipv4/conf/all/ \
  send_redirects
# echo "1" >/proc/sys/net/ipv4/conf/default/ \
  send_redirects
# echo "1" >/proc/sys/net/ipv4/conf/eth0/ \
  send_redirects
```

Genau wie beim NAT-Verfahren wird die öffentliche Schnittstelle des Loadbalancers konfiguriert. Dabei ist es wichtig, dass die Broadcast-Adresse auf die IP-Adresse gesetzt wird. Anschließend wird noch eine Host-Route auf das Interface gesetzt:

```
# ifconfig eth0:0 212.201.7.39 \
  broadcast 212.201.7.39 netmask 255.255.255.0
# route add -host 212.201.7.39 dev eth0:0
```

Somit sind alle Vorbereitungen getroffen, um die Regeln für den Loadbalancer zu setzen:

```
# ipvsadm -C
# ipvsadm -A -t 212.201.7.39:http -s rr
# ipvsadm -a -t 212.201.7:39:http \
  -r 212.201.7.17:8080 -g
# ipvsadm -a -t 212.201.7:39:http \
  -r 212.201.7.43:80 -g
```

Da auf dem Realserver *Jupiter* (212.201.7.17) keine *root*-Rechte zur Verfügung stehen, kann der Webserver dort nicht auf dem Standardport 80 laufen<sup>17</sup>. Aber auch Direct Routing kann die Zielports der Anfragepakete umschreiben. Als letzte Option wurde hier der *-g* Schalter gesetzt, der dem LVS-Dienst deutlich macht, dass für die Weiterleitung an die Realserver das Direct Routing-Verfahren eingesetzt wird.

<sup>17</sup>Ports bis einschließlich 1024 können nur durch Prozesse belegt werden, die mit Root-Rechten gestartet wurden.

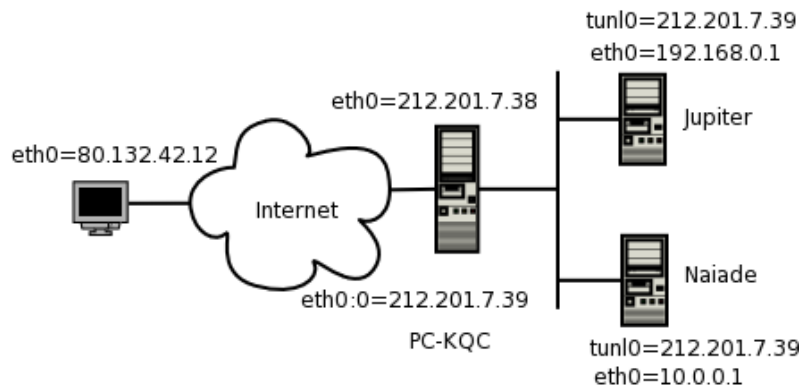


Abbildung 4.10: LVS-IP-Tunnel Konfiguration

### 4.3.3 IP-Tunneling

Das IP-Tunneling-Verfahren ist das flexibelste. Es kann auch dann verwendet werden, wenn sich Realserver und Loadbalancer in unterschiedlichen Netzen befinden. Theoretisch ist damit sogar eine Realserverfarm vorstellbar, die über das Internet verteilt ist. Allerdings wäre das für den Performance-Aspekt nicht gerade förderlich. Als Beispiel wurden *Jupiter* und *Naiade* private IP-Adressen in unterschiedlichen Netzen zugewiesen, damit noch einmal deutlich wird, dass dies über den IP-Tunnel möglich ist.

Wie beim Direct-Routing Verfahren kann die IP-Weiterleitung hierbei deaktiviert werden:

```
# echo "0" >/proc/sys/net/ipv4/ip_forward
```

Auch die Konfiguration der öffentlichen Netzchnittstelle gleicht der Konfiguration für Direct Routing:

```
# ifconfig eth0:0 212.201.7.39 \
  broadcast 212.201.7.39 netmask 255.255.255.0
```

Die Konfiguration des Loadbalancer-Kerns bietet mittlerweile auch nicht mehr viel neues.

```
# ipvsadm -C
# ipvsadm -A -t 212.201.7.39:http -s rr
# ipvsadm -a -t 212.201.7:39:http \
  -r 192.168.0.1 -i
# ipvsadm -a -t 212.201.7:39:http \
  -r 10.0.0.1 -i
```

```

IP Virtual Server version 1.0.10 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP pc-kqc.mni.fh-giessen.de:ama wrr
-> Naiade.mni.fh-giessen.de:amanda Tunnel 1 0 3
-> jupiter.mni.fh-giessen.de:amanda Route 1 1 0
-> pc-kqc.mni.fh-giessen.de:amanda Local 1 1 0
-> localhost:amanda Local 0 0 0
TCP pc-kqc.mni.fh-giessen.de:104 wrr
-> jupiter.mni.fh-giessen.de:10443 Route 1 0 0
-> pc-kqc.mni.fh-giessen.de:10443 Local 1 1 0
-> localhost:10443 Local 0 0 0
TCP pc-kqc.mni.fh-giessen.de:mysql wrr
-> Naiade.mni.fh-giessen.de:mysql Tunnel 1 0 0
-> jupiter.mni.fh-giessen.de:mysql Route 1 0 0
-> pc-kqc.mni.fh-giessen.de:mysql Local 1 1 0
-> localhost:mysql Local 1 0 0

```

Abbildung 4.11: LVS-Konfiguration im Fachbereich MNI

Um das IP-Tunneling-Verfahren zu verwenden, wird das `-i` Flag gesetzt. Eine netz-übergreifende Weiterleitung von Paketen ist somit möglich. Eine Einschränkung besitzt IP-Tunneling allerdings. Der Port für den entsprechenden Dienst muss beim Tunneln auf den Realserver gleich bleiben. Da ein IP-Tunnel kein *Port remapping* anbietet, ist es hiermit also nicht möglich bspw. Port 80 auf Port 8080 umzusetzen. Natürlich kann der Realserver durch eigenes *Port forwarding* ein empfangenes Paket vom Realserver immer noch auf einen andern Port lenken, aber direkt über den IP-Tunnel und somit über die Loadbalancer-Konfiguration ist dies nicht möglich. Für die Konfiguration der Realserver reicht daher auch die Angabe der IP-Adresse ohne den Port aus.

#### 4.3.4 Konfigurationsbeispiel Fachbereich MNI

Bei der Einrichtung des Loadbalancers und der Realserverfarm im Fachbereich MNI zeigte sich, dass die Verfahren Direct-Routing und IP-Tunneling auch hervorragend zusammen eingesetzt werden können. Da alle verwendeten Rechner im Fachbereich zum gleichen Netz gehören, war Direct Routing die erste Wahl. Das NAT-Verfahren schied von vorneherein aus, da es nur bis etwa 25 Realserver performant ist. Zwar hätte das im Fachbereich keine Probleme gegeben, da ohnehin nur zwei Realserver vorhanden waren, aber die Aufgabenstellung verlangte ein System für  $n$  Realserver. Für IP-Tunneling bestand dem ersten Anschein nach keine Veranlassung. Tests zeigten allerdings, dass der virtuelle Rechner *Naiade* nicht mit Direct Routing konfigurierbar war. Nach längerer Fehlersuche stellte sich schließlich heraus, dass ARP-Anfragen an sämtliche virtuelle Maschinen lediglich die Hardwareadresse ihres Hostrechners *Neptun* lieferten. Direct-Routing war somit für die virtuellen Maschinen des Fachbereichs nicht einsetzbar.

In Abbildung 4.11 ist zu erkennen, dass der virtuelle Rechner *Naiade* einen IP-Tunnel verwendet, während die reale Solaris-Maschine *Jupiter* über Direct-Routing erreicht wird. Auch der Loadbalancer selbst wird hier als Realserver eingesetzt. Das Wort *local* weist darauf hin. In der Abbildung werden Anfragen an die Ports 10080 (HTTP),

10443 (HTTPS<sup>18</sup>) und 3306 (MySQL) nach dem Weighted Round Robin- Verfahren (wrr) verteilt.

## 4.4 Einrichten der Realserver

Die Realserver benötigen keine spezielle Software, wie etwa die Loadbalancer. Von daher gibt es wenige Gemeinsamkeiten bei der Konfiguration.

### 4.4.1 Network Address Translation

Wie bereits mehrmals erwähnt, muss der Loadbalancer beim LVS-Nat-Verfahren als Gateway für die Antwortpakete eingerichtet werden. Der erste Schritt beim Einrichten der Realserverumgebung ist daher das Setzen des Standard-Gateways auf den Loadbalancer:

```
# route add default gw 192.168.0.1
```

Anschließend muss nur noch die IP-Weiterleitung für den Realserver ausgeschaltet werden:

```
# echo "0" >/proc/sys/net/ipv4/ip_forward
```

Das war bereits alles, was auf dem Realserver beachtet werden muss. Natürlich müssen die benötigten Dienste (in diesem Fall *httpd*) gestartet sein. Wie sich im Folgenden zeigen wird, ist das LVS-Verfahren mit NAT was die Konfiguration der Realserver betrifft mit Abstand das einfachste.

### 4.4.2 Direct Routing

Die Konfiguration für einen Direct-Routing Realserver ist aufwendiger, als die für einen NAT-Realserver. Zuerst muss auch hier das Standardgateway eingestellt werden. Dies kann ein beliebiges sein. Für die beste Performance ist es natürlich von Vorteil, wenn die verschiedenen Realserver unterschiedliche Standardgateways verwenden. Außerdem könnten beim Ausfall *eines* Gateways dann die betreffenden Realserver aus der *ipvsadm*-Tabelle entfernt werden. Danach könnten diese in Ruhe umkonfiguriert werden, während das Portal durch die übrigen Realserver weiterhin erreichbar ist:

---

<sup>18</sup>Hypertext Transfer Protocol Secure – Sichere sitzungsorientierte Version von HTTP.

```
# route add default gw 212.201.7.1
```

IP-Weiterleitung wird genau wie beim NAT-Verfahren auch abgeschaltet:

```
# echo "0" >/proc/sys/net/ipv4/ip_forward
```

Nun kommt der etwas kniffligere Teil. Da beim Direct-Routing die IP-Adressen nicht verändert werden, muss der Realserver dem IP-Paket vorgaukeln, dass es sich noch auf seinem Zielrechner befindet. Dazu wird ein Alias-Interface auf der lokalen Schnittstelle erzeugt und diesem die IP-Adresse des Loadbalancers zugewiesen:

```
# ifconfig lo:39 212.201.7.39 \
    broadcast 212.201.7.39 netmask 255.255.255.255
```

Das ähnelt der Schnittstellenkonfiguration des Direct-Routing Loadbalancers. Beachtet werden muss dabei, dass für Netzmasken von lokalen Schnittstellen alle Bits gesetzt werden (255.255.255.255).

Anschließend wird eine Host-Route für die Loadbalancer IP-Adresse auf das lokale Interface gesetzt, damit die empfangenen Pakete den Realserver nicht wieder verlassen:

```
# route add -host 212.201.7.39 dev lo:39
```

Der Realserver ist fast fertig eingerichtet. Jedoch existiert das bereits erwähnte ARP-Problem. Damit der Realserver richtig arbeitet, muss dieses Problem zuerst behoben werden. Da das ARP-Problem auch beim IP-Tunneling Verfahren auftritt und es einige verschiedene Lösungsansätze dazu gibt, ist diesem Thema ein eigener Abschnitt gewidmet. Die Lösung des ARP-Problems wird in Abschnitt 4.5 auf Seite 40 behandelt.

### 4.4.3 IP-Tunneling

IP-Tunneling ist das für einen Realserver am aufwendigsten zu aktivierende Verfahren. Um unter Linux einen IP-Tunnel aufbauen zu können, gibt es in den aktuellen Distributionen ein entsprechendes Kernel-Modul namens *ipip*. Ist dieses fest im Kernel eingebunden, entfällt natürlich der folgende Aufruf, mit dem bei einem modularen Kern das Modul geladen wird:

```
# modprobe ipip
```

Die IP-Weiterleitung muss nicht aktiviert sein:

```
# echo "0" >/proc/sys/net/ipv4/ip_forward
```

Da sich beim IP-Tunneling die Realserver ohnehin in unterschiedlichen Netzen befinden, stellt sich erst überhaupt nicht die Frage, ob das Standard-Gateway gleich sein muss:

```
# route add default gw 212.201.7.1
```

Da das `ipip`-Modul geladen wurde oder fest im Kern installiert ist, kann nun das Tunnel-Interface `tunl0` verwendet werden. Dieses wird erst einmal gestartet und später konfiguriert:

```
# ifconfig tunl0 0.0.0.0 -arp up
```

Nun sollte zuerst das ARP-Problem für die jeweils verwendete Plattform gelöst werden. Das Flag `-arp` verhindert unter Solaris, dass ARP-Anfragen durchgeführt werden. Unter Linux ist dieses Flag zwar aus Kompatibilitätsgründen Teil des `ifconfig`-Programms, zeigt jedoch keine Wirkung. Der nächsten Abschnitt zeigt Lösungswege auf, wie ARP-Abfragen unter Linux verhindert werden können.

Das Starten des `tunl0`-Interfaces ist für die Lösung des ARP-Problems je nach Ansatz notwendig. Daher wurde es zuvor unkonfiguriert gestartet. Wurden die notwendigen Einstellungen getätigt, kann das Tunnel-Interface konfiguriert werden:

```
# ifconfig tunl0 212.201.7.39 \
  broadcast 212.201.7.39 netmask 255.255.255.255
```

Abschließend wird noch genau wie beim Direct-Routing eine Host-Route auf das eben eingerichtete Interface gesetzt:

```
# route add -host 212.201.7.39 dev tunl0
```

Schließlich muss unbedingt noch verhindert werden, dass die Pakete, die über das Tunnel-Interface ankommen nach dem Entpacken gleich wieder verworfen werden. Dafür gibt es in aktuellen Linux-Kerneln das `rp_filter` Kernelflag. Der `rp_filter` ist eine `iptables`<sup>19</sup>-Kette welche Pakete verwirft, die über eine Schnittstelle ankommen, über die sie eigentlich nicht hätten eintreffen dürfen. Dieses muss für das `tunl0`-Interface deaktiviert werden:

```
# echo "0" >/proc/sys/net/ipv4/conf/tunl0/rp_filter
```

Damit ist die Konfiguration des IP-Tunnel-Realservers abgeschlossen und er ist nun einsatzbereit.

<sup>19</sup>Umfangreicher Paketfilter in Linux. Nachfolger von `ipchains`.

## 4.5 Das ARP-Problem

Bei den Verfahren Direct Routing und IP-Tunneling wird auf dem Realserver bestimmten Netzwerkschnittstellen die gleiche IP-Adresse zugewiesen, wie sie bereits der Loadbalancer verwendet. Dies ist auch korrekt und notwendig, denn beide Verfahren basieren darauf, dass die Ziel-IP-Adresse der ankommenden Pakete an den Loadbalancer gerichtet ist und von diesem nicht verändert wird. Beim Direct-Routing-Verfahren wird vom Loadbalancer die MAC-Adresse des Pakets umgeschrieben und so an den Realserver weitergeleitet. Beim IP-Tunneling wird das Paket in ein weiteres IP-Paket verpackt und vom Empfänger wieder ausgepackt.

Die Notwendigkeit auf den Realservern gleiche IP-Adressen wie auf dem Loadbalancer einzusetzen besteht, wie in Unterabschnitt 4.1.3 auf Seite 24 beschrieben darin, dass ein Realserver jedes Paket verwerfen würde, dessen Zieladresse nicht mit der Adresse einer seiner Schnittstellen übereinstimmt. Befinden sich Loadbalancer und Realserver im gleichen Netz, tritt allerdings das Problem auf, dass ARP-Anfragen durchgeführt werden. Jede IP-Adresse im gleichen Netz wird in die Hardwareadresse der Netzschnittstelle aufgelöst und in einer Tabelle im Betriebssystemkern für einige Zeit gespeichert. Das führt bei gleichen IP-Adressen an unterschiedlichen Hardwareadressen zu undefiniertem Verhalten, da nun mehrere Rechner auf eine IP-Anfrage antworten können. In der Regel stellt sich dieses Problem bei Realservern so dar, dass sie beim Ansprechen über den Lastverteiler nicht antworten, direkt jedoch ganz normal erreichbar sind. Abbildung 4.12 zeigt, wie das Problem erkannt werden kann.

Der in der Abbildung eingesetzte Loadbalancer besitzt die IP-Adresse 192.168.0.120. Ein Realserver besitzt die IP-Adresse 192.168.0.106. Es wurde das Verfahren Direct Routing angewendet, da sich der Realserver und der Loadbalancer im gleichen Netz befanden. Nachdem von einem Test-Client aus versucht wurde, über den Loadbalancer eine Website aufzurufen, die Antwort jedoch ausblieb, wurde der Realserver vom Test-Client aus direkt kontaktiert. Anschließend wurde mit dem Befehl `arp -n` die ARP-Tabelle des Test-Clients angezeigt. Wie in der Abbildung zu sehen ist, wird sowohl die IP-Adresse für den Loadbalancer (192.168.0.120) als auch die für den Realserver (192.168.0.106) in die Hardwareadresse 00:0D:54:9B:20:2C aufgelöst. Dies ist die Hardwareadresse des Realservers. Der Konflikt ist vergleichbar mit dem Problem, wenn zwei unterschiedlichen Rechnern die selbe IP-Adresse zugewiesen wird. Nur findet er hier in der darunterliegenden OSI-Schicht statt. Ein IP-Adresskonflikt tritt beim Direct Routing und IP-Tunneling deshalb nicht auf, da die Schnittstellen, die mit der IP-Adresse des Loadbalancers konfiguriert wurden (lo:39 bzw tunl0) die IP-Adresse nach außen hin nicht sichtbar machen.

### Verschiedene Ansätze zur Verhinderung der Adressauflösung:

- Unter *Solaris* genügt es, `ifconfig` beim Einrichten der Schnittstelle mit dem Parameter `-arp` aufzurufen. Das bewirkt, dass für das angegebene Gerät keine ARP-Anfragen mehr beantwortet werden. Der `-arp` Parameter kann zwar auch

```

tim@Geonosis: /home/tim
[root@Geonosis tim]# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
192.168.0.106   ether   00:0D:54:9B:20:2C  C          eth1
192.168.0.1     ether   00:40:63:C0:34:4F  C          eth1
192.168.0.120   ether   00:0D:54:9B:20:2C  C          eth1
[root@Geonosis tim]#

```

Abbildung 4.12: Hier tritt das ARP-Problem auf

in den *ifconfig*-Versionen von Linux gesetzt werden, zeigt jedoch auf aktuellen Kernen keine Wirkung.

- In der Kernelversion 2.0 von Linux kann durch den *-noarp* Schalter beim Konfigurieren einer Schnittstelle verhindert werden, dass ARP-Requests angenommen werden. Ab den 2.2er Kernen sind zur Laufzeit verschiedene Kernel-Flags konfigurierbar um dies zu verhindern. Je nach Kernelversion unterscheiden sich diese Flags. Es folgt eine Auflistung der Flags und wie sie gesetzt werden müssen, damit ARP verhindert wird.

Das *hidden* Flag und die beiden anderen schließen sich aus. In Kernen der Version  $\geq 2.4.26$  wurde *hidden* durch *arp\_announce* und *arp\_ignore* ersetzt. Es finden sich also je nach Version des Linuxkerns entweder nur die Flags der ersten Zeile oder die der beiden anderen. Der Stern (\*) steht dabei als Platzhalter für das Verzeichnis *all*, *default* und den Namen der zu versteckenden Schnittstellen:

```

# echo "1" >/proc/sys/net/ipv4/conf/*/hidden

# echo "2" >/proc/sys/net/ipv4/conf/*/arp_announce
# echo "1" >/proc/sys/net/ipv4/conf/*/arp_ignore

```

- Für Linux-Kernel der Version 2.4 und 2.6 existiert ein nachladbares Kernelmodul, dass ARP-Requests für eingestellte Schnittstellen ignoriert. Das entsprechende Paket kann von Maurizio Sartoris Homepage<sup>20</sup> bezogen werden.
- Es kann Prerouting auf dem Realserver durchgeführt werden:
  - Mit *ipchains* kann dies leicht für alle Ports festgelegt werden:

```

# ipchains -A input -j REDIRECT -d VIP -p tcp
# ipchains -A input -j REDIRECT -d VIP -p udp

```

- Mit *iptables* kann Prerouting für jeden Dienst festgelegt werden:

```

# iptables -t nat -A PREROUTING -p tcp -d VIP \
--dport VPORT -j REDIRECT --to-port VPORT

```

<sup>20</sup>noarp-Kernelmodul <http://www.masarlabs.com/noarp/>

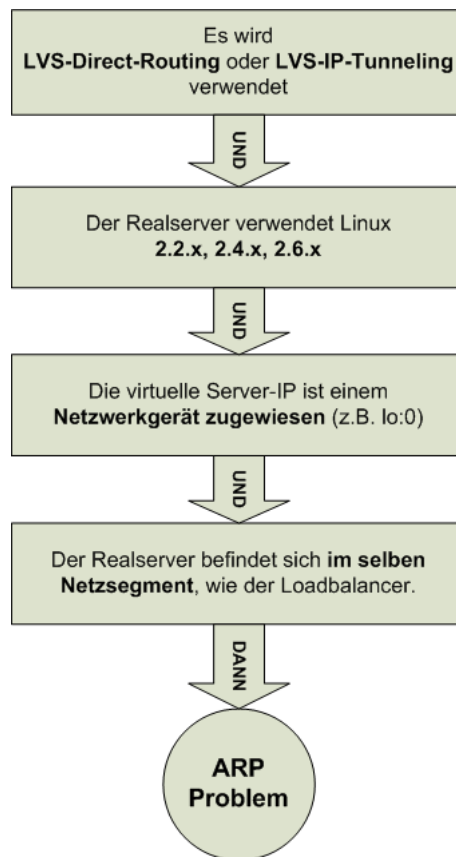


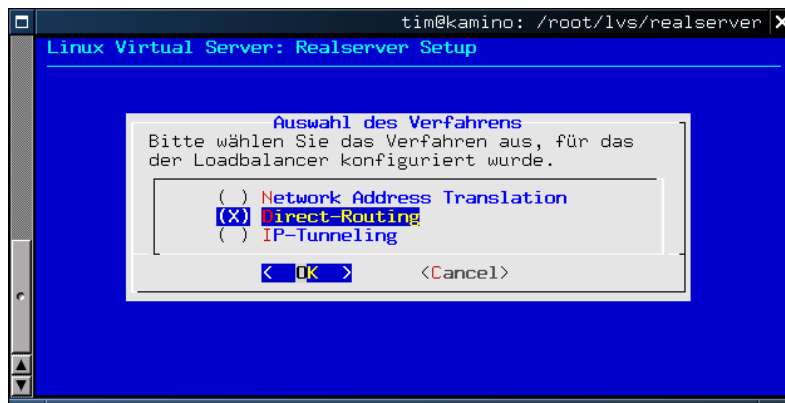
Abbildung 4.13: Wann kommt das ARP-Problem zur Geltung

- Wird ein *Transparenter Proxy* eingerichtet, kann das ARP-Problem ebenfalls umgangen werden. Ein Proxy ist ein Prozess, der zwischen einen Client und einen Dienst geschaltet wird um die Anfragen an den Dienst entgegenzunehmen und nach Bearbeitung weiterzuleiten. Auf diese Weise können Sicherheitslöcher des Dienstes im Vorhinein umgangen werden. Ein Webproxy speichert häufig geladene Internetseiten zwischen, um diese schneller abrufen zu können. Dabei muss im Browser allerdings die Adresse und der Port für den Proxy festgelegt werden. Bei einem transparenten Proxy entfällt diese Notwendigkeit. Ein transparenter Proxy kann Pakete akzeptieren, die nicht für die IP-Adresse des Rechners, auf dem er läuft bestimmt sind. Für einen Realserver bedeutet dies, dass er bei Einsatz eines Transparenten Proxies auf die Einrichtung einer virtuellen IP-Adresse des Loadbalancers verzichten und so das ARP-Problem umgehen könnte [Hor04b].

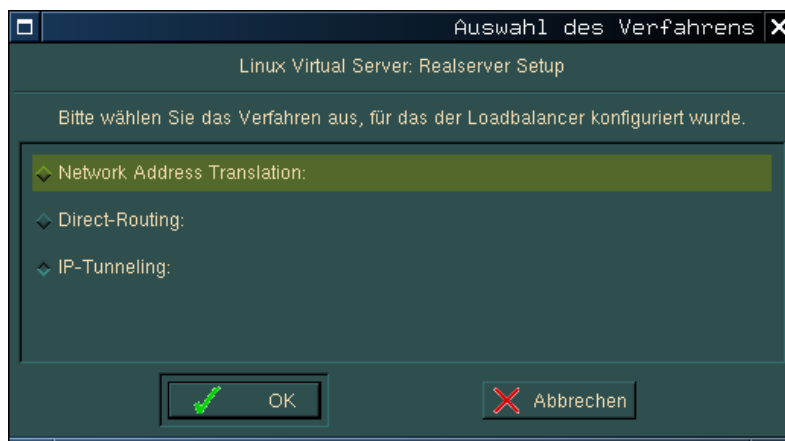
Auf der LVS-Website[Zha04a] existiert ein einfacher Algorithmus, der die Notwendigkeit zur Lösung des ARP-Problems im Vorhinein aufdecken kann. Abbildung 4.13 stellt diesen in einem Diagramm dar.

## 4.6 Realserver-Konfigurationstool

Für die komfortablere Realserverkonfiguration wurden einige Skripten erstellt, die es erlauben ohne großen Aufwand einen Realserver in der gewünschten Form zu starten. Nach Auswahl des Verfahrens (Abbildung 4.14(a)) und einiger Angaben zur Konfiguration, versucht das Tool entsprechend dieser Angaben den Realserver einzurichten. Gelingt dies nicht, wird die Fehlerursache angezeigt. In Abbildung 4.15 auf der nächsten Seite z.B. konnte das Standard-Gateway nicht eingestellt werden. In der Folgezeile wird jedoch angezeigt, dass es bereits eingerichtet war und erreichbar ist. Abbildung 4.14(b) zeigt noch einmal das gleiche Tool. Die Konfigurationsoberfläche ist auch über einen X-Dialog verfügbar.



(a) Auswahl des LVS-Verfahrens



(b) X-Version des Tools

Abbildung 4.14: Realserver-Konfigurationstool

```

tim@kamino: /root/lvs/realserver
----- Realserver für IP-Tunneling -----
Lade Modul 'ipip'... bereits geladen...           [PASSED]
Setze 'IP-Forwarding' auf '1' ...                 [ OK ]
Setze Default Gateway (192.168.0.1)...            [FEHLER]
Ist 192.168.0.1 ist das einzige gesetzte Default Gateway... [ OK ]
Ist 'Default Gateway' (192.168.0.1) erreichbar... [ OK ]
Ist 'Loadbalancer' (192.168.0.120) erreichbar...  [ OK ]
Bereite Tunnelinterface 'tun10' vor...           [ OK ]
Aktiviere Tunnelinterface 'tun10'...             [ OK ]
----- Realserver für IP-Tunneling -----
[root@kamino realserver]#

```

(a) Start des Realservers

Abbildung 4.15: Start einer Realserverkonfiguration

## 4.7 Lasttests

Um zu überprüfen, ob die Einrichtung eines Loadbalancer-Systems den erhofften Erfolg gebracht hat, wurden mehrere Lasttests durchgeführt.

Für Lasttests einer Website stehen heute verschiedene Open Source-Werkzeuge zur Verfügung. Diese vorzustellen ist ein anderes Thema und würde hier zu weit führen. Jedoch ist auf eine Vergleichsstudie bekannter Open Source-Lasttesttools von Christian Stoll und mir zu verweisen, die im Rahmen der Lehrveranstaltung Web-Performance bei Herrn Prof. Dr. Quibeldey-Cirkel im Winter 2004 durchgeführt wurde<sup>21</sup> [SP04].

Das hier verwendete Lasttesttool *Siege* hat zwar bei dieser Evaluation eher schlecht abgeschnitten, da es über keine interaktive Aufzeichnung von Skripten, sowie grafische Auswertung verfügt, es bot sich jedoch in diesem Falle an, weil es als Konsolenwerkzeug einfach über eine SSH-Konsole im lokalen Netz des Fachbereichs gestartet, und so direkt an der Quelle eingesetzt werden konnte.

### 4.7.1 Testkonfiguration

*Siege* misst folgende Kriterien:

#### Elapsed time

Dauer eines kompletten Testlaufs in Sekunden.

#### Data transfered

Anzahl der durch den Datenverkehr (inkl. der HTML-Header) an jeden virtuellen Benutzer übertragenen Bytes.

#### Response time

Durchschnittszeit in der für jeden virtuellen Benutzer geantwortet wird.

<sup>21</sup>interaktive tabellarische Lasttesttool-Evaluation inklusive grafischer Auswertung <http://tyranus.de/lasttest>

**Transaction rate**

Durchschnittliche Anzahl an Transaktionen, die pro Sekunde durchgeführt werden können. Anders ausgedrückt: Transaktionen / Gesamtzeit.

**Throughput**

Durchschnittliche Anzahl an Bytes, die pro Sekunde an den Benutzer gesendet werden.

**Concurrency**

Durchschnittliche Anzahl gleichzeitig offener Verbindungen. Ein Ansteigen dieses Wertes deutet auf sinkende Systemperformance des Servers hin.

Als Testseite wurde das in dieser Arbeit weiterentwickelte QA<sup>22</sup>-Portal (siehe Kapitel 7.1 auf Seite 86) verwendet. Folgendes einfache Skript wurde zur Lasterzeugung erzeugt:

**Beispiel 3.7.1 Siege Lastskript**

```
# Variablen
ROOT=http://pc-kqc.mni.fh-giessen.de:10080

# Startseite
$(ROOT)

# Impressum
$(ROOT)/modules.php?ZZZ_Credits

# Anmeldung
$(ROOT)/modules.php POST username=test \
  &user_password=test&random_num=423457& \
  gfx_check=& \op=login
$(ROOT)/modules.php?name=Your_Account& \
  op=userinfo&bypass=1&username=test

# Forum besuchen
$(ROOT)/modules.php?name=Forums
$(ROOT)/modules.php?name=Forums&file=viewform& \
  f=5&sid=9e5c4d920322505f86fdddc638614d9c
$(ROOT)/modules.php?name=Forums&file=viewtopic& \
  t=69&sid=9e5c4d920322505f86fdddc638614d9c
$(ROOT)/modules.php?name=Forums&file=posting \
  &mode=reply&t=69& \
  sid=9e5c4d920322505f86fdddc638614d9c

