JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Designstudie für Hochverfügbarkeit und Lastausgleich bei Firewalls

am Beispiel von Gibraltar

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

in der Studienrichtung

INFORMATIK

Eingereicht von:
Alexander Stieglecker, 9956289

Angefertigt am:
Institut für Informationsverarbeitung und Mikroprozessortechnik

Betreuung:
o.Prof. Dr. Jörg R. Mühlbacher

Linz, 16. März 2006

DANKSAGUNG

Für die Unterstützung bei der Erstellung dieser Arbeit möchte ich mich auf diesem Wege besonders bei Herrn Oberrat Dipl.-Ing. Rudolf Hörmanseder bedanken. Er gab in zahlreichen Diskussionen über das Thema immer wieder neue Denkanstöße, die mir bei der Entwicklung der praktischen Arbeit weiter halfen. Ferner sorgte er für die unkomplizierte Bereitstellung der verwendeten Testhardware im institutseigenen Netzwerklabor.

Dipl.-Ing. Dr. Rene Mayrhofer unterstützte mich tatkräftig bei der Portierung der Software für Gibraltar v2.3.

Die Firma eSYS sorgte laufend für Lizenzen für die Tests zur Portierung der Software unter Gibraltar.

Auch bei Herrn Christian Zeilinger möchte ich mich für das Korrekturlesen dieser Arbeit bedanken.

Besonderer Dank gilt meinen Eltern, von denen ich nicht nur bei dieser Magisterarbeit, sondern schon während des gesamten Studiums sowohl finanzielle, als auch moralische Unterstützung erhielt.

KURZFASSUNG

Diese Arbeit untersucht Möglichkeiten, um Linux Firewalls zum einen ausfallssicherer, zum anderen performanter zu gestalten. Um Hochverfügbarkeit zu erreichen, werden Systeme zumeist mit redundanten Komponenten ausgerüstet. Beim Ausfall des laufenden Systems übernimmt, das welches sich bis dahin im Standby Modus befand.

Es wird besonderes Augenmerk auf Lösungen gelegt, die ohne zentralen Load Balancer auskommen. Der Verkehr gelangt an alle Knoten und diese entscheiden verteilt, wer welche Teile des Verkehrs annimmt.

In der Arbeit werden eigene Lösungsansätze für Load Balancing mit Hochverfügbarkeit von Firewalls vorgestellt und deren Vor- und Nachteile diskutiert. Der beste Lösungsansatz wurde implementiert, sodass er in einer bestehenden Firewall Anwendung finden kann. Architektur und Arbeitsweise dieses Ansatzes werden in Kapitel 5 erläutert und dokumentiert. Da eine Verwendung im kommerziellen Bereich geplant ist, erfolgen in Kapitel 6 Hinweise für die Integration in Gibraltar, einer Firewall für Linux. Weiters werden Testumgebungen für Funktions- und Leistungstests dokumentiert und deren Ergebnisse präsentiert.

ABSTRACT

This thesis searches for possibilities to increase high availability (HA) as well as performance for Linux firewalls. To gain HA most systems are equipped with redundant components. If one of them fails, the other one which has been in standbymode takes over any lost resources.

The author focuses on solutions without any load balancer in front of the cluster. Incoming traffic is distributed between the nodes, which come to a distributed decision about responsibility for each packet.

The thesis gives a detailed explanation of some load balancing and HA concepts for firewalls, designed by the author. The discussion gives insights into advantages and disadvantages of each system. One concept has been implemented to fit requirements for real-live-applications. The design of this implementation is documented in Chapter 5. Chapter 6 gives practical hints on the integration of modules and scripts into the commercial firewall Gibraltar. Finally, test-environments for function and performance tests are documented and their results are presented.

1 Motivation

Die Datenverarbeitung auf elektronischem Weg ist sowohl im privaten Bereich, als auch in der Geschäftswelt zu einem unverzichtbaren Instrument geworden. Heutige Geschäftsprozesse in Unternehmen laufen vielfach mit IT-Unterstützung ab. In gleichem Maß, wie elektronische Datenverarbeitung hilft, die Produktivität zu steigern, wird die automatisierte Verarbeitung auch unverzichtbar. Ein Ausfall eines Computersystems in einem Unternehmen hat zumeist hohe finanzielle Einbußen zur Folge und kann – im Extremfall – zur Zahlungsunfähigkeit führen. Ein Beispiel dafür ist der Bankrott von 145 der 450 im New Yorker World Trade Center ansässigen Firmen [BrJa02] nach dem Bombenattentat 1993. Die verwendete IT-Struktur war nicht auf einen Ausfall ausgelegt. Die daraus entstandenen Kosten und der erlittene Imageverlust bei den Kunden trieben mehrere Unternehmen in die Insolvenz.

Genauso wichtig wie Hochverfügbarkeit ist auch die Datensicherheit. Die Sicherung sensibler Daten vor unbefugtem Zugriff und Zerstörung ist Auftrag jeder Organisation. Imageverlust und Kosten können dabei ebenfalls ins Unermessliche steigen. Obwohl die Firewall nicht das einzige Instrument für Datenschutz darstellt, so ist sie doch ein wichtiges Mittel, das Unternehmen vor unbefugtem Zugriff zu schützen. Vernachlässigung von Sicherheit und Verfügbarkeit sind also zwei Dinge, die nichtkalkulierbare Risiken verursachen.

Um Hochverfügbarkeit zu erreichen werden meist redundante Systeme gebaut. Ein System arbeitet, das andere befindet sich im Zustand „Standby“. Bei Ausfall des arbeitenden Systems übernimmt das andere. Während der gesamten Betriebszeit sind immer zwei Systeme im Einsatz, von denen aber nur eines produktiv arbeitet.

Ein anderer Ansatz ist, beide Systeme arbeiten zu lassen. Die Performance des HA-Systems ist also bis zu doppelt so hoch wie bei einer Standby Lösung. Bei einem Ausfall eines Systems übernimmt das jeweils andere die gesamte Arbeit bis zur Reparatur des beschädigten. Dies stellt einen Kompromiss zwischen Performance und Hochverfügbarkeit dar. Im Folgenden wird die Verbindung dieser beiden Themen beleuchtet. Als Ausgangsbasis dient dabei eine Arbeit über Hochverfügbarkeit unter Linux [RoSt04].

Auf Grund der guten Reputation von Linux im Hinblick auf Stabilität wurde in dieser Arbeit ein starker Fokus auf dieses Betriebssystem gesetzt. Das erarbeitete Konzept ist aber auch auf andere Systeme übertragbar.

Zunächst wird ein Einblick in grundlegende Konzepte gegeben, die zum weiteren Verständnis der Thematik benötigt werden. Als nächstes erfolgt eine Erläuterung verwandter Systeme – vorzugsweise unter Linux – gefolgt von einer Diskussion verschiedener Lösungsansätze für eine redundante Linux Firewall mit Load Balancing. Dokumentation der Module und Hinweise zur Integration in die kommerzielle Linux Firewall Gibraltar werden in Kapitel 6 geliefert. Abschließend werden in Kapitel 7 die Ergebnisse der Performancetests präsentiert.

2 Grundlegende Konzept von Linux Firewalls

Um Redundanz und Lastausgleich für Linux Firewalls zu erreichen muss zunächst eine geeignete Plattform ausgewählt werden. Unter Linux fiel die Wahl zwangsläufig auf IP-Tables, da dieses Projekt für Linux Firewalls fast überall eingesetzt wird. Daher werden als Grundlage für die weiteren Kapitel hier interne Strukturen des Netfilter Frameworks mit IPTables erläutert. Dabei wird besonderes Gewicht auf die Verarbeitung des Verkehrs gelegt, der die Firewall betritt und auch wieder verlässt, da dieser Verarbeitungsteil redundant gehalten werden muss.

2.1 Netfilter

Netfilter stellt ein Basisframework für Kernelmodule dar, die Filterung bzw. Veränderung von Paketen durchführen sollen. Beim Vorbeikommen von Paketen an wichtigen Punkten in der Paketverarbeitung des Kernels, werden die Kernelmodule von Netfilter informiert. Die Module können dann über die „Zukunft“ des Paketes entscheiden, indem sie es verändern oder löschen. Unter anderem bilden Teile von IPTables eine Gruppe von Modulen die auf Netfilter zurückgreifen.

Im Prinzip besteht Netfilter aus einer Reihe von Makros [Wil01] namens „NF_HOOK“, die an bestimmten Stellen im Paketverarbeitungscode von Linux eingefügt sind [Mau04]. Wird der Kernel ohne Option „CONFIG_NETFILTER“ übersetzt, so ist NF_HOOK als leeres Makro definiert und Netfilter greift nicht in die Paketverarbeitung von Linux ein. Wurde aber CONFIG_NETFILTER gesetzt, so wird das Paket in einem Funktionsaufruf an das Netfilter-Framework weitergeleitet. Netfilter führt dann Callback-Funktionen von anderen Kernel Modulen aus die bestimmen, ob das Paket unverändert bleiben, verändert, oder gelöscht werden soll. Ein Beispiel für den Aufruf eines Hook-Makros befindet sich in der Kerneldatei ip_input.c:

```
int ip_local_deliver(struct sk_buff *skb)
{
      .
      .
      .
      return NF_HOOK(PF_INET, NF_IP_LOCAL_IN, skb, skb->dev, NULL,
                     ip_local_deliver_finish);
}
```

Abbildung 1: Aufruf von NF_HOOK bei ankommenden Paketen [We00].

Jedes dieser Makros in der Paketverarbeitung wird als „Hook“ bezeichnet. Abbildung 2 zeigt die Position dieser Hooks während der „Reise“ eines Paketes durch den IP-Stack.

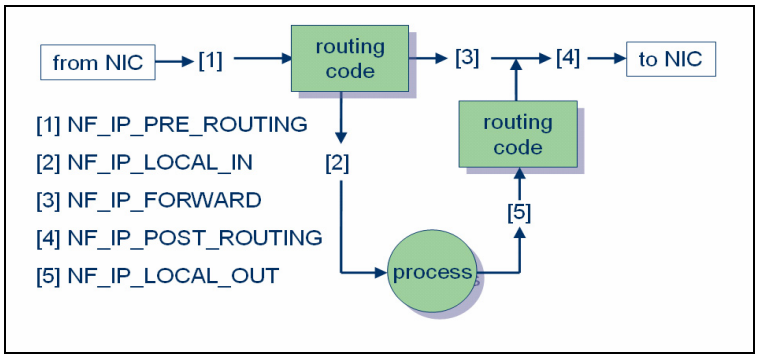

Abbildung 2: Die Netfilter Hooks [RuWe02].

Kommt das Paket von einem Netzwerk Interface, so wird zuerst NF_HOOK mit Parameter „NF_IP_PRE_ROUTING“ aufgerufen. Kehrt das Paket von der Verarbeitung durch Netfilter zurück, so wird durch den Routing Code von Linux entschieden, ob es an diesen lokalen Prozess zugestellt werden, oder über ein Netzwerk Interface nach außen weitergeleitet werden soll. Wenn das Paket für einen lokalen Prozess bestimmt ist, so wird NF_HOOK mit Parameter „NF_IP_LOCAL_IN“ aufgerufen (siehe Abbildung 1). Wenn es aber laut Routingtabelle an ein Interface gesendet werden soll, so wird der Hook „NF_IP_FORWARD“ aktiviert und es kann erneut eine Veränderung oder Löschung des Pakets durchgeführt werden. Zuletzt können die Pakete noch am Hook „NF_IP_POST_ROUTING“ inspiziert und verändert werden. Lokal generierte Pakete gehen den Weg: NF_LOCAL_OUT; Routing Code; NF_IP_POST_ROUTING.

Wie schon erwähnt, können sich Kernel Module bei Netfilter registrieren, um Callback-Funktionen aufrufen zu lassen, falls ein Paket bei einem bestimmten Hook angelangt ist. Die Funktion erhält in den Eingangsparametern einen Zeiger auf das Paket mit Hilfe dessen sie es inspizieren kann. Nach der Inspektion muss die Funktion eines von fünf „Urteilen“ fällen und die Entscheidung in Form eines Rückgabewertes Netfilter bekannt geben:

- NF_ACCEPT: Fahre mit der Verarbeitung im IP-Stack normal fort.
- NF_DROP: Lösche das Paket und beende so die Verarbeitung des Pakets.
- NF_STOLEN: Das Modul verarbeitet das Paket selbst.
- NF_QUEUE: Das Paket wurde an den Userspace weitergegeben
- NF_REPEAT: Wiederhole den Makroaufruf für dieses Paket.

Aufbauend auf dieses Gerüst kann eine sehr flexible Paketverarbeitung durch diverse Module erfolgen. In IPTables findet sich die bekannteste Gruppe von Modulen, die Gebrauch von Netfilter macht. Es gibt aber auch andere für Paketfilterung wie z.B. nf-HIPAC [Be05].

2.2 IPTables aus Benutzersicht

Mit IPTables ist es möglich, Pakete zu filtern und zu verändern. Es klinkt sich in Netfilter an den oben beschriebenen Hooks ein und unterstützt sowohl Stateless als auch Stateful[1] Regeln. Wie der Name schon sagt, besitzt IPTables verschiedene Tabellen, die für unterschiedliche Aufgaben bestimmt sind. Exemplarisch wird hier die Filtertabelle beschrieben. In ihr können Pakete nach bestimmten Kriterien gelöscht oder akzeptiert werden. Die Filtertabelle betrifft die Hooks NF_IP_LOCAL_IN, NF_IP_LOCAL_OUT und NF_IP_FORWARD. In iptables_filter.c wird jedem dieser Hooks eine Kette von Regeln zugeordnet. Die INPUT-Chain ist NF_IP_LOCAL_IN, die OUTPUT-Chain NF_IP_LOCAL_OUT und die FORWARD-Chain NF_IP_FORWARD zugeordnet[2]. Jede Chain ist eine lineare Liste von Regeln. Kommt ein Paket bei NF_IP_FORWARD vorbei, so werden alle Regeln der FORWARD-Chain abgearbeitet, bis eine davon auf das Paket passt. Die Regel bestimmt, ob das Paket gelöscht oder beibehalten und ob die Abarbeitung der Chain fortgesetzt werden soll. Passt keine Regel, so wird am Ende der Chain die Standardaktion (z.B. Löschen des Paketes) ausgeführt.
Regeln bestehen immer aus zwei Teilen. Der erste Teil bestimmt ob ein Paket auf die Regel passt. Er wird „Match“ genannt. Der zweite Teil bestimmt, was mit dem Paket geschehen soll, nachdem die Regel darauf passte.

[1] Unter Verwendung des Moduls ip_conntrack

[2] Es können auch benutzerdefinierte Chains erstellt werden, auf die in der Abarbeitung der Standard Chains verzweigt werden kann.

Matches in IPTables

Um eine Grundfunktionalität zu bieten, werden in IPTables Standard Matches angeboten. Dazu gehört z.B. die Angabe von Source/Destination IP-Adressen, Protokollen oder Interfaces. Daneben können aber auch noch Match Erweiterungen (Match Extensions) angewendet werden. Eine mögliche Match Extension ist die Angabe einer Portnummer, auf die die Pakete untersucht werden sollen.

	Standard Match	Match Extension
1	Src. Addr.: 10.0.1.0/24 Dst. Addr.: 10.0.0.0/24 Proto: TCP	Dst Port: 80 TTL: <3
2	Src. Addr.: 10.0.1.0/24 Dst. Addr.: 10.0.0.0/24 Input Interface: InternalNIC	

Tabelle 1: Mögliche Ausprägungen von Matches unter IPTables

Targets unter IPTables

Genau wie bei den Matches gibt es auch bei den Targets Standard Targets und Target Erweiterungen (Target Extensions). Die wichtigsten Standard Targets sind:

- ACCEPT: Lasse das Paket durch den Filter und breche die Abarbeitung der Chain bei der aktuellen Regel ab.
- DROP: Lösche das Paket und breche die Abarbeitung der Chain bei der aktuellen Regel ab.

Eine Target Extension ist z.B. das LOG Target. Es bestimmt, dass der Header des Paketes an einen Loggingprozess geschickt wird, wo er in eine Logdatei geschrieben wird. Nachdem damit Matches und Targets erläutert wurden, können vollständige Regeln erstellt werden:

	Standard Match	Match Ext.	Standard Target	Target Ext.
1	Src. Addr.: 10.0.1.0/24 Dst. Addr.: 10.0.0.0/24 Proto: TCP	Dst. Port: 80 TTL: <3	DROP	
2	Src. Addr.: 10.0.0.0/24 Dst. Addr.: 10.0.1.0/24 In-Iface.: InternalNIC			LOG

Tabelle 2: Beispiel für Einträge in der FOWARD Chain

Die erste Regel löscht alle TCP-Pakete mit Destination Port 80, deren TTL kleiner als drei ist, jedoch nur dann, wenn sie aus dem Subnet 10.0.1.0/24 stammen und für das Subnet 10.0.0.0/24 bestimmt sind. Regel Nummer zwei protokolliert alle Pakete vom Subnet 10.0.0.0/24 ins Subnet 10.0.1.0/24, die vom Interface namens „InternalNIC“ stammen. Um Regel Nummer eins zu erstellen, ist ein Befehl auf der Kommandozeile [Ed02] notwendig:

```
iptables -t filter -A FORWARD -s 10.0.1.0/24 -d 10.0.0.0/24 -p tcp \
         -m tcp --dport 80 -m ttl --ttl-lt 3 -j DROP
```

Abbildung 3: iptables [Ed02] wird für die Eingaben von Firewallregeln benutzt.

IPTables ist erweiterbar und so können sowohl eigene Tabellen, als auch eigene Matches und Targets für Regeln implementiert werden.

Connection Tracking

Um Stateful Inspection betreiben zu können, müssen Matches von Regeln auf eine externe Datenbasis zugreifen können, die den Status jeder Verbindung mitverfolgt. Diese externe Datenbasis stellt das Modul ip_conntrack dar. Es klinkt sich in den Netfilter Hook NF_IP_PRE_ROUTING und in den Hook NF_IP_POST_ROUTING ein, greift dort alle Pakete ab und baut daraus eine Datenbasis von Verbindungen und ihren Zuständen auf. Mögliche Zustände der Verbindungen sind:

- NEW: Die Verbindung wird gerade neu aufgebaut.
- ESTABLISHED: Die Verbindung ist fertig aufgebaut.
- RELATED: Die Verbindung wird gerade neu aufgebaut, gehört aber zu einer anderen Verbindung, die bereits den Zustand ESTABLISHED hat. Ein Beispiel wäre

eine FTP-Data Verbindung, die ja von einer FTP-Command Verbindung initiiert wurde.

Durch die Match Extension „state“ erhält eine Regel Zugriff auf die Einträge in ip_conntrack. Die Eingabe einer Regel mit Match Extension „state“ kann beispielsweise so aussehen:

```
iptables -D FORWARD -p tcp -m state --state ESTABLISHED -j ACCEPT
```

Abbildung 4: Regel, die auf die Verbindungstabelle von ip_conntrack zugreift.

Diese Regel akzeptiert alle Pakete, die einer Verbindung mit Zustand ESTABLISHED zugeordnet sind. Die Zuordnung eines Pakets zu einer Verbindung erfolgt auf Basis von IP-Adressen, Portnummern und Protokollen, die sowohl Paket als auch Verbindung speichern. Pakete, die keiner Verbindung zugeordnet werden können, erhalten den Status INVALID. Connection Tracking funktioniert für die Protokolle TCP, ICMP und UDP. Auch Network Address Translation funktioniert mit Regeln, die auf die Verbindungstabelle von ip_conntrack zugreifen. Dadurch lässt sich bestimmen, wie die Pakete verändert werden sollen. Eine genauere Beschreibung der internen Vorgänge in ip_conntrack ist unter Abschnitt 3.2.2 bzw. unter http://www.netfilter.org/ [RuWe02] zu finden.

3 Existierende HA-Softwarelösungen

Sowohl für HA[3], als auch für Load Balancing (LB) existieren bereits verschiedenste Lösungsansätze, die in diesem Kapitel erläutert werden sollen.

Bei LB ist grundsätzlich zwischen zwei Paradigmen zu unterscheiden:

- Aufteilung des Verkehrs mittels zentralem Load Balancer. Im Folgenden wird dieses Verfahren als „zentrale Methode“ bezeichnet. Ein typischer Vertreter dieser Technik ist Linux Virtual Server [ZhSh04].
- Aufteilung des Verkehrs mittels eines auf allen Knoten verteilt laufenden Entscheidungsalgorithmus (im Folgenden als „dezentrale Methode“ bezeichnet). Der Verkehr wird z.B. mittels Hub auf mehrere Rechner verteilt und diese entscheiden mit einem deterministischen, verteilten Algorithmus welcher Knoten welche Teile des Verkehrs annimmt. Clusterip [Fl05], aber auch das kommerzielle Network Load Balancing von Microsoft [Mi03] stellen Vertreter dieser Technik dar.

Bemerkenswert ist, dass sich zum Zeitpunkt der Recherchen kein freies[4] Load Balancing System für Linux Firewalls (egal ob mit zentraler oder dezentraler Methode) auffinden ließ, jedoch zahlreiche Anfragen in Mailinglisten gefunden wurden. Es scheint hier also Bedarf für Neuentwicklungen gegeben zu sein.

3.1 Clusterip

Bei Clusterip handelt es sich um ein relativ junges dezentrales LB-System, das für Lastausgleich von Anwendungsservern verwendet werden kann. Programmiert von Harald Welte nach der Idee von Fábio Olivé Leite [Le02] wird es mit Linux Kernels ab Version 2.6.10 unter der GNU Public License ausgeliefert. Es befindet sich noch im experimentellen Stadium.

3.1.1 Das Prinzip

Bei Clusterip befinden sich mehrere Server an einen Switch. Alle Server haben die gleiche IP-Adresse.

[3] High Availability

[4] Frei im Sinne der GNU Public License

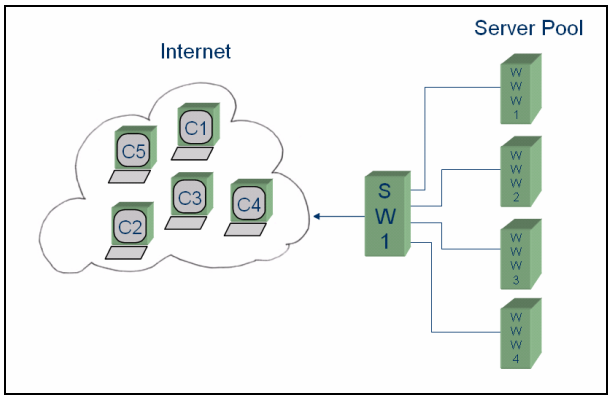

Abbildung 5: Netzwerktopologie von Clusterip.

Erster Schritt ist die Modifikation des OSI-Layers 2, um den Switch in den Flooding Modus zu versetzen. Eine genauere Beschreibung dieser Technik findet sich im Kapitel 4.2.1. An dieser Stelle sei nur bemerkt, dass der Switch durch Modifikation von ARP-Requests oder Replies den ankommenden Verkehr auf alle Knoten WWW1-WWW4 verteilt.
Davon weitgehend unabhängig ist die Arbeitsweise auf OSI-Layer 3. Jeder Knoten erhält eine Ordnungsnummer, anhand der er eindeutig identifiziert ist. Gelangt ein Paket von außen an den Switch, so wird es an alle Knoten verteilt. Jeder Knoten nimmt eine Kopie dieses Paketes an und unterzieht es folgender in Pseudo-Code dargestellten Entscheidung:

```
Hashval := Hash(SrcIP);
Hash := Hashval MODULO NumberOfNodes;
If Hash == LocalNodeNumber Then
      return IPT_CONTINUE;
Else
      return IPT_DROP;
```

Abbildung 6: Verteilte Entscheidung über Annahme von Paketen

Von der Source IP-Adresse des ankommenden Pakets wird zunächst ein Hashwert gebildet, um möglichst unabhängig von der Verteilung der Bits der Source IP-Adresse zu sein. Die Bedingung des IF-Konstruktes ist stets auf nur einem Knoten TRUE und so wird das Paket nur auf diesem Knoten angenommen. Die TCP-Verbindung wird lediglich von einem Knoten gehalten, alle anderen verwerfen die Pakete auf Grund der Source IP-Adresse.

Grenzen von Clusterip

Geht man davon aus, dass alle Client IP-Adressen in etwa gleiche Last verursachen, so erhält man eine gute Aufteilung auf alle Knoten. Befinden sich aber mehrere Clients hinter

einem Gateway, das Network Address Translation betreibt, oder hinter einem Proxy, so besitzen diese – zumindest aus der Sicht des Clusters – alle die gleiche IP-Adresse. Als Folge läuft die Kommunikation nur über einen Knoten. Der Speedup wird vollständig zunichte gemacht. Für diesen Fall kann man bei der Bildung des Hashwertes auch noch den Source Port mit berücksichtigen. Dieser variiert ja selbst bei NAT[5].

Ein weiteres Problem stellen Anwendungsprotokolle dar, die mehrere TCP-Verbindungen in unterschiedliche Richtungen gleichzeitig verwenden. Ein bekanntes Beispiel ist Active-FTP [Ie89a]. Bei Active-FTP bestehen zeitweise zwei TCP-Verbindungen in jeweils entgegen gesetzter Richtung. Eine wird von Client zum Server, später eine vom Server zum Client aufgebaut.

Bildet man den Hash nur über die Source IP-Adresse ohne Port, so lässt sich dieses Problem vermeiden.

Auch in Punkto HA kann Clusterip mit einem Konzept aufwarten. Die in Abbildung 5 beschriebenen vier Knoten sind für jeweils ein Viertel des gesamten Adressraums zuständig. Fällt ein Knoten aus, so besteht die Möglichkeit, den ausgefallenen Pool an Client-Adressen von einem anderen Knoten übernehmen zu lassen. Dieser Knoten muss folglich aber mit einer höheren Last rechnen. Nicht Teil von Clusterip ist eine Überwachung der Funktion der Knoten und ein Ressource Management, falls ein Knoten ausfällt. Hierfür kann aber z.B. die Software Heartbeat in Version 2.0 eingesetzt werden (siehe Abschnitt 3.6).

[5] Network Address Translation

3.1.2 Struktur des Clusterip Targets

Da Clusterip Pakete filtern muss, bietet sich die Verwendung der Linux Firewall IPTables an. Clusterip soll nur für Pakete, die für lokale Prozesse bestimmt sind eine Selektion durchführen. Es ist also sinnvoll, dieses Target in der INPUT-Chain zu verwenden.