# Abmeldung
```

<sup>22</sup>Quality Assurance (Qualitätssicherung)

```
$(ROOT)/modules.php?name=Your_Account&op=logout
```

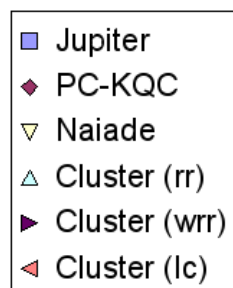
```
# Startseite
$(ROOT)/index.php
```

### 4.7.2 Tests

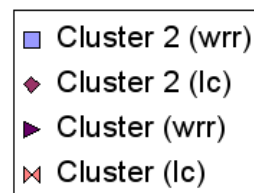
Da die Tests darauf abzielen sollten, Unterschiede zwischen den Clusterkonfigurationen und den Einzelrechnern herauszufinden, wurden diese als Stresstest (oder Benchmark) angelegt. Es wurde nicht durch variable Wartezeiten zwischen den einzelnen Anfragen das korrekte Verhalten von Benutzern simuliert, sondern darauf abgezielt, die Rechner durch eine stetig ansteigende Zahl von virtuellen Benutzern in die Knie zu zwingen.

Es wurden mehrere Lasttests durchgeführt. Beim ersten Test wurde in jedem Testdurchlauf die Anzahl der virtuellen Benutzer von 1 auf 50 erhöht, wobei jeder virtuelle Benutzer 20 Transaktionen durchführte. In Testlauf Nr. 1 wurden also 20 Transaktionen durchgeführt, in Testlauf Nr. 2 40 und in Testlauf Nr. 50 wurden mit 50 Benutzern und je 20 Transaktionen 1000 Transaktionen durchgeführt. Getestet wurden die Rechner *Jupiter*, *PC-KQC*, *Naiade*, sowie drei Clusterkonfigurationen mit unterschiedlichen Scheduling-Algorithmen. Zum Cluster gehörten jedesmal *Jupiter* (Direct Routing), *Naiade* (IP-Tunneling) und *PC-KQC* (local), also der Loadbalancer selbst (siehe Legende für Test 1 in Abbildung 4.16(a)). Folgende Clusterkonfigurationen wurden getestet:

- Round Robin (rr)
- Weighted Round Robin (wrr) mit folgender Gewichtung:  
*Jupiter*(3) : *Naiade*(1) : *PC-KQC*(4)
- *Least-Connection* (lc)



(a) Legende 1



(b) Legende 2

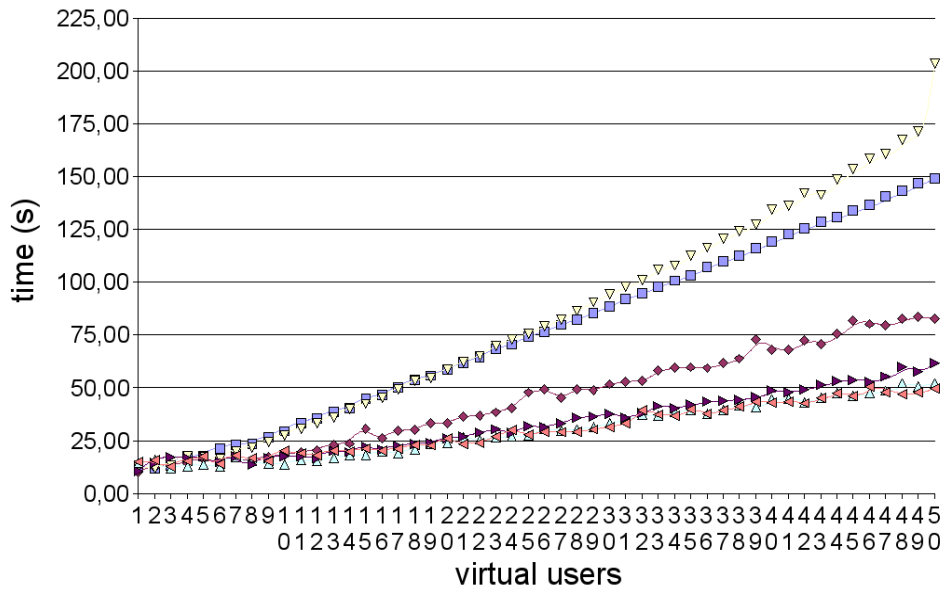
Abbildung 4.16: Legendes für die Lasttestauswertung

In einem zweiten Test wurden nur unterschiedlich konfigurierte Clusterlösungen miteinander verglichen. Auch diemal wurde die Anzahl der Benutzer von 1 bis 50 erhöht. Jeder Benutzer führte diesmal allerdings 25 Transaktionen durch. Eine Cluster-Konfiguration bestand dabei wie beim letzten Test aus *Jupiter*, *Naiade* und *PC-KQC*. Eine weitere Konfiguration setzte sich aus *Jupiter* und *Naiade* zusammen, während *PC-KQC* lediglich als Loadbalancer verwendet wurde (siehe Legende für Test 2 in Abbildung 4.16(b)). Folgende Clusterkonfigurationen wurden dabei getestet:

- Weighted Round Robin (wrr) mit folgender Gewichtung:  
*Jupiter*(3) : *Naiade*(1) : *PC-KQC*(4)
- *Least-Connection* (lc)

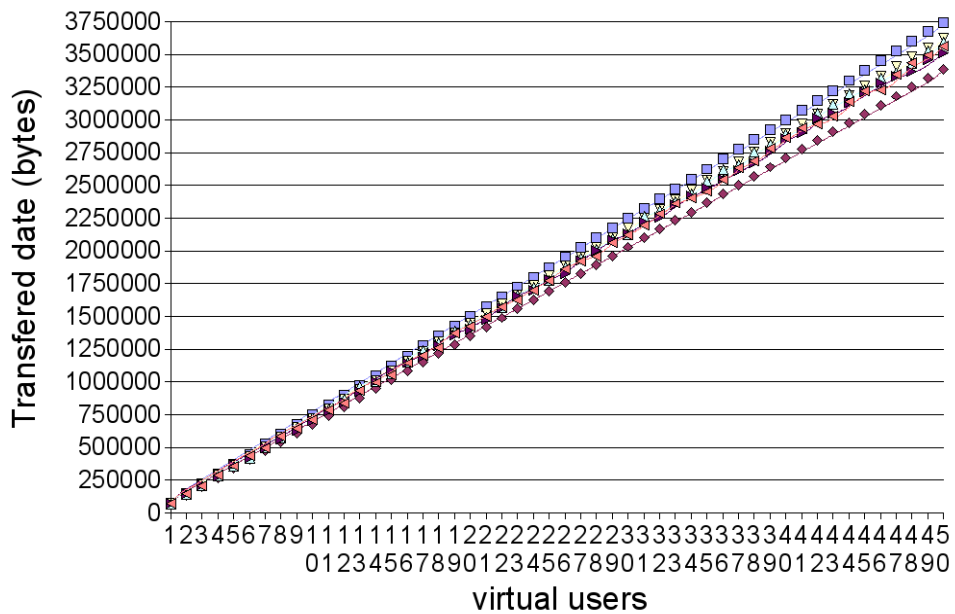
Vom zweiten Test sind hier lediglich die Diagramme für *Durchschnittliche Antwortzeit* und *Durchsatz* eingefügt, da diesen am meisten Informationen entnommen werden können.

### Elapsed time



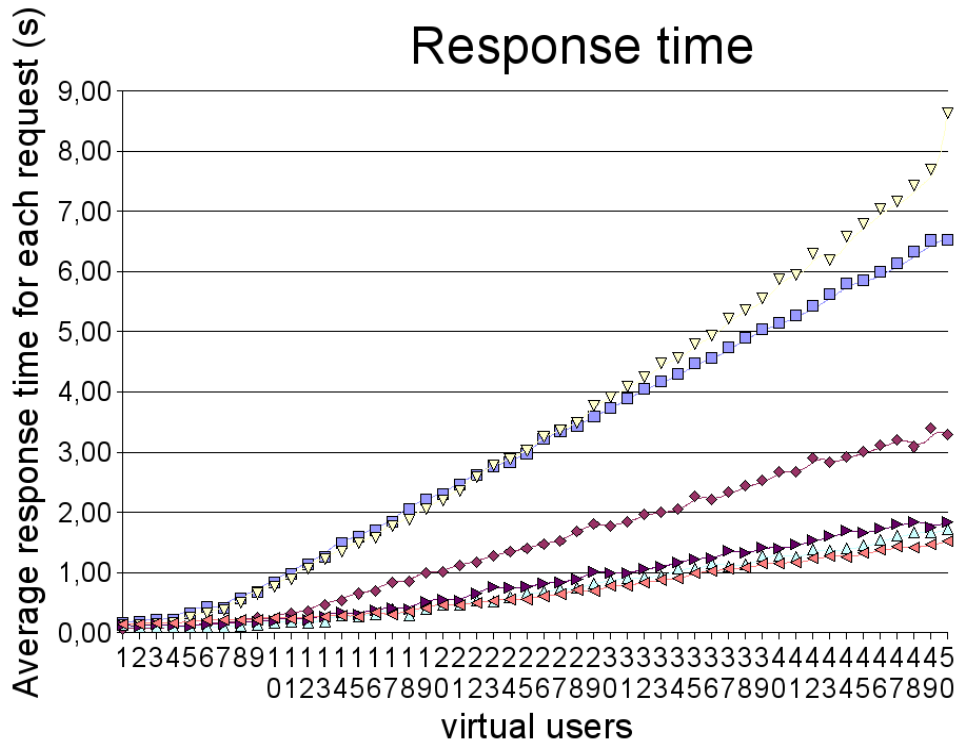
(a) Der Graph zeigt die Gesamtdauer jedes Einzeldurchlaufs in Sekunden.

### Data transfered

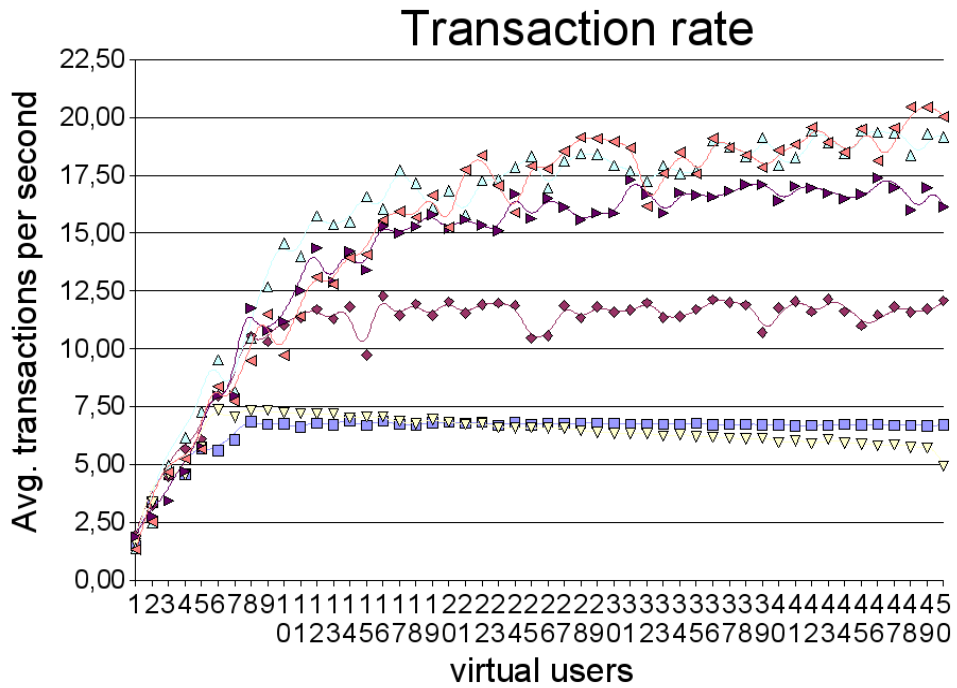


(b) Dieser Graph stellt die Anzahl der übertragenen Bytes für jeden Einzeldurchlauf dar.

Abbildung 4.17: Lasttest 1-1

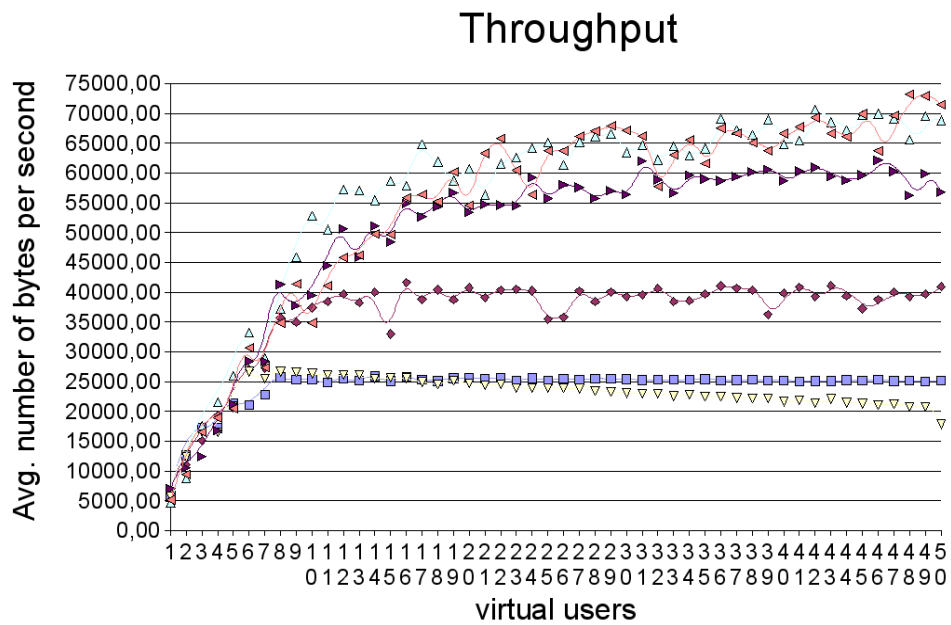


(a) Dieser Graph zeigt die durchschnittliche Antwortzeit für jede Anfrage in Sekunden.

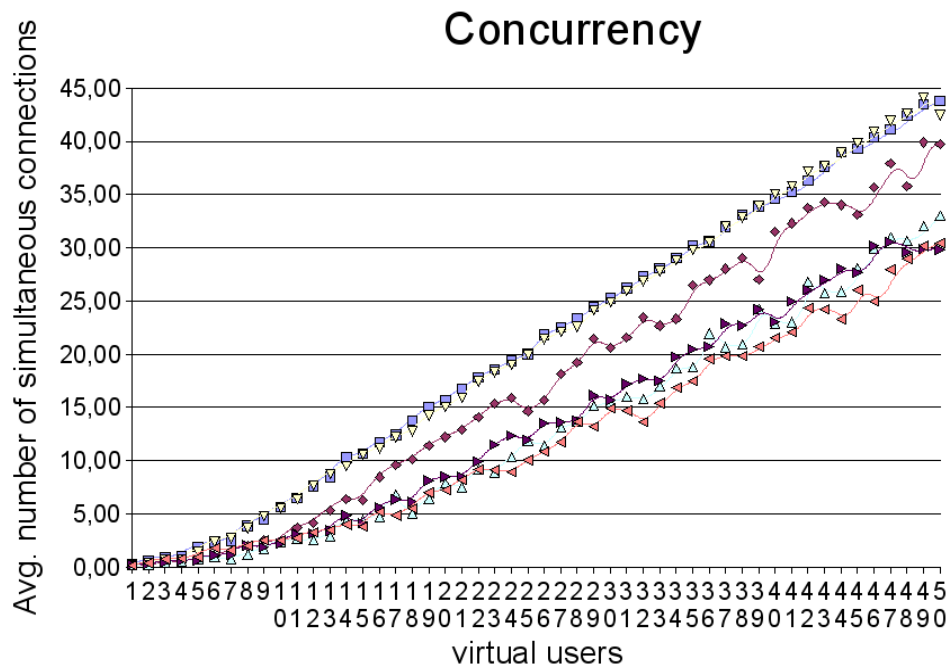


(b) Zeigt die durchschnittliche Anzahl an Transaktionen pro Sekunde.

Abbildung 4.18: Lasttest 1-2

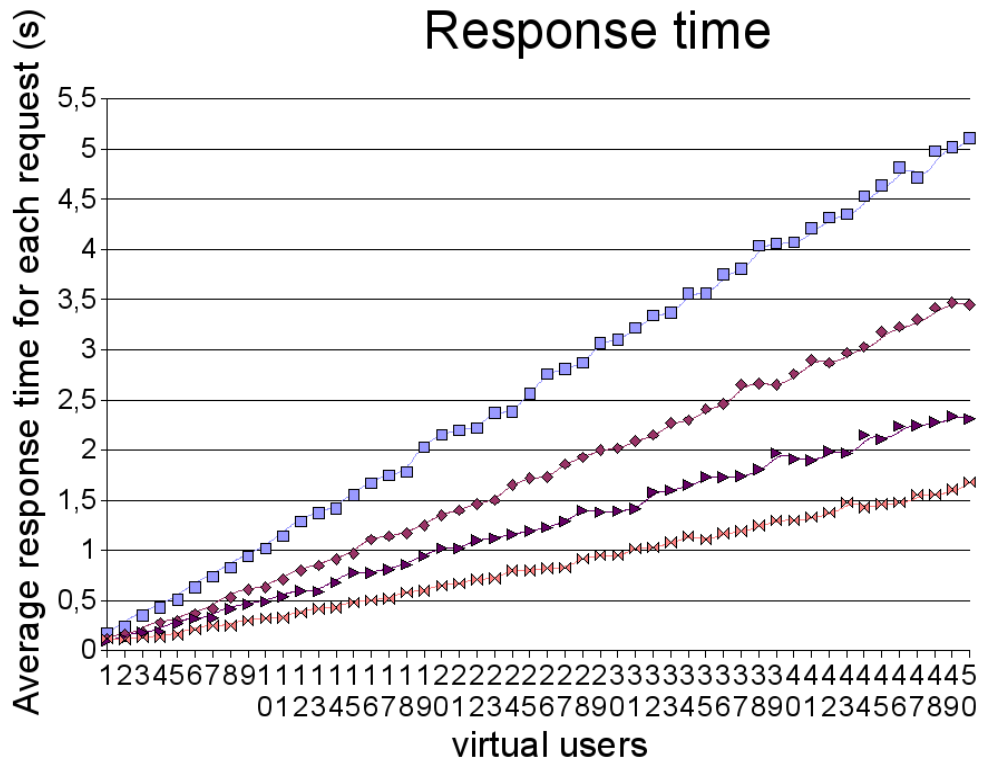


(a) Dieser Graph zeigt die durchschnittliche Anzahl von übertragenen Bytes pro Sekunde zu jedem virtuellen Benutzer an.

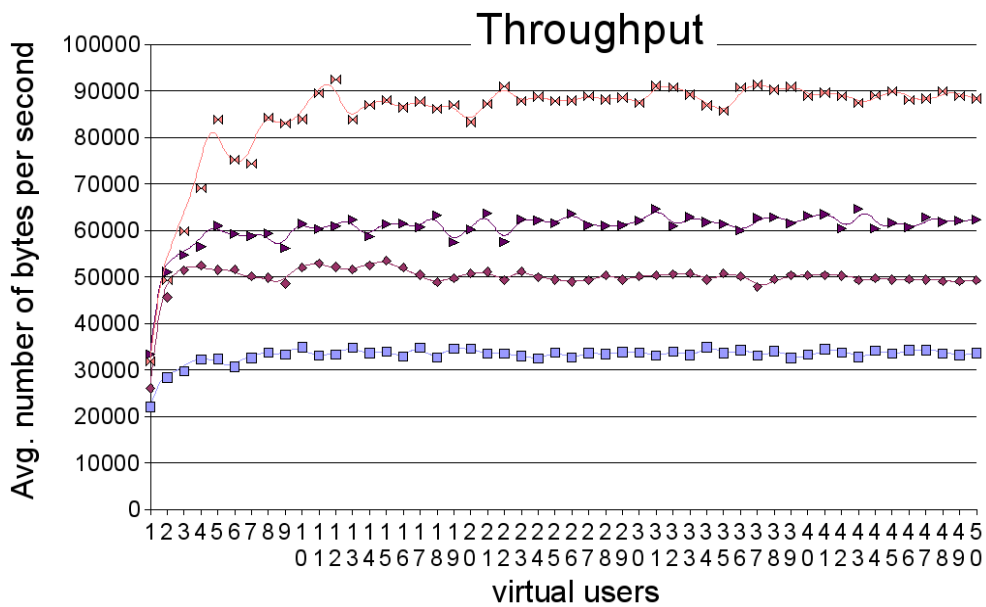


(b) Der letzte Graph von Test 1 zeigt die durchschnittliche Anzahl der gleichzeitigen offenen Verbindungen für jeden Durchgang.

Abbildung 4.19: Lasttest 1-3



(a) In diesem Graphen werden nur Antwortzeiten von Clusterlösungen verglichen.



(b) Der letzte Graph zeigt den durchschnittlichen Durchsatz pro Sekunde in Bytes bei 25 Transaktionen pro Benutzer an.

Abbildung 4.20: Lasttest 2

Das im Vergleich zu den anderen beiden Clusterlösungen schlechter abgeschnittene Weighted Round Robin-Verfahren lässt sich damit erklären, dass die Gewichtung schlecht vorgenommen worden war. Es wurde davon ausgegangen, dass *Jupiter* um einiges performanter sei, als *Naiade*. Denn es handelt sich bei *Naiade* nur um eine virtuelle Maschine. Durch die Tests stellte sich jedoch heraus, dass diese Annahme falsch war.

#### Lasttest 1-1(a)

Die Gesamtdauer der Einzeldurchläufe steigt bei den Rechnern *Jupiter* und *Naiade* sehr schnell an. Bei 50 gleichzeitigen Benutzern liegt die Dauer im Gegensatz zu den drei Clusterlösungen beim Dreifachen, während beim Rechner *PC-KQC* die Gesamtdauer gerade halb so groß ist, als bei den beiden anderen Rechnern. Die 3 getesteten Clusterlösungen liegen alle sehr dicht beisammen. Besondere Vor- oder Nachteile zwischen den Konfigurationen sind hier nicht zu erkennen.

#### Lasttest 1-1(b)

Erwartungsgemäß liegt die Anzahl der übertragenen Daten bei allen Messungen in etwa gleich. Dass *PC-KQC* ein wenig darunter liegt, lässt sich dadurch erklären, dass aufgrund der hohen CPU-Last ab 20 Benutzern einige http-Prozesse verworfen wurden.

#### Lasttest 1-2(a)

Kein System skaliert hier wirklich bis 50 Benutzer. Es muss allerdings auch klar sein, dass dies ein Stresstest ist, der darauf abzielt, die Grenzen der Hardware auszuloten. Die schnellen Anfrageintervalle der virtuellen Benutzer simulieren nicht das normale Surfverhalten mit Wartezeiten. Die Antwortzeiten der Clusterlösungen bleiben bis etwa 12 Benutzer nahezu gleich, während die Zeiten für *Jupiter* und *Naiade* mit jedem zusätzlichen Benutzer um ca. eine 1/7 Sekunde ansteigen. Die schlechteste der Clusterlösungen (*wrr*) erreicht aber auch bei 50 Benutzern nicht einmal die 2-Sekunden Marke.

#### Lasttest 1-2(b)

Bei diesem Graphen fällt der Rechner *PC-KQC* auf. Erwartungsgemäß liegt er zwar wieder über *Naiade* und *Jupiter*, springt jedoch auf und ab. Dies lässt sich auch mit der hohen CPU-Auslastung erklären, die während der Tests auftrat. Die Celeron-CPU scheint hier etwas unterdimensioniert. Dies führt auch bei den drei Clusterlösungen dazu, dass die Graphen sehr sprunghaft sind, da *PC-KQC* bei diesen auch als Realserver diente.

#### Lasttest 1-3(a)

Beim durchschnittlichen Durchsatz pro Benutzer erreichen *Despina* und *Jupiter* gerade 24 kB pro Sekunde. Während *Jupiter* diesen Wert wenigstens halten kann, geht der Durchsatz von *Naiade* mit steigender Benutzerzahl sogar zurück. Dies ist ein deutliches Zeichen, dass die Maschine völlig überlastet und nur noch damit beschäftigt ist, Systemressourcen umzuschichten. Auch hier wird das unstete Verhalten von *PC-KQC* deutlich.

**Lasttest 1-3(b)**

Da die Anfragen bei den Clusterlösungen auf drei Rechner verteilt werden, liegt natürlich die Anzahl der offenen Verbindungen durch die schnellere Abarbeitung entsprechend niedriger, als bei den Einzelrechnern.

**Lasttest 2(a)**

Wie bereits aus Abbildung 4.18 hervorging, steigen die Antwortzeiten beim 3er-Cluster mit dem Least-Connection-Verfahren am geringsten an und bleiben unter 2 Sekunden. Die 2er-Cluster mit den weniger performanten Rechnern besitzen von Beginn an durchschnittlich um 1,5 mal größere Antwortzeiten.

**Lasttest 2(b)**

Der 3er-Cluster mit dem Least-Connection-Algorithmus hebt sich beim Durchsatztest sehr stark von den drei übrigen Kandidaten ab. Erkennbar ist auch hier wieder, dass die Durchsatzgraphen der 3er-Clusterlösung wegen der hohen CPU-Auslastung von PC-KQC einen gezackteren Verlauf aufweisen, als der 2er-Cluster.

**4.7.3 Fazit**

Es ist klar ersichtlich, dass eine Loadbalancer-Lösung im Gegensatz zu einem Ein-Rechner-Webserver eine deutliche Verbesserung in puncto Antwortzeitverhalten und Durchsatz aufweist. Selbst im Gegensatz zum performantesten Rechner *PC-KQC* ist die Dauer der Antwortzeiten (4.18(a)) bei den Clusterlösungen deutlich geringer und steigt weniger steil an. Der Durchsatz liegt dabei fast doppelt so hoch (4.19(a)). Die Vergleiche der Tests zwischen den 2er- und 3er-Clustern haben gezeigt, dass der derzeit performanteste Realserver leider der *Loadbalancer* ist. Es wird daher dringend empfohlen, der Realserverfarm einen weiteren performanten Rechner hinzuzufügen, damit der Loadbalancer in Zukunft nur noch zu dem Zweck verwendet werden muss, zu dem er eingeplant ist: zur Lastverteilung.

# Kapitel 5

## Hochverfügbarkeit

„Je planmäßiger die Menschen vorgehen, desto wirksamer vermag sie der Zufall zu treffen.“

– Friedrich Dürrenmatt

Da in den letzten Jahren die Anzahl von geschäftskritischen Anwendungen im Internet sehr stark zugenommen hat, wurde es zunehmend wichtiger, Webdienste ständig verfügbar zu halten. Die Verfügbarkeits- bzw. Ausfallzeiten von Webdiensten werden heute zwischen dem Dienstanbieter und den Geschäftskunden vertraglich geregelt. Man spricht von einem Dienstgütemerkmal oder auch Service-Level-Agreement (SLA). Möchte eine Firma ihre Produkte über einen e-Shop vertreiben, so sucht sie sich einen Hostler für solche Systeme aus und vereinbart mit diesem bestimmte SLAs, die je nach Güte billiger oder teurer sind. In diesem Zusammenhang gibt es den Begriff der magischen Neunen. Diese bezeichnen die Verfügbarkeit eines Dienstes im Laufe eines Jahres. Je mehr Neunen ein Kunde in seinem Paket wählt, umso höhere Verfügbarkeit des Dienstes garantiert der Anbieter und umso höher werden die Kosten. Bei Nichteinhaltung muss er den im Vertrag vereinbarten Bedingungen Folge leisten. Tabelle 5.1 zeigt dazu eine Übersicht über Ausfallzeiten von Diensten innerhalb eines Jahres. Alan Robertson schreibt in einem Artikel vom November 2003 im Linux-Magazin [Rob04b], dass mit gewöhnlicher billiger Hardware eine Ausfallzeit von 8,8 Stunden im Jahr realistisch sei. Das entspricht einer Verfügbarkeit von 99,9%. Er merkt aber auch an, dass höhere Stufen nicht einfach durch bessere Hardware erreicht werden, sondern dass auch ganz andere Merkmale, wie ein stabiles Betriebssystem, sowie Administrator-Training beachtet werden müssen. Hochverfügbarkeit ist

1	90,00000 %	36,5 Tage
2	99,00000 %	3,7 Tage
3	99,90000 %	8,8 Stunden
4	99,99000 %	53 Minuten
5	99,99900 %	5,3 Minuten
6	99,99990 %	32 Sekunden
7	99,99999 %	3,2 Sekunden

Tabelle 5.1: Ausfallzeiten von Diensten innerhalb eines Jahres

also ein Konzept, bei dem nicht das System alleine, sondern viele andere Faktoren eine Rolle spielen.

Für die Hochverfügbarkeitslösung im Fachbereich MNI, die wie auf Seite 16 beschrieben Linux voraussetzt, eignet sich das *High-Availability Linux Project* (im Folgenden als HA-Projekt bezeichnet) hervorragend. Die folgenden Abschnitte beschreiben, die Konfiguration eines Hochverfügbarkeitsclusters.

Laut der Website des HA-Projekts [Rob04a] besteht großes Interesse an der Zusammenarbeit mit dem LVS-Projekt, welches wie in Kapitel 4 bereits ersichtlich wurde für die Performancekomponente ein existenzieller Bestandteil ist.

## 5.1 Heartbeat

Heartbeat stellt das Fundament des HA-Projekts dar. Es handelt sich dabei um eine Software, die die Verfügbarkeit von Diensten auf verteilten Systemen überwacht. Dabei gibt es mehrere Möglichkeiten der Überwachung. Es können die „Herzschläge“ verschiedener Rechner über die serielle Schnittstelle oder über Ethernet observiert werden. Meldet sich ein Rechner innerhalb einer gewissen Zeitspanne nicht mehr, so übernimmt ein anderer zuvor festgelegter Rechner dessen IP-Adresse und startet den Dienst des ausgefallenen Rechners. Der Kunde des Dienstes merkt dabei im Optimalfall nicht, dass er nun in Wahrheit den Dienst des Ersatzrechners in Anspruch nimmt. Dies setzt natürlich einen synchronen Datenbestand auf beiden Rechnern voraus, sofern der Dienst Daten anbietet.

### 5.1.1 Installation

Die Heartbeatpakete sind in einigen Linux-Distributionen, wie Suse, Mandrake oder Debian bereits enthalten. Wer eine Distribution besitzt bei der das nicht der Fall ist oder wer die aktuellsten Versionen haben möchte, der kann die folgenden Pakete aus dem Downloadbereich<sup>1</sup> des HA-Projekts beziehen:

- `heartbeat-<version>.i586.rpm`
- `heartbeat-pils-<version>.i586.rpm`
- `heartbeat-stonith-<version>.i586.rpm`
- `heartbeat-lldirectord-<version>.i586.rpm`

Für andere Rechnerarchitekturen befindet sich dort auch noch ein `heartbeat-<version>-.src.rpm` und für Linux-Distributionen ohne RPM-Paketsystem gibt es auch ein `heartbeat-<version>.tar.gz` Paket.

<sup>1</sup>Aktuelle Heartbeat-Pakete <http://linux-ha.org/download/>

Installiert werden die Pakete in folgender Reihenfolge:

```
# rpm -Uvh heartbeat-pils-<version>.i586.rpm
# rpm -Uvh heartbeat-stonith-<version>.i586.rpm
# rpm -Uvh heartbeat-<version>.i586.rpm
# rpm -Uvh heartbeat-lldirectord-<version>.i586.rpm
```

Anschließend müssen die Konfigurationsdateien angepasst werden. Bei einigen älteren Distributionen befinden sich diese direkt im Verzeichnis */etc*. Bei aktuellen Distributionen normalerweise unter */etc/ha.d*.

## 5.1.2 Konfiguration

In diesem Abschnitt geht es darum, Heartbeat so einzurichten, dass ein Loadbalancer kein *Single Point of Failure*<sup>2</sup> (SPoF) mehr darstellt. Heartbeat ist dafür bestens geeignet. Im Folgenden wird die Konfiguration am Beispiel des Loadbalancers *PC-KQC* und seines Ersatzrechners *Galatea* im Fachbereich MNI erläutert.

### 5.1.2.1 Die Datei *ha.cf*

Diese Datei enthält die Grundeinstellungen für Heartbeat. Im Beispiel 5.1.2.1 wird eine Minimalkonfiguration vorgestellt. Dabei werden serielle Verbindungen außen vorge lassen, da in der Testumgebung des Fachbereichs MNI nur Ethernet-Verbindungen eingesetzt werden. Die Einträge der Konfigurationsdatei werden durch die Kommentare erklärt. Der *Watchdog-Eintrag* bedeutet, dass Heartbeat den Rechner automatisch neu bootet, sobald der Watchdog des Kernels eine Minute lang schweigt. Soll dieses Feature verwenden, so muss das Watchdog-Modul geladen sein. Gestartet wird der Linux Watchdog mit dem Befehl:

```
# modprobe softdog
```

Zu den *node* Einträgen muss noch etwas erläutert werden. Alle dort eingetragenen Rechner bilden den Heartbeat-Cluster. Wichtig ist, dass die Namen mit der Ausgabe des Befehls „*uname -n*“ auf dem jeweiligen Rechner übereinstimmen und dass die Rechnernamen, falls es sich nicht um DNS-Namen handelt auch in die Datei */etc/hosts* auf allen beteiligten Rechnern eingetragen werden. Sonst können sich die Rechner nicht finden.

---

<sup>2</sup>Die Komponente eines Systems, welche beim Ausfall die Funktionalität des gesamten Systems beeinträchtigt, wird als *Single Point of Failure* bezeichnet. Ziel einer Hochverfügbarkeitslösung muss es sein, SPoFs durch Redundanzen möglichst zu vermeiden.

## Beispiel 4.1.2.1: Die Datei ha.cf

```
# keepalive: Anzahl der Sekunden
# zwischen Heartbeats.
keepalive 10

# deadtime: Nach dieser Zeit gilt der
# Host als Tot.
deadtime 30

# warntime: Nach dieser Zeitspanne wird
# ein verspäteter Heartbeat geloggt.
warntime 20

# initdead: Direkt nach dem Start dauert es
# länger, bis der Host für tot erklärt wird, da
# auf manchen Systemen die Initialisierung nach
# dem Booten mehr Zeit benötigt.
initdead 60

# udpport: Welchen UDP-Port verwendet Heartbeat?
udpport 694

# bcast: Über welche Schnittstelle kommuniziert
# Heartbeat?
bcast eth0

# ucast: Alternativ: Unicast Heartbeat mit
# angegebenem Ziel
ucast eth0 212.201.7.47

# Überprüft den Watchdog-Timer und bootet
# gegebenenfalls neu
/dev/watchdog

# Facility to use for syslog()/logger
logfacility local0

# Welche Rechner befinden sich im Cluster?
# Die Namen müssen "uname -n" entsprechen.
node pc-kqc
node Galatea

# IP-Fail aktivieren. Falls Pingnode nicht
# erreichbar, werden die Ressourcen abgegeben.
```

```
ping 212.201.7.39
```

### 5.1.2.2 Die Datei *haresource*

Diese Datei enthält die Regeln, nach denen Heartbeat arbeitet. Sie spezifiziert die Dienste im Cluster und legt die Rechner fest, die diese Dienste standardmäßig ausführen. Diese Datei muss auf allen Rechnern im Cluster identisch sein und enthält im einfachsten Fall nur eine einzige Zeile:

```
pc-kqc 212.201.7.39
```

Diese Zeile legt fest, dass der Rechner *pc-kqc* für sämtliche Dienste beim Start von Heartbeat der Hauptserver ist und die IP-Adresse *212.201.7.39* erhält. Heartbeat startet nun die virtuelle Netzwerkschnittstelle *eth0:0* mit der IP-Adresse *212.201.7.39*. Fällt *pc-kqc* aus, so übernimmt *Galatea* diese IP-Adresse auf dessen Interface *eth0:0*. Damit ein Dienst, der über den Namen *pc-kqc* erreichbar sein soll, weiterhin zur Verfügung steht, ist es in diesem simplen Beispiel notwendig, dass er auf dem Rechner *Galatea* bereits gestartet war. Um zu gewährleisten, dass bspw. der HTTP-Dienst und der MySQL-Dienst beim Übernehmen der IP-Adresse gestartet und bei Abgabe der IP-Adresse beendet werden, wird die Konfigurationszeile folgendermaßen modifiziert:

```
pc-kqc 212.201.7.39 httpd mysql
```

Wird ein Portal verwendet, welches andere Dienste benötigt, so können diese durch Leerzeichen getrennt hinzugefügt werden. Für die Konfiguration des MNI-Portals werden die Dienste *mon* und *ldirectord* gestartet. Wie diese arbeiten wird in den beiden Abschnitten [5.2](#) auf Seite [61](#) und [5.3](#) auf Seite [65](#) beschrieben.

Heartbeat verwendet zum Starten und Beenden der Dienste die gleichen Startskripten, wie das System, wenn es die *Runlevel*<sup>3</sup> wechselt. Dabei sucht Heartbeat im Verzeichnis */etc/rc.d/init.d*, in welchem die Startskripten normalerweise liegen. Auf dieses verweist oft der symbolische Link */etc/init.d*.

Ein übergebener Parameter *start* startet den Dienst und der Parameter *stop* beendet ihn. Mit dem Parameter *status* lässt sich feststellen, ob der Dienst gerade aktiv ist. Ein Beispiel zeigt das Starten und Beenden des HTTPD-Dienstes über die Konsole.

```
# /etc/init.d/httpd status
Apache-Webserver läuft nicht.
# /etc/init.d/httpd start
```

<sup>3</sup>Unix-Systeme verwenden unterschiedliche Runlevel, um Dienste zu gruppieren. Beim Wechsel in einen Runlevel werden zuerst Dienste beendet, die in diesem Runlevel nicht verfügbar sein sollen. Anschließend werden die Dienste gestartet, die den Benutzern zur Verfügung stehen sollen.