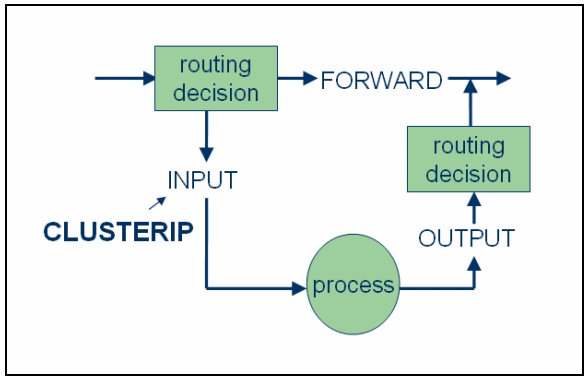

Abbildung 7: Schnittstelle zu Clusterip.

Um eine Regel in genau diese Chain zu hängen ist IPTables mit folgenden Parametern notwendig:

```
# iptables -A INPUT -d 10.0.0.1 -j CLUSTERIP \
--new --hashmode sourceip \
--clustermac 01:10:47:7D:94:94 \
--total-nodes 4 \
--local-node 1
```

Abbildung 8: Beispiel einer Clusterip-Regel [Fl05].

Die Regel in Abbildung 8 zeigt die IP-Adresse des Clusters 10.0.0.1. Der Parameter „new“ besagt, dass eine neue Regel erstellt und keine bestehende verändert werden soll. Hier wird der Hashwert nur durch die Source IP-Adresse gebildet. Um den Switch in den Flooding Modus zu bringen, muss eine passende Multicast MAC-Adresse verwendet werden, die mit Parameter „Clustermac“ übergeben wird (mehr dazu in Abschnitt 4.2.1). Als letztes werden Anzahl und Index der Knoten festgelegt. Die Anzahl der Knoten ist auf allen Knoten mit der gleichen Zahl anzugeben. Der lokale Index variiert von Knoten zu Knoten.
Diese Regel wirkt als Filter für eine bestimmte Menge von Client IP-Adressen. Wie soll aber einem Knoten mitgeteilt werden, dass er fortan alle Requests für einen ausgefallen übernehmen soll? Immerhin befinden sich die Daten „local-node“ und „total-nodes“ im Kernelspace, von dem aus sich mit dem Userspace nicht ohne weiteres kommunizieren lässt. Es besteht die Möglichkeit, die Regel über das Process Filesystem zu verändern. Ü-

ber das Process Filesystem kann auf einfache Weise eine Userspace-Kernelspace Kommunikation durchgeführt werden.
Der Befehl:

```
# cat /proc/net/10.0.0.1
1
```

fragt über das Process Filesystem die Nummern der lokal verwalteten virtuellen Knoten ab. In diesem Fall handelt es sich um Knoten 1, wie der Output des Kommandos zeigt.
Genauso kann man einen virtuellen Knoten lokal hinzufügen:

```
# echo "+2" >> /proc/net/10.0.0.1
```

Hier wurde der virtuelle Knoten 2 lokal hinzugefügt. Vom Zeitpunkt des Befehls an verwaltet der lokale Knoten auch jene Client IP-Adressen des Knotens 2. Um den virtuellen Knoten 2 wieder der lokalen Verwaltung zu entziehen, genügt der gleiche Befehl mit "-2" als Parameter.
Es ist Aufgabe eines HA-Systems, wie es z.B. Heartbeat 2.0 darstellt, die oben genannten Befehle auszuführen, wenn es zum Ausfall eines Knotens kommt. Heartbeat 2.0 bietet eine entsprechende Konfigurationsmöglichkeit für mehrere Knoten an.
Clusterip ist die erste freie Lösung für LB Systeme, die ohne zentralen Load Balancer auskommt. Zusätzlich bietet sie Schnittstellen, um HA Systeme anzubinden. Auch für Firewalls auf denen Server wie z.B. Proxies laufen, ist sie gut geeignet. Clusterip ist aber - auf Grund der Konzeption - nicht für Stateful Packet Filtering in Firewalls zu verwenden.

3.2 Netfilter-HA

3.2.1 Das Prinzip von Netfilter-HA

Nach einem Vorschlag von Harald Welte [We02], eine redundante Lösung für Netfilter zu entwerfen, wurde das Netfilter-HA Projekt ins Leben gerufen. Es ist ein Projekt, welches den redundanten Betrieb von zwei IPTables Firewall Knoten erlaubt. Wie bereits in Abschnitt 2.2 beschrieben, stellt Netfilter in Verbindung mit IPTables eine Stateful Firewall dar. Je nach Art und Anzahl der durch die Firewall aufgebauten Verbindungen ändert sich

der Zustand der Firewall. Netfilter-HA versucht laufend diesen Zustand an ein Backupsystem zu senden.

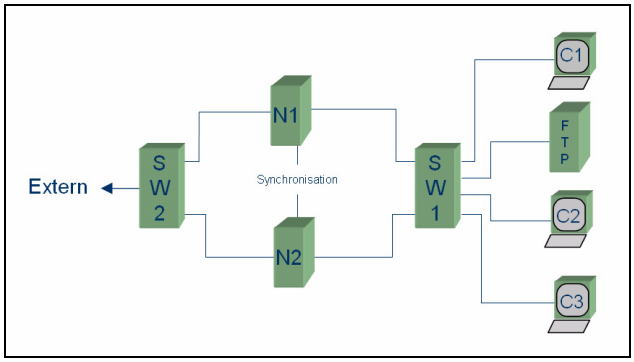

Abbildung 9: Netfilter-HA Topologie.

Dieses kann im Fehlerfall des aktiven Systems sofort alle Verbindungen übernehmen. Aus Client- und Serversicht geht der Betrieb also ohne Unterbrechung weiter. Für Load Balancing bietet Netfilter-HA eine so genannte Active/Active Variante an. Mit ihr ist ein Betrieb von zwei Knoten gleichzeitig im Sinne von Load Balancing möglich. Das gesamte Projekt ist allerdings noch im Alpha Stadium und gewährt dadurch noch keinen stabilen Betrieb.

3.2.2 Funktionsweise von Netfilter-HA

Um einen Einblick in die Funktion von Netfilter-HA zu erlangen, muss die Arbeitsweise von Connection Tracking unter Netfilter klar sein. Es wird daher ein kurzer Einblick in die Funktionsweise des Connection Tracking gegeben.

Connection Tracking mit Netfilter

Das Kernel Modul ip_conntrack.o greift alle ankommenden Pakete am Netfilter Hook NF_IP_PRE_ROUTING ab. Jedes Mal, wenn ein Paket an diesem Hook vorbeikommt, wird ein Tupel aus diesem Paket extrahiert. In diesem Tupel stehen wichtige Daten wie Source IP, Destination IP, Source- und Destination Port und Protokoll des Pakets. Dieses Tupel wird mit allen anderen Tupeln bereits bestehender Verbindungen verglichen. Zur besseren Verständlichkeit wird zunächst angenommen, dass das Paket eine neue Verbindung aufbaut. Somit wird kein bestehendes Tupel mit gleichen Attributen im Tupelhash gefunden.

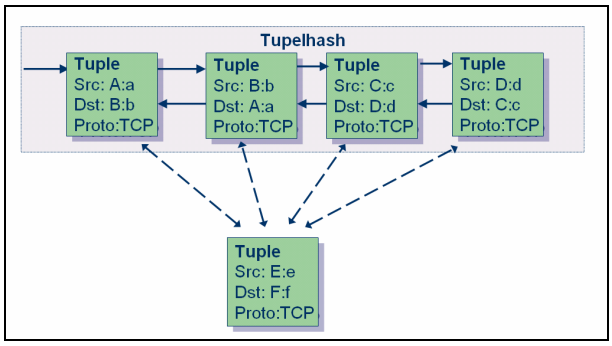

Abbildung 10: Suche nach bestehenden Tupeln.

Daraufhin wird eine Datenstruktur ip_conntrack erzeugt. Sie repräsentiert eine neue Verbindung und enthält alle dafür wichtigen Daten wie z.B. ein Tupel in Original- und Antwortrichtung. Das Tupel für die Antwortrichtung wird durch Vertauschen von Source IP:Port mit Destination IP:Port erzeugt.

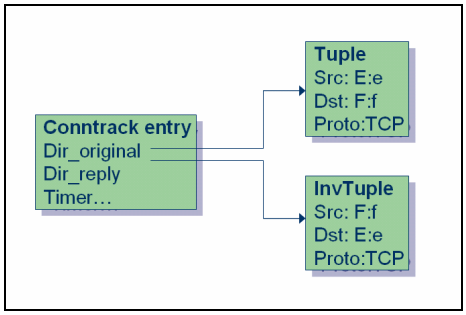

Abbildung 11: Conntrack Struktur.

Das Paket, das diese Vorgänge verursacht hat, wandert durch die Filter der Firewall hindurch. Falls es dabei verworfen wird, werden auch die in Abbildung 11 dargestellten Datenstrukturen wieder gelöscht. Verlässt es aber die Firewall, so wird die besagte Datenstruktur in den Tupelhash eingehängt und künftige Pakete dieser Verbindung werden diesem Eintrag zugeordnet.

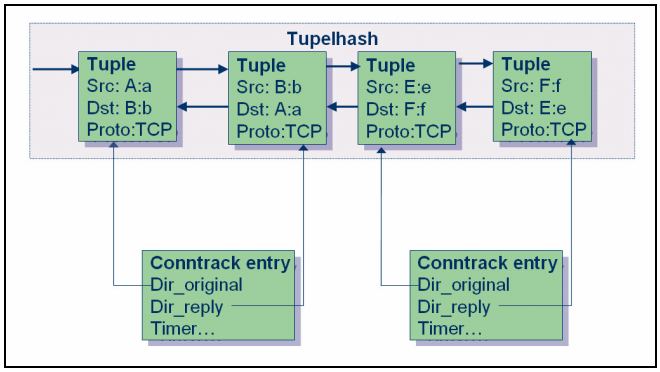

Abbildung 12: Der Tupelhash mit Conntrack Strukturen.

Der beschriebene Mechanismus wird für verschiedene Protokolle wie z.B. TCP oder UDP verwendet. Die dabei entstehende Datenstruktur wie sie in Abbildung 12 dargestellt ist, unterliegt natürlich dem gleichzeitigen Zugriff mehrerer Prozesse und muss daher mittels Semaphoren, bzw. beim SMP-Kernel[6] mittels Spinlocks, gesichert werden. Diese Sperren sollten aus Gründen der Performance jeweils nur wenige Millisekunden aufrecht sein.

Struktur von Netfilter-HA

Um nahtlose Stateful Inspection zu garantieren muss, beim Ausfall des aktiven Knoten, die Verbindungstabelle auch am Backupsystem vorhanden sein. Eine Übertragung der gesamten Verbindungstabelle erfordert eine Lesesperre des ganzen Tupelhashes inklusive der Conntrack-Objekte am aktiven Rechner. Da die Übertragungszeit, selbst bei schneller Netzwerkverbindung zum Backupknoten, weit mehr als nur wenige Millisekunden beträgt, können immer nur kleine Teile der Datenstruktur übertragen werden. Es handelt sich dabei um so genannte Update-Nachrichten. Dies birgt jedoch die Gefahr der Inkonsistenz der Datenstruktur auf dem Backupsystem. Bei hoher Last kann nicht garantiert werden, dass alle Conntrack Einträge sofort auf dem Backup-System vorhanden sind. Geht man aber davon aus, dass wichtige Verbindungen längere Zeit hindurch bestehen, so können zumindest diese mit der Standby Firewall synchronisiert werden.
Im Wesentlichen besteht Netfilter-HA aus dem Kernel Modul ct_sync.o. Aus Performancegründen übernimmt es sowohl Empfang als auch Versand von Update-Nachrichten zur Standby Firewall.

[6] Kernel mit Symmetric Multi Processing Unterstützung

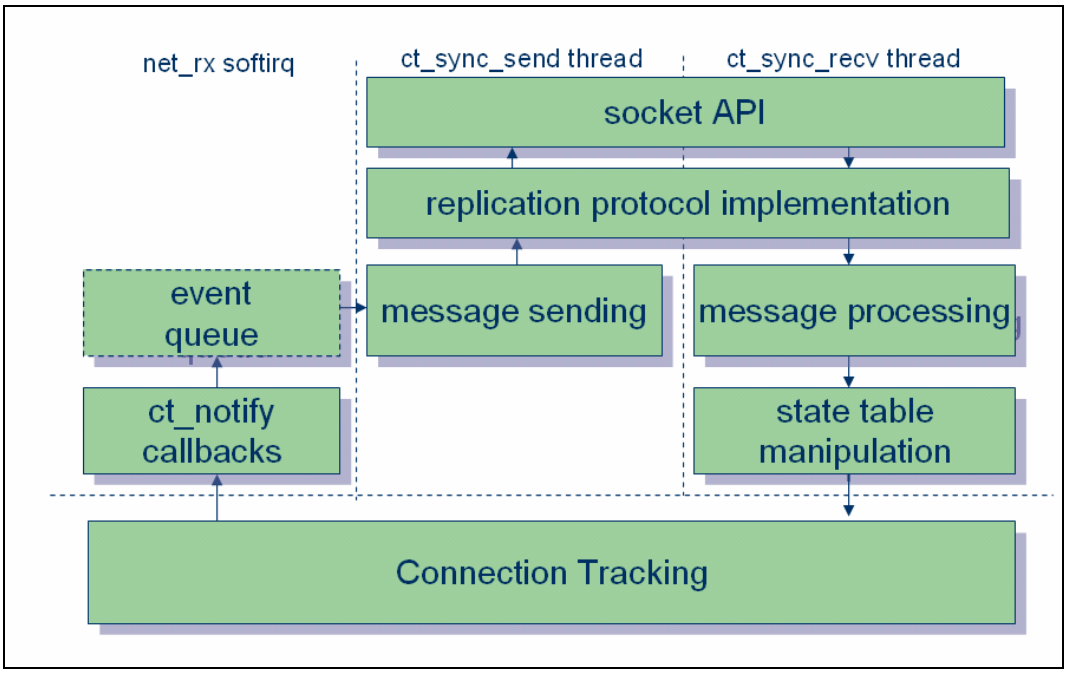

Abbildung 13: Interne Struktur von ct_sync des Netfilter-HA Projekts [We05].

Das ct_sync Modul interagiert mit ip_conntrack größtenteils über Callback-Funktionen. Diese informieren ct_sync über Updates der Datenstrukturen in ip_contrack. Diese Updates werden in einer Queue zwischengespeichert und an eine Standby Firewall über Netsockets gesendet.

Bis jetzt funktioniert normale Stateful Packet Inspection sowie Source-NAT. Auf Grund des guten Ansatzes, die Verbindungstabelle zu übertragen ist zu vermuten, dass diese Lösung in Zukunft einige andere Load Balancing Systeme für Firewalls ablöst. Bis dies der Fall sein kann, muss Netfilter-HA jedoch das Beta-Stadium der Entwicklung verlassen haben. Unter dem Betriebssystem OpenBSD (http://www.openbsd.org/) existiert bereits eine Lösung namens pfsync, welche die oben beschriebene Funktionalität von redundanten Verbindungstabellen realisiert. Pfsync (http://www.openbsd.org/faq/pf/carp.html) ist stabil genug um im Alltagsbetrieb eingesetzt werden zu können. Netfilter-HA wurde von dieser Entwicklung inspiriert.

3.3 Linux Virtual Server

Linux Virtual Server (LVS) wird für Lastausgleich mit Hochverfügbarkeit im Serverbereich eingesetzt. Um verschiedene Anforderungen zu erfüllen, kann LVS in drei Betriebsarten verwendet werden. In jedem Fall verfolgt es das Paradigma von zentralem Load Balancing, also einem vorgeschalteten Redirector.

3.3.1 Linux Virtual Server mit NAT (LVS/NAT)

Abbildung 14 zeigt die Netzwerktopologie von LVS/NAT. Der Load Balancer leitet eingehende Requests nach einem Scheduling Algorithmus durch Destination-NAT an unterschiedliche Server weiter. Rückantworten von den Servern werden auf dem Weg durch den Load Balancer wieder mit den originalen IP-Adressen versehen.

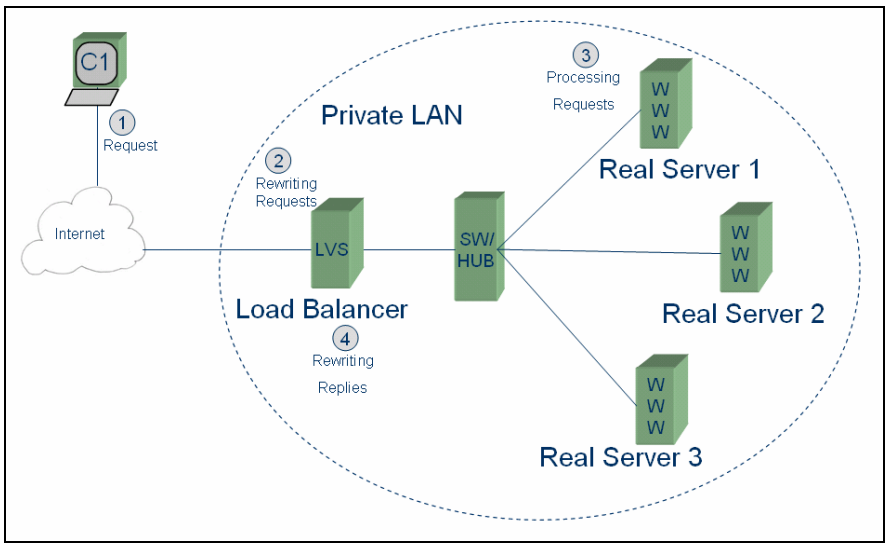

Abbildung 14: Netzwerktopologie von Linux Virtual Server mit NAT [Zh00].

Dadurch können sich die Server in einem privaten Netzwerk befinden und die interne Netzwerkstruktur wird gegenüber außen versteckt. Bei dieser Methode besteht sowohl für die Server als auch für die Clients Transparenz. Das heißt auf den Servern kann jedes beliebige Betriebssystem und jeder beliebige Server-Typ laufen. Eine Limitierung besteht jedoch in der Anzahl von möglichen Servern. Es müssen sowohl Pakete in Client-Server als auch in Server-Client Richtung durch den Load Balancer verändert werden. Durch diesen Verarbeitungsaufwand skaliert diese Lösung nur bis zu einem Maximum von etwa 20 Servern [Zh00].

3.3.2 Linux Virtual Server mit IP Tunneling (LVS/TUN)

Bei LVS/TUN können sich Load Balancer und Server irgendwo im Internet befinden. Zwischen Load Balancer und Servern kann ein WAN liegen und es könnte auch ohne Umweg über den Load Balancer Verbindung mit den Servern aufgenommen werden.

Clientseitige Pakete von Requests gehen zum Load Balancer, der sie mittels IP-over-IP [Ie96] wiederum in IP-Pakete verpackt. Diese werden mittels Scheduling Algorithmus an

einen der Server geschickt. Im Unterschied zu LVS/NAT antworten hier aber die Server mit ihren Requests nicht über den Load Balancer, sondern schicken die Antwortpakete direkt an den Client ohne IP-over-IP zurück.
Üblicherweise sind Requests von Clients kleiner als die Responses der Server. Es ist also davon aus zu gehen, dass der Großteil der Pakete zwischen Server und Client ohne Umweg über den Load Balancer versendet wird (all jene Pakete in Server-Client Richtung). Darum ist die Skalierbarkeit höher als bei LVS/NAT. Es können bis zu 100 Server betrieben werden. Einzige Vorraussetzung für die Funktion ist die Unterstützung von IP-over-IP am Server.

3.3.3 Linux Virtual Server mit Direct Routing (LVS/DR)

Wie bei LVS/NAT ist auch hier gefordert, dass sich sowohl Server als auch Load Balancer in einem LAN-Segment befinden. ARP-Replies müssen in diesem LAN an den Servern unterbunden sein. Load Balancer und Server besitzen die gleiche IP-Adresse.
Erhält der Load Balancer ein Paket von einem Client, so stellt er es nicht einem lokalen Prozess zu, sondern entscheidet auf Grund eines Scheduling Algorithmus, an welchen Server es weitergeleitet werden soll. Ihm ist auch ohne ARP-Replies bekannt, welche MAC-Adresse welcher Server besitzt. Das IP-Paket wird unverändert in ein Ethernet-Frame mit passender Destination MAC-Adresse verpackt, sodass es vom richtigen Server angenommen wird. Eine daraus folgende Antwort des Servers wird direkt und nicht über den Load Balancer an den Client weitergeleitet. Die Antwort besitzt aber die richtige Source IP-Adresse die der Client in der Antwort erwartet.
Diese Lösung kommt ohne die Verwendung des IP-over-IP Protokolls aus, was sich positiv auf die Performance auswirkt. Die Arbeitsweise benötigt aber virtuelle Interfaces und IP-Adressen. Server mit Betriebssystemen, die keine virtuellen Interfaces unterstützen, können daher nicht eingesetzt werden. Die Skalierbarkeit von LVS/DR ist mit der von LVS/TUN vergleichbar.

3.3.4 Grenzen von LVS

LVS ist eine gute Lösung für Anwendungen, bei denen die Server identischen Inhalt anbieten. Es bietet eine hohe Verfügbarkeit und der Single Point of Failure, den der Load Balancer darstellt, kann durch einen zweiten Load Balancer im Standby Modus eliminiert wer-

den. Übernimmt jedoch der zweite, so gehen alle aufgebauten Verbindungen verloren. Für den Einsatz von Load Balancing bei Firewalls lässt sich LVS nicht gebrauchen, da es nur die Requests zu einer Cluster IP-Adresse auf die Server aufteilt. Ganze Subnetze können nicht verwaltet werden.

3.4 Hochverfügbarkeit

Unter Hochverfügbarkeit oder High Availability (HA) versteht man Systeme die durch ihr Design außerordentlich geringe Ausfallzeiten ermöglichen.
Rein rechnerisch lässt sich die Verfügbarkeit (engl.: availability) folgendermaßen beschreiben:

$$A = \frac{MTBF}{MTBF + MTTR}$$

MTBF[7] ist der Kehrwert der Fehlerrate. Hat also ein System innerhalb einer Million Betriebsstunden 2 Ausfälle, so beträgt die MTBF 1.000.000h/2=500.000h. Sie ist nicht zu verwechseln mit der MTTF[8]. Es ist die durchschnittliche Zeit bis zum ersten Ausfall eines Systems und wird vorzugsweise bei nicht reparierbaren Bauteilen wie z.B. Glühbirnen angegeben.

MTTR[9] ist die durchschnittliche Zeit die vergeht, um ein ausgefallenes System wieder zu reparieren.

Wie aus obiger Formel ersichtlich, erhöht sich die Verfügbarkeit A, wenn man MTBF erhöht, da die Erhöhung im Zähler stärker zum Tragen kommt. Bei einer Verringerung der durchschnittlichen Reparaturzeit MTTR steigt A ebenfalls, da der Term im Nenner sinkt. Die folgende Tabelle soll zur Veranschaulichung der Größenordnungen von Ausfallzeiten dienen.

[7] Mean Time Between Failures

[8] Mean Time To Failure

[9] Mean Time To Repair

Prozentuale Verfügbarkeit	Größenordnung	Ausfallszeit pro Jahr
99%	2 Neunen	3,6 Tage
99,9%	3 Neunen	8,76 Stunden
99,99%	4 Neunen	52 Minuten
99,999%	5 Neunen	5 Minuten
99,9999%	6 Neunen	30 Sekunden

Tabelle 3: Veranschaulichung der prozentualen Verfügbarkeit [BrJa02]

Auch ein gutes HA-System kann keine hundertprozentige Verfügbarkeit garantieren. Als Faustregel kann man aber durch den Einsatz von Redundanz eine Verbesserung um eine Neun erlangen [Ro05b]. Mit steigender Verfügbarkeit steigen die Kosten für Bau und Wartung des Systems wie Abbildung 15 zeigt.

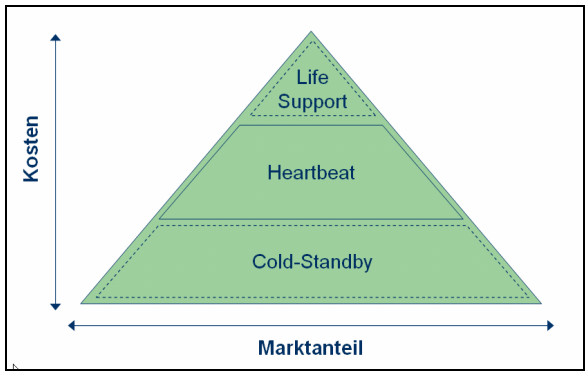

Abbildung 15: Die Heartbeat Pyramide [Ro05b].

Aus der oben dargestellten Formel folgt die Tatsache, dass die Verfügbarkeit auf zwei Arten erhöht werden kann. Zum einen kann MTBF erhöht, zum anderen MTTR verringert werden.

Maßnahmen zur Verringerung der Reparaturzeiten MTTR

Die Reparaturzeit kann auf viele verschiedene Arten minimiert werden. Hier wird nur eine kurze Liste für die Verringerung der MTTR angeführt:

- Komplexität des Systems so gering wie möglich halten.
- Hardware im Cold-Standby, die schnell ausgetauscht werden kann.
- Entwurf von Notfallplänen.
- Gute Dokumentation des Systems für schnelle Behebung des Fehlers.
- Laufende Schulung der Systembetreuer (hilft auch zur Erhöhung der MTBF).

- Schleichende Fehler durch Abschalten und Alarm ersetzen.

In vielen Fällen werden die genannten Punkte vernachlässigt, da sie ein Szenario behandeln, das nach Möglichkeit gar nicht erst auftreten soll. Die Maßnahmen tragen aber wesentlich zur Verbesserung der Verfügbarkeit bei!

Vergrößerung der MTBF:

Hauptaufgabe der meisten HA-Systeme ist es dennoch, einen Ausfall eines Systems zu vermeiden. Dies kann auf verschiedene Arten erreicht werden:

- Verwenden von hochwertiger Hardware (siehe Weltraumtechnik).
- Redundantes Design möglichst vieler Komponenten.
- Alarmierung der Systembetreuer bei Ausfall redundanter Subsysteme.
- Verringerung der Komplexität der Subsysteme.
- Verwenden von stabilen Plattformen und Betriebssystemen.
- Soft- und Hardware Monitoring.
- Ausschließen möglichst vieler SPOF[10] im System.

Die Erhöhung der MTBF ist ein komplexes Themengebiet. Aus diesem Grund ist die Aufzählung nicht vollständig und könnte noch beliebig fortgesetzt werden.

3.4.2 Identifikation von kritischen Punkten im System

Ein wichtiger Schritt beim Entwurf eines HA-Systems ist die Identifikation von SPOF. Die Gründe für SPOFs sind mannigfaltig und es ist auch eine Frage des zur Verfügung stehenden Budgets, wie viele davon ausgeschlossen werden können. Jeder SPOF beinhaltet das Risiko eines Gesamtausfalls des Systems. Für eine Erfassung möglichst vieler SPOFs ist viel Erfahrung der Systemdesigner erforderlich. Mit Blick auf IT-Systeme wird an dieser Stelle eine Liste von SPOFs angegeben [Ma04].

- **Infrastruktur**: Stromversorgung, Klimatisierung, Kabel, Stecker, Personal für Systemwartung, Standort des Rechenzentrums.
- **Netzwerk:** Netzanbindung, Firewalls.
- **Server:** Lüfter, Laufwerke, Netzwerkkarten, Software, schlechte Dokumentation.

[10] Single Point of Failure

Manchmal sind SPOFs im Design von Netzwerk- oder Controllerkarten nur im Schaltplan ersichtlich und lassen sich dadurch nur schwer erkennen [Ro05a]. Auch ein RAID-Controller, bei dem die Information über die Reihenfolge der angeschlossenen Festplatten verloren ging, stellt beispielsweise einen SPOF dar.
Üblicherweise werden viele SPOF durch Redundanz eliminiert. Netzwerkkarten werden doppelt belegt, Massenspeicher redundant angelegt. In vielen Fällen wird ein Parallelbetrieb von Computern in Erwägung gezogen. Das System besteht dann aus zwei oder mehreren parallel laufenden Systemen.

3.4.3 Cluster für Hochverfügbarkeit

Bei daraus entstehenden Clustersystemen, die zumeist mehr als nur zwei Rechner beinhalten können, unterscheidet man einen Active/Passive und einen Active/Active Betrieb. Im Active/Passive Betrieb arbeitet immer nur ein System, das andere befindet sich im Standby Modus. Das bedeutet, es „wartet“ auf den Ausfall des arbeitenden Systems. Bei Active/Active Lösungen arbeiten beide Systeme parallel. Kommt es zum Ausfall eines Systems, so übernimmt jeweils das andere. Dieses Modell kann auch auf mehrere Knoten erweitert werden, sodass mehr als nur zwei Knoten arbeiten und somit ein höherer Speedup erreicht werden kann. Load Balancing Systeme arbeiten nach diesem Prinzip. Diese Lösung wurde im praktischen Teil dieser Arbeit angestrebt und implementiert.
Jeder Knoten im Cluster muss die Fähigkeit haben zu entscheiden, welche seiner Nachbarn noch arbeiten und welche ausgefallen sind. Für diesen Zweck muss ständig eine Interkommunikation zwischen den Knoten stattfinden. Üblicherweise werden so genannte „Heartbeats“ ausgetauscht. Durch diese Heartbeats erhalten alle Knoten eine einheitliche Sicht auf den Cluster. Bleiben solche Heartbeats von einem Knoten aus, so können alle anderen darauf reagieren und ausgefallene Services übernehmen. Diese Aufgabe wird zumeist von eigenen Prozessen übernommen. Unter Linux kann das Softwarepaket „Heartbeat“ dafür eingesetzt werden. Ein gutes HA-Management System sollte aber noch über weitere Fähigkeiten verfügen.

Resource Management

Ein Cluster betreibt zumeist mehrere Ressourcen über seine Knoten verteilt. HA-Systeme sorgen für eine sinnvolle Aufteilung der Ressourcen. Dies wird unter dem Begriff Resour-

ce Management zusammengefasst. In ausgefeilten Setups können Abhängigkeiten zwischen Ressourcen festgelegt werden, die bestimmen, auf welchen Knoten eine bestimmte Ressource laufen darf und auf welchen nicht.
Die Entscheidung, wer die Services übernehmen soll, darf aber nicht von einer übergeordneten Instanz gefällt werden, da diese wieder ein SPOF wäre. Es muss also per Voting entschieden werden, wem die Übernahme zugesprochen wird.

Konsistenz im Cluster

Zum Austausch der Heatbeats ist ein Kommunikationsnetz erforderlich, das – bei nicht redundanter Ausführung – einen SPOF in sich birgt. Fällt dieses Netz aus, so handelt jeder einzelne Knoten so, als wäre er als einziger übrig. Dieses Phänomen wird auch als Partitionierung des Clusters bezeichnet. Als Folge daraus übernimmt jeder Knoten im Netz alle Services, was unweigerlich zu Inkonsistenz der Daten im Cluster führt. Diesen Vorgang nennt man „Splitbrain Syndrome“ [Ma04]. Eine Abhilfe dagegen ist eine redundante Ausführung des Heartbeat Netzes. Bei starker Last auf diesen Netzen besteht aber immer noch die Gefahr, dass nicht alle Knoten Heartbeats von allen anderen erhalten. Auch dagegen müssen Mechanismen bereitgestellt werden. Im Hinblick auf Load Balancing bekommt das Split Brain Syndrome noch eine weitere Dimension. Dienste, die wegen einer Cluster Partitionierung doppelt gestartet werden, verursachen ebenfalls Inkonsistenz im Cluster. Auch dagegen müssen Mechanismen geschaffen werden.

Resource Monitoring

Jeder Knoten muss die gleiche Sicht auf alle Services/Ressourcen haben, die vom Cluster angeboten werden. Dies ist wichtig, damit eine ausgefallene Ressource sofort von einem anderen Knoten übernommen werden kann. Zumindest ist eine Abfrage notwendig, ob die Ressource noch läuft. Besser ist es aber auch gleich die Qualität dieser Ressource zu prüfen. Ein fehlerhaftes Ethernet Interface, das fälschlicherweise im Half-Duplex Modus arbeitet, kann zwar möglicherweise die Ressource noch anbieten, aber nur mit einer unzureichenden Performance. Eine Überwachung von Existenz und Qualität der Ressourcen wird als Resource Monitoring bezeichnet.
Es stellt sich die Frage, wer dieses Ressource Monitoring betreibt. Eine einzige externe Instanz ist zu wenig, da sie wieder einen SPOF darstellen würde. Darum sind mehrere externe Instanzen oder ein gegenseitiges Testen der einzelnen Knoten besser geeignet.

Abgrenzung von ausgefallenen Knoten (Fencing)

Es gibt eine Vielzahl unterschiedlicher Arten von Ausfällen für einen Knoten. Neben Totalausfällen auf Grund von Netzteilfehlern oder Abstürzen mit Reboots können möglicherweise auch nur Subsysteme des Knotens ausfallen. In diesem Fall kann der fehlerhafte Knoten für inkonsistente Zustände im Cluster sorgen, indem er nach wie vor sein bereits von einem anderen Knoten übernommenes Service anbietet. Es muss daher möglich sein, den fehlerhaften Knoten bis zu seiner Reparatur sicher aus dem Cluster auszugrenzen. Diesen Vorgang nennt man Fencing. Er ist fast immer mit der Verwendung von Spezialhardware verbunden, die den fehlerhaften Knoten sicher aus dem Cluster nimmt.

Redundanter Datenzugriff

Egal ob es sich um eine redundanten Webserver, eine Datenbank oder ein Netzwerkdateisystem handelt, der Schutz der Daten vor Zerstörung hat stets Priorität. Ohne Daten existiert kein Service und darum muss darauf spezielles Augenmerk liegen. Besonders wenn mehrere Systeme auf den gleichen Daten schreiben bzw. lesen ist aber die Gefahr von Inkonsistenz und Datenverlust besonders hoch.
Es gibt zwei Arten, wie sich Daten in einem Cluster gut verwalten lassen: Durch Replikation auf alle Knoten, oder durch spezielle Hardware Unterstützung für gemeinsamen Zugriff.
Replikation bietet den Vorteil, dass sie mit relativ geringen Kosten verbunden ist. Aus den vielen vorhandenen Ansätzen, sei hier nur „drbd“ herausgegriffen. Es handelt sich um eine Lösung unter Linux, die Daten von einem aktiven auf einen passiven Knoten überträgt. Nähere Informationen darüber unter: http://www.drbd.org/.
Gemeinsamer Zugriff lässt sich z.B. mit Shared SCSI erreichen. Dabei kann ein RAID-System wahlweise von zwei Controllern bedient werden. Der eine befindet sich im aktiven Knoten, der andere im Failover-Modus.

3.5 Heartbeat Release 1

Eine der bekanntesten HA-Lösungen unter Linux ist Heartbeat. Seit 1999 wird es unter der Leitung von Alan Robertson entwickelt und ist unter der GNU Public License frei erhältlich (http://www.linux-ha.org/). Heartbeat konnte sich neben anderen Produkte fest etablie-

ren und wird heute von Konzernen wie MAN Nutzfahrzeuge AG, FedEx, BBC, Sony und vielen anderen Unternehmen erfolgreich eingesetzt [Ro05a].

In der Release 1 besitzt Heartbeat die Möglichkeit, Heartbeat Nachrichten zwischen zwei Knoten auszutauschen. Bleiben beim Standby Knoten die Nachrichten vom aktiven Knoten aus, so übernimmt der Standby Knoten die Aufgaben des ausgefallenen. Auch die IP-Adresse kann übernommen werden, sodass die Clients ohne Modifikation auskommen.

Auch ein Active/Active Betrieb ist möglich, in dem beide Knoten arbeiten und im Notfall ein Knoten alle Aufgaben übernehmen kann.

Heartbeat Release 1 verfügt aber nicht über die Möglichkeit, mehr als zwei Knoten gleichzeitig zu betreiben. Durch Abschalten des Ressource Managements und durch Verwendung des API des Kernprozess von Heartbeat kann aber ein Mehrknotenbetrieb erreicht werden [Ho03]. Durch die Existenz von Heartbeat Release 2 tritt diese Möglichkeit jedoch in den Hintergrund.

Release 1 bietet für ein redundantes System viele wichtige Funktionen wie Resource Management, Konsistenzprüfung im Cluster, Authentifizierung der Heartbeats und Fencing. Resource Monitoring muss über ein externes Tool wie MON [Li01] erfolgen.

Das **Resource Management** ist auf die Unterstützung von zwei Knoten limitiert. Dadurch ist das Management sehr einfach. Jede Ressource kann unabhängig von allen anderen entweder auf Knoten 1 oder auf Knoten 2 laufen. Alle dafür wichtigen Einträge befinden sich in der Konfigurationsdatei haresources [Weis02] (siehe 6.6.2 Abschnitt „haresources“).

Die **Authentifizierung** der Heartbeats zwischen den Knoten kann in unterschiedlichen Sicherheitsklassen durchgeführt werden. Es stehen dazu drei verschiedene Arten zur Verfügung: CRC, MD5 und SHA [Weis02] (siehe 6.6.2 Abschnitt „authkeys“).

Konsistenzprüfung im Cluster wird in Heartbeat Release 1 mit dem Cluster Consensus Membership Prozess durchgeführt. Er sorgt für eine einheitliche Sicht der beiden Knoten auf den Cluster.

Die Abgrenzung von beschädigten Komponenten mittels „**Fencing**“ wird mit dem so genannten STONITH[11] Subsystem durchgesetzt. Es greift auf Plugins zurück, die defekte Teile des Clusters durch Spezialhardware abschalten.

Die **Interkommunikation** der Knoten mittels Heartbeats kann über Ethernet und/oder über ein serielles Nullmodemkabel erfolgen. Die Installation mehrerer paralleler Kommunikati-

[11] STONITH ist die Abkürzung für Shoot The Other Node In The Head

onskanäle ist möglich. Die Heartbeats werden mit UDP-Broadcasts oder Unicasts ausgetauscht. Auf die Implementation einer Flusskontrolle wurde verzichtet, da keine großen Datenmengen übertragen werden.
Release 1 erfüllt die Anforderungen von Anwendungen mit mittlerem Budget gut, ist aber für mehr als zwei Knoten wegen unzureichendem Ressource-Management nicht einsetzbar.

3.6 Heartbeat Release 2

Heartbeat in Release 2 ist seit August 2005 verfügbar und auch zu kommerziellen Produkten wie LifeKeeper von SteelEye (http://www.steeleye.com/products/linux/), oder Veritas Cluster Server von Symantec (http://www.veritas.com/) konkurrenzfähig [Ro04]. Es ist auf http://www.linux-ha.org/DownloadSoftware unter der GNU Public License frei verfügbar.
Das Hauptproblem von Heartbeat Release 1 war seine Limitierung auf zwei Knoten. Mit Release 2 wurde eine neue Lösung geschaffen werden, die den Betrieb von mehr als zwei Knoten erlaubt. Dies erfordert aber auch ein ausgefeiltes Resource Management. Beim Ausfall eines von drei Knoten ist nicht sofort klar, von welchem der verbleibenden die Ressourcen übernommen werden sollen. Release 2 ist eine Implementation des Open Cluster Frameworks [Ro00], welches einen Standard für das einheitliche Design von Clustersystemen vorschlägt. Folgende Verbesserungen wurden im Vergleich zu Release 1 vorgenommen:

- Unterstützung für mehr als zwei Knoten.
- Adaptierung des STONTH Frameworks für mehr als zwei Knoten.
- Komplexeres Resource Management mit der Möglichkeit Abhängigkeiten für Ressourcen zu definieren.
- Integration von Monitoring in Heartbeat.

Die Handhabung von Release 2 unterscheidet sich am auffälligsten im **Resource Management** von ihrer Vorversion. Es wurde ein Cluster Ressource Manager (CRM) implementiert, der verteilt über alle Knoten für das Resource Management sorgt. Die Datei haresources, in der sonst alle im Cluster verfügbaren Ressourcen eingetragen sind, fällt weg und wird durch die XML-Datei cib.xml ersetzt. CIB steht für Cluster Information Base. Sie ist die Konfigurationsdatei von CRM. In ihr werden alle im Cluster vorhandenen Ressourcen deklariert und ihre Abhängigkeiten zueinander festgeschrieben. Die einfachste Ausprägung dieser Datei beinhaltet drei wichtige Abschnitte:

```
<cib>
    <configuration>
        <crm_config/>
        <nodes/>
        <resources/>
        <constraints/>
    </configuration>
    <status/>
</cib>
```

Abbildung 16: Die Datei cib.xml und ihre einzelnen Abschnitte.

Abschnitte wie <status> oder <nodes> werden vom CRM selbst ausgefüllt und sind für den Benutzer nicht von Bedeutung. In <crm_config> können Grundeinstellungen für den CRM bestimmt werden. Dazu gehört z.B. ob für die Beschlussfähigkeit des Clusters ein Quorum erforderlich ist: <nvpair id = "require_quorum name = "require_quorum" value = "true"/>.

In <resources> werden alle im Cluster verfügbaren Ressourcen mit ihren Aufrufparametern angeführt. Dabei gibt es verschiedene Klassen von Ressourcen:

- OCF-Ressourcen entsprechen dem Open Cluster Framework Standards.
- Heartbeat-Ressourcen werden aus Kompatibilitätsgründen zu Release 1 weiter unterstützt.
- LSB-Ressourcen können normale Init-Scripts sein, die den Linux Standard Base[12] Konventionen entsprechen.
- Stonith-Ressourcen sind ausschließlich für das Fencing defekter Komponenten im Cluster zuständig und den OCF-Ressourcen sehr ähnlich.

[12] LSB ist ein Standard für die Erhöhung der Kompatibilität zwischen Linux® Distributionen (http://www.Linux®base.org/).

```
<resources>
  <primitive id="ip_res" class="ocf" type="IPaddr" provider="heartbeat">
    <instance_attributes>
      <attributes>
        <nvpair name="ip" value="1.2.3.4"/>
      </attributes>
    </instance_attributes>
  </primitive>
  <primitive id="apache" class="heartbeat" type="apache">
    <instance_attributes>
      <attributes>
        <nvpair name="1" value="/apache1dir/httpd.cf"/>
      </attributes>
    </instance_attributes>
  </primitive>
</resources>
```

Abbildung 17: Ressourcen mit Ressouceklasse „ocf“ und „heartbeat“

Das Schlüsselwort „primitive“ deklariert eine neue Ressource. Die Parameter, die dieser Ressource mitgegeben werden, können in Form von Instanzattributen angegeben werden. Um viele Ressourcen besser handhaben zu können, ist es möglich, Ressourcen zu Gruppen zusammen zu fassen: <group id=”webserver”><primitive/><primitive/></group>.
Der Abschnitt „constraints” bestimmt auf welchen Knoten Ressourcen oder Ressource-Gruppen laufen sollen.

```
<constraints>
  <rsc_location>
    <rule/>
    <rule/>
  </rsc_location>
</constraints>
```

Abbildung 18: Das Grundgerüst des Abschnitts <constraints>.

Jede dieser Resource-Locations kann eine Menge von Regeln beinhalten, die Aussagen über Ressourcen und ihren Ausführungsort machen:

```
<constraints>
  <rsc_location id="run_webserver" rsc="apache">
    <rule id="rule_webserver" score=100>
      <expression attribute="#uname" operation="eq" value="node1"/>
    </rule>
  </rsc_location>
</constraints>
```

Abbildung 19: Constraints Abschnitt der cib.xml.

Die Ressource-Location in Abbildung 19 gibt für die Ressource mit der ID „apache“ eine Präferenz von 100 zum Lauf auf Knoten „node1“ an. Dies bewirkt, dass diese Regeln, gegenüber anderen mit niedrigerer Gewichtung als 100, „gewinnt“. Es sind noch weitaus

komplexere Constraints möglich. Eine nähere Beschreibung über die Cluster Information Base befindet sich unter 6.6.2 Abschnitt „Kurzbeschreibung der Konfigurationsdatei cib.xml“ sowie unter http://www.linux-ha.org/ClusterInformationBase_2fUserGuide.

4 Dezentrales Load Balancing für Firewalls

4.1 Aufgabenstellung

Für HA und Load Balancing existiert eine Fülle von Lösungen. Einige davon wurden bereits in Kapitel 3 erläutert. Vor allem im Bereich von Anwendungsservern, wo Load Balancing aber auch HA besonders gefragt sind, sind solche Lösungen allgegenwärtig. Für Linux bietet LVS eine flexible Plattform, die fast jede Anforderung erfüllen kann.
In punkto Load Balancing und HA für Netfilter Firewalls ist – zumindest im Open-Source Bereich – keine etablierte Lösung auf dem Markt. Dies mag zum einen am jungen Alter von Netfilter, zum anderen an der Komplexität des Themas liegen. Aufgrund dieses Umstandes wurde für den praktischen Teil der Arbeit die Implementierung eines HA Systems mit Load Balancing gewählt. Die genauen Anforderungen lauteten wie folgt:

- Möglichst geringe Hardwareanforderungen: kein zentraler Redirector vor der Firewall, keine administrierbaren Layer 2 Geräte vor oder hinter der Firewall.
- Alle Knoten sollen arbeiten, um Load Balancing betreiben zu können.
- Erweiterbarkeit auf beliebig viele Knoten für den Firewall Cluster.
- Unterstützung möglichst vieler Protokolle.
- Unterstützung möglichst vieler Betriebsarten von Firewalls.
- Lauffähigkeit auf dem Firewall Produkt Gibraltar der Firma eSYS.
- Bereitstellung von Schnittstellen für die Integration in die graphische Benutzeroberfläche von Gibraltar.
- Lauffähigkeit auf Linux Kernels ab Version 2.4.27 inklusive neuerer 2.6er Versionen.
- Hohe Transparenz für den Administrator der Firewall.
- Hundertprozentige Transparenz für alle an die Firewall angeschlossenen Rechner.

Die Elimination des zentralen Redirector begründet sich darin, dass solch teure Spezialhardware nicht zur Verfügung stand. Außerdem wurde als Zielplattform Gibraltar gewählt, eine Firewall, die für Klein- und Mittelbetriebe konzipiert ist. Niedrige Kosten sind in diesem Segment meist erwünscht.

Als Besonderheit ist die Erweiterbarkeit auf mehr als nur zwei Knoten zu werten. Für gebräuchliche Anwendungen wird ein Zweiknoten-Betrieb ausreichen. Dennoch können mehrere Knoten – schon allein wegen der besseren Performance – einen Vorteil mit sich bringen.
Wichtiger Punkt in der Liste von Anforderungen ist auch die Transparenz dem Administrator gegenüber. Bei der Bedienung der Firewall soll er nach Möglichkeit den Eindruck haben, er administriere eine einzige Firewall. Dennoch sollen sich alle Änderungen stets auf alle Knoten im Cluster auswirken.

4.2 Lösungsansätze

Mit Blick auf diese Anforderungspunkte wurden mehrere Lösungsansätze durchdacht, Prototypen implementiert, die beste Lösung verfeinert und in Gibraltar integriert. Prinzipien wie z.B. jenes von Clusterip wurden als Ausgangsbasis für diese Lösungsansätze verwendet.

4.2.1 Verteilung des Verkehrs auf Layer 2

Da Load Balancing ohne zentralen Redirector angestrebt wird, ist zunächst darauf zu achten, dass an jeden Knoten eine Kopie des ankommenden Paketes gelangt. Es wurde Transparenz gegenüber allen angeschlossenen Geräte gefordert. Dadurch dürfen keine Veränderungen an ihnen vorgenommen werden. Der administrative Aufwand wäre zu hoch. Lediglich an den Clusterknoten darf das Verhalten am Layer 2 verändert werden.
Bei der Verwendung von Ethernet Hubs ist eine Verteilung ohnehin schon gewährleistet. Die angeschlossenen Ethernet Netzwerkkarten müssen nur in den Promiscuous Mode versetzt werden, damit die IP-Stacks aller Knoten mit den Paketen versorgt werden. Im Zeitalter von Layer 2 Switches ist dies selbst in einem Broadcast-Medium wie Ethernet nicht ohne weiteres möglich. Es müssen spezielle „Tricks“ angewendet werden, um den Switch für den ankommenden Verkehr wie einen Hub reagieren zu lassen. Hierfür stehen zwei Varianten zur Verfügung.

Die Unicast-Methode

Ein Ethernet Switch versucht, die MAC Adresse der angeschlossenen Interfaces zu lernen, indem er die Source MAC-Adresse jedes ankommenden Ethernet Frames abspeichert und

sie der Portnummer, an dem der Frame empfangen wurde, zuordnet. Im Laufe der Kommunikation zwischen verschiedenen Endgeräten erhält der Switch so eine Tabelle von MAC-Adressen und Port Nummern, die es ihm ermöglicht, eine direkte Zustellung der Ethernet Frames durchzuführen. Kommt aber ein Frame mit unbekannter Destination MAC-Adresse an den Switch, so stellt er diesen an alle Ports zu. Endgeräte, auf die die MAC-Adresse passt, werden ihn annehmen, alle anderen verwerfen ihn. Dieser Mechanismus kann für ein Broadcasting ausgenutzt werden.
Dabei muss vermieden werden, dass der Switch einen MAC ↔ Port Eintrag für die Firewallknoten generiert. Folgende Vorgehensweise ist dazu geeignet:

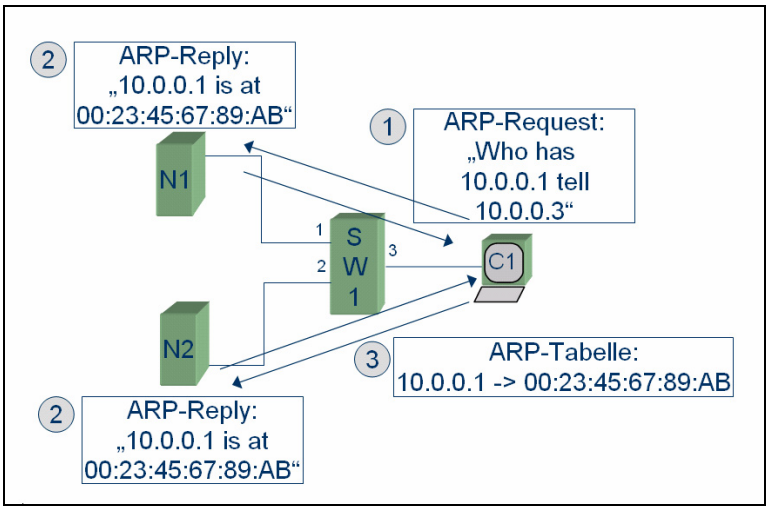

Abbildung 20: Die Unicast-Methode.

Alle Firewall Knoten senden unter einer anderen MAC-Adresse als der, unter der sie empfangen. Wie in Abbildung 20 dargestellt, erhält der Cluster eine Cluster MAC Adresse, unter der er ARP-Requests beantwortet.

1. C1 sendet einen ARP-Request an alle Knoten: „Who has 10.0.0.1 tell 10.0.0.3“
2. N1 und N2 beantworten diesen ARP-Request mit der Antwort:
   ”10.0.0.1 is at 00:23:45:67:89:AB“.
   Achtung: Dieser Frame hat aber im Source MAC Feld nicht 00:23:45:67:89:AB sondern irgendeine andere MAC-Adresse stehen!
3. C1 speichert in seinem ARP-Cache folgenden Eintrag ab:
   10.0.0.1 -> 00:23:45:67:89:AB.

Nach Ausführung von Punkt drei befinden sich folgende Einträge im Switch:

Port	MAC-Adresse
1	Beliebige MAC von N1
2	Beliebige andere MAC von N2
3	Beliebige MAC von C1

Tabelle 4: Inhalt der Tabelle im Switch

Sendet nun C1 Frames an den Cluster, so führen diese die Destination MAC-Adresse 00:23:45:67:89:AB im Header. Diese MAC-Adresse findet der Switch in seiner Tabelle nicht und muss folglich diese Frames an alle Ports schicken. Genauso wie es für diesen Anwendungsfall erforderlich ist.

Es ist zu bemerken, dass die beschriebene Arbeitsweise auch für clusterseitige ARP-Requests funktionieren muss, da der ARP-Cache von C1 auch durch ARP-Requests des Clusters lernt!

Die Multicast-Methode

Die Multicast-Methode ist der Unicast-Methode relativ ähnlich. Wie auch andere Layer 3 Protokolle verfügt das IP-Protokoll über die Möglichkeit, Pakete per Multicast an mehrere Rechner gleichzeitig zu senden. Ein Shared-Medium wie das Ethernet ermöglicht es, die aus den Paketen entstehenden Frames gleichzeitig an mehrere Netzwerkinterfaces zu senden. Hubs benötigen dazu keine speziellen Funktionalitäten. Ihre Aufgabe ist es ohnehin, jeden Frame an alle Ports zu verteilen. Switches stellen nach Möglichkeit jeden Frame nur gezielt an den richtigen Port zu. Für Multicast bedienen sie sich einer speziellen Technik. Ein Multicast Paket wird bei der IP-Ethernet Encapsulation [Ie84] als Multicast Frame gekennzeichnet. In der Destination MAC-Adresse wird im ersten Oktet das niederwertigste Bit (das Multicast Bit) gesetzt [Ie88]. Damit erhält der Switch die Information, dass dieses Paket mehrere Empfänger haben kann. Als Folge daraus stellt der Switch diese Multicast-Pakete an alle Ports im VLAN zu.