```
Starting httpd2: [ OK ]
# /etc/init.d/httpd status
Apache-Webserver läuft. httpd2: 29604 29603 29602
# /etc/init.d/httpd stop
```

Zwar haben die meisten Linux-Systeme diese Methode von *Redhat* übernommen, jedoch gibt es einige, die diese Skripten nicht verwenden. In diesem Fall ist selbst Hand anlegen gefragt, denn Heartbeat durchsucht auch das Verzeichnis */etc/ha.d/rc.d* nach Skripten mit dieser Struktur. Dort hinein können eigene Skripten für verschiedene Dienste abgelegt werden, für die in */etc/init.d* kein Startskript vorhanden sein soll, da sie nur von Heartbeat benötigt werden. Ein Modellskript dazu befindet sich im Anhang A.1 auf Seite 92. Dieses kann als Vorlage genutzt und leicht angepasst werden.

### 5.1.2.3 Die Datei *authkeys*

Damit nicht jeder dahergelaufene Rechner die IP-Adresse des Dienstrechners übernehmen kann, müssen sich die Maschinen im Cluster untereinander authentifizieren. Das geschieht mit Hilfe der Datei *authkeys*. Sie enthält beliebige Zeilen mit Schlüsseln. Dazu gibt es drei unterschiedliche Verfahren: *crc*<sup>4</sup>, *md5*<sup>5</sup> und *sha1*<sup>6</sup>. Ein *crc*-Hash ist am schnellsten erzeugt, hat dafür aber auch keinerlei kryptographischen Wert und sollte nur in sicheren Netzen verwendet werden. Mit einem Wert von 160 Bit Länge erzeugt der *sha1*-Algorithmus zwar am meisten Aufwand, gilt jedoch als sicherer. Dazu ist zu sagen, dass keiner der drei Algorithmen eine Verschlüsselungsfunktion ist. Sie stellen alle eine Einweg-Hashfunktion dar, welche bei beliebigem Input einen Output fester Länge liefert. Außerdem besitzt eine solche Funktionen keine Umkehrfunktion. Nur über eine *Brute Force*<sup>7</sup>-Vergleichsmethode kann deren Ausgangswert zurückerhalten werden. Über der Schlüsselzeile wird mit *auth <nr>* eine der unteren Zeilen als Schlüssel ausgewählt. Auch diese Datei muss auf allen Rechnern des Clusters identisch sein.

```
auth 3
1 crc ein_unsicherer_schluessel_fuer_sichere_netze
2 md5 ein_etwas_schwaecherer_hash_algorithmus
3 sha1 das_ist_unser_schluessel
```

Wichtig hierbei ist, dass die Rechte für die Datei *authkeys* auf *600* gesetzt sein müssen, damit kein anderer als der *root*-Benutzer sie auslesen kann. Sollte dies nicht der Fall sein, verweigert Heartbeat den Start.

<sup>4</sup>Cyclic Redundancy Check

<sup>5</sup>Message Digest Algorithm 5

<sup>6</sup>Secure Hash Algorithm 1

<sup>7</sup>Brute Force: Systematisches ausprobieren aller Permutationen von Zeichen für beliebige Schlüssel-längen



(a) Webmin nach der Anmeldung

(b) Heartbeat Startseite

Abbildung 5.1: Webmin und Heartbeat

### 5.1.3 Heartbeat und Webmin

Webmin<sup>8</sup> ist ein Konfigurationswerkzeug für Unix-Systeme, welches über einen Webbrowser abläuft. Webmin bringt seinen eigenen Webserver mit. Der Server lauscht standardmäßig an Port 10000 und sollte wegen dem vollen Rootzugriff auf das System nur über das mit *SSL* verschlüsselte Protokoll *https* verwendet werden. Webmin kann die Konfiguration des Systems vereinfachen. Allerdings muss dabei auch beachtet werden, dass es meistens mehr Optionen gibt, als Webmin anbietet. Für Heartbeat bietet es die Standard-Konfiguration an, wie sie im vorhergehenden Beispiel manuell durchgeführt wurde.

Webmin kann bei der Administrierung eines entfernten Systems sehr hilfreich sein, da es Standardaufgaben sehr leicht grafisch über den Browser erledigen lässt. Abbildung 5.1(a) zeigt die Startoberfläche direkt nach dem Anmelden. Von hier aus können die verschiedenen Untermenüs ausgewählt werden. Die Heartbeat Konfiguration befindet sich im Menü unter *Cluster*. In Abbildung 5.1(b) sind die drei Einstellungsmöglichkeiten zu sehen, die zuvor per Hand eingerichtet wurden.

Webmin bietet für viele Systemkonfigurationen weitreichende Möglichkeiten und wird hier nur einmal am Beispiel von Heartbeat vorgestellt. Abbildung 5.2 zeigt die Konfiguration der Datei `/etc/ha.d/ha.cf`. Sie bietet alle Einstellungen, die auch beim direkten Bearbeiten an der Konfigurationsdatei vorgenommen werden können.

Der Vorteil von Webmin liegt darin, dass es oft einfacher ist, als die Konfiguration über die Konsole. Außerdem ist Webmin, da es sich um eine Webanwendung handelt, für jede Linux-Distribution einsetzbar und es bedarf keiner Umgewöhnungsphase, wenn zwischen Distributionen gesprungen wird. Nachteilig ist zu nennen, dass Webmin bei dem meist sehr hohen Konfigurationsangebot der Dienste niemals alle Einzelheiten berücksichtigen kann. Ein Dienst kann so konfiguriert werden, dass er

<sup>8</sup>Webmin <http://www.webmin.com>

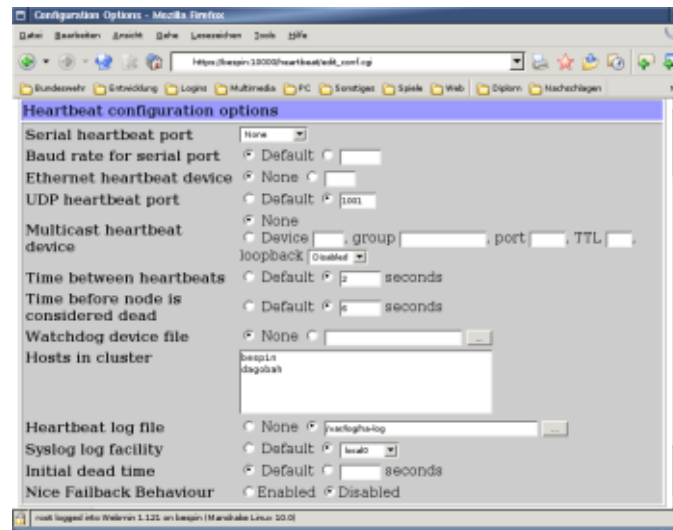


Abbildung 5.2: Heartbeat Konfiguration mit Webmin

läuft. Für Feinabstimmungen muss jedoch die Konfigurationsdatei dennoch häufig direkt bearbeitet werden.

## 5.2 Mon

Das letzte Kapitel hat gezeigt, dass Heartbeat es ermöglicht, die Verfügbarkeit von Rechnern in einem Netzwerk zu überprüfen und aufgrund des Ergebnisses zu reagieren. Verfügbarkeit des Rechners im Netz ist jedoch nicht gleichzusetzen mit der Verfügbarkeit eines Dienstes. In der Realität kommt es viel häufiger vor, dass ein Dienstprogramm sich unsachgemäß beendet oder keine Antwort mehr liefert. Nun kann Heartbeat aber nicht feststellen, ob z.B. der HTTP-Server oder der MySQL-Server noch aktiv sind. Wird der Server im Betrieb abgeschaltet, so merkt Heartbeat davon gar nichts. Alle ankommenden Anfragen laufen ins Leere.

An dieser Stelle kommt *Mon* ins Spiel. *Mon* ist ein Überwachungsmonitor für Softwaredienste, der sehr komfortabel zu konfigurieren ist und sehr individuell agieren kann, falls sich ein zu überwachender Dienst verabschiedet. Im Grunde ist *Mon* nicht ein Monitor, sondern bildet ein Grundgerüst für beliebige Monitore, die in Form von Perl-Skripten hinzugefügt werden können.

### 5.2.1 Installation von Mon

*Mon* ist Teil vieler Distributionen. Es kann natürlich auch von der *Mon*-Seite<sup>9</sup> direkt bezogen werden. Die Installation gestaltet sich einfach, da es keine Abhängigkeiten und Paketreihenfolgen zu beachten gibt. Es gibt lediglich ein aktuelles *.tgz*-Paket.

<sup>9</sup>Mon <http://www.kernel.org/software/mon/>

Damit Mon sachgerecht funktioniert, wird *Perl 5* benötigt. Mit Modulen aus älteren Versionen gab es Probleme. Folgende Perl-Module<sup>10</sup> werden vorausgesetzt:

- Time::Period
- Time::HiRes
- Convert::BER
- Mon::\*

Nachdem das `mon-<version>.tar.gz`-Archiv entpackt wurde, liegen in dem neuen Verzeichnis eine größere Anzahl Unterverzeichnisse und Dateien vor. Als *root* werden nun die Anweisungen in der README-Datei ausgeführt, um Mon zu installieren. Ein Beispielskript für die Mon-Installation befindet sich in Anhang A.2 auf Seite 94. Nach der Installation sollte *mon* mit dem Befehl

```
# /etc/init.d/mon start
```

gestartet werden können. Ist dies nicht der Fall, so fehlt mit ziemlicher Sicherheit noch mindestens eines der benötigten Perl-Module.

Der Mon-Monitor wurde so konfiguriert, dass er nicht beim Booten des Rechners startet. Das ist Absicht, denn er soll bei Bedarf von Heartbeat aus gestartet werden, damit immer nur ein Mon die Dienste überwacht. Die Voraussetzungen für einen Start von Heartbeat aus sind über das Startskript bereits erfüllt. Nun geht es an die Konfiguration. Es muss eine neue Datei `/etc/mon/mon.cf` erstellt werden. Die alte ist am besten vorher zu sichern, da sie als Beispiel dienen kann.

Beispiel 4.2.1: Die Datei *mon.cf*

```
# Festlegen von Gruppensegmenten
hostgroup web 192.168.0.120
hostgroup routers 192.168.0.1

# Gruppensegmente
watch web
service http
    interval 30s
    monitor http.monitor -p 80 -u /index.shtml
    period wd {Mon-Fr} hr {0am-23pm}
    alert bring-ha-down.alert -S \
        "web server down" webmaster@tyranus.de
    upalert mail.alert -S \
        "web server is back up" webmaster@tyranus.de
```

<sup>10</sup>CPAN: Alles über Perl <http://www.perl.com/CPAN/>

```
    alertevery 1h
    alertafter 2
    period wd {Sat-Sun}
    alert bring-ha-down.alert -S \
        "web server down" webmaster@tyranus.de
    upalert mail.alert -S \
        "web server is back up" webmaster@tyranus.de

watch routers
service ping
interval 10s
monitor ping.monitor
allow_empty_group
period wd {Mon-Sun}
    alert bring-ha-down.alert -S \
        "Router is down" admin@tyranus.de
    upalert mail.alert -S \
        "Router is back up" admin@tyranus.de
    alertevery 10m
```

Die Mon Konfigurationsdatei in Beispiel 5.2.1 dient lediglich der Demonstration, wie flexibel Mon eingesetzt werden kann. Sie hat nichts mit der Konfiguration im Fachbereich MNI zu tun. Diese wird in Abschnitt 5.2.2 behandelt.

Mit *hostgroups* werden Rechner auf denen gleiche Dienste laufen gruppiert, damit diese später gleich behandelt werden. Mit *watch <gruppenname>* wird die Überwachung für eine Gruppe gestartet. Je nach Art des Dienstes wird mit der Zeile *service <dienst>* die Überwachungsart eingestellt. Bei der *web*-Gruppe wird bspw. *http* ausgewählt. In der *monitor* Zeile wird das Perl-Skript angegeben, welches als Monitor verwendet werden soll. Die Monitorskripten für Mon liegen alle im Verzeichnis */usr/lib/mon/mon.d*. Je nach Skript können hier verschiedene Parameter übergeben werden.

Mit der *period*-Zeile kann sogar unterschieden werden, was passieren soll, wenn der angegebene Dienst zu unterschiedlichen Zeiten ausfällt. Hier kann bspw. eine email-Benachrichtigung je nach Wochentag an den zuständigen Administrator geschickt werden. Die *alert*- und *upalert*-Zeilen legen fest, was beim Ausfall des Dienstes geschehen soll. Auch hier handelt es sich wieder um Perl-Skripten. Diese werden aus dem Verzeichnis */usr/lib/mon/alert.d* gelesen.

Mit diesen Informationen sollte es möglich sein, Mon entsprechend so zu konfigurieren, dass er für die eigenen Bedürfnisse eingesetzt werden kann. Auch für Mon gibt es ein Webmin-Modul, welches sehr umfangreich ist.

## 5.2.2 Ausfallsicherung der Realserver mit Mon

Mon läuft in der MNI-Testumgebung auf dem Loadbalancer *PC-KQC* und falls dieser ausfällt mit der gleichen Konfiguration auf *Galatea*. Mon kann so konfiguriert werden, dass es, im Falle eines nicht mehr erreichbaren Realservers diesen automatisch aus der Liste des Loadbalancers entfernt und sobald der Dienst sich wieder meldet erneut hinzufügt.

Beispiel 4.2.2: Ausschnitt aus der Datei *mon.cf* für die Realserverüberwachung

```
# Hostgruppen, hier nur ein Realserver pro Gruppe
hostgroup rs1 jupiter.mni.fh-giessen.de
hostgroup rs2 naiade.mni.fh-giessen.de
# Realserver 1
watch rs1
  service http
    interval 10s
    monitor http.monitor
    period wd {Sun-Sat}
    alert mail.alert webadmin
    upalert mail.alert webadmin
    alert lvs.alert -P tcp \
      -V 212.201.7.39:10080 \
      -R 212.201.7.17 -W 5 -F dr
    upalert lvs.alert -P tcp \
      -V 212.201.7.39:10080 \
      -R 212.201.7.17 -W 5 -F dr

# Realserver 2
watch rs2
  service http
    interval 10s
    monitor http.monitor
    period wd {Sun-Sat}
    alert mail.alert webadmin
    upalert mail.alert webadmin
    alert lvs.alert -P tcp \
      -V 212.201.7.39:10080 \
      -R 212.201.7.17 -W 5 -F dr
    upalert lvs.alert -P tcp \
      -V 212.201.7.39:10080 \
      -R 212.201.7.17 -W 5 -F dr
```

Beispiel 5.2.2 zeigt die Einstellungen für eine Überwachung der Realserver im Fachbereich MNI. Das verwendete Alarmskript *lvs.alert* gehört nicht zum Standardum-

fang von Mon. Es stammt von Wensong Zhang, dem Initiator des LVS-Projektes und befindet sich im Anhang A.3 auf Seite 95 [Zha04b].

## 5.3 LDIRECTORD

Dieses Werkzeug mit dem seltsamen Namen *ldirectord* ist ein für das LVS-Projekt abgestimmter Dienst. Sein Name bedeutet *Loaddirector daemon*. Es handelt sich um einen Monitor, der anstelle von Mon für die Überwachung von Realservern eingesetzt werden kann. Direkt zu beziehen ist der Monitor nur über CVS<sup>11</sup>. Allerdings ist er in den meisten neuen Distributionen enthalten, die auch bereits einen Kernel mit LVS-Unterstützung verwenden. Er konnte auch bereits als Teil des Heartbeat-Pakets mitinstalliert werden (siehe Abschnitt auf Seite 55).

In Abschnitt 5.2 wurde gezeigt, wie mit dem *Mon* Dienstüberwachungsmonitor die Ausfallsicherung von Realservern betrieben werden kann, indem diese dynamisch aus dem System genommen und wieder eingefügt werden können. Mit *ldirectord* geht das noch ein bisschen eleganter.

Beispiel 4.3: Konfigurationsdatei für *ldirectord*

```
# /etc/ha.d/rs.cf für ldirectord
#
# Anzahl der Sekunden nach denen der Realserver als
# tot erklärt wird
checktimeout = 15

# Anzahl der Sekunden zwischen den Checks
checkinterval = 5

# Notseite, falls alle Realserver ausfallen sollten
fallback=127.0.0.1:10080

# Lädt Konfig-Datei automatisch neu, wenn diese
# sich geändert hat.
autoreload=yes

# Entfernt tote Realserver nicht von der
# Serverliste, sondern setzt ihre Gewichtung auf 0
quiescent=yes

# HTTP-Dienst
virtual=212.201.7.39:10080
```

<sup>11</sup>ldirectord Homepage <http://www.vergenet.net/linux/ldirectord/>

```
protocol=tcp
scheduler=wrr
real = 212.201.7.38:10080 gate 1
real = 212.201.7.17:10080 gate 1
real = 212.201.7.43:10080 ipip 1
service=http
request = "/.testpage"
receive = "test page"

# HTTPS-Dienst
virtual=212.201.7.39:10443
real=212.201.7.38:10443 gate 1
real=212.201.7.17:10443 gate 1
real=212.201.7.43:10443 ipip 1
fallback=127.0.0.1:10443
service=https
scheduler=wrr
request="/.testpage"
receive="test page"
protocol=tcp
```

Die Konfiguration ist etwas einfacher als für Mon. Beispiel 5.3 zeigt die Konfiguration von *ldirectord* für den Loadbalancer *PC-KQC* bzw dessen Stellvertreter *Galatea*.

Für den Fall, dass einmal sämtliche Realserver ausfallen, kann hier immer noch eine Notlösung festgelegt werden. Im Falle des Fachbereichsportals von MNI wird das Portal vom Loadbalancer selbst geladen. Eine andere interessante Option ist *quiescent*. Wird diese auf *yes* gesetzt, wird ein Realserver nicht aus der Loadbalancer-Tabelle entfernt, sondern seine Gewichtung wird auf 0 gesetzt. Das hat den Vorteil, dass die Loadbalancer-Tabelle übersichtlich bleibt und dass der Realserver mit weniger Aufwand wieder ans Netz gebracht werden kann. Verbindungsorientierte Dienste, wie *HTTPS*, die mit dem *-p* Flag gestartet wurden, werden dann allerdings so lange an den nicht mehr vorhandenen Realserver weitergeleitet, bis der Verbindungs-Timeout erreicht ist.

Die folgenden Einstellungen bedürfen keiner Erklärung mehr, da sie lediglich das Starten der Realserver aus Kapitel 4.3 abbilden. Lediglich die Zeilen *request* und *receive* müssen noch gesondert erläutert werden. Die *request*-Zeile gibt eine HTML-Datei relativ zum Wurzelverzeichnis des Webservers an, die bei den periodischen Tests abgefragt wird. Die *receive*-Zeile gibt an, was die HTML-Datei innerhalb ihres *<body>*-Tags enthalten soll, damit das Ergebnis positiv verläuft.

## 5.4 IPVSADM Client/Server

Einen Beitrag zur Hochverfügbarkeit leistet auch das Kontrollprogramm des Loadbalancers selbst. Um einen *Single Point of Failure* (SPoF) zu vermeiden, werden

zwei Lastverteiler verwendet. Damit die Loadbalancer-Tabellen immer auf beiden Rechnern aktuell gehalten und so wirklich jede Benutzersitzung bei einem Fehler des Haupt-Loadbalancers gerettet werden kann, muss jede Anfrage an den Loadbalancer auch an dessen Stellvertreter weitergeleitet werden. Dazu hat das *ipvsadm*-Kontrollprogramm eine Client/Server Applikation eingebaut, die dies gewährleisten kann. Wird auf dem Loadbalancer der Server mit folgendem Befehl gestartet:

```
# ipvsadm --start-daemon=master \  
--mcast-interface=eth0
```

und auf dem Backup-Loadbalancer den Client mit folgendem Befehl:

```
# ipvsadm --start-daemon=backup \  
--mcast-interface=eth0
```

ist es möglich, eine aktuelle Sitzung beim Wechsel der Dienste zu erhalten. Dabei muss allerdings auch bedacht werden, dass nach Angaben der Dokumentation dieses Dienstes [Zha04c] auf der LVS-Projekt Website jede Verbindung eines Clients zum Loadbalancer einen zusätzlichen Netzwerk-Overhead von 24 bytes verursacht. Es sollten also Überlegungen angestellt werden, ob der Einsatz des Dienstes bei einem Webportal notwendig ist, dessen Hauptziel nicht darin besteht, möglichst keine Kunden zu verlieren.

Gestoppt wird der Dienst mit dem Befehl:

```
# ipvsadm --stop-daemon
```

## 5.5 Datensynchronisation

Bisher wurde die Möglichkeit geschaffen, Performance durch Lastverteilung zu erzeugen. Die Hochverfügbarkeit wurde dabei ebenso durch Lastverteilung im Zusammenhang mit den zuvor vorgestellten Monitoren, sowie durch eine Backuplösung des Lastverteilers erreicht.

Dabei sind folgende Probleme aufgetreten:

- Findet ein Schreibzugriff in eine Datenbank statt, müssen diese Daten auf allen Realservern verfügbar gemacht werden.
- Finden Änderungen im Dateisystem statt, müssen auch diese Daten auf sämtliche Realserver verteilt werden.

Erste Überlegungen gingen dahin, ein verteiltes Dateisystem zu verwenden. Dieses würde sich automatisch um Änderungen der Daten kümmern. Allerdings sind nicht auf allen Realservern *root*-Rechte verfügbar und es wäre besser, wenn Realserver auch ohne *root*-Rechte realisiert werden könnten. Der zweite Grund gegen den Einsatz eines verteilten Dateisystems besteht darin zu gewährleisten, dass die Realserver möglichst einfach und schnell hergerichtet und kurzfristig ausgetauscht werden können.

Aus diesen Gründen wurde entschieden, die Datenbanksynchronisation mittels eines in MySQL eigens dafür eingebauten Mechanismus vorzunehmen. Die Synchronisation von Daten im Dateisystem soll – falls eine Änderung eintritt – per *rsync* vorgenommen werden. Die beiden nächsten Unterabschnitte beschäftigen sich näher mit diesen Themen.

### 5.5.1 Datenbanksynchronisation

Um Daten zwischen Datenbanken zu kopieren wurden in der Vergangenheit meist sehr fragwürdige Mechanismen, wie *mysqldump*<sup>12</sup> oder ähnliches verwendet. Damit werden aus der gesamten Datenbank SQL-Statements in eine Datei geschrieben, diese auf einen anderen Rechner kopiert und dort wieder eingespielt. Dieser Vorgang ist für ein als möglichst performant ausgelegtes System nicht brauchbar. MySQL bietet jedoch seit einiger Zeit das Feature *Replication* an.

Das Open Source-Datenbanksystem *MySQL* enthält seit der Version 3.23.xx einen Replikationsmechanismus für Datenbanken. Dieser hat sich ab der Version 4.0 stark verändert. Um es vorweg zu nehmen ist es also ratsam, noch vorhandene antiquierte Versionen, wie sie Debian gerne anbietet durch neue zu ersetzen.

Das Prinzip der MySQL Datenbankreplikation beruht auf der Client-Server Architektur. Es gibt dabei Master und Slaves, wobei ein Master nichts über seine Slaves weiß. Ein Master loggt jede Datenbanktransaktion in einer Binärdatei mit. Diese Binärdatei ist die Grundlage für den Replikationsmechanismus. Ein Slave kann sich zu seinem Master verbinden und wartet bis sich etwas in dieser Binärdatei verändert. Diese Änderung liest er anschließend ein und führt die gelesene Transaktion bei sich aus. Auf diese Weise enthält ein Slave immer die gleiche Datenbasis wie sein Master, so lange die Verbindung nicht unterbrochen wird.

Dabei müssen, wie Jeremy D. Zawodny in seinem Buch *High Performance SQL* [ZB04] beschreibt, folgende Regeln eingehalten werden:

- Jeder Slave muss eine eindeutige Server ID besitzen.
- Jeder Slave darf nur *einen* Master besitzen.
- Jeder Master darf beliebig viele Slaves besitzen.
- Slaves können gleichzeitig Master für andere Slaves sein.

<sup>12</sup>Konsolenwerkzeug zum Umwandeln von Datenbanken in SQL-Befehle und zum Zurückspielen der SQL-Befehle in eine Datenbank.

### 5.5.1.1 Optimale Datenbankreplikation für ein leseintensives Webportal

Für ein Hochverfügbarkeitscluster, welches ein Webportal enthält auf dem Lesezugriffe auf die Datenbank überwiegen, ergibt sich somit folgende optimale Struktur.

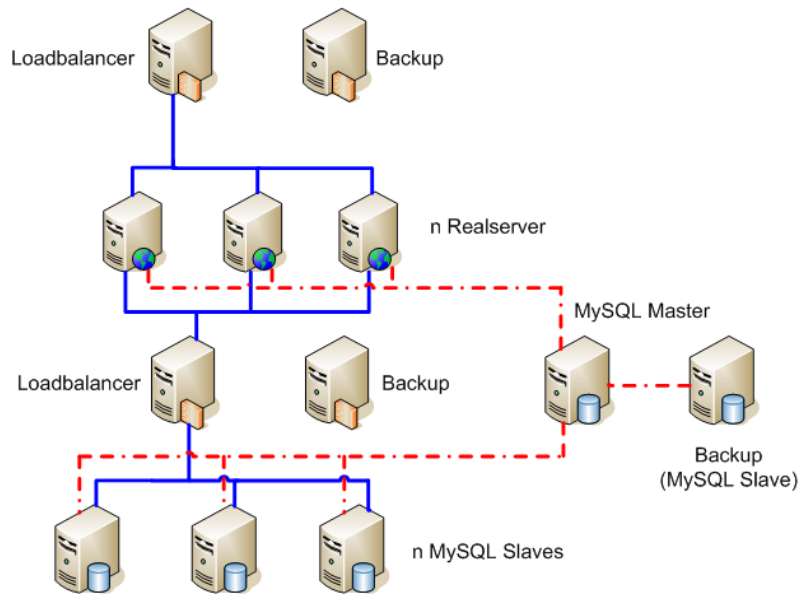


Abbildung 5.3: Replikation bei leseintensiven Anwendungen

Abbildung 5.3 zeigt den Aufbau einer Clusterlandschaft für Applikationen, bei denen Leseaktionen auf Datenbanken überwiegen. Dazu zählen Content Management Systeme. Leseaktionen werden bei diesen Webportalen praktisch bei jedem Mausklick durchgeführt, während Schreibaktionen nur bei Änderungen des Inhaltes, also z.B. bei Forumseinträgen stattfinden. Die oberen zwei Rechnerreihen zeigen das bereits bekannte Loadbalancer/Realserver Szenario. Die unteren beiden Zeilen sind Neuland und zeigen die Komponente der Datenbasis.

Sämtliche Schreibzugriffe (hier rot gestrichelt dargestellt) werden auf den MySQL-Masterrechner geleitet. Dort werden sie eingetragen. Die MySQL-Slaves inklusive des Backup-Masters, der so lange der MySQL-Master aktiv ist auch einen normalen Slave darstellt, gleichen ihre Tabellen mit denen des MySQL Masters ab. Sollte der MySQL-Master ausfallen übernimmt hier, genau wie bei den Loadbalancern der Backuprechner die Schreibzugriffe der Realserver.

Lesezugriffe von den Realservern auf die Datenbank werden über einen weiteren Loadbalancer auf die MySQL Slaves verteilt. Das Verfahren funktioniert dabei genau so, wie in Kapitel 4 beschrieben. Um keinen *Single Point of Failure* zu bilden, benötigt der MySQL Loadbalancer natürlich auch ein Backupsystem, dass die gleiche Konfiguration enthält. Da Lese- und Schreibzugriffe bei diesem Aufbau strikt getrennt sind, muss das verwendete Content Management-System dies natürlich unterstützen. Ist dies nicht der Fall, wie etwa bei dem QA-Portal, müssen Wrapper-

Funktionen geschrieben werden, die bei Lese- und Schreibzugriffen auf unterschiedlichen Verbindungen zugreifen.

Selbst wenn sowohl für Realserver, als auch MySQL-Slaves nur je 2 Rechner eingeplant werden, ist für diese Lösung – da SPoF's vermieden werden müssen – ein Minimum von 10 Rechnern notwendig. Dieser Lösungsansatz musste leider aufgrund nicht ausreichend vorhandener Hardware verworfen werden.

### 5.5.1.2 Multi-Master-Ring

Die Regeln bei der MySQL-Replikation erlauben es, dass jeder Slave eines Masters gleichzeitig ein Master eines Slaves sein kann. Im einfachsten Falle kann somit eine *Dual Master*-Konfiguration eingerichtet werden, bei der es zwei MySQL-Server gibt, von denen jeder sowohl Master als auch Slave des anderen ist.

Diese Konfiguration ist jedoch nur ein Spezialfall einer weiteren Konfiguration. Kommt mindestens ein dritter MySQL-Server hinzu, existiert bereits ein Multi-Master-Ring Konfiguration.

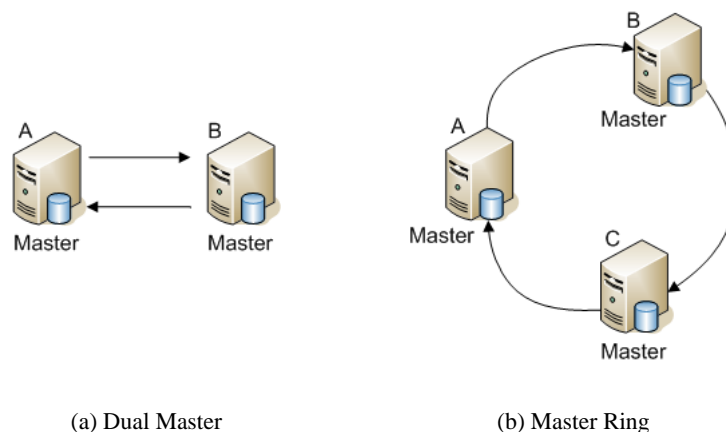


Abbildung 5.4: Multi Master-Konfigurationen

Folgende Vorteile besitzt ein Multi-Master-Ring:

- Es wird keine zusätzliche Hardware benötigt, da jeder Realserver sein eigener MySQL-Master sein kann.
- Ein Content Management System muss nicht unterscheiden zwischen Lese- und Schreibverbindungen, da beide auf dem eigenen Rechner stattfinden.
- Es ist die wohl performanteste Methode, da alle Zugriffe lokal erfolgen und durch die Aufspaltung der Realserver trotzdem unterschiedliche Benutzer auf unterschiedliche Rechner zugreifen.

```

[tim@Geonosis bin]$ ./replication.monitor --host=pc-kqc-1
-----= Realserver =-----
jupiter.mni.fh-giessen.de:      Active  Slave_IO_Running  Slave_SQL_Running
naiade.mni.fh-giessen.de:      Yes     Yes                Yes
pc-kqc-1.mni.fh-giessen.de:    Yes     Yes                Yes

-----= Realserver =-----
                                qa      perftest
jupiter.mni.fh-giessen.de      13422  6
naiade.mni.fh-giessen.de       13422  6
pc-kqc-1.mni.fh-giessen.de     13422  6

Clean disconnect!
[tim@Geonosis bin]$

```

Abbildung 5.5: Replikationsmonitor - Alles in Ordnung

Diese Konfiguration wurde mit den drei Rechnern *PC-KQC*, *Naiade* und *Jupiter* aufgebaut und eingehend getestet. Ein Master-Ring ist im Grunde aus Sicht der Verfügbarkeit eine heikle Sache. Da die Daten ringförmig von einem zum nächsten Slave verteilt werden müssen, ist sie zwar sehr performant, da sie am wenigsten Netzlast verursacht, dafür aber äußerst problematisch, sobald einer der Rechner im Ring ausfällt. Aus diesem Grund wurde ein Perl Monitorskript für Mon geschrieben, welches alle aktiven Realserver überprüft und testet, ob deren Replikations-Threads (Slave\_SQL) laufen und deren Zugriff auf den Master gegeben ist (Slave\_IO). Außerdem prüft dieser Monitor als Integritätstest, ob die Summe aller Tabellenzeilen der zu replizierenden Datenbanken übereinstimmt.

Zum Test wurde dafür die Datenbank des *Quality Assurance Portal* (QA-Portal) verwendet. Zusätzlich wurde noch eine kleine Testdatenbank (perftest) verwendet. So lange alle drei MySQL-Server liefen, gab es keinerlei Probleme bei der Datenreplikation in dem Ring. Abbildung 5.5 zeigt, dass auf allen drei Rechnern die Replikation ordnungsgemäß abläuft. Nachdem umfangreiche *INSERT* und *DELETE* Abfragen durchgeführt worden waren, stimmten die Anzahl der Zeilen in den beiden zum Test verwendeten Datenbanken noch immer überein.

Daraufhin wurden einige Überlegungen angestellt, was zu tun ist, wenn ein Rechner aus dem Ring ausfällt. Daraus konnte der Schluss gezogen werden, dass ein Mon Alarmskript den Ring schließen kann, bis der ausgefallene Rechner wieder online geht. Voraussetzung dafür ist, dass der Loadbalancer auf dem Mon läuft, eine Liste von aktiven Realservern und deren Master-Slave-Beziehungen besitzt. Eine solche Liste wurde als Tabelle in einer Datenbank auf dem Loadbalancer abgebildet. Abbildung 5.6 zeigt einen vorerst noch intakten Ring. Ein Master befindet sich darin immer am Anfang eines Pfeils und ein Slave am Pfeilende. Fällt Rechner A aus, bemerkt Mon dies ohnehin und kann in seiner Realservertabelle A auf *nicht verfügbar* setzen. Der Mon-Replikationsmonitor aus Abbildung 5.5 erkennt daraufhin, dass ein Rechner nicht mehr aktiv ist. Es läuft folgender Algorithmus ab:

- Mon sucht über eine Tabelle den Slave (=B) des ausgefallenen Rechners (A)
- Mon sucht den alten Master von A (=C) und weißt B diesen als neuen Master

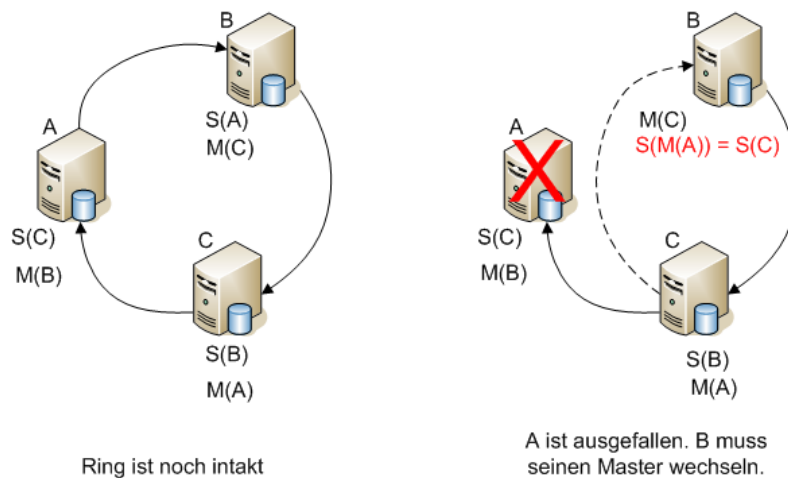


Abbildung 5.6: Failover beim Ausfall eines Rechners im Ring

zu.

Damit ist der Ring geschlossen. Wird Rechner A wieder aktiv, besitzt Rechner B zwei Slaves. Das ist kein Problem, denn eine Replikationsregel besagt, dass jeder Master beliebig viele Slaves haben darf. Nachdem A nun wieder einsatzbereit ist und alle Daten von C übernommen hat, verwendet Mon folgenden Algorithmus:

- Mon sucht über die Realservertabelle den ursprünglichen Slave (=B) des nun wieder intakten Masters (A).
- Mon weist dem gefundenen Slave (B) seinen alten Master (A) wieder zu.

Somit ist der Ring wieder geschlossen. Theoretisch ist so ein *failover*<sup>13</sup> auf diese Weise mittels Multi-Master-Ring möglich. Praktisch allerdings treten dabei immer wieder Probleme auf.

Das grundlegende Problem dabei ist, dass die Position des binären Logfiles, an der sich der Master beim Absturz befand nicht bekannt ist. Zwar lässt sich diese über einen SQL Befehl auslesen, jedoch ist der Master nicht mehr verfügbar. Es muss sich daher damit abgefunden werden, dass beim Wechseln der Master automatisch die letzte Logposition gewählt und evtl. auftretende Folgefehler bei der Replikation übersprungen werden. Das Ergebnis ist in Abbildung 5.7 zu sehen. 3 MySQL-Server, 3 unterschiedliche Datenbasen. Wahrlich keine guten Voraussetzungen für ein Hochverfügbarkeitssystem. Vor allem da sämtliche Rechner in einem Ring gleichberechtigt sind und es sich nicht feststellen lässt, welche Datenbasis die korrekte ist. Zudem kommt bei einem ringförmigen Aufbau ein weiteres Problem hinzu. Sobald gleichzeitig auf zwei Rechnern in der gleichen Tabelle eine Zeile angelegt wird, kommt

<sup>13</sup>Übernahme von Diensten eines anderen Rechners innerhalb eines Clusters im Fehlerfall.

```

[tim@Geonosys bin]$ ./replication.monitor --host=pc-kqc-1
-----= Realserver =-----
jupiter.mni.fh-giessen.de:   Active  Slave_IO_Running  Slave_SQL_Running
                             Yes      Yes                Yes
naiade.mni.fh-giessen.de:   Yes      Yes                Yes
pc-kqc-1.mni.fh-giessen.de: Yes      Yes                Yes

-----= Realserver =-----
                             qa    perftest
jupiter.mni.fh-giessen.de   13500 6
naiade.mni.fh-giessen.de   13374 6
pc-kqc-1.mni.fh-giessen.de  13375 6

Clean disconnect!
[tim@Geonosys bin]$

```

Abbildung 5.7: Replikationsmonitor - Inkonsistenzen

es zum Problem bei *AUTO\_INCREMENT*<sup>14</sup>-Feldern. Primärschlüssel werden häufig durch einen solchen Zähler automatisch erzeugt. Werden nun auf unterschiedlichen Rechnern gleichzeitig Einträge in eine gleiche Tabelle getätigt und diese Tabelle enthält *UNIQUE KEYS*, die mit der *AUTO\_INCREMENT*-Funktion hochgezählt werden, so kann dieser Eintrag nicht mehr auf den anderen Rechner repliziert werden. Zwei Rechner enthalten also unterschiedliche Daten. Je nach Position der Rechner im Ring enthält im schlechtesten Fall die Hälfte aller MySQL-Slaves nach der Replikation die Zeile des einen Rechners und die andere Hälfte der Slaves die Zeile des anderen. Der Vorteil, dass die Content Management-Systeme nicht an die Datenbankstruktur angepasst werden müssen ist somit hinfällig. *AUTO\_INCREMENT*-Felder dürfen nicht verwendet werden. Zusammengefasst ergeben sich folgende Nachteile:

- Es dürfen keine *AUTO\_INCREMENT* Felder verwendet werden.
- Nach dem Ausfall eines Rechners können über das gesamte System hinweg Dateninkonsistenzen auftreten.
- Der Aufbau widerspricht dem Hochverfügbarkeitsprinzip *Keep it simple*.

Ein Multi-Master-Ring sollte daher mit Vorsicht verwendet werden. Zukünftige MySQL Versionen werden die aufgetretenen Probleme vielleicht lösen können, aber im Moment ist gerade der Umstand, dass bei einer Dateninkonsistenz kein Rechner existiert, der mit Sicherheit die korrekten Daten besitzt der Hauptgrund, dieses Verfahren abzulehnen.

### 5.5.1.3 Datenreplikation für wenige Rechner

Da die ersten beiden Replikationsvorschläge im Fachbereich MNI nicht umsetzbar sind, musste ein Kompromiss gefunden werden, um gleiche Datenbasen nach Schreibzugriffen zu gewährleisten.

<sup>14</sup>Interne MySQL-Funktion, die auf numerische Felder angewendet werden kann. Sie fügt beim Anlegen einer neuen Tabellenzeile eine Zahl in das Feld und zählt ihren Zähler um eins hoch.

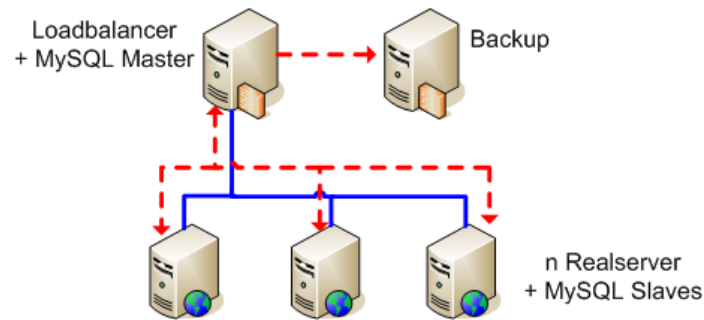


Abbildung 5.8: Lowcost-Replikation

Abbildung 5.8 zeigt eine *Lowcost*-Variante des Modells aus Verfahren 5.5.1.1. Dabei fallen der extra Loadbalancer, dessen Backuplösung, der extra MySQL-Master-Server und dessen Backuplösung weg. Schreibzugriffe werden an den normalen Loadbalancer oder im Fehlerfall dessen Stellvertreter geleitet. Die Realserver replizieren dessen Datenbasis. Lesezugriffe sind schneller, als in der Variante mit einem extra Loadbalancer, da sie direkt auf dem Realserver stattfinden, der die Daten benötigt. Auch das Netz wird im Falle des Fachbereichs MNI nicht *mehr* belastet als bei Verfahren 5.5.1.1. Lediglich der Loadbalancer muss mehr arbeiten als sonst. Da nun mehr Netzwerkverkehr über ihn läuft, ist es empfehlenswert, ihn mit einer weiteren Netzchnittstelle auszustatten.

Hier noch einmal die Vor- und Nachteile dieser Methode. Zuerst die Vorteile:

- Es wird nicht mehr Hardware benötigt als ohne Datenbanken.
- Lesezugriffe sind genauso performant, wie in der Master-Ring Lösung, da sie direkt vom Realserver ausgeführt werden.
- Im Fehlerfall der Realserver selbst besitzt der Loadbalancer als MySQL Master immer die korrekte Datenbasis.
- Im Fehlerfall des Loadbalancers besitzt dessen Backuprechner die korrekte Datenbasis.
- Es entstehen keine zusätzlichen Kosten.
- Eine spätere Umrüstung auf die zuerst vorgestellte Methode ist einfach möglich.

Nachteile:

- Der Loadbalancer wird durch Schreibzugriffe auf die Datenbank zusätzlich belastet.

- Lese- und Schreibzugriffe müssen in der Portal-Software getrennt behandelt werden. Entweder ein Content Management-System unterstützt die Unterteilung von Lese- und Schreibzugriffen auf unterschiedliche Datenbankobjekte, oder der Code muss entsprechend angepasst werden.

Diese Version eignet sich von den vorgestellten am besten für den Fachbereich MNI und wird dort eingesetzt.

## 5.5.2 Dateisynchronisation

Für die Datensynchronisation wird das Unix-Tool *rsync* eingesetzt. Da die meisten Daten bei einem verwendeten Content Management-System ohnehin in einer Datenbank abgespeichert werden, müssen lediglich bestimmte Verzeichnisse des Portals synchronisiert werden. Das sind Upload-Verzeichnisse, welche Dateien enthalten, die ein Benutzer in das Portal einstellen kann. Dabei kann ein *cron*<sup>15</sup>-Job in regelmäßigen Abständen überprüfen, ob Änderungen in dem Upload-Verzeichnis erfolgt sind.