Modernere Layer 3 Switches haben auch die Möglichkeit, die Frames nur an die Ports zu schicken, an denen sich ein Endgerät befindet, das an den Daten interessiert ist. Um zu wissen an welchen Ports ein solches Gerät angeschlossen ist, muss der Switch gesendete IGMP-Queries [Ie88; Ie97] verstehen. Er kann dann eine Tabelle von Portnummern mit IGMP-Groups anlegen.

Frames mit gesetztem Multicast Bit werden also an alle Ports gesandt. Diese Arbeitsweise lässt sich für die Verteilung des ankommenden Verkehrs zweckentfremden. Ähnlich wie bei der Unicast-Methode beantworten alle Knoten der Firewall ARP-Requests, diesmal jedoch mit einer Multicast MAC-Adresse.

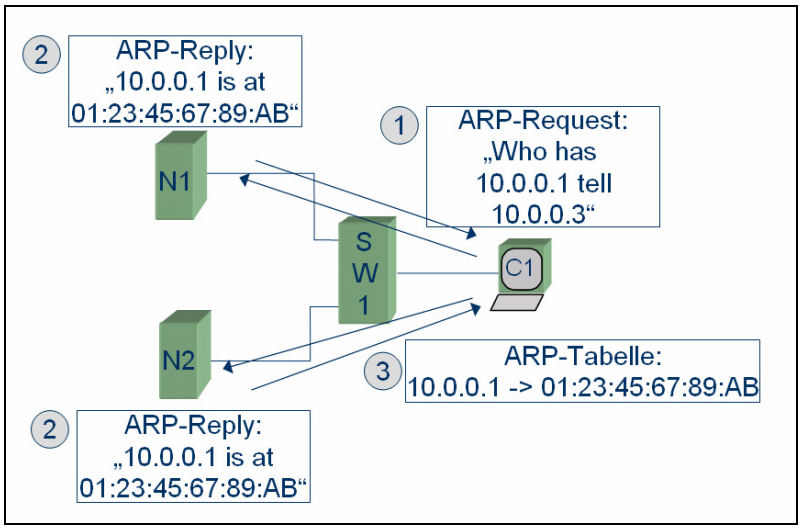

Abbildung 21: Die Multicast-Methode.

1. C1 sendet einen ARP-Request an alle Knoten: „Who has 10.0.0.1 tell 10.0.0.3“
2. N1 und N2 beantworten diesen ARP-Request mit der Antwort:
   ”10.0.0.1 is at 01:23:45:67:89:AB“.
   Achtung: Dieser Frame hat aber im Source MAC Feld nicht 01:23:45:67:89:AB sondern irgendeine andere MAC-Adresse stehen! Dadurch lernt der Switch diese andere MAC-Adresse, der Client C1 jedoch die Multicast MAC-Adresse.
3. C1 speichert in seinem ARP-Cache folgenden Eintrag ab:
   10.0.0.1 -> 01:23:45:67:89:AB.

Es wird dadurch bei C1 ein ARP-Cache Eintrag mit einer Multicast MAC-Adresse erzwungen. Ansonsten ist die Multicast-Methode für die Clients transparent.
Für die Realisierung reicht es nicht aus, das Multicast Bit der MAC-Adressen auf den Netzwerkkarten der Clusterknoten zu setzen. ARP-Requests/Replies würden zwar ordnungsgemäß durchgeführt werden. Sendet aber ein Clusterknoten mit einer Multicast MAC-Adresse, so verwirft C1 den Frame. Nur Frames mit Unicast IP-Adresse und Unicast-MAC werden angenommen. Darum müssen ARP-Requests/Replies, so wie oben beschrieben, verändert werden.

Manche Switches reagieren auf die gleiche Art und Weise. Sie verwerfen Frames mit einer Unicast-IP/Multicast-MAC Kombination. Diese Switches sind in Verbindung mit der Multicast-Methode nicht einsetzbar.

Multicast- vs. Unicast-Methode

Sowohl Multicast- als auch Unicast-Methode haben Vor- und Nachteile. Um einen direkten Vergleich anstellen zu können, wird nachfolgend eine tabellarische Auflistung von Vor- und Nachteilen der beiden Methoden vorgenommen.

	Unicast-Methode	Multicast-Methode
RFC-Konform	ja	nein [Ie89b]
Beschränkung der Broadcast- Domain[13]	nein	ja (bei IGMP-fähigen Switches)
Erhöhte Abhörsicherheit	nein	ja (Beschränkung der Broadcast Domain)
Transparenz für Clients	ja	ja
niedrige CPU-Last auf den Clusterknoten	nein (NIC muss im promiscuous Mode sein)	ja (NIC kann Multicast Filter verwenden)

Tabelle 5: Vergleich der Unicast- und Multicast Methode.

Für beide Varianten gibt es Einsatzgebiete, wobei jedoch in der Praxis eher die Multicast-Methode eingesetzt wird. Der Hauptgrund dafür ist die geringere CPU-Belastung, weil sich die Netzwerk Interfaces nicht im promiscuous Mode befinden müssen. Sie können durch einen eingebauten Multicast-Filter die CPU entlasten. Außerdem lässt sich die Broadcast Domain einschränken, was der Sicherheit zugute kommt.

4.2.2 Load Balancing mit Packet Filtering[14]

Nachdem die Verteilung der Pakete im vorangegangenen Abschnitt erläutert wurde, kann auf die eigentliche Aufteilung des Verkehrs eingegangen werden.
Geht man von einer Firewall im Sinne von Packet Filtering aus, so kann eine einfache Methode für Load Balancing herangezogen werden. Die ankommenden Pakete können, jedes

[13] Möglichkeit die Broadcast-Domain auf Ports der Clusterknoten zu beschränken.

[14] Packet Filtering ist die Filterung von Paketen ohne internen Zustand der Firewall.

für sich, als unabhängig voneinander betrachtet werden. Es spricht also nichts dagegen, die Verarbeitung der Pakete gleichmäßig auf alle Knoten aufzuteilen. Routingtabelle und Regelwerk sind – zumindest für den Host übergreifenden Verkehr[15] – auf allen Knoten gleich. Wichtig ist nur, dass ein verteilter Entscheidungsprozess existiert, in dem die Knoten entscheiden, wer das anliegende Paket annimmt. Diese Entscheidung sollte möglichst bald in der Verarbeitung in den Knoten erfolgen, um den Speedup hoch zu halten.
Nach dem Vorbild von Clusterip (siehe Abschnitt 3.1) kann ein geeigneter Hashcode über Source IP-Adresse und Source Port jedes ankommenden Paketes gebildet werden. Abhängig davon wird das Paket von einem der Knoten angenommen, von den anderen verworfen. Antwortpakete werden möglicherweise über andere Knoten zum Absender zurückgeschickt. Es könnte ein TCP SYN-Paket über einen Knoten N1 nach außen und das Antwortpaket über einen Knoten N4 zurück an den Absender geleitet werden.

4.2.3 Load Balancing mit Stateful Inspection

Der im vorigen Abschnitt beschriebene Parallelbetrieb von Firewalls stößt an seine Grenzen, wenn Stateful Inspection ins Spiel kommt. Eine Möglichkeit wäre, auch laufend die Verbindungstabellen der Knoten zu synchronisieren. Dies bringt jedoch jene Probleme mit sich, die schon in Abschnitt 3.2.2 mit Netfilter-HA dargestellt wurden.
Um also TCP-Verbindungen mitverfolgen zu können, ist es notwendig, dass alle Pakete einer Verbindung den gleichen Knoten passieren. Ein Drei-Wege-Handshake kann z.B. anders nicht mitverfolgt werden. Jeder Knoten hält nur jene Einträge in seiner Verbindungstabelle, für die er verantwortlich ist. Für ein Failover bedeutet dies leider, dass beim Ausfall eines Knotens alle Verbindungen, die auf jenem Knoten gespeichert wurden, abbrechen. Sie können aber über einen neuen Knoten wieder aufgebaut werden.
Um eine Aufteilung der TCP-Verbindung auf die Knoten zu erreichen, muss hinterfragt werden, wie eine einzelne TCP-Verbindung eindeutig identifiziert wird. Es ist das Quadrupel aus Client IP-Adresse, Client-Port, Server IP-Adresse und Server-Port. Mit einer Hashfunktion, die diese vier Werte einer Zahl von 1 bis N zuordnet, partitioniert man die Menge aller TCP-Verbindungen in N Teilmengen. Eine Hashfunktion mit diesen Eigenschaften ist trivial zu finden. Die Modulo-Funktion ist eine Variante. Dennoch sollte eine bessere

[15] Jener Verkehr, der nicht für lokale Prozesse auf den Clusterknoten bestimmt ist.

Hashfunktion wie z.B. der im Linux Kernel vorhandene Jenkins Hash [Je00] verwendet werden.

Die Berechnung der Hashfunktion – in Zukunft als Selektion bezeichnet – erfordert als Eingangsparameter die schon oben genannten vier Daten, Client IP-Adresse, Client-Port, Server IP-Adresse und Server-Port. Aus der Sicht der Firewall ist aber nicht klar, welches IP/Port Paar zum Server und welches zum Client gehört. Je nachdem ob das Paket vom Client oder vom Server stammt, befinden sich die IP/Port Paare in den Source- bzw. Destination Feldern des IP-Headers. Connection Tracking kann die Information über die Herkunft des Paketes liefern. Andererseits sollte die Entscheidung, ob ein Paket für einen Knoten bestimmt ist, aus Gründen der Performance möglichst bald fallen, also vor den zeitintensiven Operationen von Connection Tracking.

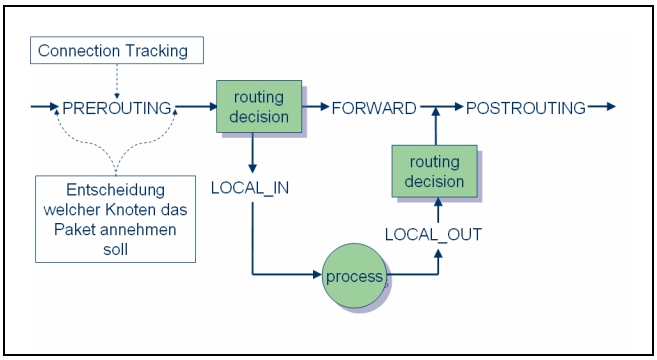

Abbildung 22: Connection Tracking und Vorauswahl.

Abbildung 22 zeigt die beiden Möglichkeiten, die sich bieten, um eine Vorauswahl der Pakete durchzuführen. Connection Tracking findet am PREROUTING-Hook statt. Die Vorauswahl sollte ebenfalls unter PREROUTING durchgeführt werden. Die erste Möglichkeit, ist sie mit einer höheren Priorität als Connection Tracking auszustatten, um sie als erstes vorzunehmen. Die zweite Möglichkeit besteht darin, sie mit niedrigerer Priorität zu starten, also nach Connection Tracking, was einerseits langsamer ist, andererseits eine bessere Paketselektion erlaubt. In 4.2.4 und 4.2.5 werden beide Möglichkeiten besprochen.

4.2.4 Load Balancing nach Connection Tracking

Ein Kernel Modul namens FWSELECT wird ip_conntrack einfach nachgeschaltet. Es bewertet nur Pakete, die neue Verbindungen initiieren. Alle anderen werden ohne Auswahl zur Verarbeitung ins weitere Regelwerk der Firewall durch gelassen. Abbildung 23 be-

schreibt in einem Ablaufdiagramm den Entscheidungsprozess für die Annahme eines Paketes.

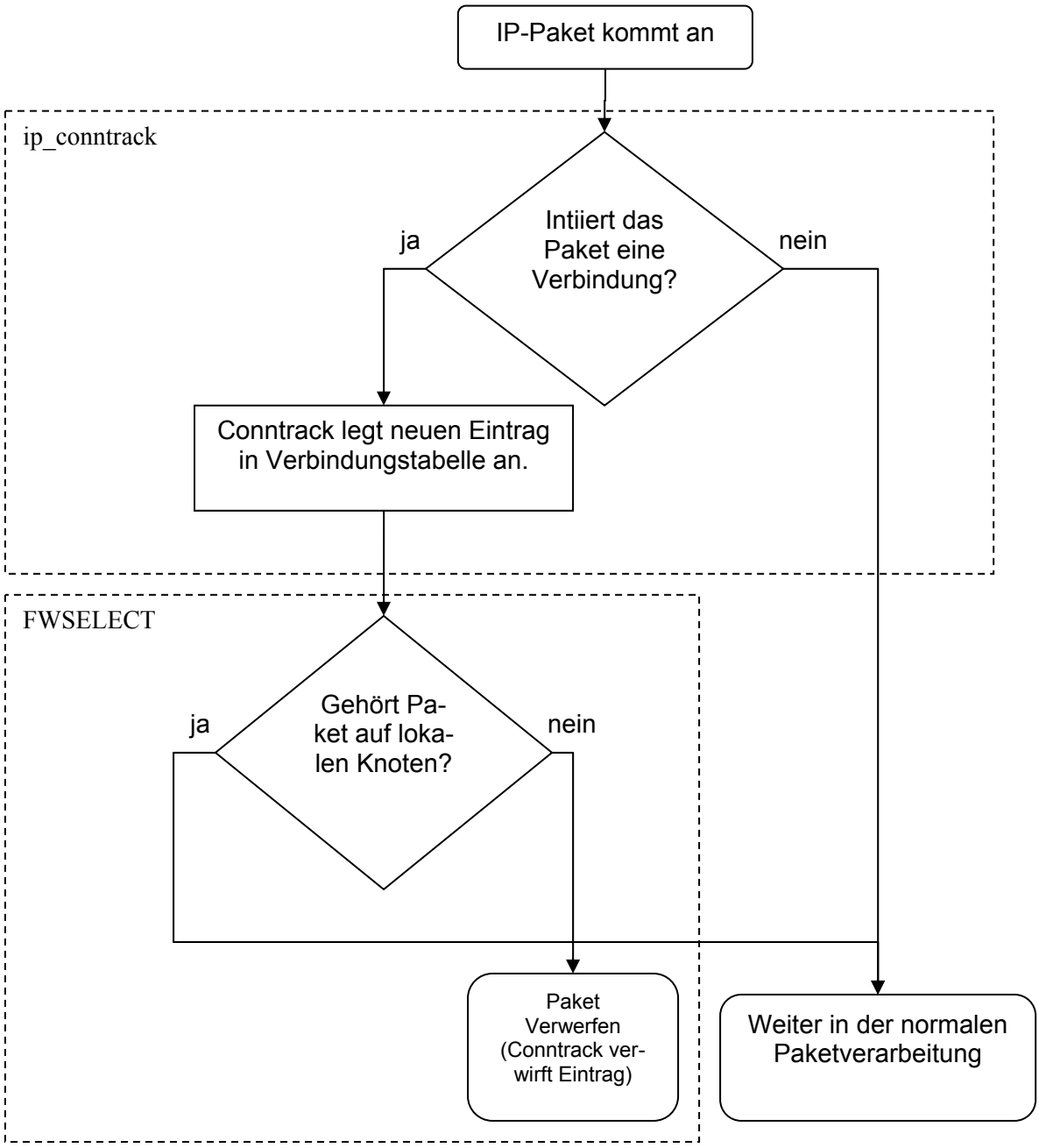

Abbildung 23: Entscheidung über die Annahme von Paketen.

Nach obigem Ablaufdiagramm werden nur jene Pakete, die neue Verbindungen initiieren, mit der Hashfunktion selektiert. Alle anderen werden ohne Clusterselektion zur weiteren Paketverarbeitung zugelassen. Pakete, die eigentlich für andere Knoten bestimmt wären und für die am lokalen Knoten kein Eintrag in der Verbindungstabelle existiert, werden von einer Regel eliminiert, die alle Pakete mit Status „INVALID“ verwirft. Das Design der Firewall muss also folgende Eigenschaften aufweisen:

- Pakete, mit dem Status „INVALID“ werden sofort verworfen.
- Pakete die zu bereits aufgebauten Verbindungen gehören, werden immer akzeptiert. (Das sind Pakete mit dem Status „ESTABLISHED“)
- Pakete, die zu bereits aufgebauten Verbindungen verwandt sind, werden immer akzeptiert. (Das sind Pakete mit dem Status „RELATED“)
- Zum Erlauben von Applikationsprotokollen, die mehrere TCP-Verbindungen benötigen (z.B. FTP, FXP, H.323), werden immer nur Stateful Regeln benutzt.
- Regeln zum Erlauben neuer Verbindungen haben immer folgende Gestalt:

```
target prot opt source       destination
ACCEPT tcp  --  10.0.0.0/24  any  tcp spts:1024: dpt:www state NEW
```

Abbildung 24: Musterregel für Firewall Cluster.

- Die Firewall verändert Client IP-Adresse, Client-Port, Server IP-Adresse und Server-Port im Laufe ihrer Paketverarbeitung nicht.

Regelüberdeckungen im Firewall Cluster

Ein nach den oben angeführten Punkten gültiges Regelwerk könnte also z.B. folgendermaßen aussehen:

```
Chain FORWARD (policy DROP)
target prot opt source      destination
ACCEPT tcp  --  anywhere    anywhere tcp state ESTABLISHED
ACCEPT tcp  --  anywhere    anywhere tcp state RELATED
ACCEPT tcp  --  10.0.0.0/24 anywhere tcp spts:1024:65535 dpt:ftp state
NEW
```

Abbildung 25: Gültiges Regelwerk im Cluster

Ein IPTables Regelwerk der folgenden Art wäre hingegen ungültig:

```
Chain FORWARD (policy DROP)
target prot  source      destination
ACCEPT tcp   anywhere    anywhere tcp state ESTABLISHED
ACCEPT tcp   anywhere    anywhere tcp state RELATED
ACCEPT tcp   10.0.0.0/24 anywhere tcp spts:1024:65535 dpt:ftp state NEW
ACCEPT tcp   anywhere    10.0.0.0/24 tcp spts: ftp-data dpt: 1024:65535
```

Abbildung 26: Ungültiges Regelwerk im Cluster

Hier wäre die Gefahr von mehrfach versendeten TCP-Paketen gegeben, da auf jeden Fall der Knoten, der die FTP-Command Verbindung übernimmt, auch die FTP-Data Verbindung zulassen würde. Zusätzlich aber werden die anderen Knoten, die FTP-Data Verbindung als unabhängige neue Verbindung ansehen und wegen der letzten Regel in Abbildung 26 ebenfalls erlauben. Die dynamisch erstellte Regel von ip_conntrack überdeckt sich mit der statischen FTP-Data Regel. **Regelüberdeckungen von statischen und dynamischen Regeln müssen aus diesem Grund vermieden werden.**

Eigentlich ist die letzte Regel im obigen Regelwerk überflüssig und macht die Firewall unsicher. Man bedenke aber, dass es noch komplexere Protokolle mit mehreren TCP-Verbindungen gibt als FTP [Ie85]. Bei ihnen werden über eine Parent-Connection mehrere Child-Connections initiiert, die zufällig gewählte Source- und Destination-Ports belegen. Für diese Protokolle werden dann von ip_conntrack dynamische Regeln erzeugt, welche sich mitunter mit statischen Regeln überdecken können, was – wie oben gezeigt – zu mehrfach versendeten Paketen führen kann. In einem Regelwerk mit mehreren tausend Regeln ist diese Gefahr aber niemals hundertprozentig auszuschließen. Das Problem tritt auch bei gegenseitiger Regelüberdeckung zweier dynamischer Regeln auf.

Load Balancing nach Connection Tracking ist ein Ansatz, der ein interessantes Grundkonzept darstellt, da er in der Entscheidungsfindung, welcher Knoten ein Paket annehmen darf, Zugriff auf die Verbindungsdaten hat. Dennoch verletzt dieser Ansatz eine Anforderung, die gestellt wurde. Durch die Vorschrift über das Design der Firewallregeln gewährt er keine hohe Transparenz in der Administration des Clusters und widerspricht damit einer in Abschnitt 4.1 aufgestellten Forderung. Außerdem sollte die Paketselektion möglichst bald in der Verarbeitung geschehen, um Zeit zu sparen. Der beschriebene Ansatz verschiebt die Selektion jedoch hinter das Connection Tracking. Die Selektion sollte aber vor dem Connection Tracking passieren, weil damit die CPU-intensiven Listenoperationen nicht mehrfach durchgeführt werden müssen.

4.2.5 Load Balancing vor Connection Tracking

Eine weitere Möglichkeit ist, die Selektion zum frühesten Zeitpunkt geschehen zu lassen, der für Netfilter möglich ist. Der erste Hook, an dem jedes Paket vorbei kommt, ist der PREROUTING Hook. Hier greift ip_conntrack erstmals die Pakete ab. Ein Kernel Modul, das eine Selektion noch vor ip_conntrack durchführt, vermeidet die zeitintensiven Operationen, die durch Connection Tracking entstehen. Leider besteht dadurch aber auch keine Möglichkeit zu entscheiden, ob das Paket vom Server oder vom Client kommt. Die in Abschnitt 4.2.3 beschriebene Hashfunktion kann also nicht mit den Parametern in richtiger Reihenfolge aufgerufen werden (hash(Client IP, Client Port, Server IP, Server Port)), weil nicht klar ist, wo im IP-Header z.B. die Client-IP und Server-IP zu finden sind. Um ohne diese Information auszukommen, muss eine spezielle Hashfunktion verwendet werden, bei der sowohl Client-IP und Server-IP als auch Client-Port und Server-Port jeweils vertauschbar sind. Es muss also stets gelten:

$$h(ClientIP,ClientPort,ServerIP,ServerPort) = h(ServerIP,ServerPort,ClientIP,ClientPort)$$

Abbildung 27: Eigenschaft der Hashfunktion für LB ohne Connection Tracking.

Eine Möglichkeit ist, die Paare von ClientIP und ServerIP bzw. ClientPort und ServerPort zu multiplizieren und erst dann den eigentlichen Hashwert zu bilden. Damit wird jedes Paket, egal ob in Server-Client oder Client-Server Richtung, immer über den gleichen Knoten geleitet, was Stateful Inspection auf den einzelnen Knoten ermöglicht. Eine FTP-Verbindung ist dennoch nicht möglich, da sowohl Command-, als auch Data-Connection über den gleichen Knoten laufen müssen. Die Data-Connection hat zwar gleiche Source- und Destination IP-Adressen wie die Command-Connection, aber andere Source- und Destination Ports. Eine Spezialbehandlung der Ports 20 und 21 in der Firewall, würden diese Probleme zumindest für FTP ausschalten. Um die Funktion aber für alle Multi-TCP-Protokolle zu erlauben, müssen Ports bei der Berechnung des Hashwertes weg gelassen werden. Es gilt also nur noch:

$$h(ClientIP,ServerIP) = h(ServerIP,ClientIP)$$

Abbildung 28: Hashfunktion für Preselect.

Jetzt können auch Protokolle mit mehreren TCP-Verbindungen in unterschiedliche Richtungen ablaufen, falls diese nur zwei Hosts involvieren. Alle TCP-Verbindungen einer

FTP-Session werden immer über den gleichen Knoten verbunden, egal ob im Active- oder Passive Mode. Der Nachteil beim Weglassen der Ports besteht darin, dass sich nicht die TCP-Verbindungen auf die Knoten verteilen, sondern Paare von IP-Adressen. D.h. die Granularität der kleinsten aufteilbaren Einheit ist eine größere.

Nach wie vor nicht möglich sind Protokolle mit TCP-Verbindungen, die mehr als zwei Hosts involvieren. Ein gebräuchlicher Vertreter dieser Protokolle ist das File Exchange Protokoll (FXP). Es ermöglicht den Datenaustausch zweier FTP-Server, gesteuert von einem Client.

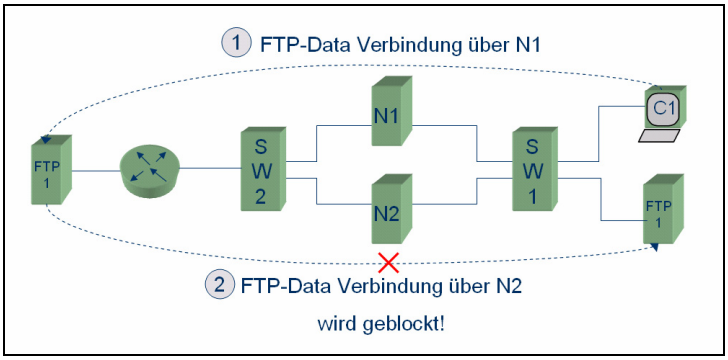

Abbildung 29: FXP Verbindung über den Cluster.

Der Client C1 baut über einen Knoten eine Command-Verbindung zu FTP1 auf. Er weist FTP1 an, eine Daten-Verbindung zu FTP2 aufzubauen, welche wiederum über die Firewall führt. Diesmal wird aber – auf Grund der IP-Adresse von FTP2 – Knoten 2 für die Verbindung ausgewählt. Die Verbindung schlägt fehl, da Knoten 2 keine FTP-Data Verbindung zu lässt, diese wird nur von N1 erwartet. Der Einsatz von Load Balancing mit Connection Tracking könnte hier eine sinnvolle Lösung sein.

In Punkto Transparenz ist zu bemerken, dass sich beim Aufbau des Regelwerks der Firewall keine Restriktionen ergeben. Er erfolgt genau wie bei einer Firewall ohne Cluster-Design.

4.2.6 Network Address Translation im Cluster

Unabhängig davon, ob man Load Balancing mit oder ohne Connection Tracking verwendet, ergeben sich für Network Address Translation (NAT) Probleme. Als Einführung wird hier kurz auf die Funktionsweise von Source NAT mit IPTables eingegangen.

NAT erfordert per Definition Connection Tracking, da Source- bzw. Destination IP-Adressen auf Basis der Verbindungstabelle ersetzt werden. Für Source NAT wird die

Source IP-Adresse durch eine andere ersetzt und falls erforderlich, auch ein anderer Source Port verwendet. Bei IPTables finden diese Modifikation kurz nach dem Eintreten bzw. kurz vor dem Verlassen des Paketes aus dem Netfilter Framework statt [RuWe02].

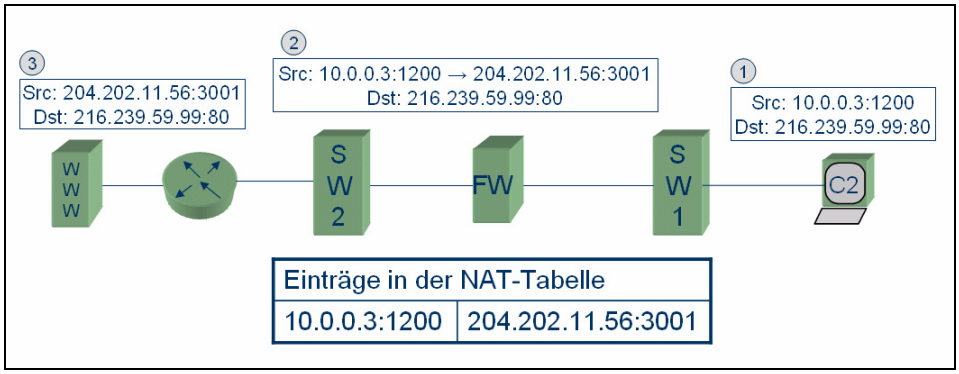

Abbildung 30: Source NAT.

IPTables hat dadurch die Möglichkeit, die Pakete zu verändern, bevor bzw. nachdem sie durch Routing- und Filter-Verarbeitung gehen. Pakete vom Client werden am POSTROUTING Hook mit der Source Adresse des Interfaces ersetzt, an welchem sie wieder ins Netz geschickt werden. Falls notwendig, wird außerdem der Source Port durch einen unbelegten Port der Firewall belegt. Die Änderung von Source IP-Adresse und Port wird in ip_conntrack vermerkt, da auch Antwortpakete auf Basis dieser Information manipuliert werden. Die Destination Adressen und Ports der Antwortpakete werden am PREROUTING Hook mit denjenigen Adressen ersetzt, die in der NAT-Tabelle stehen.

Für den Einsatz von dezentralem Load Balancing ergibt sich dadurch ein Problem. Die Entscheidung, welcher Knoten ein Paket weiter verarbeitet, basiert auf Basis der Source/Destination IP-Adressen/Ports. NAT verändert aber genau diese. Würde diese Veränderung immer unmittelbar nach dem Eintreten der Pakete in die Knoten passieren, so ergäbe sich daraus kein großes Problem. Die Entscheidung, ob das Paket lokal angenommen wird, müsste einfach auf unmittelbar nach der Ersetzung verschoben werden. Pakete in Client-Server Richtung werden aber erst am POSTROUTING Hook, also nach der Paketverarbeitung, verändert. Eine Selektion nach dieser Ersetzung hätte keinen Sinn, da keine Performancesteigerung zu einer normalen Firewall zustande käme. Ein Paket würden von den N Knoten parallel verarbeitet werden um anschließend von N-1 Knoten verworfen zu werden.

Abbildung 31 zeigt ein Setup, das den Betrieb von Source-NAT erlaubt.

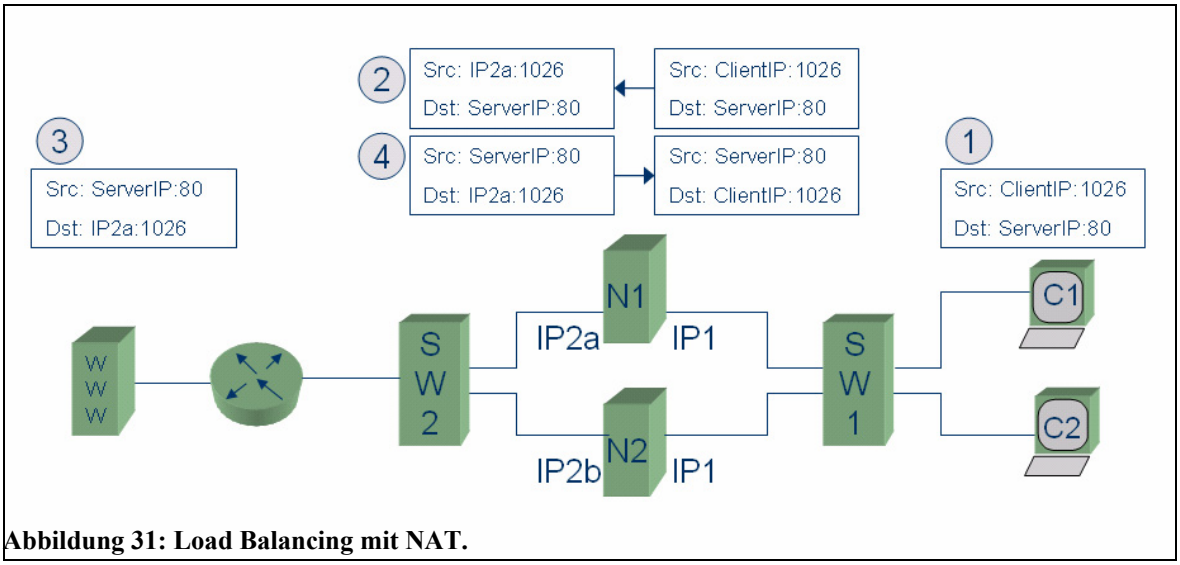

Abbildung 31: Load Balancing mit NAT.

Im öffentlichen Netz wird nicht eine einzige gemeinsame IP-Adresse verwendet, sondern es existieren für jeden Knoten separate IP-Adressen. Ein TCP SYN-Paket wird von C1 an den Webserver gesendet. In Punkt 2 wird die Source IP-Adresse des Pakets mit IP2a, einer IP-Adresse die nur von N1 verwendet wird, überschrieben. Der Server sendet nach Erhalt des SYN-Paketes eine Antwort. Punkt 3 beschreibt die Daten im Header des Antwortpaketes. Dieses Paket wird nicht durch den Switch SW2 an beide Knoten zugestellt, sondern gelangt lediglich an N1, der die Rückübersetzung (Punkt 4) vornimmt. Die Server-Client Pakete unterliegen dadurch keiner Selektion, die bei NAT – wie oben beschrieben – zu Problemen führen kann.

Nachteilig ist nur, dass nicht exakt der gleiche Regelsatz auf beiden Knoten laufen kann, da die NAT-Regeln an die jeweilige IP-Adresse angepasst werden müssen (IP2a bzw. IP2b) und dass zwei IP-Adressen nach außen benötigt werden.

4.3 Diskussion der Lösungsvorschläge

Nachdem Vor- und Nachteile sowohl von Load Balancing mit Connection Tracking, als auch ohne Connection Tracking beleuchtet wurden, kann nun ein Vergleich der beiden Methoden angestellt werden.

	LB mit Conntrack	LB ohne Conntrack
Hohe Laufzeiteffizienz	Nein, Listenoperationen in ip_conntrack dauern lange.	Ja, besser als bei LB mit Hilfe von ip_conntrack.
Transparenz für Administration	nein (vorgegebenes Regeldesign).	ja
Mehrfachversand von Paketen ausgeschlossen	nein	ja
Multi Host Protokoll fähig[16]	ja	nein
NAT fähig	ja	ja

Tabelle 6: Vergleich der beiden beschriebenen Load Balancing Algorithmen.

Die ausschlaggebenden Argumente, die den Einsatz von Load Balancing ohne Conntrack sinnvoller machen, sind die ersten beiden. Zum einen ist eine gute Laufzeiteffizienz der Hauptgrund, warum man Load Balancing überhaupt anstrebt, zum anderen ist die Transparenz ein wichtiger Faktor zur Akzeptanz einer Firewall bei den Benutzern. Einschränkungen im Regeldesign schränken auch die Flexibilität der Firewall ein. Aus diesem Grund wurde im Modul preselect die Implementation von Load Balancing ohne Connection Tracking vorgenommen. Bei LB ohne Connection Tracking werden keine Multi Host Protokolle unterstützt. Auch eine Kombination von beiden gegenübergestellten Konzepten bringt hier keine Verbesserung. Verbindungen, bei denen es sein könnte, dass sie zu einem Multi Host Protokoll gehören könnten, würden ohne Selektion durch Preselect zur weiteren Verarbeitung zugelassen werden. Zu entscheiden, welche Pakete vermutlich zu einem Multi Host Protokoll gehören, läuft aber auf eine Heuristik hinaus, die durchaus auch fehlschlagen kann. Aus diesem Grund wurden auf solch eine Lösung in Preselect verzichtet.

[16] Protokolle an denen mehr als zwei Computer beteiligt sind (z.B. FXP)

5 Dokumentation der Kernel Module und Skripte

Im Wesentlichen lassen sich Module und Skripte in zwei Bereiche unterteilen. Zum einen in jenen der für die Aufteilung der Pakete auf ISO-OSI Layer 2 sorgt. Diese Aufgabe wird von Arpfake übernommen. Im Folgenden wird das Modul arpfake inklusive Skripte und Konfigurationsdateien stets als „Arpfake“ bezeichnet. Zum anderen existiert das Subsystem Preselect, das für die Selektion der Pakete an den einzelnen Subnetzen sorgt. Es besteht aus dem Module ip_preselect inklusive Resource Script und wird im Folgenden als „Preselect“ bezeichnet. Preselect arbeitet eng mit Heartbeat zusammen und sorgt neben der Selektion auch noch für ein ordnungsgemäßes Failover, falls ein Knoten ausfällt. Arpfake ist von Preselect unabhängig und kann auch für andere Einsatzgebiete verwendet werden, wo eine Aufteilung des Verkehrs auf mehrere Rechner erforderlich ist. Alle Teilkomponenten von Arpfake und Preselect wurden zunächst auf dem Linux Kernel in Version 2.6.10 getestet. Später wurde ein Backport auf Versionen der 2.4er Reihe durchgeführt.

5.1 Arpfake

5.1.1 Arpfake vs. Arptables

Arpfake sorgt für die Verteilung des Verkehrs auf alle Knoten. Die Theorie dazu wurde bereits in Abschnitt 4.2.1 erklärt. Diese beschriebenen Aufgaben könnten zum Teil auch von dem Kernel-Subsystem ARPTables erledigt werden. ARPTables ist IPTables nachempfunden und man kann damit Pakete des Address Resolution Protokolls [Ie82] filtern und verändern. Mit ARPTables ist es dadurch möglich, ausgehende ARP-Requests/Replies zu verändern, ganz so wie es im Falle der Unicast- und Multicast-Methode erforderlich ist. Für die Multicast Methode ist es aber zusätzlich noch erforderlich, die Netzwerkkarte auf den Empfang von Multicast Ethernet Frames zu programmieren, was von ARPTables nicht unterstützt wird. Darum wurden keine Target Extensions für ARPTables geschrieben, sondern ein eigenes Subsystem namens Arpfake implementiert. Ein eigenes Modul hat auch den Vorteil, dass bessere Transparenz dem Administrator gegenüber besteht. D.h. es gibt keine Regeln, die bereits in ARPTables vorhanden sind und versehentlich gelöscht werden könnten. Arpfake bietet also eine All-In-One Lösung. Bei Bedarf können aber die bestehenden Sourcen des Moduls arpfake in eine Target-Extension eingeflochten werden.

Arpfake besteht im Prinzip aus einem Kernel Modul namens arp_arpfake.o, das bei Bedarf geladen werden kann. Der Ladevorgang wird durch das Init-Script arpfake erleichtert. Wie jedes andere Init-Script kann es beim Hochfahren oder beim Umstieg in andere Runlevels geladen bzw. entladen werden. Es liest seine Konfiguration aus der Datei arpfake.conf, die es an das Kernel Modul arp_arpfake weitergibt. Diese Daten beschreiben, mit welcher MAC-Adresse ankommende ARP-Requests/Replies beantwortet werden sollen. Beim Stoppen des Scripts wird arp_arpfake einfach entladen und es erfolgt keine weitere Manipulation der ARP-Requests/Replies.

5.1.2 Struktur von Arpfake

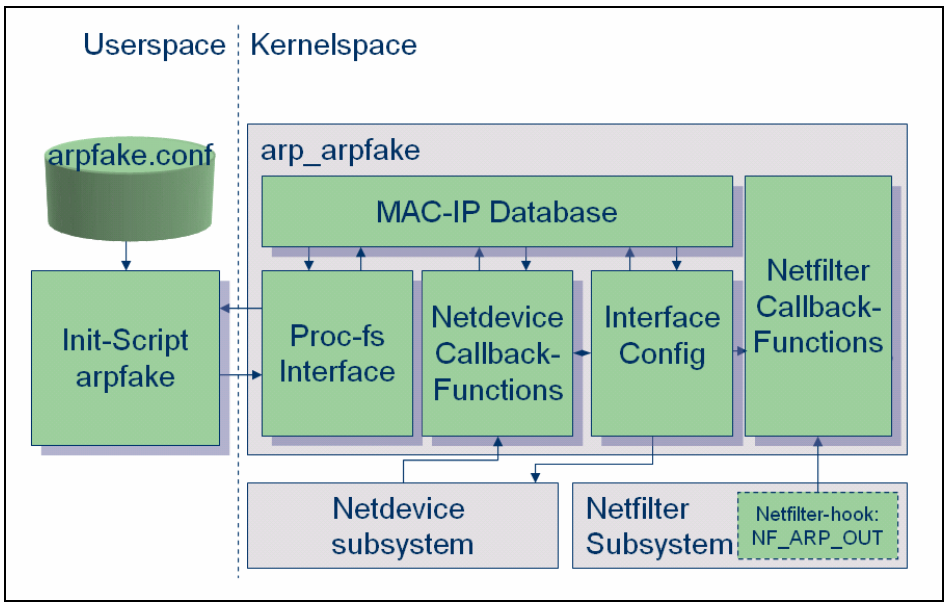

Abbildung 32: Interne Struktur von Arpfake.

Das Kernel Modul verwendet unter anderem Systemaufrufe des Nefilter-Frameworks. Neben den in Abschnitt 2.1 beschriebenen Hooks für IP-Pakete gibt es auch noch Hooks für das Address Resolution Protocol. Es wird dabei der Hook NF_ARP_OUT verwendet, der beim Versenden von ARP-Paketen durch den Kernel aktiv wird. Die dadurch aufgerufene Callback-Funktion ersetzt die ARP-Requests/Replies mit der MAC-Adresse, die in der MAC-Interface Database steht. Bei ARP-Requests/Replies wird lediglich das erste Feld nach dem ARP-Header – die „Sender Hardware Address“ – durch eine Multicast MAC-Adresse ersetzt, die in der MAC-IP Database steht.

Eine weitere Besonderheit sind die Netdevice Callback-Functions. Diese werden immer dann aufgerufen, wenn ein Ethernet-Interface ge- oder entladen wird oder sich die IP-

Adresse eines Interfaces verändert. Dadurch können neu installierte Interfaces ohne Neuladen von arp_arpfake unterstützt werden. Kommt ein neu installiertes Interface hinzu, so wird arp_arpfake sofort über sämtliche IP-Adressen informiert, die dieses Interface mit sich bringt. Standardmäßig wird aber keine Manipulation der ARP-Requests/Replies für diese neuen Interfaces durchgeführt. Diese Funktion muss vom Init-Script eingeschaltet werden.

Das Process File Interface

Die MAC-IP Datenbank hält jene MAC Adressen die für unterschiedliche IP-Adressen zur Manipulation der ARP-Requests/Replies verwendet werden sollen. Diese Daten können über das Process File Interface [Qu04; CorRub05] abgefragt und verändert werden. Eine Abfrage über Manipulationsdaten der IP-Adresse 10.0.0.1 kann folgendermaßen aussehen:

```
cat /proc/net/10.0.0.1
01:23:45:67:89:AB|disabled
```

Die Ausgabe bedeutet, dass ARP-Requests/Replies vom Interface, das die IP-Adresse 10.0.0.1 besitzt, mit der MAC-Adresse 01:23:45:67.89:AB manipuliert werden. „Disabled“ bedeutet, dass diese Manipulation aber nicht aktiv ist.
Eine Veränderung der MAC Adresse funktioniert wie folgt:

```
echo "01:23:45:67:89:AB" > /proc/net/10.0.0.1
```

Das Aktivieren der Manipulation erfolgt durch:

```
echo "1" > /proc/net/10.0.0.1
```

Das Ausschalten erfolgt analog mit „0“.
Um diesen Default-Wert auf „1“ zu ändern, muss beim Laden des Moduls der Parameter mangle=1 angegeben werden.

```
insmod ./arp_arpfake.o mangle=1
```

In Kernel-Builds, in denen kein Process File System vorhanden ist, kann arp_arpfake ebenfalls verwendet werden. Es können dann keine Veränderungen der MAC Daten vorge-

nommen werden. Es wird immer das Multicast-Bit in der MAC-Adresse des Interfaces gesetzt und mit dieser gearbeitet. Außerdem ist der Ladeparameter mangle=1 obligatorisch.

Multicast Filter bei Netzwerkkarten

Es ist zu empfehlen, Netzwerkkarten zu benutzen, bei denen Multicast-MAC Filter integriert sind. Diese erlauben eine Filterung von zumeist 16 Multicast MAC Adressen bereits auf der Netzwerkkarte. Dies spart Rechenzeit und fördert dadurch die Leistung des Systems. Die meisten Netzwerkkarten verfügen über einen solchen Filter. Von arp_arpfake wird er durch das Interface-Config Subsystem eingeschaltet.
Arpfake kann auch unabhängig von Preselect überall dort eingesetzt werden, wo eine Verteilung des Verkehrs auf mehrere Knoten erforderlich ist. Es besteht keine Verbindung zwischen Arpfake und Preselect. Bei der Verwendung von Hubs kann preselect auch ohne arpfake verwendet werden, wenn die Ethernet Interfaces im Promiscuous Mode sind.

5.2 Preselect

Preselect sorgt für die Verteilung des IP-Verkehrs über die einzelnen Knoten. Entsprechend dem Namen führt es eine Vorauswahl vor allen anderen Modulen, die den NF_IP_PRE_SELECT Hook nützen, durch. Die Selektion läuft so ab wie schon im Abschnitt 4.2.4 beschrieben.
Die zweite Aufgabe von Preselect ist es, ausreichende Funktionalität für ein Failover zu bieten, um Hochverfügbarkeit selbst bei Ausfall eines oder mehrerer Knoten zu erreichen. Dieses Thema soll im folgenden Abschnitt genauer erläutert werden.

5.2.1 Failover bei Preselect

Preselect läuft auf allen Knoten und lässt jedes Paket nur auf einem bestimmten Knoten passieren. Auf allen anderen wird es verworfen. Fällt dieser Knoten aus, so werden bestimmte IP-Adresspaare nicht mehr durch gelassen. Ein Failover der verbleibenden Knoten muss dafür sorgen, dass die Adresspaare von anderen Knoten übernommen werden. Es gibt mehrere Möglichkeiten, wie diese Adresspaare aufgeteilt werden können.

Selektion mit variablem Index und variabler Knotenanzahl

Die Hashfunktion h(ClientIP,ServerIP) liefert für einen Cluster mit N Knoten Werte von 1 bis N. In preselect läuft diese Berechnung so ab:

```
index = (h(ClientIP,ServerIP)%N)+1;
if index == localindex then
     return CONTINUE_PACKET;
else
     return DROP_PACKET;
```

Abbildung 33: Selektionsalgorithmus.

Zu dieser Berechnung benötigt man die Anzahl der aktiven Knoten (N) und den Index des lokalen Knotens. Beim Ausfall eines Knotens verändern sich genau diese beiden Parameter. Ändert sich aber die Zahl der Knoten und der lokale Index, so werden Pakete bestehender Verbindungen von anderen Knoten angenommen. Ohne Connection Tracking würde dies keine weiteren Probleme bereiten. Connection Tracking verlangt aber, dass die Verbindungen ihre gesamte Lebensdauer lang über einen Knoten laufen, weil nur auf diesem Knoten die Einträge in der Verbindungstabelle existieren. Würden die Pakete nach einem Failover über einen anderen Knoten laufen, so würde dieser die Pakete verwerfen.
Es ist also dafür Sorge zu tragen, dass die bestehenden Verbindungen über die ursprünglichen Knoten aufrechterhalten bleiben. Pakete neuer Verbindungen hingegen werden mit der neuen Anzahl von Knoten und dem neuen lokalen Index bewertet. Dies ist wie folgt zu realisieren. An jedem existierenden Eintrag in der Verbindungstabelle müssen folgende Daten gebunden sein:

- Der Index , den der Knoten zum Zeitpunkt des Aufbaus der Verbindung hatte.
- Die Gesamtanzahl der arbeitenden Knoten, die zum Zeitpunkt des Aufbaus der Verbindung aktiv waren.

Mit diesen Daten kann eine korrekte Zuteilung der Pakete an die Knoten erfolgen. Leider setzt dies voraus, dass ip_preselect Funktionen des Connection Tracking Moduls ip_conntrack verwendet, was aber – wie schon in Abschnitt 4.2.3 erläutert – zu Performanceeinbußen führt. Darum wurde in ip_preselect die folgende Lösung implementiert.

Selektion mit konstantem Index und konstanter Knotenanzahl

Eine andere Möglichkeit ist, Index und Knotenanzahl in der Berechnung immer konstant zu halten. Ermöglicht wird das durch die Einführung virtueller Knoten.
Ein Cluster mit vier Knoten erhält vier virtuelle Knoten, die er nach Belieben auf diese vier realen Knoten aufteilen kann. Im Normalbetrieb entspricht jeder virtueller Knoten einem realen Knoten. Fällt aber ein Knoten aus, so geht der frei werdende virtuelle Knoten auf einen anderen realen Knoten über. In Zukunft arbeitet also ein Knoten der drei verbleibenden für jenen, der ausgefallen ist und für seinen ursprünglichen virtuellen Knoten. So ändert sich in der Berechnung die Anzahl der Knoten nicht, und aufgebaute Verbindungen bleiben auf ihrem Platz. Nachteilig ist jedoch, dass durch die Mehrbelastung eines Knotens Leistungsengpässe entstehen können. Geht man jedoch davon aus, dass ein Cluster mit vier Knoten den Großteil der Zeit auch mit allen seinen vier Knoten in Betrieb ist, so scheint es sinnvoll, diesen Kompromiss einzugehen. Außerdem bietet sich die Möglichkeit, drei virtuelle Knoten pro realen Knoten zu definieren und diese im Falle eines Ausfalles auf die verbleibenden drei zu verteilen. Auch in verwandten Projekten wie Clusterip wird mit Selektion mit konstantem Index und konstanter Knotenanzahl verfahren. Da die Administration und Laufzeiteffizienz hier besser sind, wurde auch für Preselect dieser Lösungsansatz gewählt.

5.2.2 Systemarchitektur von Preselect

Abbildung 34 zeigt die einzelnen Systemkomponenten von Preselect und ihr Zusammenspiel mit Heartbeat. Ähnlich wie Arpfake besteht Preselect aus einem Kernel Modul und einem Ressource Script, welches hier aber nicht als normales Init-Script ausgeführt ist, sondern ein Ressource Script im Sinne von Heartbeat darstellt.

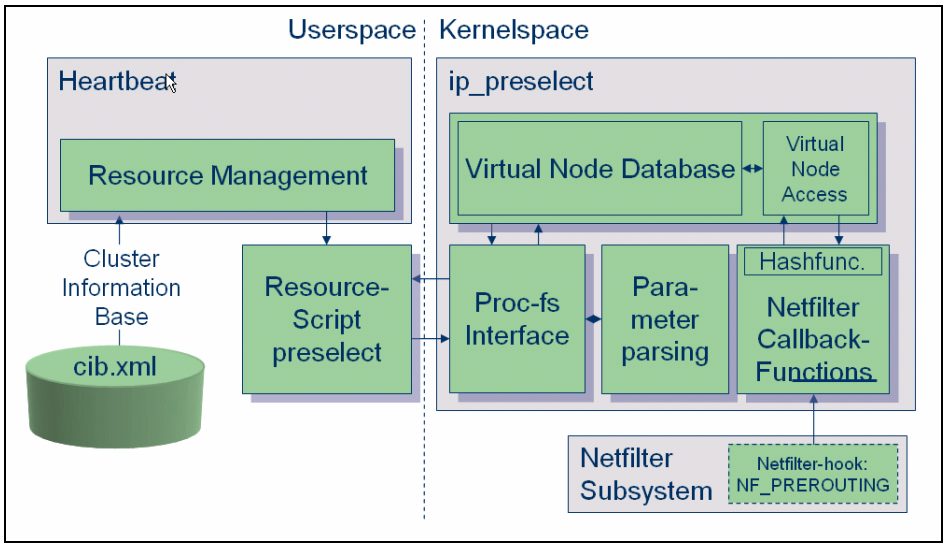

Abbildung 34: Interne Struktur von Preselect

Damit ip_preselect mit Prozessen vom Userspace kommunizieren kann, greift es auf das Process Filesystem zu. Das Kernel Modul ip_preselect verfügt dazu über eine Process Filesystem - Input/Output Schnittstelle.

Das Process Filesystem

In erster Linie wird das Process File System dazu benutzt, Anzahl und Indizes der am lokalen Knoten verwalteten virtuellen Knoten zu lesen und zu schreiben. Um auszulesen, welche Knotenindizes lokal verwaltet werden, ist wieder folgender Befehl notwendig:

```
cat /proc/net/preselect/nodes
1,2
```

Die Ausgabe im konkreten Fall zeigt die Verwaltung der virtuellen Knoten mit Index 1 und Index 2 an. D.h. der lokale Knoten nimmt alle Pakete an, bei denen die Selektion (h(ClientIP,ServerIP)%N)+1 eins oder zwei ergibt.
Im Falle eines Failovers des virtuellen Knoten 3 müsste der Index 3 lokal hinzugefügt werden. Dies kann wie folgt geschehen:

```
echo "+3" > /proc/net/preselect/nodes
```

Vom Zeitpunkt dieses Befehls an wird auch der virtuelle Knoten 3 lokal verwaltet. Analog dazu wird beim Entfernen eines virtuellen Knotens der gleiche Befehl mit dem Vorzeichen „minus“ verwendet.