```
# Prüfe, ob sich seit dem letzten mal etwas
# geändert hat
ls -laR $UPLOAD_ROOT | md5sum >$TMPFILE
cmp $CMPFILE $TMPFILE >/dev/null 2>&1
RVAL=$?
if [ $RVAL -eq 0 ]; then
    exit 0;
fi

# Es gab Änderungen. Aktualisiere das $CMPFILE
cp -f $TMPFILE $CMPFILE
```

Auf diese Weise wird ein MD5-Hashwert aus einer detaillierten *ls* Ausgabe des Upload-Verzeichnisses erzeugt. Dieser wird mit dem Hashwert verglichen, der beim letzten Aufrufen des Skripts erzeugt wurde. Sind die Hashwerte gleich, so hat sich nichts geändert und *rsync* muss gar nicht erst aufgerufen werden. Das spart Netzwerkreressourcen.

Fand eine Änderung statt, so wird *rsync* gestartet. Das Tool erzeugt beim ersten Start eine Datei mit Informationen der zu synchronisierenden Verzeichnisse des entfernten Rechners. In den folgenden Durchläufen werden ausgehend von dieser Datei, nur noch Änderungen übertragen. Ändert sich bspw. eine Textdatei minimal wird auch nur diese Änderung übertragen. Allerdings muss die Synchronisation mit jedem Realserver durchgeführt werden. Besser ist es auch hier, wenn ein Realserver sich immer nur mit dem Loadbalancer synchronisiert und dieser in einem eigenen cron-Job

<sup>15</sup>Unix-Dienst zum zeitgesteuerten Ausführen von Programmen.

dafür sorgt, dass alle Realserver synchronisiert werden. Der Netzverkehr ist dadurch zwar etwas höher, jedoch existiert ein Master, der im Fehlerfall eine neue konsistente Datenbasis auf den Realservern erstellen kann. Ein vollständiges Synchronisationskript, wie es ein Loadbalancer einsetzen kann, befindet sich im Anhang [A.4](#) auf Seite [95](#).

# Kapitel 6

## Sicherheit

„Wer die Freiheit aufgibt, um Sicherheit zu gewinnen, wird am Ende beides verlieren.“

– Benjamin Franklin

Sicherheit im Internet ist spätestens seit *dem* Tag ein Thema, seit dem es um bares Geld geht. Im Zeitalter von *e-banking*, *e-commerce* und *e-government* bieten die Internetdienste, die eigentlich nie für Sicherheit ausgelegt wurden und heute zum Teil für andere Anwendungsformen genutzt werden, als ursprünglich konzipiert, standardmäßig nur wenig Sicherheit. Wie bereits in der Kapitelübersicht erwähnt, ist dieses Thema eigentlich kein Kapitel für sich, sondern vielmehr ein Überbegriff, der vor allem bei Performance-Themen berücksichtigt werden muss. Exzessiv betriebene Sicherheit und die bestmögliche Performance schließen sich gegenseitig aus. Bessere Performance geht oft zu Lasten der Sicherheit, aber häufig kann Performance auch dadurch verbessert werden, dass unnötige Sicherheitsaspekte deaktiviert werden. Das klassische Beispiel hierfür ist eine SSL<sup>1</sup>-Sitzung. Sicherlich könnte ein gesamtes Webportal nur über HTTPS mit SSL-Verschlüsselung angeboten werden. Da eine SSL-Sitzung jedoch auf dem Webserver im Vergleich zu normalem HTTP unverhältnismäßig viele Systemressourcen verbraucht, sollte sich darauf beschränkt werden, wirklich nur für Seiten eine SSL-Verbindung aufzubauen, bei denen sensible Daten übermittelt werden.

In den beiden folgenden Abschnitten wird beim Thema Sicherheit kurz auf die unterschiedlichen Bereiche *Datenschutz* und *Datensicherheit* eingegangen. Es werden kurze Beispiele vorgestellt, wie beides erreicht werden kann und somit eine kurze Übersicht über das Themengebiet gegeben. Die *Sicherheits-Checkliste*, die im Laufe dieser Arbeit erstellt wurde, enthält Tipps und Vorgehensweisen, um die Plattform eines Webportals abzu härten und dieses Kapitel zu ergänzen. Sowohl was Datenschutz, als auch Datensicherheit angeht. Außerdem prüft sie eine existierende Plattform darauf, inwiefern diese den Kriterien entspricht.

---

<sup>1</sup>Secure Socket Layer

## 6.1 Datenschutz

Das größte Sicherheitsproblem beim Datenschutz ist häufig, dass sich aufgrund bereits getroffener Sicherheitsvorkehrungen in Sicherheit gewogen wird. Viele Nutzer setzen eine Desktop-Firewall gleich mit dem Begriff *Firewallsystem* und sind nach der Installation einer solchen der Meinung, ihr System sei nun gegen sämtliche Angriffe geschützt. Solch eine trügerische Sicherheit wird auch als *snake oil* oder *Schlangenöl* bezeichnet. Eine wirkungsvolle Firewall jedoch setzt ein Sicherheitskonzept voraus. Norbert Pohlmann schreibt in seinem Buch *Firewall Systeme* auf Seite 41: „Nicht das Firewall-System macht sicher, sondern mit einem Firewall-System kann man etwas sicher machen, wenn es richtig betrieben wird.“ [Poh00] Dabei müssen vorher Tabellen mit Diensten der verschiedenen Rechner aufgestellt werden und es muss klar sein, von wo aus diese erreichbar sein sollen. Vor dem Einsatz eines Firewall-Systems müssen die grundlegenden Anforderungen an dieses feststehen.

Eine Firewall ist nicht alles. Zum Datenschutz gehören auch *administrative* Sicherheitsaspekte, denen in der Checkliste eine eigene Unterkategorie gewidmet ist. Dabei müssen Sicherheitsaspekte bei der Fernadministration berücksichtigt werden. Die Passwörter der Administratoren müssen bestimmte Sicherheitsvoraussetzungen erfüllen und dürfen nur über eine sichere Verbindung übertragen werden. Neben Technischen Aspekten werden in der Checkliste auch einige soziale Probleme z.B. bei der Verwendung von Passwörtern angesprochen.

Als letzten Aspekt behandelt die Checkliste schließlich auch die *Angriffsprävention*. Dabei geht es vor allem darum, mögliche Angriffe rechtzeitig aufzudecken oder Angreifer bewusst in Fallen zu locken, die nur dem Ziel dienen, diese zu registrieren. Solche Fallen werden als *honey pods* oder *booby traps* bezeichnet.

### 6.1.1 Risikoanalyse

Im Buch *SICHERE SERVER mit LINUX* beschreibt der Autor Michael D. Bauer, wie die jährliche Verlusterwartung, für durch Sicherheitslücken in der Infrastruktur entstandene Schäden, berechnet werden kann [Bau03]. Dabei gilt für die Kosten jeder Sicherheitslücke die einfache Formel:

$$K_g = j \cdot \sum k$$

$K_g$  := Jährliche Gesamtkosten für eine Sicherheitslücke

$j$  := Voraussichtliche Ausnutzung dieser Sicherheitslücke pro Jahr

$k$  := Einzelkostenpunkt

Um die Berechnung zu verdeutlichen hier ein Beispiel, welches auch zeigt, dass Sicherheitslücken nicht nur in Software auftreten können:

Sicherheitslücke: Verlust des Schlüssels für einen Serverraum in einem Kleinunternehmen:

Tabelle 6.1: Kostenbeispiel einer Sicherheitslücke

Arbeitskosten des Schlüsseldienstes	100 Euro
Neues Sicherheitsschloss	100 Euro
Nachmachen der Schlüssel für alle zugangsberechtigten Personen	75 Euro
Gesamtkosten	375 Euro

Angenommen ein solcher Verlust tritt durchschnittlich alle 3 Jahre auf, ergeben sich für diese Sicherheitslücke jährliche Gesamtkosten von  $K_g = \frac{1}{3} \cdot 375 \text{ Euro} = 125 \text{ Euro}$ .

Eine solche Aufstellung kann für sämtliche Sicherheitslücken, sei es ein Sendmail-Bug oder ein wütender Mitarbeiter aufgestellt werden. Die Zusammenfassung ergibt bereits eine sehr gute Sicherheitsanalyse, da sie direkt eine Übersicht über Kosten liefert. Informationen über das vorraussichtliche Auftreten einer Sicherheitslücke pro Jahr kann dabei durch Analysen über bisherige Ausnutzung von Sicherheitslücken erhalten werden. Außerdem können Überlegungen angestellt werden, ob es sich lohnt, die Sicherheitslücke ganz zu schließen, falls dies überhaupt möglich ist. In diesem Beispiel kann sie durch eine andere Zugangskontrolle, wie ein Nummernschloss geschlossen werden. Dabei ergibt sich allerdings bereits eine neue Sicherheitslücke, denn ein vergesslicher Zugangsberechtigter könnte sich die Nummer irgendwo aufgeschrieben haben...

Datenschutz muss also in allen Punkten gewährleistet sein in denen mit sensiblen Daten umgegangen wird und jedes gute Datenschutzkonzept setzt eine Risikoanalyse voraus. Im Bereich eines Open Source-Portals sind vor allem Zugangsdaten der Benutzer zu schützen. Gleichzeitig ist die Plattform, auf der das Portal läuft gegen Angriffe abzusichern und diesen vorzubeugen. Hilfen und Konfigurationstipps, wie dies geschehen kann finden sich in der Sicherheitscheckliste.

### 6.1.2 Firewalls

Um dem bereits performanten und für Hochverfügbarkeit ausgelegtem System die nötige Sicherheit zu geben, werden bestimmte Zugriffskontrollen installiert.

In der Abbildung 6.1 ist der typische Aufbau eines Firewallsystems gezeigt. Die Firewall-Komponente bildet dabei ein Router mit zwei Netzwerkschnittstellen. Eine der Schnittstellen ist mit dem unsicheren Netz verbunden, die andere Schnittstelle gehört zu einem lokalen, sicheren Netz oder zumindest zu einem Netz dessen Betreiber ein Interesse daran hat, den ein- und ausgehenden Datenverkehr zu kontrollieren. Der Firewallrechner nimmt dabei die Funktion eines Pförtners wahr. Er nimmt Anfragen entgegen, die aus dem einen in das andere Netz gerichtet sind, prüft sie und entscheidet anhand eines Regelwerkes, ob das Paket passieren darf oder nicht. Diese Regeln sind häufig sehr umfangreich, was besonders bei schlecht gewählten Durchlauf-Reihenfolgen ein Performanceverlust darstellen kann. Die Firewall, die das MNI-Netz 212.201.7.0/24 vom Internet trennt war für das Projekt nicht zugänglich. Außerdem müssen innerhalb des Testsystems viele Dienste, wie z.B. *Heartbeat*,

*ldirectord* und *Mon* verwendet werden, die auch für bestimmte Rechner *innerhalb* dieses Netzes nicht zugänglich sein sollten. Aus diesem Grund wurde beschlossen, jeden in der Abbildung angezeigten Rechner selbst mit Maßnahmen auszustatten, die eine ausreichende Zugriffskontrolle für die Verbindung mit den darauf laufenden Dienste bieten.

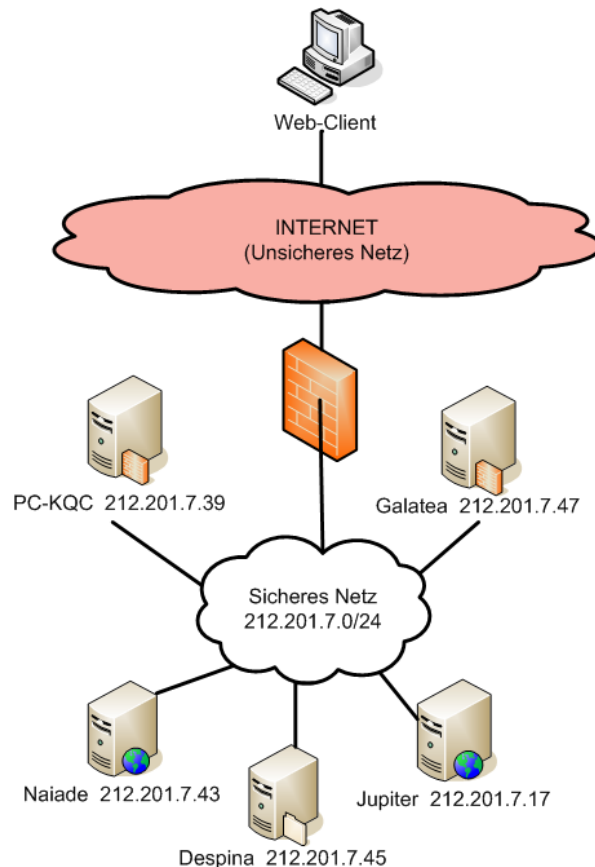


Abbildung 6.1: Trennung von Netzen

Dem Einrichten von Sicherheitsmaßnahmen geht – wie bereits erwähnt – eine Ist-Analyse des zu schützenden Systems voraus. Dazu bieten sich Portscans an. Ein Portscan kann alle oder einen beliebigen Bereich an Ports durchlaufen und testen, ob an diesem ein Dienst lauscht. Das Unix-Werkzeug *nmap* oder in seiner Frontend-Version *xnmap* ist ein Portscanner und für diesen Zweck bestens geeignet. Je nach Konfiguration des Portscanners werden *root*-Rechte benötigt. Folgender Aufruf von *nmap* startet einen ausführlichen TCP-Portscan:

```
# nmap -A -T5 host
```

Natürlich bietet *nmap* sehr viel mehr Einstellungsmöglichkeiten. Für eine komplette Auflistung der Möglichkeiten sei auf die Manpage verwiesen<sup>2</sup>. Beispielhaft für al-

<sup>2</sup>nmap Manpage <http://athena.fit.qut.edu.au/cgi-bin/man/man2html?nmap>

le Rechner des Systems zeigt Tabelle 6.2 auf der nächsten Seite das Ergebnis eines Portscans von *PC-KQC* über das Internet. Alle dort nicht aufgelisteten Ports sind geschlossen. An diesen lauscht kein Dienst. Ein *open*-Eintrag in der *STATE*-Spalte weist darauf hin, dass ein Dienst lauscht und Verbindungen akzeptiert. Ein *filtered*-Eintrag bedeutet, dass der Port bereits durch einen Mechanismus verfügt, der *nmap* zurückgewiesen hat. Dies kann unterschiedliche Ursachen haben. Zum einen verfügen viele Dienste bereits von Haus aus über Konfigurationsmöglichkeiten, mit denen festgelegt werden kann, von wo aus ein Dienst erreichbar sein soll. Zum anderen können bereits eingesetzte Paket- oder Dienstfilter eine Anfrage zurückweisen. Unter Berücksichtigung dieser Erkenntnis weist ein *filtered*-Eintrag also nicht zwangsläufig auf die Existenz eines lauschenden Dienstes hin. Die Spalte *SERVICE* gibt den Namen des gefundenen Dienstes an. Das ist natürlich nur bei offenen Ports möglich. In der *VERSION*-Spalte stehen nähere Informationen zu dem laufenden Dienst.

Die letzten beiden Spalten gehört nicht zur Ausgabe von *nmap*. Sie dienen bereits der Anforderungsanalyse an das Firewall-System. Die *SOLL*-Spalte enthält Informationen, von wo aus die gefundenen Dienste in Zukunft erreichbar sein sollen. In der (+/-)-Spalte wird markiert, ob Dienste derzeit wirklich laufen. Welche Dienste auf dem eigenen System laufen kann auch mit dem Befehl:

```
# netstat -lnp
```

ermittelt werden. Dabei können je nach Konfiguration von *nmap* sogar noch Dienste angezeigt werden, die der Portscan übergangen hat. Ist dies der Fall, werden sie der Liste hinzugefügt, wie im Beispiel die lauschenden UDP-Ports. An Ports wie *netbios*(137-139), die von *nmap* zwar als *filtered* deklariert wurden, jedoch in der *netstat*-Ausgabe nicht auftraten lauscht kein Dienst.

Nach dem Vervollständigen der Tabelle werden die entsprechenden Dienste nach der angestrebten Erreichbarkeit sortiert, dabei wird der nicht ermittelbare Dienst an UDP-Port 10000 erst einmal nicht nach außen hin zugänglich gemacht. Sollte er zu Webmin gehören und die Sperrung dieses Ports Probleme bereiten, kann er immer noch der *ALL*-Gruppe hinzugefügt werden:

```
ALL      ftp(21/tcp), ssh(22/tcp), http(80/tcp), https(443/tcp), cvs(2401/tcp),
         webmin(10000/tcp), http(10080/tcp), http(10443/tcp)
```

```
LNET     Xvnc(5806/tcp), Xvnc(5906/tcp), ntpd(123/tcp)
```

```
CLUSTER  mon(2583/tcp), mysqld(3306/tcp), rsync(10837/tcp), mon(2583/udp)
```

```
LOCAL    smtp(25/tcp)
```

```
DESPINA  telnet(23/tcp)
```

```
GALATEA  heartbeat(694/udp), heartbeat(54534/udp), heartbeat(54535/udp)
```

Tabelle 6.2: Dienstanalyse von PC-KQC

PORT	STATE	SERVICE	VERSION	SOLL	(x/-)
21/tcp	open	ftp	vsFTPd	<b>ALL</b>	+
22/tcp	open	ssh	OpenSSH	<b>ALL</b>	+
23/tcp	open	telnet		<b>DESPINA</b>	+
25/tcp	filtered	smtp		<b>LOCAL</b>	+
80/tcp	open	http	Apache httpd	<b>ALL</b>	+
135/tcp	filtered	msrpc		/	-
136/tcp	filtered	profile		/	-
137/tcp	filtered	netbios-ns		/	-
138/tcp	filtered	netbios-dgm		/	-
139/tcp	filtered	netbios-ssn		/	-
443/tcp	open	https	Apache httpd	<b>ALL</b>	+
445/tcp	filtered	microsoft-ds		/	-
617/tcp	filtered	sco-dtmgr		/	-
2401/tcp	open	cvspserver	cvs pserver	<b>ALL</b>	+
2583/tcp	open	unknown	(MON)	<b>CLUSTER</b>	+
3306/tcp	open	mysql	MySQL 4.0.15	<b>CLUSTER</b>	+
5806/tcp	open	unknown	(Xvnc)	/	+
5906/tcp	open	unknown	(Xvnc)	/	+
6006/tcp	open	X11	(access denied)	/	+
10000/tcp	open	http	Webmin httpd	<b>ALL</b>	+
10080/tcp	open	http	Apache httpd	<b>ALL</b>	+
10443/tcp	open	ssl/http	Apache httpd	<b>ALL</b>	+
10837/tcp	open	rsync	rsync	<b>CLUSTER</b>	+
54534/udp	open	unknown	(heartbeat)	<b>GALATEA</b>	+
54535/udp	open	unknown	(heartbeat)	<b>GALATEA</b>	+
10000/udp	open	unknown	?	(?)	+
2583/udp	open	unknown	(MON)	<b>CLUSTER</b>	+
694/udp	open	unknown	(heartbeat)	<b>GALATEA</b>	+
123/udp	open	unknown	(ntpd)	<b>LNET</b>	+

Aus der Aufstellung ergeben sich 6 Gruppen für diesen Rechner, von denen Zugriff auf die zur Gruppe gehörenden Dienste gestattet wird. In der *ALL*-Gruppe befinden sich alle Dienste, die von überall erreichbar sein sollen, während die Dienste der *LNET*-Gruppe lediglich im lokalen Netz benötigt werden. Das *smtp*-Mailgateway soll sogar nur von *PC-KQC* selbst verwendet werden können. Der Telnet-Server wurde nur deshalb gestartet, um als Hintertür zu dienen, falls der SSH-Server ausfallen sollte<sup>3</sup>. Lediglich vom Rechner *Despina* aus ist die Verwendung des Dienstes daher gestattet. Die Gruppen *CLUSTER* und *GALATEA* erhalten Zugangsberechtigung zu Diensten, die nur für die Kommunikation und Verwaltung innerhalb des Clusters eine Rolle spielen. Andere Rechner sollen darauf keinen Zugriff besitzen.

Die aufgestellten Gruppen werden in den beiden Konfigurationsdateien *ports.tcp* und *ports.udp* abgebildet, die später leicht erweitert werden können. Als Beispiel für den Aufbau der Dateien wird hier nur ein Ausschnitt aus der Datei *ports.tcp* angegeben:

```
ALL:      21 22 80 443 2401 10000 10080 10443
LNET:    123 5806 5906
```

Passend zu den Konfigurationsdateien wurde ein Firewall-Skript mittels *iptables*-Paketfilter erstellt. Dieses und die beiden Konfigurationsdateien sind im Anhang A.6 zu finden. Abbildung 6.2 zeigt die Ausgaben des Firewall-Skripts beim Start. Es empfiehlt sich, das Skript mit einem Startskript, wie es in Anhang A.1 abgedruckt ist zu versehen. Auf diese Weise kann es beim Start des Rechners automatisch die eingetragenen Gruppen und Ports aus den Konfigurationsdateien auslesen und die Firewall einrichten.

Der *iptables*-Paketfilter ist ein mächtiges Werkzeug und sehr flexibel einsetzbar. Er kann DOS-Attacken verhindern, Portscans unterbinden oder auch HTTP-Pakete ausfiltern, welche Viren enthalten. Genaue Informationen zur Benutzung von *iptables* können der Manpage<sup>4</sup> entnommen werden.

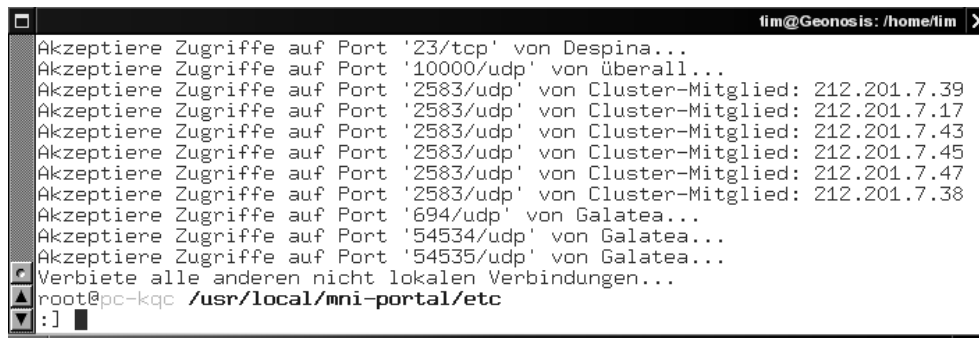
Mit *iptables* kann im Übrigen auch das Problem direkter Zugriffe auf die Webdienste der Realserver verhindert werden. Dazu bietet es sich an für das *firewall*-Skript eine Gruppe mit dem Namen *LOADBALANCER* zu erzeugen. Dieser können die Ports der Dienste zugewiesen werden, welche der Loadbalancer verteilt. Über eine Filterregel, die bspw. diese Dienste nur von den MAC-Adressen der beiden Loadbalancer akzeptiert wird verhindert, dass ein Realserver direkt kontaktiert und der Loadbalancer umgangen wird.

## 6.2 Datensicherung

Die Datensicherung ist ein weiterer Punkt, den die Plattform eines hochverfügbaren Systems bieten muss. Gerade ein solches System kann sich keinen Datenverlust er-

<sup>3</sup>Da über Telnet alle Daten inklusive der Passwörter unverschlüsselt übermittelt werden, sollten sich Benutzer über diesen Dienst nicht einloggen können.

<sup>4</sup>*iptables* Manpage <http://athena.fit.qut.edu.au/cgi-bin/man/man2html?iptables>



```
lim@Geonos is: /home/lim X
Akzeptiere Zugriffe auf Port '23/tcp' von Despina...
Akzeptiere Zugriffe auf Port '10000/udp' von überall...
Akzeptiere Zugriffe auf Port '2583/udp' von Cluster-Mitglied: 212.201.7.39
Akzeptiere Zugriffe auf Port '2583/udp' von Cluster-Mitglied: 212.201.7.17
Akzeptiere Zugriffe auf Port '2583/udp' von Cluster-Mitglied: 212.201.7.43
Akzeptiere Zugriffe auf Port '2583/udp' von Cluster-Mitglied: 212.201.7.45
Akzeptiere Zugriffe auf Port '2583/udp' von Cluster-Mitglied: 212.201.7.47
Akzeptiere Zugriffe auf Port '2583/udp' von Cluster-Mitglied: 212.201.7.38
Akzeptiere Zugriffe auf Port '694/udp' von Galatea...
Akzeptiere Zugriffe auf Port '54534/udp' von Galatea...
Akzeptiere Zugriffe auf Port '54535/udp' von Galatea...
Verbiete alle anderen nicht lokalen Verbindungen...
root@pc-kqc /usr/local/mni-portal/etc
:]
```

Abbildung 6.2: Ausgaben des Firewallskripts

lauben. Daher sind regelmäßige automatische Sicherungen des Dateisystems und der Datenbanken unumgänglich. Um den Performanceaspekt zu wahren, sollten diese Sicherungen zu Nulllastzeiten vorgenommen werden. Auf die Analyse des Nutzerverhaltens wurde bereits auf Seite 18 kurz eingegangen. Automatische Datensicherungen bergen allerdings auch wieder die Gefahr der vorgetäuschten Sicherheit. Es muss daher kontrolliert werden, ob sie wirklich das sichern, was sie sollen und ob die Datensicherung überhaupt abläuft. Dazu könnten z.B. E-mail Reports an den Administrator verschickt werden. Die typische Gefährdung für Daten eines Web-Portals sind:

- Technisches Versagen
- Angriffe
- Katastrophen
- Administrator

Wegen technischem Versagen von Komponenten wird bereits die in Kapitel 5 beschriebene Hochverfügbarkeitslösung eingesetzt. Dadurch, dass gleiche Daten auf verschiedenen Maschinen liegen, ist die Wahrscheinlichkeit eines Datenverlusts durch technisches Versagen gering. Befinden sich alle Maschinen des Hochverfügbarkeitssclusters allerings in der selben Umgebung, im selben Gebäude oder gar im selben Raum, können Katastrophen wie z.B. Feuer oder Flut alle Datenbestände auf einmal vernichten. Daher sollten die Daten auch an einem räumlich getrennten Ort aufbewahrt werden. Je nach Datenmenge ist das ohne eine Standleitung ein Problem, daher wird in großen Betrieben häufig auf Speichermedien, wie Bänder gesichert. Ein Open Source-Webportal verwendet in der Regel keine so großen Datenmengen. Daher ist eine Sicherung auch über ein externes Netz möglich. Hierbei sollte allerdings an die Verschlüsselung von sensiblen Daten über unsichere Netze gedacht werden. Sicherungen sind aber nicht nur wegen technischem, sondern auch wegen menschlichem Versagen notwendig. Das versehentliche Löschen eines ganzen Verzeichnisbaums geht sehr schnell. Auch aus diesem Grund sind Sicherungen sehr zu

empfehlen. Wird auf die lokale Festplatte gesichert, bietet dies keinen Schutz vor technischem Versagen, aber immerhin vor versehentlich falsch durchgeführten Konfigurationen, fälschlichem Löschen oder Überschreiben von Dateien. Da als Portalplattform Linux verwendet wird, können dort Sicherungen des Dateisystems und der Datenbanken zeitgesteuert über einen cron-Job erfolgen. Dafür wurden im Laufe dieser Arbeit einige Backup-Skripten geschrieben, die verschiedene Verzeichnisse mit deren Unterverzeichnissen, sowie die auf dem Rechner befindlichen Datenbanken sichern und auf einen entfernten Rechner kopieren. Damit die Festplatten nicht aufgrund alter Sicherungsdateien volllaufen, werden Sicherungen, die älter sind als ein Monat automatisch gelöscht, sofern neuere Sicherungen vorliegen. Vorlagen für diese Skripten befinden sich im Anhang [A.5](#). Weitere Informationen zur Datensicherung finden sich in der Sicherheitscheckliste.

# Kapitel 7

## Checklisten

Im Rahmen dieser Arbeit wurden Checklisten erstellt, die bestehende Webportale auf Sicherheits-, Hochverfügbarkeits- und Performance-Aspekte hin bewerten. Diese Checklisten wurden nicht in Papierform erzeugt, sondern sind Teil eines QA-Portals<sup>1</sup>, welches innerhalb des Fachbereichs MNI erstellt wurde.

### 7.1 QA-Portal



Abbildung 7.1: QA-Portal

Das QA-Portal der Fachhochschule Gießen-Friedberg ist ursprünglich das Ergebnis eines Teams von Studenten der Veranstaltung Softwaretechnik II aus dem Wintersemester 2003/2004 bei Herrn Prof. Dr. Klaus Quibeldey-Cirkel. Das Team, bestehend aus Lothar Lattermann, Michel Krämer, Florian Schultheis und Andreas Kiebach hatte als Content Management System PHP-Nuke<sup>2</sup> gewählt. Basierend auf diesem Sy-

<sup>1</sup>QA-Portal <http://pc-kqc.mni.fh-giessen.de>

<sup>2</sup>PHP-Nuke Homepage <http://phpnuke.org/>

stem hatte es mit der Entwicklungsmethode *Extreme Programming* ein objektorientiertes Checklistenbackend-System geschaffen und ein Frontend zur Benutzung, Bewertung und Sicherung der Checklisten in das bestehende CMS<sup>3</sup> eingebunden. Die im Laufe dieser Diplomarbeit entstandenen Checklisten sind in dieses Portal aufgenommen worden.

Es zeigte sich jedoch, dass die vorhandenen Klassen und auch das Frontend für das Vorhaben nicht ausreichten. Bisher war lediglich die Eintragung eines kurzen Kommentars zu jeder Frage in der Checkliste vorgesehen. Da jedoch diese Checkliste nicht nur ein bereits vorhandenes System bewerten, sondern auch dabei helfen sollte, die geforderten Kriterien zu erfüllen, musste ein erweitertes System zum Eintragen und Formatieren von Text und Bildmaterial geschaffen werden. Abbildung 7.2 zeigt

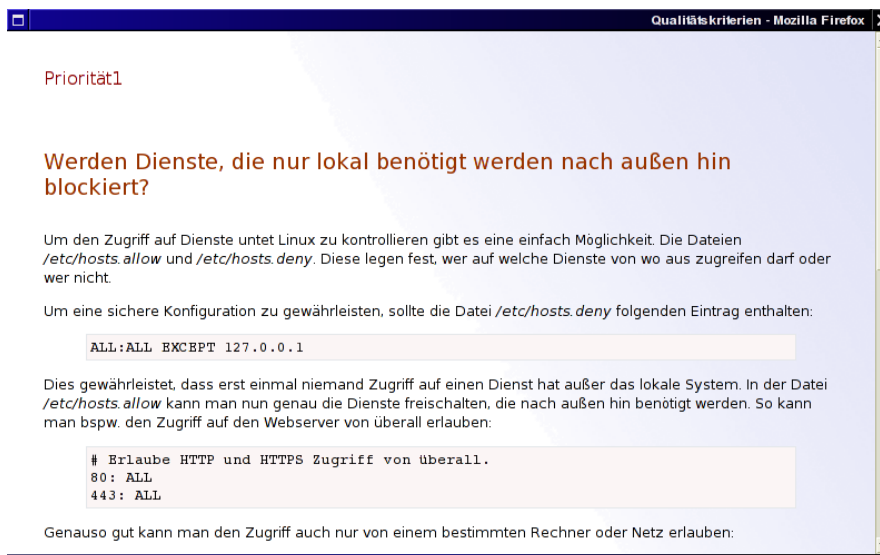


Abbildung 7.2: Neue Hilfe zu Checklistenfragen

ein Beispiel für die neuen Hilfen zu den Checklistenfragen. Diese können nun Bilder, externe und interne Links, sowie formatierten Text enthalten. Auch das Quellsystem wurde überarbeitet, denn das alte System ließ pro Frage nur eine weiterführende Quelle zu. Wie Abbildung 7.3 zeigt, können pro Frage nun beliebig viele Quellen zu weiterführender Literatur oder Websites angegeben werden. Durch einen Klick auf eine Online-Quelle wird die entsprechende Website geöffnet. Bei einer Bücherquelle erscheint eine Seite mit näheren Informationen zum Buch. Schließlich wurde noch das Design angepasst und auf eine angemessene Darstellung für die gängigsten Browser erweitert.

<sup>3</sup>Content Management System

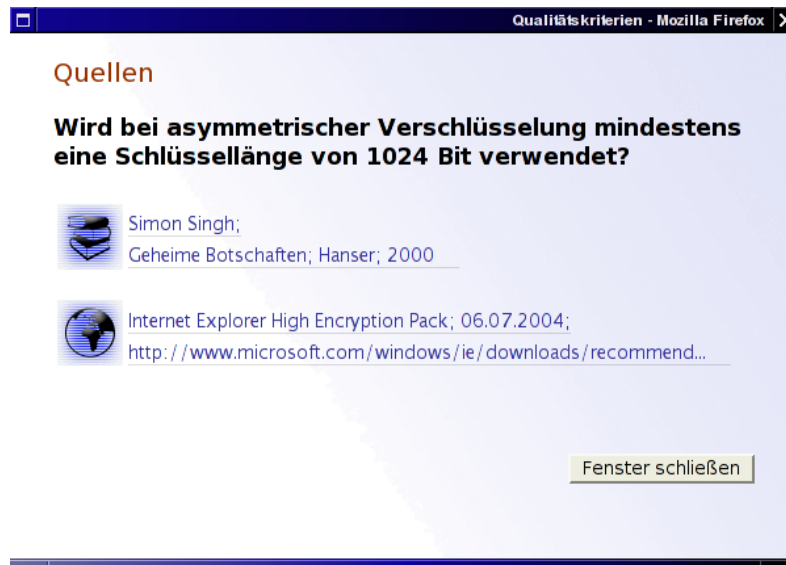


Abbildung 7.3: Neues Quellensystem

## 7.2 Evaluationschecklisten

Die im Laufe dieser Arbeit erstellten Checklisten unterteilen sich, genau wie der Titel der Arbeit in die Bereiche *Verfügbarkeit*, *Performance* und *Sicherheit*. Dabei wurde versucht, eine Vielzahl von Aspekten aus allen drei Teilbereichen zu berücksichtigen. Einige der Themen, Hinweise und Erläuterungen stammen aus dieser Arbeit, es werden jedoch auch andere Aspekte angesprochen, welche hier keine Erwähnung finden.

Priorität1	<b>Wird bei symmetrischer Verschlüsselung mindestens eine Schlüssellänge von 128 Bit verwendet?</b>	<input type="radio"/> ja	<input type="radio"/> nein	<input checked="" type="radio"/> irrelevant	<a href="#">Hilfe</a> <a href="#">Quellen</a>
Priorität1	<b>Wird bei asymmetrischer Verschlüsselung mindestens eine Schlüssellänge von 1024 Bit verwendet?</b>	<input type="radio"/> ja	<input type="radio"/> nein	<input checked="" type="radio"/> irrelevant	<a href="#">Hilfe</a> <a href="#">Quellen</a>
Priorität1	<b>Werden Benutzereingaben über reguläre Ausdrücke überprüft?</b>	<input type="radio"/> ja	<input type="radio"/> nein	<input checked="" type="radio"/> irrelevant	<a href="#">Hilfe</a> <a href="#">Quellen</a>
Priorität2	<b>Werden Passwörter aus Websitzungen vor der Übertragung über das Netz verschlüsselt?</b>	<input type="radio"/> ja	<input type="radio"/> nein	<input checked="" type="radio"/> irrelevant	<a href="#">Hilfe</a> <a href="#">Quellen</a>
Priorität2	<b>Werden Mitgliederbereiche auf Webseiten durch Passwörter geschützt?</b>	<input type="radio"/> ja	<input type="radio"/> nein	<input checked="" type="radio"/> irrelevant	<a href="#">Hilfe</a> <a href="#">Quellen</a>

Abbildung 7.4: Ausschnitt aus der Sicherheitscheckliste

Die Kriterien der Checkliste lassen sich in drei Kategorien aufteilen:

**Priorität 1** Punkte, die beachtet werden *müssen*.

**Priorität 2** Punkte, die beachtet werden *sollten*.

**Priorität 3** Punkte, die beachtet werden *können* (*nice to have*).

Je nach Priorität werden die Punkte bei der Auswertung gewichtet. Grundsätzlich setzt sich die Checkliste aus *Ja / Nein* Fragen zusammen. Trifft eine spezielle Frage auf die Anforderungen des Benutzers nicht zu, kann er diese als *irrelevant* kennzeichnen. Diese Frage wird dann bei der Auswertung nicht berücksichtigt. Der Aufbau der Checklisten ist in Abbildung 7.4 auf der vorherigen Seite beispielhaft dargestellt. Sämtliche Punkte sind zunächst als *irrelevant* gekennzeichnet. Auf diese Weise müssen nicht alle Fragen für eine Auswertung beantwortet werden. Nachdem die Checkliste durchgegangen wurde, kann eine Auswertung gestartet werden. Diese zeigt die erreichte Punktzahl an. Das erreichte Ergebnis kann abgespeichert und mit denen anderer Mitgliedern des Portals verglichen werden. Alle als irrelevanten deklarierten Kriterien werden in einer Liste am Ende der Auswertung angezeigt. Anschließend wird, wie in Abbildung 7.5 zu erkennen ist, eine Liste aller nicht erfüllter Kriterien zusammengefasst. Über einen Klick auf die Frage kann sich darüber informieren werden, wie das Kriterium erfüllt werden kann. Zusätzlich öffnet ein Klick auf das Quellensymbol links das bereits beschriebene Quellenfenster zu weiterführenden Informationen.

#### Folgende Kriterien werden nicht eingehalten:






	<a href="#">Wird bei symmetrischer Verschlüsselung mindestens eine Schlüssellänge von 128 Bit verwendet?</a>
	<a href="#">Werden Benutzereingaben über reguläre Ausdrücke überprüft?</a>
	<a href="#">Werden Passwörter aus Websitzungen vor der Übertragung über das Netz verschlüsselt?</a>
	<a href="#">Sind nur Dienste gestartet, die benötigt werden?</a>
	<a href="#">Verwendet das System ein Firewall-Konzept?</a>

Abbildung 7.5: Nicht erfüllte Kriterien

# Kapitel 8

## Abschließende Betrachtung

### 8.1 Zusammenfassung

Nach einer Vorstellung der im Fachbereich MNI der Fachhochschule Gießen-Friedberg zur Verfügung stehenden Rechnerhardware, wurden die Grundvoraussetzungen beschrieben, die an die Hardware und Software für die Testumgebung eines hochperformanten und hochverfügbaren Webportals gestellt werden, welches mittels des Linux Virtual Server Projects realisiert werden soll. Anschließend wurde die Installation und Verwendung der Performance-Komponente mittels eines Linux Loadbalancers und beliebigen Realservern durchgeführt und beispielhaft dargestellt. Dabei wurde auch auf die Loadbalancer-Grundlagen, wie Scheduling-Algorithmen und Loadbalancer-Verfahren eingegangen und diskutiert, wo Vor- und Nachteile liegen. Nach Beispielen zur Einrichtung von Loadbalancern und Realservern wurde auf das ARP-Problem eingegangen, welches bei verschiedenen Verfahren existiert und einige Lösungen dafür vorgeschlagen. Anschließend fanden Last- und Performance-tests eines mittels Lastverteilung betriebenen Portals statt. Der nächste Teil stelle die Hochverfügbarkeitskomponente vor, mit der das zuvor aufgestellte System aufgerüstet wurde. Hier wurde die Einrichtung mehrerer Möglichkeiten zum *failover* des Loadbalancers gegeben. Schließlich wurde die Datenbankreplikation von MySQL besprochen, verschiedene Replikationssysteme vorgestellt und deren Vor- und Nachteile aufgezeigt. Daran anschließend wurden kurz die Grundlagen für Datenschutz und Datensicherheit erläutert. Im letzten Teil schließlich wurde das weiterentwickelte Quality Assurance Portal und die Checklisten Hochverfügbarkeit, Performance und Sicherheit vorgestellt. Anhand dieser Checklisten können Besucher des Portals ihre Web-Plattformen abhärten und bekommen Anleitungen und Tipps, was an diesen verbessert werden kann.

### 8.2 Ausblick

Der rapide Anstieg der Nutzung des Internets in den letzten Jahren hat aus einem zuvor als Nischenlandschaft abgetanen Geschäftszweig neue Märkte und Informationsplattformen geschaffen. In Zeiten des Informationszeitalers ist es für Betreiber eines Webportals immer wichtiger geworden, sich mit den Begriffen Verfügbarkeit,

Performanz und Sicherheit auseinanderzusetzen. Qualitätssicherung der Webauftritte ist heute ein *muss*.

Eine äußerst erfolgreiche Website kann heute nur noch durch Lastverteilung aufrecht erhalten werden. Das bedeutet aber auch, dass bei einem immer stärkeren Aufkommen solcher Systeme, wieder mehr auf die Administration geachtet werden muss. Ein stiefmütterlich behandeltes System kann nicht als hochverfügbar angesehen werden. Die Verfügbarkeit und Performance hängen nicht allein von der Technik ab, sondern vom Wissen der Administratoren um den Ist-Zustand des Systems.

Gerade im Open Source-Bereich haben sich Hochverfügbarkeits- und Performance-lösungen in letzter Zeit etabliert. Michael Soltau schreibt in seinem Buch *Unix/Linux Hochverfügbarkeit* [Sol02] auf Seite 77: „Dabei haben viele Pakete keinesfalls den „Bastelstatus“, der ihnen vielfach noch nachgesagt wird. Trotzdem gibt es auch immer mehr kommerzielle Produkte, die vom Prinzip her in direkter Konkurrenz zu den freien Produkten stehen“. Open Source-Einzelprodukte können es im Verbund also durchaus mit kostenpflichtigen Komplettpaketen aufnehmen. Für die Zukunft wird es wohl auch im Open Source-Bereich Komplettpakete geben. Ein Beispiel dafür ist das *Ultra Monkey*<sup>1</sup> Projekt, welches ein Open Source-Komplettpaket anbietet. Auch im Bereich der Datenbanken gibt es Neuerungen: *MySQL Cluster* befindet sich im Alpha-Stadium und wirbt damit, hohe Performance und 99.999% Verfügbarkeit mit herkömmlichen Rechnern zu gewährleisten. Auch der aktuelle Erfolg von Typo3 und anderen Content Management Systemen zeigt einen fortwährenden Aufschwung in diesem Bereich. Die Anwendungen für Clustersysteme im Web ist beinahe unbeschränkt und die Entwicklungen innerhalb der Open Source-community sowieso.

---

<sup>1</sup>Ultra Monkey <http://www.ultramonkey.org/>

# Anhang A

## Quellcode

### A.1 Startskriptvorlage für Heartbeat

```
#!/bin/bash
#
# Startet oder beendet einen Dienst
# (c) Tim Pommerening / 2004
#

### VARIABLEN #####
NAME=$(basename $0)      # Name des Dienstes
SNAME=$(basename $0)d    # Name der Dienstdatei
USER=root                # User des Dienstes
### FUNKTIONEN#####
function start() {
    rval=`ps -u $USER | grep -c $SNAME`
    if [ "$rval" != "0" ]; then
        echo -n " already running..."
        return 1
    fi
    $SNAME
    return $?
}

function stop() {
    rval=`ps -u $USER | grep -c $SNAME`
    if [ "$rval" != "0" ]; then
        pids=`ps -u $USER | grep $SNAME | awk '{ print$1 }'`
        for pid in $pids; do
            kill $pid >/dev/null 2>&1
            sleep 1
        done
        sleep 1
        rval=`ps -u $USER | grep -c $SNAME`
        return $rval
    else
        echo -n " was not running..."
        return 1
    fi
}
```

```
}

function status() {
    rval=`ps -u $USER | grep -c $SNAME`
    if [ "$rval" != "0" ]; then
        echo -e "\nThere are $rval $SNAME instances running."
    else
        echo -e "$SNAME isn't running."
    fi
    return 0
}

function restart() {
    rval=`ps -u $USER | grep -c mysqld`
    if [ "$rval" != "0" ]; then
        stop;
        sleep 1;
        start
        return $?
    else
        return 1
    fi
}

### MAIN #####

case $1 in
start)
    echo -n "Starting $NAME ..."
    start
    if [ $? -eq 0 ]; then
        echo "done"
        logger -t $NAME "Start successfull"
    else
        echo "failed"
        logger -t $NAME "Start failed"
    fi
;;
stop)
    echo -n "Stopping $NAME ..."
    stop
    if [ $? -eq 0 ]; then
        echo "done"
        logger -t $NAME "Stop successfull"
    else
        echo "failed"
        logger -t $NAME "Stop failed"
    fi
;;
restart)
    echo -n "Restart $NAME ..."

```

```

restart
if [ $? -eq 0 ]; then
    echo "done"
    logger -t $NAME "Restart successfull"
else
    echo "failed"
    logger -t $NAME "Restart failed"
fi
;;
status)
    status
;;
*)
    echo "Usage: $NAME {start|stop|restart|status}"
esac
exit 0

```

## A.2 Mon-Installationskript

```

#!/bin/bash
# MON install-script
# Hilft bei der Installation des
# mon-<version>.tar.gz Pakets
# Muss im Verzeichnis gestartet werden,
# in dem mon entpackt wurde.
# 2004 Tim Pommerening
#####
monroot=/usr/lib/mon
monconfig=/etc/mon
initd=/etc/rc.d/init.d
#####
echo -n „Creating mon root...”
mkdir -p $monroot
cp -r alert.d mon.d mon $monroot
echo „[done]”
echo -n „Creating config dir...”
mkdir -p $monconfig
cp etc/auth.cf etc/example.cf $monconfig
cp etc/very-simple.cf $monconfig/mon.cf
echo „[done]”
echo -n „Creating starting script...”
if [ -d „$initd” ]; then
    cp etc/S99mon $initd/mon
    echo „[done]”
else
    echo „[failed]”
fi

```

## A.3 Mon-Skript für Linux-Virtual-Server

```
#!/usr/bin/perl
#
# lvs.alert - Linux Virtual Server alert for mon
#
# It can be activated by mon to remove a real server
# when the service is down, or add the server when the
# service is up.
#
#
use Getopt::Std;
getopts ("s:g:h:t:l:P:V:R:W:F:u");
$ipvsadm = "/sbin/ipvsadm";
$protocol = $opt_P;
$virtual_service = $opt_V;
$remote = $opt_R;
if ($opt_u) {
    $weight = $opt_W;
    if ($opt_F eq "nat") {
        $forwarding = "-m";
    } elsif ($opt_F eq "tun") {
        $forwarding = "-i";
    } else {
        $forwarding = "-g";
    }
}

if ($protocol eq "tcp") {
    system("$ipvsadm -a -t $virtual_service -r $remote -w \
        $weight $forwarding");
} else {
    system("$ipvsadm -a -u $virtual_service -r $remote -w \
        $weight $forwarding");
}
} else {
    if ($protocol eq "tcp") {
        system("$ipvsadm -d -t $virtual_service -r $remote");
    } else {
        system("$ipvsadm -d -u $virtual_service -r $remote");
    }
}
};
```

## A.4 Rsync-Skript für Dateiabgleich

### A.4.1 /etc/globals

```
#!/bin/bash
#
# Tim Pommerening (c) 2004
# Globale Konstanten für das MNI-Portal
```

```

#
# Diese Datei enthält Datenbank-Passwörter.
# Vergewissern Sie sich, dass sie nur vom Benutzer
# lesbar, beschreibbar und ausführbar ist
# (chmod 700 globals).
#
# Benutzername und Passwort der Datenbank für das Portal
DB_USER="benutzer"
DB_PASS="passwort"

# Datenbankname
DATABASE="datenbank"

# Temp-Pfad für die Synchronisation relativ zu $MNIDIR
SYNCPATH=tmp/dbsync

# Root-Verzeichnis des Portals (MIT '/' am Ende)
WEB_ROOT="/var/www/html/devel/qa/"

# 0 wenn Dateien, die auf dem Quellrechner gelöscht
# wurden, auf den Zielrechnern nicht gelöscht werden
# sollen. Sonst 1. Vorsicht: Geben Sie in
# '$MNIDIR/etc/rsync_excludes' Dateien oder
# Verzeichnisse an, die nicht synchronisiert werden
# sollen. Dazu gehören z.B. Verzeichnisse, in die User
# über das Webinterface Dateien hochladen können. Geben
# dieses Verzeichnis nicht in der Datei
# '$MNIDIR/etc/rsync_excludes' an, und verwenden
# RSYNC_DELETE=1, werden diese Dateien gelöscht,
# da sie auf dem Testrechner nicht vorhanden sind.
RSYNC_DELETE=1

```

#### A.4.2 /lib/common\_functions

```

#!/bin/bash
#
# Tim Pommerening (c) 2004
# Oft benötigte allgemeine Funktionen
#
. $MNIDIR/lib/tp_functions

#####
# Zeigt die Fehlermeldung an, falls errno != 0
# $1 errno Rückgabewert eines Programms
# $2 msg Fehlermeldung
function show_err() {
    local errno=$1
    local msg=$2
    if [ "$#" != "2" ]; then

```

```
        gprintf "Syntax: show_err {errno} {msg}\n"
        return 1
    fi

    if [ $((errno)) -ne 0 ]; then
        gprintf "FEHLER %s: %s\n" "$errno" "$msg"
    fi
    return 0
}

#####
# Testet, ob der eingegebene Rückgabewert ok ist.
# $1 rval Rückgabewert des Programms
function check_rval() {
    local rval=$1
    if [ $((rval)) -eq 0 ]; then
        echo_success;echo
        return 0
    else
        echo_failure;echo
        return 1
    fi
}