Die Gesamtanzahl an virtuellen Knoten im Cluster (bei der Selektion als N bezeichnet) wird so ausgelesen:

```
cat /proc/net/preselect/totalnodes
3
```

In diesem Beispiel ist die Gesamtanzahl virtueller Knoten 3. Bei einer Erweiterung des Clusters auf mehr als drei Knoten kann auch dieser Wert vergrößert werden:

```
echo "4" > /proc/net/preselect/totalnodes
```

Ohne spezielle Einstellungen führt ip_preselect eine Selektion aller Pakete durch, die von außen in das System gelangen. Eine gezielte Administration der einzelnen Knoten würde dadurch unmöglich gemacht werden. SSH Verbindungen würden – abhängig von der Quelladresse – von manchen Knoten nicht akzeptiert werden. Das Modul bietet daher die Möglichkeit, manche Adressen von der Selektion auszuschließen. Um eine Liste der bereits ausgeschlossenen IP-Adressen zu erhalten, ist folgender Befehl zweckmäßig:

```
cat /proc/net/preselect/localips
10.0.0.11,10.0.0.12,10.0.0.255
```

Hier sollten all jene IP-Adressen angegeben werden, die den Knoten zur Administration dienen oder zum Austausch von Heartbeats und Synchronisationsdaten verwendet werden. Ein Hinzufügen von IP-Adressen ist mit diesem Befehl möglich:

```
echo "+10.0.0.13" > /proc/net/preselect/localips
```

Analog kann mit dem Vorzeichen „minus“ die betreffende IP-Adresse wieder entfernt werden.
Um bereits beim Laden des Moduls bestimmte Voreinstellungen treffen zu können, ist es möglich, dem Modul Ladeparameter zu übergeben. Folgende Parameter sind möglich:

totalnodes Anzahl virtueller Knoten, die insgesamt im Cluster verfügbar sind.

localenode Index des virtuellen Knotens, der lokal verwaltet werden soll. Eine Angabe einer Liste von Knoten ist nicht möglich.

ip_list ..... Liste von IP-Adressen, die von Preselect ignoriert werden sollen. Die IP-Adressen müssen mit einem Unterstrich voneinander getrennt sein. Es können bis zu 16 IP-Adressen angegeben werden. Anmerkung: Für Module der Kernel Version 2.6 müssen die IP-Adressen mit einem Beistrich anstatt mit einem Unterstrich getrennt werden.

Ein Beispiel, wie das Modul geladen werden kann:

```
insmod ./ip_preselect.o totalnodes=3 localenode=2 \
ip_list=10.0.0.11_10.0.0.12_10.0.0.13
```

Interne Struktur des Kernel Moduls

Die Funktionen des Process File Interfaces greifen direkt auf die Virtual Node Database zu, die eine lokale Sicht auf die lokal laufenden virtuellen Knoten hält. Um möglichst schnellen Zugriff auf die Daten zu erhalten – lesender Zugriff erfolgt bei jedem ankommenden Paket – wurden alle Daten in Arrays abgelegt. Das Netfilter Callback-Functions Subsystem greift auf das Netfilter Framework zurück. Es klinkt sich in den Hook NF_IP_PRE_ROUTING ein, sodass jedes Mal, wenn ein Paket vom Netz empfangen wird, die Callback-Funktion ausgeführt wird. Diese Callback-Funktion ruft ihrerseits die Hashfunktion in Netfilter Callback-Functions Subsystem auf. Sie wurden „inline“ definiert, um die Zeit für Sprung und Rücksprung in und aus der Funktion zu sparen. Bei der Berechnung des Hashwertes wurde auf den Jenkins Hash [Je00] zurückgegriffen. Die Berechnung ist performant und hat sich bereits in Projekten wie Clusterip bewährt.

Interaktion mit Heartbeat

Heartbeat kennt nur Dienste, die auf verschiedenen Knoten laufen. Ein Webservice kann z.B. auf drei Knoten laufen und teilt sich die anfallenden Requests. Bei Ausfall eines Knotens übernimmt einer der verbleibenden Knoten die Arbeit. Um die gleiche Arbeitsweise für Preselect zu erreichen, müssen die virtuellen Knoten von Heartbeat als Dienste aufgefasst werden. Verfügt also ein Cluster über drei reale Knoten und somit drei virtuelle Knoten, so werden diese von Heartbeat als Dienste aufgefasst. Alle Dienste heißen Preselect und haben eine Nummer, die dem Index des virtuellen Knoten entspricht. Der Dienst „Preselect1“ wird auf Knoten 1 gestartet, „Preselect2“ auf Knoten 2 und „Preselect3“ auf Knoten 3. Bei Ausfall von Knoten 1 geht Preselect1 auf einen der verbleibenden beiden Knoten

über. Welcher Knoten Preselect1 übernimmt, lässt sich durch Constraints in der Cluster Information Base bestimmen. Es kann z.B. bestimmt werden, dass Preselect1 immer auf Knoten 2, aber in keinem Fall auf Knoten 3 übergehen soll. Wird für den Ausfall eines Knotens keine Übernahmeregel angegeben, so wir der Dienst einfach dem Knoten mit den wenigsten bereits laufenden Diensten zugeordnet. Eine ausführlichere Behandlung des Themas ist unter Abschnitt 3.6 zu finden.

```
<rsc_location id="run_Preselect1" rsc="Preselect1">
    <rule id="pref_run_WebServerIP" score="100">
        <expression attribute="#uname" operation="eq" value="node1"/>
    </rule>
</rsc_location>
<rsc_location id="dont_run_Preselect1" rsc="Preselect1">
    <rule id="pref_dont_run_rsc" score="-INFINITY">
        <expression attribute="#id" operation="eq"
                    value="de937e3d-0309-4b5d-b85c-f96edc1ed8e3"/>
    </rule>
</rsc_location>
```

Abbildung 35: Beispiel von Constraints in der Cluster Information Base

Abbildung 35 zeigt einen Ausschnitt aus dem Constraints Abschnitt der Cluster Information Base. Preselect1 soll mit dem Score von 100 auf Node1 laufen, jedoch niemals auf dem Knoten mit ID de97ed-0309-4b5d-b85c-f96edc1ed8e3.

Auf diese Weise lassen sich relativ komplexe Regeln erstellen, die bestimmen, welche Ressource auf welchem Knoten wann laufen darf. Mit Blick auf die Betriebssicherheit des Clusters sollte aber die Komplexität der Constraints nicht zu groß werden [Ma04]. Sich widersprechende Regeln können leicht zum Abschalten eines Dienstes führen. Im folgenden Kapitel ist aus diesem Grund ein sehr einfaches Beispiel einer Cluster Information Base angeführt.

Um Heartbeat den Betrieb von Preselect zu gewährleisten, wurde ein Resource-Script erstellt, das dem Kernel Modul die benötigten Parameter zur Verfügung stellt. Das Script wird von Heartbeat mit den geeigneten Parametern aufgerufen und teilt dem Kernel Modul über das Process File System mit, für welche virtuellen Knoten es von jetzt an verantwortlich ist. Weiters ist das Script auch für das Laden und Entladen des Kernel Moduls ip_preselect.o verantwortlich, falls dieses noch nicht geladen wurde bzw. nicht mehr gebraucht wird.

6 Praktische Hinweise zur Integration in Gibraltar v2.3

Gibraltar ist eine auf Linux basierende Firewall. Es ist daher möglich, Preselect und Arpfake in Gibraltar zu integrieren. In diesem Kapitel werden Hinweise gegeben, die eine Integration in Gibraltar erleichtern sollen. Es kann auch unabhängig von den vorangegangenen Kapiteln verwendet werden. Ziel dieses Kapitel ist es, eine Grundlage für die Erstellung eines Webinterface zu bieten, da Preselect und Arpfake zunächst unabhängig von Gibraltar entwickelt und getestet wurden.

Preselect wurde unter Verwendung von Heartbeat 2.0.x entwickelt, in Gibraltar v2.3 kam jedoch Heartbeat 1.0.3 zum Einsatz. Der Vorteil von Heartbeat 2.0.x zu Heartbeat Release 1 besteht unter anderem in einem verbesserten Ressource Management und einer Unterstützung für mehr als zwei Knoten (siehe Abschnitt 3.5 und 3.6). Der entscheidende Vorteil von Heartbeat in Release 1 ist bessere Stabilität. Der Einsatz von Heartbeat 1.0.3 erscheint unter diesem Aspekt als sinnvoll, zumal Hochverfügbarkeit immer eng mit Stabilität verknüpft ist und ein Betrieb von mehr als zwei Knoten für eine kompakte Firewall wie Gibraltar unwahrscheinlich ist.

6.1 Zwei- Knoten vs. Mehr-Knoten-Betrieb

Obwohl Preselect und Arpfake für einen Mehr-Knoten-Betrieb konzipiert ist, wird hier aus oben genannten Gründen die Integration in einen Zwei-Knoten-Betrieb erläutert.

Anmerkung: Im Folgenden werden die einzelnen Dateien zur Integration beschrieben. Da sich zwischen Zwei-Knoten-Setup und Mehr-Knoten-Setup Unterschiede in der Konfiguration ergeben, wird bei Bedarf auf diese Unterschiede gesondert hingewiesen.
Unter Gibraltar v2.3 ist nur ein Zwei-Knoten-Setup verfügbar.
Unter Linux Distributionen mit Heartbeat 2.0.x ist auch ein Mehr-Knoten-Setup verfügbar.

6.2 Das Referenzsetup

Im Zuge der Tests von Preselect/Arpfake wurde folgendes Referenzsetup mit zwei Knoten erstellt.

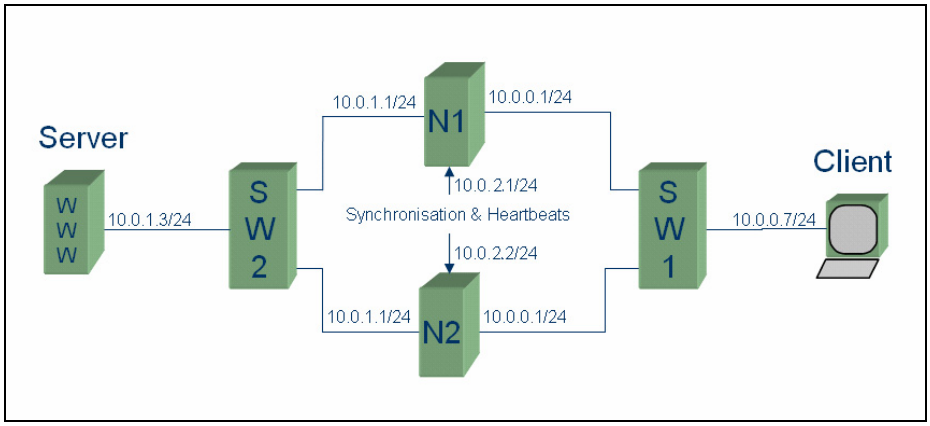

Abbildung 36: Referenzsetup für Gibraltar.

Zur besseren Nachvollziehbarkeit des Setups befinden sich die Konfigurationsimages der beiden Knoten N1 u. N2 auf CD-ROM im Verzeichnis **`/refsetup/images/node1/`** bzw. **`/refsetup/images/node2/`**.

Rechner	ClientPool-Netz		ServerPool-Netz		Heartbeat-Netz
Subnet	10.0.0.0/24		10.0.1.0/24		10.0.2.0/24
Interfaces	primär	virtuell	primär	virtuell	primär
Client	10.0.0.7	10.0.0.20-10.0.0.27 (für Tests)	Nicht verbunden	Nicht verbunden	Nicht verbunden
Node1	10.0.0.1	10.0.0.11 (Administration)	10.0.1.1	Keines	10.0.2.1
Node2	10.0.0.1	10.0.0.12 (Administration)	10.0.1.1	Keines	10.0.2.2
Server	Nicht verbunden	Nicht verbunden	10.0.1.3	10.0.1.20-10.0.1.27 (für Tests)	Nicht verbunden

Tabelle 7: Adressaufteilung für Referenzsetup.

Die Subnetzmasken wurden für alle Netze mit 24 Bit gewählt. Sowohl Client als auch Server verfügten bei den Tests über mehrere Internet-Adressen, um Funktion und Performance von Load Balancing zu testen.

Achtung! Betrifft alle Knoten: Interfaces in gleichen Netzen müssen immer die gleiche Bezeichnung tragen. So trägt z.B. im Referenzaufbau das Interface im ClientPool-Netz

sowohl am Node1 als auch am Node2 die Bezeichnung „eth0“. Um dies zu erreichen, kann das Kommandozeilen-Werkzeug **nameif** oder **ifrename** verwendet werden, oder die Änderungen können per Web-Interface durchgeführt werden. Dieser Schritt ist notwendig, um ein globales Regelwerk auf allen Knoten gültig zu halten.

6.3 Administration der Knoten

Node1 und Node2 wurden im ClientPool Netz mit einer zusätzlichen virtuellen IP-Adresse ausgerüstet, um eine selektive Administration der einzelnen Knoten zu erlauben. Grundsätzlich ist aber auch eine Konfiguration über die Adresse des Standardgateways (10.0.0.1 bzw. 10.0.1.1) möglich. Dabei hängt aber der tatsächlich angesprochene Knoten auch von der Source IP-Adresse des Webbrowsers ab! Aus diesem Grund sollte nur die virtuelle IP-Adresse zur Administration verwendet werden.
Um ein virtuelles Interface mit IP-Adresse 10.0.0.11 bzw. 10.0.0.12 zu erstellen, wurde im Referenzsetup folgender Eintrag in Datei **/etc/network/interfaces** eingefügt:

```
auto eth0:0
iface eth0:0 inet static
        address 10.0.0.11
        netmask 255.255.255.0
        network 10.0.0.0
        broadcast 10.0.0.255
```

Abbildung 37: Auszug aus der Datei /etc/network/interfaces

Auf Node2 existiert der gleiche Eintrag mit IP-Adresse 10.0.0.12. Bei einem Mehr-Knoten-Setup muss dieser Eintrag auf allen Knoten vorhanden sein.
Jeder Knoten kann somit für sich konfiguriert werden und die Änderungen können anschließend bei Bedarf auf alle anderen Knoten per **/etc/preselect/sync-etc.sh** verteilt werden.

6.4 Verteilung der Konfiguration

Um die Verteilung der Konfiguration auf andere Knoten zu gewährleisten, müssen folgende Dateien konfiguriert sein:

- **/etc/preselect/preselect.conf** muss eine Liste von IP-Adressen aller Knoten im Cluster beinhalten (für Zwei-Knoten-Setup also nur zwei Einträge). Üblicherweise werden die IP-Adressen des Synchronisationsnetzes angegeben. Ein

gültiger Eintrag beginnt mit dem Schlüsselwort **node**, einem oder mehreren Whitespaces und der IP-Adresse des Synchronisations-Interfaces des jeweiligen Knotens. Alternativ kann auch ein DNS-Hostname verwendet werden, wenngleich aus Gründen der Hochverfügbarkeit ausdrücklich davon abgeraten wird. Kommentare beginnen mit dem Zeichen **#** und enden mit dem Zeilenende.

Achtung: Fehlen Knoten in dieser Datei, so sind diese von der Synchronisation ausgeschlossen! Ein Beispiel für eine solche preselect.conf Datei ist auf CD-ROM unter **/refsetup/node1/preselect/preselect.conf** zu finden.

- **/etc/preselect/sync-exclude** muss eine Liste von Dateien und Verzeichnissen beinhalten, die nicht synchronisiert werden sollen. Üblicherweise enthält sie Dateien, die Hardware-spezifisch sind und von Knoten zu Knoten anders aussehen müssen (z.B. **/etc/modules**). Sie sollte aber auch Dateien wie Random-Seeds, Zertifikate und Public- bzw. Private Keys beinhalten. Da sync-exclude je nach Art der Anwendung stark variieren kann, obliegt es dem Benutzer, sie ordnungsgemäß zu erstellen und zu warten. Ein Beispiel für eine sync-exclude Datei ist unter **/refsetup/node1/preselect/sync-exclude** zu finden.
- **/etc/preselect/passwordless.sh**. Dieses Script muss pro Knoten nur einmal am Beginn der Konfiguration ausgeführt werden. Es ermöglicht das gegenseitige Einloggen der einzelnen Knoten über das Synchronisationsnetz. Damit es funktioniert, muss **preselect.conf** gültig konfiguriert sein (siehe oben).
- **/etc/preselect/sync-etc.sh**. Für eine Synchronisation wird dieses Script genau dann ausgeführt, wenn Änderungen in der Konfiguration eines Knotens vorgenommen wurden. Damit es funktioniert, müssen **preselect.conf** und **sync-exclude** gültig konfiguriert sein, sowie Passwordless-Login eingerichtet worden sein (z.B. mittels **passwordless.sh**).

Achtung: Nach einer Synchronisation muss auch noch darauf geachtet werden, dass die veränderten Dienste neu gestartet werden, um die Änderungen auch auf allen Knoten wirksam zu machen. Da dieser Schritt eng mit der Web-Oberfläche verbunden ist, wurde auf ein entsprechendes Script verzichtet. Es wird vorgeschlagen, dass jedes Mal, wenn auf einen Save-Button geklickt wird, zuerst eine Synchronisation mit **sync-etc.sh** stattfindet und dann ein entsprechendes Script

ausgeführt wird. Dieses Script führt betroffene Dienste auf allen Knoten, die in **`preselect.conf`** aufgeführt sind, neu aus. D.h. es wird z.B. beim Ändern einer Regel im Firewall Regelwerk das Script **`firewall.sh`** auf allen Knoten ausgeführt.

6.5 Konfiguration des ISO-OSI Layer 2

Der gesamte ankommende Verkehr muss auf alle Knoten im Cluster verteilt werden. Dies geschieht mittels „Fälschung“ der ARP-Requests/Replies seitens der Clusterknoten. Um den Switch in den Flooding Modus zu versetzen, senden alle am Firewall Cluster angeschlossenen Rechner per Multicast ihre gefälschte MAC-Adresse. Dies wird durch Modifikation von ARP-Requests und ARP-Replies erreicht, die von den Knoten gesendet werden. Eine genaue Erklärung der Theorie findet sich auch im Abschnitt 4.2.1.

6.5.1 Funktion von arpfake

Das dafür verantwortliche Kernel Modul heißt **`arp_arpfake.o`** und befindet sich im Verzeichnis **`/etc/arpfake/`**. Arpfake scannt automatisch nach allen im Knoten verfügbaren Ethernet Interfaces, die ARP verwenden und verwaltet diese.

6.5.2 Funktion des Init-Scripts arpfake

Mit **`/etc/init.d/arpfake start`** wird arp_arpfake.o geladen bzw. mit Parameter **`stop`** entladen. Beim Hochlauf von Gibraltar kann arpfake zum gewünschten Zeitpunkt geladen werden. Dieser sollte so gewählt sein, dass bereits alle Netzwerk Interfaces vorhanden sind. Das Script liest außerdem die Datei **`/etc/arpfake/arpfake.conf`**. In dieser Datei befindet sich eine Tabelle von IP-Adressen und Multicast MAC-Adressen. Jede Zeile sollte ein Firewall-Netz repräsentieren. Die erste Spalte enthält die Adresse, die in diesem Netz als Standardgateway fungiert. Die zweite Spalte enthält eine Multicast MAC-Adresse, die einmalig in allen verwendeten Netzen sein soll. Ein Beispiel für eine mögliche arpfake.conf ist auf der CD-ROM unter:
`/refsetup/node1/arpfake/arpfake.conf` zu finden. Die in arpfake.conf vorhandenen Informationen werden vom Init-Script an das Kernel Modul weitergegeben, so-

dass dieses die ARP-Requests/Replies mit den geeigneten Multicast-MAC Adressen fälschen kann.

Anmerkung: Das Init-Script arpfake wird im beschriebenen Referenzaufbau nicht mittels Runlevels aufgerufen, sondern muss manuell aktiviert werden. Für Folgeversionen von Gibraltar v2.3 wird jedoch eine Integration in die Startup Scripts empfohlen.

6.6 Konfiguration von Heartbeat in Verbindung mit preselect

Ein von arpfake unabhängig einsetzbares Subsystem stellt Preselect dar. Es arbeitet eng mit dem Ressource Manager von Heartbeat zusammen und sorgt für Load Balancing unter den Clusterknoten. Eine detaillierte Beschreibung der theoretischen Konzepte von Preselect finden sich unter 5.2, sowie von Heartbeat unter 3.5 und 3.6.
Im Folgenden wird daher die Funktionsweise von preselect nur kurz umrissen:
Preselect nimmt auf jedem Clusterknoten eine Auswahl der ankommenden IP-Pakete vor. So wird eine Aufteilung des IP-Verkehrs auf alle Knoten erlangt. Fällt ein Knoten aus, so wird ein Teil der Pakete nicht mehr angenommen. Durch Heartbeat gesteuert übernimmt nach kurzer Zeit ein anderer Knoten die Annahme dieser Pakete zusätzlich zu seinen eigenen. Ist der Cluster also für drei Knoten konfiguriert, so existieren immer drei Dienste. Fällt ein Knoten aus, so wandert der Dienst von diesem Knoten zu einem anderen. Bei einem Zwei-Knoten-Setup sind auf Grund des Konzeptes nur zwei Knoten – also zwei Dienste – möglich. Jedem Dienst ist eine Nummer zugeordnet. Bei drei Preselect Diensten existieren also Preselect1, Preselect2 und Preselect3.

6.6.1 Konfiguration von Preselect

Um ip_preselect.o über Heartbeat wie oben beschrieben zu steuern, benötigt man ein Ressource-Script. Dieses heißt **`/etc/heartbeat/resource.d/preselect`**. Ähnlich wie ein Init-Script lässt sich preselect mit Parameter „start“ oder „stop“ aufrufen. Damit startet/stoppt man den Dienst. Zusätzlich dazu muss dem Script per Parameter mitgeteilt werden, welcher Preselect Dienst gestartet werden muss. Die Gesamtanzahl der Dienste muss ebenfalls mitgegeben werden. Will man auf einem bestimmten Knoten also den Dienst Nummer 2 von insgesamt drei im Cluster laufenden Diensten stoppen, so lautet der Befehl:

```
/etc/heartbeat/resource.d/preselect 2 3 none stop
```

Der erste Parameter bezeichnet den gemeinten Dienst, der zweite die Totalanzahl an Diensten, der dritte wird weiter unten in diesem Abschnitt erklärt und der vierte bestimmt, was mit dem Dienst gemacht werden soll. Neben einem Start oder Stopp existiert auch noch die Möglichkeit mit **`status`** den Status des Dienstes abzufragen:

```
/etc/heartbeat/resource.d/preselect 2 3 none status
```

Der dritte Parameter besteht aus einer Liste von mit einem Unterstrich getrennten IP-Adressen, die von Preselect immer angenommen werden sollen. Alternativ dazu kann auch **`none`** angegeben werden, um keine Ausnahmen zu erlauben.
Diese Befehle können zwar händisch ausgeführt werden, werden aber normalerweise nur von Heartbeat getätigt.

Achtung: Ein Ausführen von Preselect mit inkorrekten Parametern kann ein Denial-Of-Service nach sich ziehen. Es muss unbedingt auf passende Parameter geachtet werden. Im Zweifelsfall muss lokal am Knoten gearbeitet werden.

6.6.2 Konfiguration von Heartbeat

Eine Beschreibung der Theorie von Heartbeat ist unter Abschnitt 3.5 und 3.6 zu finden.
Eine wichtige Entscheidung ist, wie die Heartbeats übertragen werden sollen. Heartbeat benötigt mindestens eine Möglichkeit, das Signal zu übertragen. Für die aktuelle Version existieren die Möglichkeiten:

- Seriell via Nullmodemkabel.
- Ethernet mittels Crossoverkabel.
- Ethernet verbunden mit Switch oder Hub, meist zur Realisierung größerer Cluster.

Im Hinblick auf High Availability sollten mehrere Übertragungsarten gleichzeitig gewählt werden. Dadurch kann ein SPOF – nämlich der Heartbeat Übertragungskanal – ausgeschlossen werden. Für den vorliegenden Fall bietet sich die Kombination von serieller Übertragung und Ethernet an. Nullmodem Übertragung ist die am fehlerunanfälligste; Ethernet kann zusätzlich zu den Heartbeats für die Übertragung der Synchronisationsdaten genutzt werden. Für den Fall, dass mehrere Knoten in Betrieb sind, scheidet die Übertragung über Crossoverkabel aus.

Achtung: Es ist möglich, die Heartbeats über das ClientPool-Netz auszutauschen. In der Praxis sollte aber – erstens aus Gründen der Sicherheit, zweitens aus Gründen der Hochverfügbarkeit – immer zusätzlich ein eigenes VLAN für Heartbeats verwendet werden. Dieses Netz muss abhörsicher sein, sodass Heartbeats nicht abgefangen oder gefälscht werden können.

Um Heartbeat starten zu können, ist es notwendig, dieses zuvor zu konfigurieren.

Konfiguriert werden müssen folgende Dateien:

- `ha.cf`
- `authkeys`
- `haresources` für Zwei-Knoten-Setup
- `cib.xml` für Mehr-Knoten-Setup

Mit Ausnahme von cib.xml befinden sich diese Dateien im Ordner **`/etc/heartbeat/`**.

Kurzbeschreibung der Konfigurationsdatei ha.cf

Diese Konfigurationsdatei beinhaltet alle Einstellungen, die von Knoten zu Knoten verschieden sein können. Die Beispieldatei des Referenzsystems (in diesem Fall ist sie auf beiden Knoten gleich) befindet sich unter **`/refsetup/node1/heartbeat/ha.cf`**.

`serial /dev/ttyS0`

Verwenden des seriellen Heartbeatkanals. Weitere serielle Anschlüsse sind wie folgt benannt: **`/dev/ttyS1, /dev/ttyS2,`** etc... . **`/dev/ttyS0`** entspricht in der Windows-Welt dem COM1-Port. Werden mehrere Einträge „serial“ vergeben, so wird über jedes einzelne empfangen/gesendet. Beim Mehr-Knoten-Setup müssen die Knoten per serieller Schnittstelle verkettet werden. Die Heartbeats werden dann in zwei Richtungen von Knoten zu Knoten weitergereicht (ähnlich der Arbeitsweise eines Tokenrings).

`bcast eth0`

Verwenden des Ethernet Interfaces eth0. Mit diesem Eintrag wird Heartbeat veranlasst, die Heartbeats per UDP-Broadcasts über die Schnittstelle eth0 zu schicken. Beim Mehr-Knoten-Setup empfiehlt sich die Verwendung dieses Eintrags.

`ucast eth0 10.0.2.1`

Ist nur für ein Zwei-Knoten-Setup sinnvoll (also für Gibraltar v2.3 möglich). In diesem Beispiel bezeichnet eth0 die Schnittstelle, über die die Heartbeats gesendet werden sollen und 10.0.2.1 die Destination IP-Adresse, die die gesendeten Heartbeats besitzen sollen.

`watchdog /dev/watchdog`

Watchdog ist ein Timer, der den eigenen Rechner neu startet, falls dieser eine Minute lang kein Heartbeatsignal mehr sendet.

Um diese Funktionalität zu nutzen ist es notwendig, zuvor das softdog Kernel Modul zu laden "**`insmod softdog`**". Als nächstes folgt die Eingabe **`"grep misc /proc/devices"`** welche als Output eine Zahl liefert (vermutlich 10). Als nächstes gibt man ein **`"cat /proc/misc | grep watchdog"`** und notiert sich dessen Antwort (vermutlich 130). Mit diesen Informationen kann dann das Node-File erzeugt werden **`"mknod /dev/watchdog c 10 130"`**.

`keepalive 1`

Zeit zwischen den gesendeten Heartbeatsignalen in Sekunden. Mit dem Postfix „ms“ können auch Millisekunden angegeben werden.

`warntime 10`

Zeit nach der eine “late heartbeat-Warnung“ in die Logdatei ausgegeben wird.

`deadtime 20`

Falls nach dieser Zeit kein Heartbeat empfangen wurde, so gilt der betreffende Knoten als ausgefallen. Falls der lokale Knoten die ausgefallenen Ressourcen übernehmen kann, werden alle nötigen Schritte zur Ressource Übernahme eingeleitet.

`initdead 40`

Bei manchen Konfigurationen benötigt das Netzwerk einige Zeit, um nach einem Neustart seine Arbeit aufzunehmen. Für diesen Fall gibt es die initdead Zeit, welche mindestens doppelt so groß wie die deadtime sein soll.

hopfudge 1

Gibt für Ringtopologien mit der seriellen Schnittstelle die Anzahl der erlaubten Hops an (abhängig von der Anzahl der Knoten im Cluster). Diese Funktion wird nicht für das Zwei-Knoten-Setup gebraucht.

baud 19200

Geschwindigkeit der seriellen Verbindung in bps. Sie sollte nur so hoch wie nötig eingestellt werden, um Übertragungsfehler zu vermeiden.

udpport 694

Verwendete Portnummer bei einer bcast bzw. ucast Verbindung. Der Standardport ist 694 und sollte nur für Spezialanwendungen verändert werden.

nice_failback off

Achtung: Nur für Zwei-Knoten-Setup. Für ein Active/Active Setup, wie es im Falle von Preselect erforderlich ist, sollte nice_failback stets auf off gesetzt werden. Genauere Informationen sind aus der Heartbeat Dokumentation zu entnehmen (http://www.linux-ha.org).

auto_failback legacy

Achtung: Nur für Mehr-Knoten-Setup. Im Falle von Preselect übernimmt der Cluster Ressource Manager die Failback Strategie. Auto_failback ist wirkungslos, sollte aber laut Heartbeat Dokumentation auf „legacy“ gesetzt sein. Genauere Informationen sind aus der Heartbeat Dokumentation zu entnehmen, die unter der Adresse http://www.linux-ha.org/ha_2ecf_2fAutoFailbackDirective zu finden ist.

node node1

node node2

Liste von Knoten im Cluster. Für ein Zwei-Knoten-Setup werden zwei Node Einträge benötigt. Für ein Mehr-Knoten-Setup entsprechend mehr. Die Knotenbezeichnung (hier node1/node2) muss dem Output des Befehls **„uname -r“** auf dem jeweiligen Knoten entsprechen.

stonith

Diese Direktive ist nur in Heartbeat Release 1 verwendet. Das Stonith Subsystem von Heartbeat ermöglicht es, durch den Einsatz von Spezialhardware den ausgefallenen Knoten, der den Cluster möglicherweise in einen inkonsistenten Zustand führt, abzuschalten. Da entsprechende Spezialhardware nicht zur Verfügung stand, wurde im Referenzaufbau von dieser Möglichkeit nicht Gebrauch gemacht. Es existiert also kein stonith Eintrag. Dennoch sollten im normalen Betrieb Spezialhardware wie z.B. Network Power Switches zum Abschalten der ausgefallenen Knoten eingesetzt werden. Nähere Informationen hierzu unter http://linux-ha.org/ha_2ecf_2fStonithDirective.

coredumps true

crm yes

Diese Einträge werden nur für ein Mehr-Knoten-Setup benötigt. Sie sind erst in Heartbeat Release 2 verfügbar und dürfen in früheren Versionen wie z.B. Heartbeat 1.0.3 nicht verwendet werden. Um coredumps Unterstützung zu erhalten, wird coredumps auf true gesetzt. Genauere Informationen sind aus http://www.linux-ha.org/GettingCoreDumps zu entnehmen.

Um den Cluster Ressource Manager zu verwenden, wird crm auf yes gesetzt. Der Cluster Ressource Manager erlaubt ein komplexes Resource-Management über mehrere Knoten hinweg. Um erweiterte Unterstützung für Resource-Management auf Basis der cib.xml zu erhalten, muss diese Direktive gesetzt werden.

Kurzbeschreibung der Konfigurationsdatei authkeys

Die zweite Datei, welche konfiguriert wird, behandelt die Authentifizierung der Heartbeats. Diese werden in authkeys festgelegt. Aus Sicherheitsgründen muss diese Datei root gehören und darf nicht World-Readable sein. Dies wird durch folgende Kommandozeilenbefehle erreicht:

```
chown root.root /etc/heartbeat/authkeys
chmod 600 /etc/heartbeat/authkeys
```

Bei Heartbeat stehen dabei drei Arten der Authentifizierung zur Auswahl:

- **CRC** ist die schnellste Art, die Verwendung bei einem sicheren Netzwerk finden sollte.
- **MD5** stellt einen Kompromiss aus Geschwindigkeit und Sicherheit dar.
- **SHA1** ist die sicherste Authentifizierung unter Heartbeat, belastet aber die CPU stark.

Der Dateieintrag hat wie folgt auszusehen:

```
auth <number>
<number> <authmethod> [<authkey>]
```

```
auth 1
1 sha1 PutYourSuperSecretKeyHere
```

Abbildung 38: Beispiel einer einfachen authkeys-Datei.

Egal, welcher Index nach dem Schlüsselwort auth verwendet wird: Wichtig ist, dass genau dieser auch vor der Authentifizierungsmethode steht!

Kurzbeschreibung der Konfigurationsdatei haresources

Verwendet man ein Zwei-Knoten-Setup, benötigt man die Datei haresources. Sie legt fest, welcher Preselect Dienst welchem Knoten zugeordnet ist.

Achtung: Diese Datei muss auf allen Knoten identisch sein!

Von hier können Init- und Resource-Scripts aufgerufen werden. Im vorliegenden Fall muss das Resource-Script Preselect mit den Parametern wie in Abschnitt 6.6.1 beschrieben aufgerufen werden.

Der Syntax für die Datei haresources ist folgendermaßen:

`nodename  scriptname{::parameter}`

Im Referenzsetup lauten die Einträge folgendermaßen:

```
node1     preselect::1::2::10.0.0.11_10.0.0.12_10.0.0.255
node2     preselect::2::2::10.0.0.11_10.0.0.12_10.0.0.255
```

Abbildung 39: Die Datei haresources des Referenzsystems

In der ersten Spalte befinden sich die Bezeichnungen der Knotennamen so wie sie der Befehl „`uname -r`“ auf den jeweiligen Knoten liefert. Dann folgt – für diesen Anwen-

dungsfall obligatorisch – der Name des Resource Scripts „preselect“. Ähnlich wie in Abschnitt 6.6.1 beschrieben steht in der nächsten Spalte die Dienstnummer, die dem jeweiligen Knoten zugeordnet wurde. Spalte 4 hält die Anzahl der im Cluster laufenden Dienste von Preselect. Diese Zahl muss in jeder Zeile gleich sein. Der letzte Parameter bezeichnet eine Liste von IP-Adressen, die von Preselect nicht beachtet werden sollen. Werden Heartbeats über Ethernet Interfaces mit der ucast Methode ausgetauscht (siehe Abschnitt „Kurzbeschreibung der Konfigurationsdatei ha.cf“), so ist es unbedingt erforderlich, die Interface-Adressen der einzelnen Heartbeat Interfaces anzuführen. Die Heartbeats selbst sollen ja keiner Selektion durch Preselect unterliegen. Wird die Methode mit bcast verwendet (siehe Abschnitt „Kurzbeschreibung der Konfigurationsdatei ha.cf“), so ist die Broadcast-Adresse des Heartbeat Netzes anzuführen. Wenn keine Heartbeats über Ethernet ausgetauscht werden, kann auch das Schlüsselwort „`none`“ angegeben werden.

Kurzbeschreibung der Konfigurationsdatei cib.xml

Diese Datei existiert nur für Heartbeat Release 2 bei eingeschaltetem Cluster Ressource Management (ha.cf: crm yes). In diesem Fall muss die Datei haresources leer sein. An ihre Stelle tritt die Datei **`/var/lib/heartbeat/crm/cib.xml`**. Anders als in der haresources können in der Cluster Information Base (CIB) auch Constraints formuliert werden, die Abhängigkeiten zwischen einzelnen Ressourcen definieren. Entsprechend der Erweiterung handelt es sich um eine XML-Datei. Eine genauere Beschreibung findet sich unter Abschnitt 3.6.

Es erfolgt die Erklärung jener **`cib.xml`**, die für die Tests im Mehr-Knoten-Setup benutzt wurde.

```
<cib>
  <configuration>
    <crm_config>
      <nvpair id="require_quorum" name="require_quorum" value="false"/>
      <nvpair id="symmetric_cluster" name="symetric_cluster"
                                                        value="true"/>
      <nvpair id="transition_timeout" name="transition_timeout"
                                                       value="30000"/>
      <nvpair id="suppress_cib_writes" name="suppress_cib_writes"
            value="true"/>
    </crm_config>
    <nodes/>
    <resources>
      <primitive id="rsc_PreSelect1" class="heartbeat" type="preselect"
                                                    provider="heartbeat">
```

```
        <instance_attributes>
          <attributes>
            <nvpair name="1" value="1"/>
            <nvpair name="2" value="2"/>
            <nvpair name="3"
                  value="10.0.0.11,10.0.0.12,10.0.0.13,10.0.0.255"/>
          </attributes>
        </instance_attributes>
      </primitive>
      <primitive id="rsc_PreSelect2" class="heartbeat" type="preselect"
                                                    provider="heartbeat">
        <instance_attributes>
          <attributes>
            <nvpair name="1" value="2"/>
            <nvpair name="2" value="2"/>
            <nvpair name="3" value=
                          "10.0.0.11,10.0.0.12,10.0.0.13,10.0.0.255"/>
          </attributes>
        </instance_attributes>
      </primitive>
    </resources>
    <constraints>
      <rsc_location id="run_rsc_PreSelect1" rsc="rsc_PreSelect1">
        <rule id="pref_run_rsc_PreSelect1" score="100">
          <expression attribute="#uname" operation="eq" value="node1"/>
        </rule>
      </rsc_location>
      <rsc_location id="run_rsc_PreSelect2" rsc="rsc_PreSelect2">
        <rule id="pref_run_rsc_PreSelect2" score="100">
          <expression attribute="#uname" operation="eq" value="node2"/>
        </rule>
      </rsc_location>
    </constraints>
  </configuration>
  <status/>
</cib
```

Abbildung 40: Die Cluster Information Base des Referenzsetups.

Für die Konfiguration sind vor allem die drei Abschnitte crm_config, ressources und constraints wichtig.

Der Abschnitt crm_config

Der Abschnitt crm_config beinhaltet folgende Name/Value Paare:

- **`require_quorum=true`** Um Aktionen ausführen zu können, müssen mindestens 50% aller Knoten vorhanden sein.
- **`symmetric_cluster=true`** Ressourcen dürfen (falls in der constraints section nicht anders angegeben) auf jedem Knoten laufen.
- **`transition_timeout=30000`** Zeit, nach der eine (Ressource)-Transition als gescheitert gilt.

- **`suppress_cib_writes=true`** Ergänzungen des Cluster Ressource Manager werden nicht in die Datei zurück geschrieben, sondern nur im Speicher gehalten.

Der Ressource Abschnitt

Der Ressource Abschnitt definiert die im Cluster vorhandenen Preselect Ressourcen. An jeder Ressource hängen verschiedene Eigenschaften.

- Eine eindeutige ID, wie hier „rsc_PreSelect“.
- Eine Klasse der Ressource. Neben „heartbeat“ ist auch noch „OCF“ oder „LSB“ möglich. „heartbeat“ bedeutet, dass das Script ein Ressource Script ist. „LSB“ würde z.B. bedeuten, dass es sich im ein Init Script handelt.
- Der Typ der Ressource bezeichnet den Namen des Scripts, das in Verzeichnis /etc/heartbeat/resource.d/ steht. In diesem Fall ist er „preselect“.
- Der Anbieter der Ressource kann ein beliebiger Bezeichner sein.

Zusätzlich kann über „Attributes“ bestimmt werden, mit welchen Parametern das Script aufgerufen werden soll. Hier sei auf Abschnitt 6.6.1 verwiesen, wo diese beschrieben sind. Da in Ressource Scripts der Klasse Heartbeat nur „positional parameters“ erlaubt sind, gibt „name“ immer die Reihenfolge der Parameter an und „value“ den tatsächlichen Wert.

Der Constraints Abschnitt

Constraints ermöglichen die Angabe von ausgefeilten Abhängigkeiten zwischen verschiedenen Ressourcen. Für Preselect wurden lediglich einfache Constraints definiert, weil keine Abhängigkeiten zwischen den Dienstnummern existieren.
Die Resource Locations (RL) bestimmen, wo eine Dienstnummer im Normalfall zu laufen hat. Mit Scores können Gewichtungen vorgenommen werden. Jede RL hat bestimmte Eigenschaften:

- Eine eindeutige ID für die RL. Hier wurde sie mit run_<ressource-ID> angenommen.
- Die Ressource, für die dieser Constraint gelten soll.
- Eine Regel mit Gewichtung, die logisch verknüpfte Ausdrücke enthalten kann. Die Gewichtung wurde hier für alle gleich gewählt. Je höher ein Gewichtungsscore einer RL ist, desto eher gewinnt er gegenüber anderen RL. Eine Score von 0 ist De-

fault. Es sind auch Scores von „INFINITE“ oder „-INFINITE“ möglich. Regeln mit Ausdruck in der vorliegenden cib.xml sagen aus, dass Ressource rsc_PreSelect1, falls möglich, immer auf node1 laufen soll, rsc_PreSelect2 immer auf node2.

7 Zusammenfassung der Ergebnisse und Ausblick

Neben der Implementierung von Arpfake und Preselect wurden auch Funktions- und Leistungstests durchgeführt. Um eine Rekonstruktion der durchgeführten Tests zu ermöglichen, werden die Setups hier beschrieben.

7.1 Funktionstests für Failover

Um die Funktion von Failover zu testen, wurde ein Cluster mit drei Knoten aufgebaut. Dieser war mit zwei Netzen verbunden, einem ClientPool Netz und einem ServerPool Netz. Heartbeats und Synchronisation wurden ebenfalls über das ClientPool Netz durchgeführt.

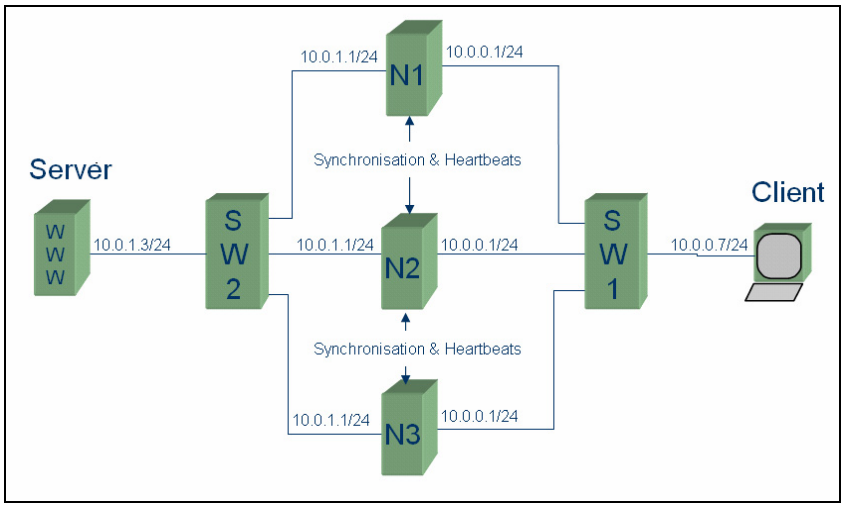

Abbildung 41: Setup mit drei Knoten.

Zur Simulation wurden auf dem Servernetz von einem Apache Webserver mehrere Testdateien angeboten, die von der Clientseite mittels mehrerer Prozesse des Tools wget [Fsf05] gleichzeitig herunter geladen wurden. Es wurde darauf geachtet, dass die unterschiedlichen Prozesse nicht alle die gleichen Source- und Destination IP-Adressen benutzen.

Da für acht Clients und acht Webserver zu wenig Hardware zur Verfügung stand, wurden mit einem Rechner im ClientPool Netz und einem Rechner im ServerPool Netz jeweils acht Clients/Server simuliert. Dafür wurde sowohl im ServerPool als auch im ClientPool Netz jeweils ein Rechner angeschlossen. Am Client-Rechner liefen acht Client-Prozesse, die mit jeweils acht Source IP-Adressen HTTP Requests an die Server schickten. Jeder

dieser Prozesse kommunizierte über eine eigene Source IP-Adresse zu unterschiedlichen Server IP-Adressen. Am Server-Rechner lief ein Apache Webserver, mit dem über acht verschiedenen IP-Adressen kommuniziert werden konnte.

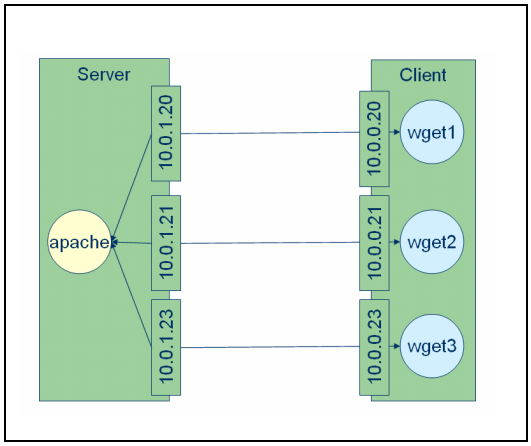

Abbildung 42: Prozessaufteilung am Client und Server

Auf diesem Rechner wurden acht virtuelle Interfaces angelegt. Weiters wurden acht Prozesse gestartet, die jeweils über eines dieser virtuellen Interfaces kommunizierten. So kommunizierten unterschiedliche Client IP-Adressen mit unterschiedlichen Server IP-Adressen.

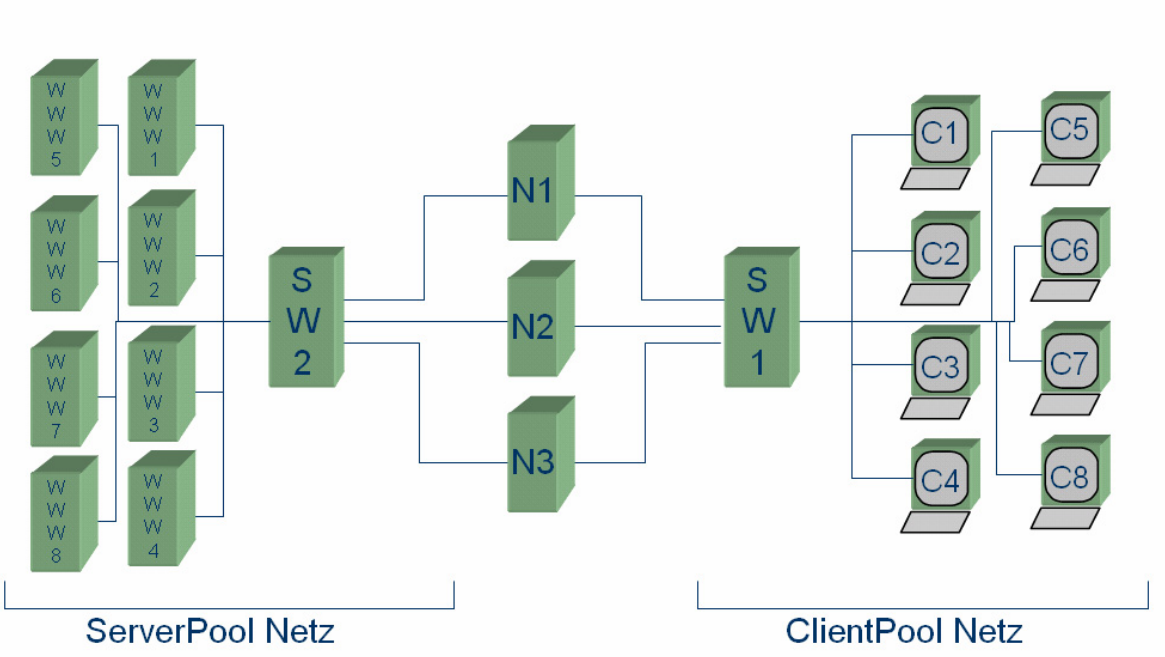

Abbildung 43: Virtuelle Clients und Server für Funktionstest

Auf den Clients musste dafür gesorgt werden, dass nicht nur über die primäre IP-Adresse kommuniziert wurde. Dazu wurden alle ausgehenden Pakete in NF_IP_LOCAL_OUT mit Nummern von 1 bis 8 markiert. Beim Hook NF_IP_POST_ROUTING wurde Network Address Translation durchgeführt. Abhängig von der vergebenen Nummer wurden unterschiedliche Source IP-Adressen vergeben. So kommunizierte der Prozess wget0 mit der Source IP-Adresse 10.0.0.20, wget1 mit 10.0.0.21 und so fort.
Für eine mehrmalige Wiederholung des Tests wurde folgendes Script erstellt:

```
#!/bin/bash

killall wget0 2> /dev/null
killall wget1 2> /dev/null
killall wget2 2> /dev/null
killall wget3 2> /dev/null
killall wget4 2> /dev/null
killall wget5 2> /dev/null
killall wget6 2> /dev/null
killall wget7 2> /dev/null

rm -rf ./bindir
mkdir ./bindir
iptables -t mangle -F
iptables -t nat -F

cp ./wget ./bindir/wget0
cp ./wget ./bindir/wget1
cp ./wget ./bindir/wget2
cp ./wget ./bindir/wget3
cp ./wget ./bindir/wget4
cp ./wget ./bindir/wget5
cp ./wget ./bindir/wget6
cp ./wget ./bindir/wget7

ifconfig eth0:0 10.0.0.20 netmask 255.255.255.0
ifconfig eth0:1 10.0.0.21 netmask 255.255.255.0
ifconfig eth0:2 10.0.0.22 netmask 255.255.255.0
ifconfig eth0:3 10.0.0.23 netmask 255.255.255.0
ifconfig eth0:4 10.0.0.24 netmask 255.255.255.0
ifconfig eth0:5 10.0.0.25 netmask 255.255.255.0
ifconfig eth0:6 10.0.0.26 netmask 255.255.255.0
ifconfig eth0:7 10.0.0.27 netmask 255.255.255.0

# we mark packets from wget<x> with label <x+1>
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget0 -j MARK \
    --set-mark 1
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget1 -j MARK \
    --set-mark 2
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget2 -j MARK \
    --set-mark 3
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget3 -j MARK \
    --set-mark 4
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget4 -j MARK \
    --set-mark 5
```

```
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget5 -j MARK \
    --set-mark 6
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget6 -j MARK \
    --set-mark 7
iptables -t mangle -A OUTPUT -m owner --cmd-owner wget7 -j MARK \
    --set-mark 8

# We do local source-NAT to use all virtual interfaces by the certain
wget
#  processes
iptables -t nat -A POSTROUTING -m mark --mark 1 -j SNAT --to-source\
    10.0.0.20
iptables -t nat -A POSTROUTING -m mark --mark 2 -j SNAT --to-source\
    10.0.0.21
iptables -t nat -A POSTROUTING -m mark --mark 3 -j SNAT --to-source\
    10.0.0.22
iptables -t nat -A POSTROUTING -m mark --mark 4 -j SNAT --to-source\
    10.0.0.23
iptables -t nat -A POSTROUTING -m mark --mark 5 -j SNAT --to-source\
    10.0.0.24
iptables -t nat -A POSTROUTING -m mark --mark 6 -j SNAT --to-source\
    10.0.0.25
iptables -t nat -A POSTROUTING -m mark --mark 7 -j SNAT --to-source\
    10.0.0.26
iptables -t nat -A POSTROUTING -m mark --mark 8 -j SNAT --to-source\
    10.0.0.27

(./bindir/wget0 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.20/testfile0 && echo "Download #0 finished" || \
    echo "Download #0 failed" ) &
(./bindir/wget1 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.24/testfile1 && echo "Download #1 finished" || \
    echo "Download #1 failed" ) &
(./bindir/wget2 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.27/testfile2 && echo "Download #2 finished" || \
    echo "Download #2 failed" ) &
(./bindir/wget3 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.22/testfile3 && echo "Download #3 finished" || \
    echo "Download #3 failed" ) &
(./bindir/wget4 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.25/testfile4 && echo "Download #4 finished" || \
    echo "Download #4 failed" ) &
(./bindir/wget5 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.21/testfile5 && echo "Download #5 finished" || \
    echo "Download #5 failed" ) &
(./bindir/wget6 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.23/testfile6 && echo "Download #6 finished" || \
    echo "Download #6 failed" ) &
(./bindir/wget7 --limit-rate=1024k -q -t 1 -Y off -T 10 -O /dev/null\
    http://10.0.1.24/testfile7 && echo "Download #7 finished" || \
    echo "Download #7 failed" ) &
```

Abbildung 44: Testscript für den Cluster

Die Downloadrate wurde in diesem Fall auf 1024 KBytes/s beschränkt. Bei verschiedenen Tests wurden aber auch andere Downloadraten eingestellt. Auf der Seite des Webservers

wurden acht virtuelle Interfaces erstellt, denen die Ziel IP-Adressen 10.0.1.20-10.0.1.27 zugewiesen wurden.

Die Funktionstests beinhalteten auch ein Regelwerk in den Firewalls, das Stateful Inspection einsetzte:

```
Chain FORWARD (policy DROP)
target  prot source       destination
ACCEPT  tcp  anywhere     anywhere  tcp state ESTABLISHED
ACCEPT  tcp  anywhere     anywhere  tcp state RELATED
ACCEPT  tcp  10.0.0.0/24  anywhere  tcp spts:1024:65535 dpt:ftp state NEW
ACCEPT  tcp  10.0.0.0/24  anywhere  tcp spts:1024:65535 dpt:www state NEW
```

Abbildung 45: Regelsetup für den Test der Firewall.