#####
# Testet, den letzten Befehl
function check_last_rval() {
    if [ $? = 0 ];
        then echo_success;echo
        return 0
    else
        echo_failure;echo
        return 1
    fi
}
```

### A.4.3 Rsync-Skript

```
#!/bin/sh
#
# Tim Pommerening (2004)
#
# gleicht alle Daten, die auf dem Loadbalancer im
# angegebenen Verzeichnis liegen auf die Realserver
# (aus der Datei 'realserver' ab.
#
# Hinweis: Automatischer Ablauf benötigt SSH-Public-Keys
# für Auto-Login
#
```

```

# Globale Konstanten laden
. $MNIDIR/etc/globals
. $MNIDIR/lib/common_functions

# Datei mit den Realserverdaten
REALSERVER=$MNIDIR/etc/realserver
RSYNC_EXCLUDES=$MNIDIR/etc/rsync_excludes

#####
# Prüft, ob seit dem letzten Upload Änderungen vorkamen
CMPFILE=$MNIDIR/tmp/qafs.md5 TMPFILE=$MNIDIR/tmp/qafs.md5~

# Prüfe, ob sich seit dem letzten mal etwas geändert hat
ls -laR $WEB_ROOT | md5sum >$TMPFILE cmp $CMPFILE \
    $TMPFILE >/dev/null 2>&1 RVAL=$?
if [ $RVAL -eq 0 ]; then
    echo "Keine Änderungen am Dateisystem seit " \
        "dem letzten Upload."
    exit 0;
fi

# Es gab Änderungen. Aktualisiere das $CMPFILE
cp -f $TMPFILE $CMPFILE

#####
# Erzeuge Listen aus den Spalten der Datei
users=(`grep -v ^# $REALSERVER | awk '{print $1}'`)
hosts=(`grep -v ^# $REALSERVER | awk '{print $2}'`)
webdirs=(`grep -v ^# $REALSERVER | awk '{print $4}'`)
usrgrps=(`grep -v ^# $REALSERVER | awk '{print $5}'`)
excludes=`grep -v ^# $RSYNC_EXCLUDES | awk '{print " \
    --exclude " ; print $1}'`
MAXHOSTS=${#hosts[*]} n=0

# Alle Realserver aus der Konf-Datei synchronisieren
while [ "$n" -lt "$MAXHOSTS" ] ; do
    user=${users[$(n)]};
    host=${hosts[$(n)]};
    webdir=${webdirs[$(n)]};
    usrgrp=${usrgrps[$(n)]};
    n=$((n+1))
    ZIEL=$user@$host:$webdir

    echo ""
    echo "- - - -"
    echo "Synchronisiere Daten mit '$host'"

# Daten synchronisieren

```

```
if [ "$RSYNC_DELETE" == "1" ]; then
    rsync -azvL -e ssh \
        --delete $excludes $WEB_ROOT $ZIEL
else
    rsync -azvL -e ssh $excludes $WEB_ROOT $ZIEL
fi

# Rechte setzen
ssh $user@$host chown $usrgrp $webdir -R
done
exit 1
```

## A.5 Backupskripten

### A.5.1 Konfigurationsdatei (backupconst.cfg)

```
#!/bin/bash
# Variablen und Konstanten für Backup-Skripten.
# (c) 2004 by Tim Pommerening
#
BACKUPROOT=/home/backup
REMOTEROOT=backup@remotehost:/home/backup
```

### A.5.2 Skript zum Löschen alter Backups (backupdel)

```
#!/bin/bash
#
# Tim Pommerening (2004)
#
# Löscht alte Backupdaten im übergebenen Verzeichnis.
# Als ALT werden Dateien gesehen, wenn sie:
#
# Ungleich dem aktuellen Monat sind.
# UND
# (
#   Der Monat kein Vormonat ist
#   ODER
#   (
#     Der Tag bei einem Vormonat kleiner ist, als
#     der Tag des aktuellen Monats.
#   )
# )
#
# Bsp.:
# Heute: Mon Jun 14 13:29:05 CEST 2004
#
# Verzeichnis mit Backupdaten:
# (1) -rw-r--r-- 199M May 9 01:11 www_040509.tgz
```

```

# (2) -rw-r--r-- 314M May 16 01:11 www_040516.tgz
# (3) -rw-r--r-- 342M Jun  1 08:47 www_040601.tgz
#
# (1) Wird gelöscht, da Monat != aktueller Monat
#     und Tag < aktueller Tag
# (2) Wird nicht gelöscht, da zwar
#     Monat != aktueller Monat,
#     aber Tag > aktueller Tag
# (3) Wird nicht gelöscht, da
#     Monat == aktueller Monat.
#
#####
VERZ=$1
ENDUNG=$2
DATUM=`date +%y%m` TAG=`date +%d`

if [ $# -lt 2 ]; then
    echo "backupdel {PFAD} {ENDUNG}"
    echo "Entfernt ältere Dateien als ein Monat" \
        " sind sofern es neuere gibt."
    echo "PFAD      = Pfad zu einem Verzeichnis," \
        "welches die Backupdateien enthält"
    echo "ENDUNG    = Endung der Backupdateien" \
        "(OHNE PUNKT) z.B. 'tgz'"
    echo ""
    exit 0;
fi

#####
ANZ=0
# Teste zuerst, ob es überhaupt schon Sicherungen
# von diesem Monat gibt.
for x in `ls -lah --time-style="+%y%m %d" \
    $VERZ | grep -e "^.*\.$ENDUNG" \
    | awk '{ print $6 }'; do
    if [ $((DATUM)) -eq $((x)) ]; then
        ANZ=$((ANZ+1))
    fi
done

echo -n `date +%y-%m-%d,%H:%M` `
echo -n " $VERZ : Anzahl aktuelle Sicherungen:" \
    " $ANZ"

if [ $((ANZ)) -eq 0 ]; then
    # Abbruch, wenn keine Sicherungen aus dem
    # aktuellen Monat vorhanden
    echo " => Abbruch!"
    exit 1
else

```

```

echo ""
fi

#####
# Gehe alle gewählten Dateien im Verzeichnis
for x in `ls -lah --time-style="+%y%m %d" \
$VERZ | grep -e "^.*\.$ENDUNG" \
| awk '{ print $6 ":" $7 ":" $8}'`; do
# Aufteilen der Werte in yymm, dd, Dateiname
tmp=${x%:*} # Alles vor letzten Sem. ist Datum
DAT_DATUM=${tmp%:*} # alles vorm ersten ist yymm
DAT_TAG=${tmp#*:} # alles nach dem letztem ist dd
tmp=${x#*:} # Alles vorm letzten ist Datum
DAT_NAME=${tmp#*:} # Alles danach ist Name

# Ist eine Datei ungleich dem aktuellen Monat?
if [ $(DAT_DATUM) -eq $(DATUM) ]; then
echo -n `date +%y-%m-%d,%H:%M` \
echo " Aktueller Monat --> $DAT_NAME" \
"(erhalten)"
else
DAT_DIFF=$((DATUM-DAT_DATUM))
if [ $DAT_DIFF -eq 1 -o $DAT_DIFF -eq 89 ]
then
# 89 wegen Jahreswechsel
# Prüfe, ob die Datei aus dem Vormonat eine
# kleinere Tageszahl hat
if [ $TAG -gt $DAT_TAG ]; then
# Diese Dateien werden gelöscht
echo -n `date +%y-%m-%d,%H:%M` \
echo " Älter als ein Monat --> " \
"$DAT_NAME (entfernt)"
rm -f $VERZ/$DAT_NAME
else
echo -n `date +%y-%m-%d,%H:%M` \
echo " Vormonat --> " \
"$DAT_NAME (erhalten)"
fi
else
# Alle anderen Dateien sind noch älter
# Und diese Dateien werden gelöscht
echo -n `date +%y-%m-%d,%H:%M` \
echo " Älter als ein Monat --> " \
"$DAT_NAME (entfernt)"
rm -f $VERZ/$DAT_NAME
fi
fi
done

```

### A.5.3 Backupskript für MySQL Datenbanken

```
#
# Backup der gesamten MySQL Datenbank
# (c) 2004 by Tim Pommerening
#
. backupconst.cfg

#####
FILE=$BACKUPROOT/sql/sql_`date +%y%m%d`.sql.gz
TARGET=$REMOTEROOT/sql

#####
mysqldump -A -l --opt --allow-keywords \
  | gzip > $FILE scp $FILE $TARGET

# Löschen veralteter Backups
./backupdel /home/backup/sql gz \
  >> /home/backup/backupdel.log 2>&1
```

### A.5.4 Backupskriptvorlage für Verzeichnisse

```
#
# Backup von /etc
# (c) 2004 by Tim Pommerening
#
. backupconst.cfg

#####
FILE=$BACKUPROOT/etc/etc_`date +%y%m%d`.tgz
TARGET=$REMOTEROOT/etc

#####
tar cvzf $FILE /etc scp $FILE $TARGET

# Löschen veralteter Backups
./backupdel /home/backup/etc tgz \
  >> /home/backup/backupdel.log 2>&1
```

## A.6 Firewall-Skript

```
#!/bin/bash
# Firewall-Skript
# Tim Pommerening (2004)
FILES="ports.tcp ports.udp"

# Einzelrechner und Netze
LNET="212.201.7.0/24"
```

```
DESPINA="212.201.7.45"
GALATEA="212.201.7.47"

# Rechner- und Netzgruppen
CLUSTER="212.201.7.39 212.201.7.17 212.201.7.45 \
        212.201.7.43 212.201.7.47 212.201.7.38"

# Alle existierenden Regeln löschen
iptables -F

# Wende alle Regeln auf TCP und auf UDP Ports an
for file in $FILES; do
    # Erlaube Zugriffe aus der ALL-Gruppe von überall.
    for port in `grep -v ^# $file | grep ALL \
| awk -F":" '{ print $2 }'; do
        proto=`echo $file | awk -F"." '{ print $2 }'`
        echo "Akzeptiere Port '$port/$proto' von überall..."
        iptables -A INPUT -t filter -p $proto --dport $port \
            -j ACCEPT
    done
done

# Erlaube Zugriffe aus der LNET-Gruppe aus lokalem Netz.
for port in `grep -v ^# $file | grep LNET \
| awk -F":" '{ print $2 }'; do
    proto=`echo $file | awk -F"." '{ print $2 }'`
    echo "Akzeptiere Port '$port/$proto' von $LNET..."
    iptables -A INPUT -t filter -p $proto --source $LNET \
        --dport $port -j ACCEPT
done

# Erlaube Zugriffe aus der LOCAL-Gruppe vom Localhost
for port in `grep -v ^# $file | grep LOCAL \
| awk -F":" '{ print $2 }'; do
    proto=`echo $file | awk -F"." '{ print $2 }'`
    echo "Akzeptiere Port '$port/$proto' von localhost"
    iptables -A INPUT -t filter -p $proto \
        --source 127.0.0.1 --dport $port -j ACCEPT
done

# Erlaube Zugriffe aus CLUSTER-Gruppe von diesen Rechnern.
for port in `grep -v ^# $file | grep CLUSTER \
| awk -F":" '{ print $2 }'; do
    proto=`echo $file | awk -F"." '{ print $2 }'`
    for member in $CLUSTER; do
        echo "Akzeptiere Port '$port/$proto' \
            von Cluster-Mitglied: $member"
        iptables -A INPUT -t filter -p $proto \
            --source $member --dport $port -j ACCEPT
    done
done
```

```
done

# Erlaube Zugriffe aus DESPINA-Gruppe von diesen Rechnern.
for port in `grep -v ^# $file | grep DESPINA \
| awk -F":" '{ print $2 }'; do
    proto=`echo $file | awk -F"." '{ print $2 }'`
    echo "Akzeptiere Port '$port/$proto' von Despina..."
    iptables -A INPUT -t filter -p $proto \
        --source $DESPINA --dport $port -j ACCEPT
done

# Erlaube Zugriffe aus GALATEA-Gruppe von diesen Rechnern
for port in `grep -v ^# $file | grep GALATEA \
| awk -F":" '{ print $2 }'; do
    proto=`echo $file | awk -F"." '{ print $2 }'`
    echo "Akzeptiere Port '$port/$proto' von Galatea..."
    iptables -A INPUT -t filter -p $proto \
        --source $GALATEA --dport $port -j ACCEPT    done
done

# Alle eingehenden nicht lokalen Startup Anfragen,
# die oben nicht expliziet erlaubt wurden,
# werden verboten
iptables -A INPUT -t filter -p tcp \
    --source ! 127.0.0.1 --syn -j REJECT
```

### A.6.1 Konfigurationsdatei *ports.tcp*

```
# TCP-Gruppen
#
# Syntax:
# {GRUPPE:} [port] ...
#
ALL:      21 22 80 443 2401 10000 10080 10443
LNET:     123 5806 5906
CLUSTER:  2583 3306 10883
LOCAL:    25
DESPINA:  23
GALATEA:
```

### A.6.2 Konfigurationsdatei *ports.udp*

```
# UDP-Gruppen
#
# Syntax:
# {GRUPPE:} [port], ...
#
ALL:
LNET:
```

```
CLUSTER: 2583  
LOCAL:  
DESPINA:  
GALATEA: 694 54534 54535
```

# Anhang B

## CD zur Diplomarbeit

Einen Hauptteil der Arbeit stellte das Erzeugen von Checklisten und die damit verbundene Erweiterung des Quality-Assurance-Portals dar. Dieses Portal mit allen Checklisten befindet sich auf der CD zur Diplomarbeit. Außerdem beinhaltet sie viele nützliche Bash- und Perlskripten, die für die Systemadministration einer Webportalumgebung verwendet werden können. Neben weiteren Hintergrundinformationen und Papers finden sich auch die Lasttesttools, sowie diese Diplomarbeit auf der CD.

### Inhalt der CD

- index.html
  - ↳ Diplomarbeit
    - ▣ PDF-Version
      - ▣ Farbversion
      - ▣ Graustufenversion
      - ▣ Doppelseitendruckversion
    - ▣ LyX-Version
      - ▣ Sämtliche verwendete Grafiken
      - ▣ Sämtliche doch nicht verwendete Grafiken
  - ↳ QA-Portal
    - ▣ CMS-Code
    - ▣ Datenbanken
    - ▣ Installationsanleitung
  - ↳ Skriptschmiede
    - ▣ MySQL Replikationsmonitore für MON
    - ▣ rsync Datenreplikationsmonitor für MON
    - ▣ Weitere Datenreplikationsskripten
    - ▣ Konfigurationstools zur Einrichtung der Realserver
    - ▣ Backup-Skriptkollektion
    - ▣ Firewall-Skript
    - ▣ Startskriptvorlage
    - ▣ Sonstiges
  - ↳ Tools

- ▣▣▣ ipvsadm
- ▣▣▣ Heartbeat
- ▣▣▣ Mon
- ▣▣▣ Ldirectord
- ▣▣▣ Siege
- ▣▣▣ NoARP-Modul

↳ Lasttests

- ▣▣▣ Lasttests als Rohdaten
- ▣▣▣ Lasttests als Excel und Open-Office-Datei
- ▣▣▣ Lasttestevaluation als PDF
- ▣▣▣ Tabellarische Lasttestevaluation als interaktives Web-Tool

↳ Weitere Dokumente

- ▣▣▣ LAMP-Howto: Installation eines LAMP-Systems ohne root-Rechte
- ▣▣▣ VNC-Howto
- ▣▣▣ Performance-Studie über Loadbalancingverfahren

# Abbildungsverzeichnis

3.1	Portalaufbau des Fachbereichs MNI . . . . .	12
4.1	Tagesleistungskurve . . . . .	18
4.2	Webserverstatistik von pc-kqc.mni.fh-giessen.de im Mai 2004 . . . . .	19
4.3	Überlasteter Webserver . . . . .	20
4.4	Loadbalancingverfahren: NAT . . . . .	23
4.5	Loadbalancingverfahren: Direct Routing . . . . .	25
4.6	Loadbalancingverfahren: IP-Tunneling . . . . .	27
4.7	ipvsadm: Ausgabe nach der Konfiguration . . . . .	31
4.8	LVS NAT-Konfiguration . . . . .	32
4.9	LVS Direct Routing-Konfiguration . . . . .	33
4.10	LVS-IP-Tunnel Konfiguration . . . . .	35
4.11	LVS-Konfiguration im Fachbereich MNI . . . . .	36
4.12	Hier tritt das ARP-Problem auf . . . . .	41
4.13	Wann kommt das ARP-Problem zur Geltung . . . . .	42
4.14	Realserver-Konfigurationstool . . . . .	43
4.15	Start einer Realserverkonfiguration . . . . .	44
4.16	Legenden für die Lasttestauswertung . . . . .	46
4.17	Lasttest 1-1 . . . . .	48
4.18	Lasttest 1-2 . . . . .	49
4.19	Lasttest 1-3 . . . . .	50
4.20	Lasttest 2 . . . . .	51
5.1	Webmin und Heartbeat . . . . .	60
5.2	Heartbeat Konfiguration mit Webmin . . . . .	61
5.3	Replikation bei leseintensiven Anwendungen . . . . .	69
5.4	Multi Master-Konfigurationen . . . . .	70
5.5	Replikationsmonitor - Alles in Ordnung . . . . .	71
5.6	Failover beim Ausfall eines Rechners im Ring . . . . .	72
5.7	Replikationsmonitor - Inkonsistenzen . . . . .	73
5.8	Lowcost-Replikation . . . . .	74
6.1	Trennung von Netzen . . . . .	80
6.2	Ausgaben des Firewallskripts . . . . .	84
7.1	QA-Portal . . . . .	86
7.2	Neue Hilfe zu Checklistenfragen . . . . .	87

7.3	Neues Quellensystem . . . . .	88
7.4	Ausschnitt aus der Sicherheitscheckliste . . . . .	88
7.5	Nicht erfüllte Kriterien . . . . .	89

# Tabellenverzeichnis

4.1	Beispiel: LVS-NAT . . . . .	24
5.1	Ausfallzeiten von Diensten innerhalb eines Jahres . . . . .	54
6.1	Kostenbeispiel einer Sicherheitslücke . . . . .	79
6.2	Dienstanalyse von PC-KQC . . . . .	82

# Literaturverzeichnis

- [Bau03] Michael D. Bauer. *Sichere Server mit Linux*, volume 1. O'Reilly, 2003.
- [Hor04a] Horms. Lvs-nat performance. [http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.LVS-NAT.html#lvs\\_nat\\_performance](http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.LVS-NAT.html#lvs_nat_performance), 12.05.2004.
- [Hor04b] Horms. Lvs: Transparent proxy (tp or horms' method). [http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.transparent\\_proxy.html](http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.transparent_proxy.html), 20.06.2004.
- [OK04] Patrick O'Rourke and Mike Keefe. Performance evaluation of linux virtual server. <http://www.missioncriticallinux.com/products/lvs.pdf>, 12.06.2004.
- [PD00] Larry L. Peterson and Bruce S. Davie. *Computernetze*. dpunkt.verlag, 2000.
- [Poh00] Norbert Pohlmann. *Firewall-Systeme*, volume 3. MITP-Verlag, 2000.
- [Rob04a] Alan Robertson. High-availability linux project. <http://linux-ha.org/>, 08.05.2004.
- [Rob04b] Alan Robertson. Highly-affordable high availability yes, you can squeeze high availability into your budget. [http://www.linux-mag.com/2003-11/availability\\_01.html](http://www.linux-mag.com/2003-11/availability_01.html), 23.05.2004.
- [Sol02] Michael Soltau. *Unix/Linux Hochverfügbarkeit*. mitp-Verlag, 2002.
- [SP04] Christian Stoll and Tim Pommerening. Evaluation von lasttest-tools und performance studie. <http://tyranus.de/web/include/studium/Lasttestevaluation.pdf>, 28.02.2004.
- [WZH04] Mike Wangsmo, Wensong Zhang, and Horms. Manpage: Ipvadm(8). [http://gd.tuwien.ac.at/linuxcommand.org/man\\_pages/ipvadm8.html](http://gd.tuwien.ac.at/linuxcommand.org/man_pages/ipvadm8.html), 25.07.2004.
- [ZB04] Jeremy D. Zawodny and Derek J. Balling. *High Performance MySQL*. O'REILLY, 2004.

- [Zha04a] Wensong Zhang. Linux virtual server project.  
*<http://www.linuxvirtualserver.org/>*, 04.04.2004.
- [Zha04b] Wensong Zhang. High availability.  
*<http://www.linuxvirtualserver.org/HighAvailability.html>*, 12.05.2004.
- [Zha04c] Wensong Zhang. Ipv6 connection synchronization.  
*<http://www.linuxvirtualserver.org/docs/sync.html>*, 14.06.2004.
- [Zsc04] Oliver Zschau. Contentmanager.de das content management portal.  
*<http://www.contentmanager.de>*, 12.07.2004.

# Index

- ARP, 25, 41
- ARP-Problem, 25, 26, 38–40, 42
- ARP-Tabelle, 40
- booby traps, 78
- Brute Force, 59
- Checkliste, 8
- Cluster, 11
- Content Management, 69, 73, 75, 86
- cron, 75, 85
- Data Frame, 25
- Destination Hashing, 29
- Direct Routing, 24, 27, 30, 33, 37, 40, 46
- DNS, 22
- DOS, 83
- eStudy, 19
- failover, 72
- Firewall, 78
- Full-NAT, 22
- Google, 11
- Hardware, 11
- Heartbeat, 55
- heartbeat, 82
- Hochverfügbarkeit, 7
- honey pods, 78
- HTTP, 15, 21, 33, 58, 61, 77
- HTTPS, 37, 60, 66, 77
- ICMP, 31
- ICMP-Redirect, 31
- IP-Tunneling, 26, 35, 38, 40, 46
- iptables, 39, 41, 83
- Lasttest, 8
- ldirectord, 58, 65
- Least-Connection, 28, 46, 47, 53
- Leistungsspitze, 20
- Loadbalancer, 11
- Locality Based Least-Connection, 28
- Locality Based Least-Connection with Replication, 29
- MAC-Adresse, 24, 40
- Mon, 58, 61, 65, 71, 72, 82
- MySQL, 68
- mysqldump, 68
- NAT, 22
- netstat, 81
- Network Address Translation, 22, 31, 37
- Never Queue, 29
- nmap, 80
- Open-Source, 6, 7
- OSI, 21, 24
- Paketfilter, 11, 83
- Performance, 8
- PHP-Nuke, 86
- Port remapping, 27, 36
- Portscan, 80
- Prerouting, 41
- Quality Assurance, 71
- Realserver, 11

---

Redirects, 34  
Redundanz, 11  
Replication, 68  
Round Robin, 22, 28, 46  
Runlevel, 58

Schlangenöl, 78  
Service-Level-Agreement, 54  
Shortest Expected Delay, 29  
Sicherheit, 8  
Sieben-Sekunden-Fenster, 6  
Single Point of Failure, 66, 69  
snake oil, 78  
Socket, 30  
Source Hashing, 29  
SSH, 22, 30  
Stresstest, 46

Transparenter Proxy, 42  
Typo3, 6

Ultra Monkey, 91

Watchdog, 56  
Webmin, 60  
Weighted Least-Connection, 28  
Weighted Round Robin, 28, 30, 46, 47,  
52

xnmap, 80

Yahoo, 11

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Gießen, den 29. Juli 2004

Tim Pommerening