Der Inhalt der Cluster Information Base cib.xml, der ha.cf und authkeys liegt in elektronischer Form unter **`/sources/v2.6/scripts/cib.xml`** bei.

Während des Betriebs wurden zuerst N3 und später N2 außer Betrieb gesetzt. Laufende Downloads brachen während des simulierten Ausfalls ab, konnten jedoch nach dem Failover wiederholt werden. Der Zeitraum zwischen Ausfall und Übernahme durch einen Anderen Knoten wurde lediglich von den Timeouts in Heartbeat bestimmt, nicht aber von Preselect. Die Testdateien konnten mit jeder beliebigen Konfiguration an laufenden Rechnern erfolgreich auf den Client geladen werden.

7.2 Performancetest ohne Content Scanning

Um einen Geschwindigkeitsgewinn nachzuweisen, wurde ein Zwei-Knoten-Setup benutzt. Bei der Verarbeitung eines Paketes wird eine lineare verkettete Liste abgearbeitet, bei der jedes Element einer Regel entspricht, bis eine Regel gefunden wurde, die auf das Paket passt [KaPá05]. Es ist also davon aus zu gehen, dass Firewalls mit wachsender Anzahl an Regeln einen erhöhten Verarbeitungsaufwand zu bewältigen haben. Dieser Verarbeitungsaufwand führt zu einer Reduktion des Datendurchsatzes durch die Firewall. In Performancetests wurde überprüft, ob und wann die Parallelverarbeitung von zwei Firewalls einen Speedup verursachen kann.

Als Testaufbau diente ein Zwei-Knoten-Setup. Es ist das gleiche, das bereits in Abschnitt 6.2 als Referenzsetup benutzt wurde. Als Knoten wurden folgende Rechner verwendet:

	Knoten N1	Knoten N2
Prozessor-Typ	Intel Pentium 4	Intel Pentium 3
Taktfrequenz	2,8GHz	600MHz
L2 Cache	512KB	512KB
Arbeitsspeicher	1024MB	256MB
Systembus	200Mhz (800FSB)	133MHz
Linux Kernel Version	2.6.10 mit Debian-Patches	2.6.10 mit Debian-Patches
Swapspace	200MB	200MB

Tabelle 8: Testparameter für Performancetest ohne Content Scanning

Als Netzwerkverbindungen dienten überall Fast-Ethernet Leitungen. Die Leistung von Client und Server Rechner war unerheblich, zumal sie stets ausreichte, um die Verbindungen zu den Knoten vollständig auszulasten.

Da wget keine brauchbare Auskunft über den Datendurchsatz eines abgeschlossenen Downloads liefert, wurde auf das Performancetool iperf [Di05] zurückgegriffen. Iperf misst den Datendurchsatz zwischen zwei Rechnern. Die Tests können wahlweise über TCP oder UDP durchgeführt werden. Iperf lässt sich als Server oder auch als Client starten. Abbildung 46 und Abbildung 47 zeigen einen einfachen Test.

```
Server:~# iperf –s –B 10.0.1.20
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Abbildung 46: Verwendung von Iperf am Server

```
Client:~# iperf -c 10.0.1.20 –B 10.0.0.20
------------------------------------------------------------
Client connecting to 10.0.1.20, TCP port 5001
TCP window size: 8.00 KByte (default)
------------------------------------------------------------
[884] local 10.0.0.20 port 1177 connected with 10.0.1.20 port 5001
[ ID] Interval       Transfer     Bandwidth
[884]  0.0-10.0 sec  72.2 MBytes  60.4 Mbits/sec
```

Abbildung 47: Verwendung von Iperf am Client

Vorteilhaft ist, dass sich sowohl Iperf-Client als auch Iperf-Server an ein bestimmtes Interface binden lassen. Das erspart lokales NAT wie es bei wget erforderlich war. Iperf eignet sich also hervorragend für die Messung des Datendurchsatzes.

Um einen Worst Case Testfall herbeizuführen, mussten für die Knoten Regeln generiert werden, die eine Abarbeitung der linearen Liste der FORWARD-Chain bis zu ihrem Ende erzwangen. Dazu wurden im Match-Teil der Regeln zufällig generierte Source- und Destination IP-Adressen verwendet. Um diesen Vorgang zu automatisieren, wurde das Script genrules.sh erstellt. Es ist auf der mitgelieferten CD-ROM im Verzeichnis **`/testsetup/clienttests/root/testsuit/iperftest/genrules.sh`** zu finden.

Um bei steigender Anzahl von Regeln einen Verlauf des Datendurchsatzes zu erhalten, wurde die Anzahl der Regeln in Schritten vergrößert.

Der Testablauf wie er in

`/testsetup/clienttests/root/testsuit/iperftest/iperf_test.sh`

implementiert wurde:

1. N:=200
2. Generiere am Client ein Regelwerk mit N Regeln und erhöhe dann N um 200.
3. Sende das Regelwerk per scp auf den/die Knoten und wende es dort an.
4. Starte am Client M verschiedene TCP-Verbindungen über iperf parallel, jede Verbindung mit unterschiedlicher Source- und Destination IP-Adresse.
5. Warte bis alle Downloads beendet wurden.
6. Werte die Outputs der Prozesse von Iperf aus und speichere sie ab.
7. Fahre mit Punkt 2 fort bis N>25000.

Um einen Vergleich zum Setup mit zwei Knoten anstellen zu können, wurde zuerst eine Performancemessung mit nur einem Knoten durchgeführt. Am schnelleren Knoten N1 wurden die Regeln in der FORWARD-Chain sukzessive erhöht und dabei der Datendurchsatz gemessen. Später wurden für die gleiche Messung zwei Knoten im Load Balancing Modus betrieben. Abbildung 48 zeigt den Verlauf beider Messungen in einem Diagramm.

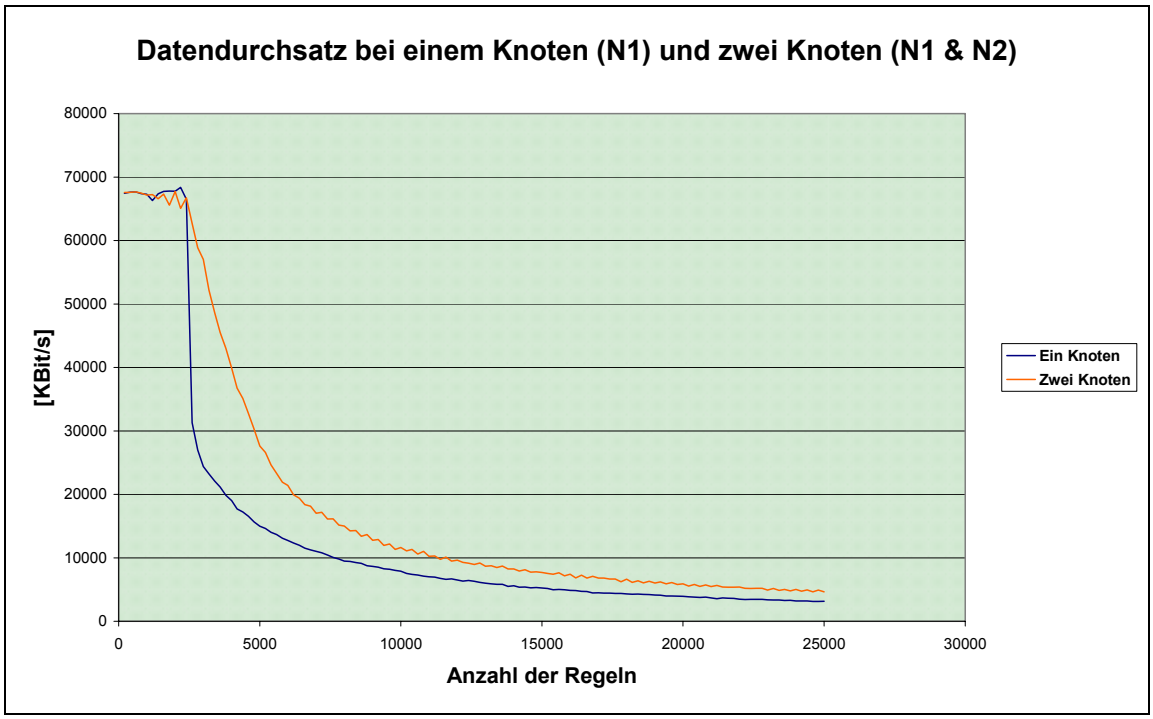

Abbildung 48: Datendurchsatz bei einem und zwei Knoten

Zunächst wird die maximal verfügbare Bandbreite vom Fast-Ethernet auf etwa 68000 kbits/s begrenzt. Bei dieser Regelanzahl ist die Verarbeitung der Liste von Regeln im Vergleich zur Übertragungszeit sehr gering und somit nicht maßgebend. Dabei fällt auf, dass der Einzelbetrieb geringfügig performanter ist als der Zwei-Knoten-Betrieb. Dies mag mit der Tatsache zusammen hängen, dass die CPU-Belastung der Firewallknoten durch das Verwerfen der Pakete höher ist als beim Einknoten-Betrieb. Ab einer Regelanzahl von etwa 2600 ergeben sich, sowohl für den Einzel- als auch für den Parallelbetrieb, Geschwindigkeitseinbußen. Vermutlich passen wichtige Teile der Regelliste ab dieser Größe nicht mehr in den Cache des Prozessors und es müssen Teile aus dem RAM nachgeladen werden. Im Einzelbetrieb bricht der Durchsatz beim Übergang von 2400 auf 2600 Regeln auf etwa 45% ein. Im Parallelbetrieb ist ein weniger starker Abfall zu beobachten. Der Quotient aus der Bandbreite im Parallelbetrieb zur Bandbreite im Einzelbetrieb ergibt den Faktor um den sich der Durchsatz steigerte. Abbildung 49 zeigt den Speedup, der durch den Parallelbetrieb erreicht wurde.

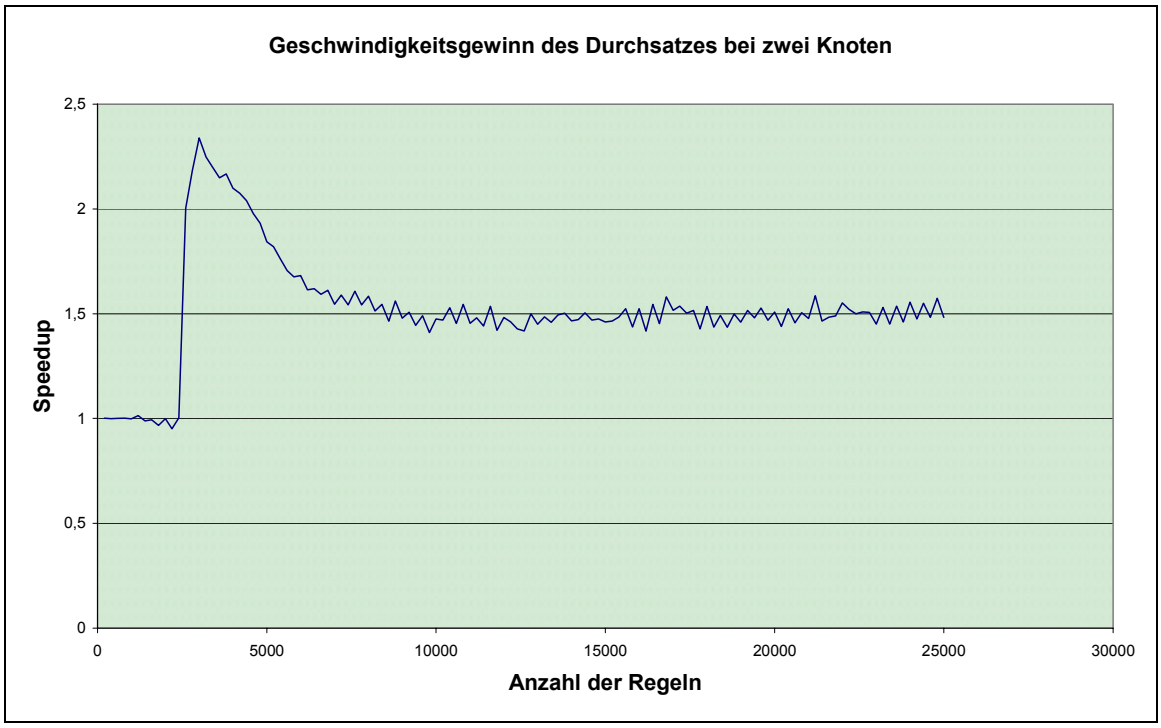

Abbildung 49: Geschwindigkeitsgewinn des Durchsatzes

Es sei hier noch mal angemerkt, dass das verwendete Regelwerk für jedes Paket eine Abarbeitung des gesamten Regelwerks erzwang. Dies entspricht nicht der normalen Arbeitsweise einer Firewall. Ein Speedup von mehr als 2 scheint unrealistisch und stammt vermutlich von der schlechten Performance des Einzelbetriebes. Bei hoher Regelzahl pendelt er sich auf den Faktor 1,5 ein. Dies ist ein realistischer Wert, da im Einzelbetrieb der bei weitem schnellere Knoten als Vergleichssystem herangezogen wurde. Ab 10000 Regeln verändert sich der Geschwindigkeitszuwachs kaum mehr. Man kann also davon ausgehen, dass Load Balancing für Firewall Cluster mit mindestens zwei Knoten auch einen Geschwindigkeitsvorteil bringen. Besonders empfehlenswert scheint ein Einsatz ab einer Regelgröße von 2400 Regeln zu sein. Diese Grenze ist aber auch maßgeblich von der verwendeten Hardware abhängig.

Im vorliegenden Testszenario wurden die Knoten nicht mit zusätzlichen Prozessen wie Proxy Servern oder ähnlichem belastet. Die Praxis zeigt aber, dass die Verwendung solcher Server auf der Firewall üblich ist. Der Betrieb von Applikationsservern verbraucht viel Rechenleistung. Als Folge daraus könnte der Einsatz von Firewall Clustern bereits bei weniger Regeln sinnvoll sein. In jedem Fall bietet er aber die Vorteile von Hochverfügbarkeit im System.

7.3 Performancetest mit Content Scanning

Durch die steigenden Prozessorleistungen ist Content Scanning auf Firewalls zu einer immer häufiger gewordenen Maßnahme geworden. Gerade in diesem Bereich kommt es aber nach wie vor zu Leistungsengpässen. Lastausgleich wie ihn Arpfake bietet, kann hier Abhilfe schaffen. Um die Leistungssteigerung im Zwei-Knoten-Betrieb zu dokumentieren, wurde unter Gibraltar folgender Test durchgeführt.
Die Netzwerktopologie war die gleiche wie bereits in Abschnitt 6.2 beschrieben. Einziger Unterschied war, dass sich im ServerPool-Netz zwei Server befanden, die die IP-Adressen 10.0.1.3 und 10.0.1.4 besaßen.

	Knoten N1	Knoten N2
Prozessor-Typ	Intel Pentium 3	Intel Pentium 3
Taktfrequenz	1,7GHz	1,7GHz
L2 Cache	256KB	256KB
Arbeitsspeicher	512MB	512MB
Linux Version	Gibraltar v2.3 ohne Patches	Gibraltar v2.3 ohne Patches

Tabelle 9: Testparameter für Performancetests mit Content Scanning

Als Content Scanner wurde der in Gibraltar enthaltene Virenscanner ClamAV verwendet. Er lässt sich mit dem HTTP-Proxy Squid verwenden. Squid wurde mit einem RAM-Speicher von 128MB und einer maximalen Objektgröße von 4096KB gestartet. Die maximale Objektgröße, bis zu der ClamAV scannen sollte war auf 2MB eingestellt. Die Konfiguration der Clients, der Server und der beiden Clusterknoten befinden sich auf CD-ROM im Verzeichnis **`/labortest/`**. Die Geschwindigkeit der Netzwerkverbindungen war unerheblich, da sich bei den Tests herausstellte, dass ClamAV die Bandbreite auf einen Bruchteil der durch die Hardware möglichen Obergrenze limitierte. Bei den Tests wurde weiters beobachtet, dass sich mit steigender Dateigröße die Bandbreite verringerte. Daher wurden schrittweise Dateien mit steigender Größe von den beiden Servern auf die virtuellen Clients geladen. Die Testdateien, die zum Download verwendet wurden, enthielten zufällige Bitverteilung. Auch beim Download von Dynamic Link Libraries, die in eine Zip-Datei verpackt waren, war kein Unterschied im Bandbreitenverhalten zu erkennen.

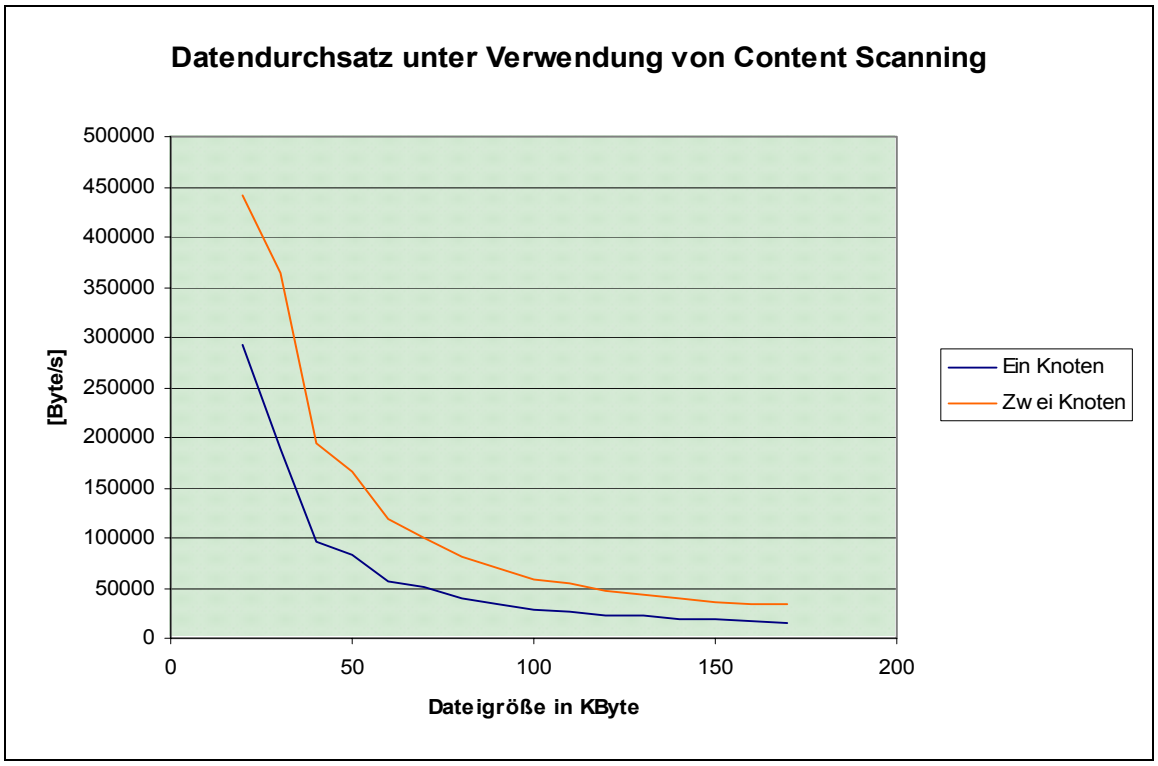

Abbildung 50: Datendurchsatz mit Content Scanning.

Es wurden Downloads von acht Prozessen mit unterschiedlichen Source IP-Adressen zu zwei Servern aufgebaut. Auf CD-ROM befindet sich das Testscript zur Rekonstruktion unter **`/labortest/client/testsuit/wgetscan/testscanner`**. In Abbildung 50 fällt auf, dass der Durchsatz außerordentlich gering ausfällt, was an der mangelhaften Performance von ClamAV im Zusammenspiel mit Squid liegt. Die CPU-Auslastung der Knoten lag stets bei annähernd 100%, wobei fast die gesamte CPU-Zeit von Squid verbraucht wurde. Bei Abschalten von Content Scanning war die Performance um ein vielfaches höher. Für die durchgeführten Tests spielte die Performance von ClamAV jedoch keine Rolle, da ohnehin eine hohe CPU-Auslastung durch Content Scanning für das Testszenario wünschenswert war. Das Diagramm zeigt einen guten Performancegewinn im Zwei-Knoten-Betrieb. Sowohl im Transparent-Proxy- als auch im Normal-Modus waren keine signifikanten Unterschiede in der Performance zu erkennen. Wie schon im vorangegangenen Test kann auch hier ein Diagramm für den Speedup errechnet werden.

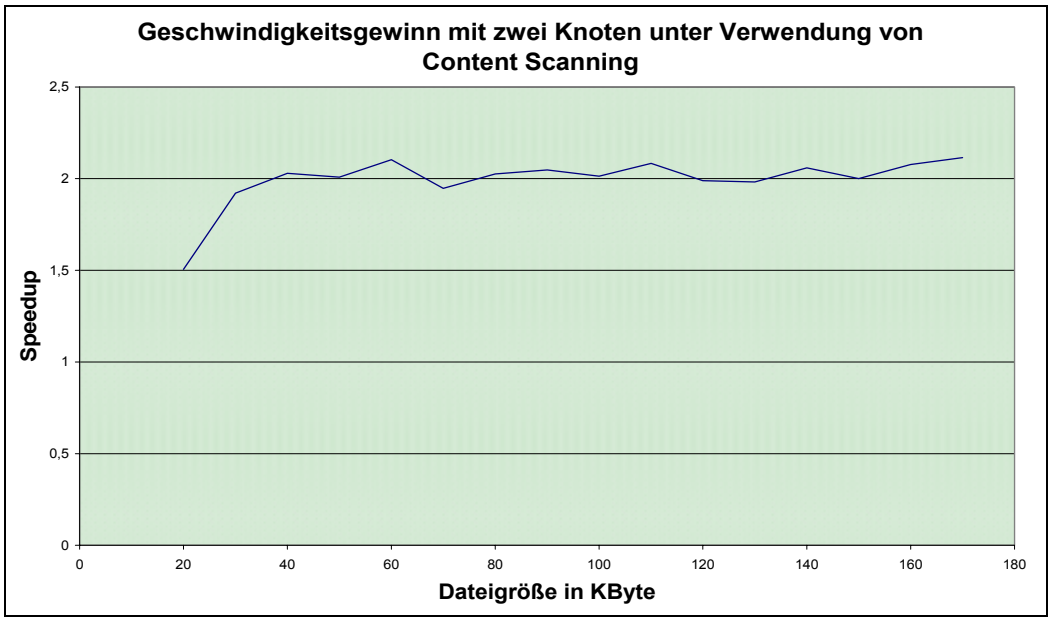

Abbildung 51: Geschwindigkeitsgewinn bei Content Scanning

Ein erzielter Speedup von zwei beweist, dass die Lösung auch für Content Scanning einen Vorteil bringt. Es ist aber zu beachten, dass die Aufteilung der Downloads zu gleichen Teilen auf die beiden Server statt fand. Bei schlechterer Aufteilung liegt der Bereich des Speedups zumindest zwischen eins und zwei. Ein weiterer Vorteil bei der Verwendung des Squid Proxies besteht darin, dass im Zwei-Knoten-Betrieb eine Synchronisation des Caches über das Internet Cache Protocoll (ICP) [Wes97] erfolgen kann. Bei Bedarf können Inhalte, die im lokalen Cache nicht vorhanden sind, vom Nachbarknoten angefordert werden. Dies ermöglicht auch ohne Content Scanning einen Geschwindigkeitsvorteil.

7.4 Zusammenfassung und Ausblick

Die gesammelten Erkenntnisse zeigen, dass Lastenausgleich mit Hochverfügbarkeit ohne zentralen Load Balancer grundsätzlich auch für Firewalls einsetzbar ist. Dennoch können nicht alle Betriebsarten von Firewalls unterstützt werden. Die Architektur von ip_conntrack und IPTables wurde nicht für den Einsatz in parallelen Systemen ausgelegt. Beim Eintritt eines Paketes in eine Firewall kann noch nicht gesagt werden, ob und mit welchen IP-Adressen und Portnummern das Paket das System wieder verlässt. Genau davon hängt aber die Rückantwort der Gegenstelle ab. Um Connection Tracking durchführen zu können, muss aber die Rückantwort vom gleichen Knoten angenommen werden, was bei unbekannten Portnummern und IP-Adressen nicht möglich ist. Nur durch den Einsatz von mehreren IP-Adressen ist es möglich, Firewall Cluster mit Source Network Address

Translation zu betreiben. Eine weitere Abhilfe dafür bestünde in einer Synchronisation der Verbindungstabelle zwischen den Knoten. Projekte wie Netfilter-HA verfolgen diesen Ansatz, der aber bei hoher Verkehrslast eher schnell seine Grenzen erreicht.

Die Verwendung von Virtual Private Networks ist grundsätzlich möglich, wurde aber bei der Behandlung des Themas ausgeklammert. Hier ist Bedarf für weiterführende Arbeiten gegeben. Generell gilt, dass es bei allen Betriebsarten, bei denen Veränderungen von Portnummer und IP-Adresse unterbleiben, Chancen für den Einsatz des beschriebenen Ansatzes gibt. Die durchgeführten Performancetests zeigen auch das Potential der Leistungssteigerung, das durch diesen Ansatz gegeben ist. Besonders bei großer Regelanzahl in der FORWARD-Chain oder beim Einsatz von Content Scanning können gute Leistungssteigerungen erzielt werden. Diese Lösung ist eine der wenigen, die Redundanz für eine Netfilter Firewall bereitstellt. Netfilter-HA ist beispielsweise vom Aufbau komplexer und noch nicht stabil genug, um den Anforderungen unter realen Bedingungen zu genügen, was aber im Hinblick auf Hochverfügbarkeit unbedingt notwendig ist. Darum hat Preselect für geeignete Anwendungen durchaus seine Berechtigung.

Preselect und Arpfake wurden unabhängig voneinander entwickelt. Das Teilsystem Arpfake kann daher auch für andere Anwendungen wie z.B. für Load Balancing von Anwendungsservern als eigenständiges Softwarepaket verwendet werden.

Aus der Praxis ist bekannt, dass Load Balancing und Verfügbarkeit auch für Firewalls immer wichtigere Themen werden. Diese Arbeit zeigt, auf welche Probleme beim Design von solchen Systemen geachtet werden muss und welche Möglichkeiten dezentrales Load Balancing für Firewalls bietet. Der praktische Teil liefert eine Lösungsvariante für die Implementierung eines Firewall Clusters mit den Eigenschaften von Lastausgleich und Hochverfügbarkeit.

Kurzreferenz

Dateien für Preselect und Arpfake

Dateiname	Ident	Kurzbeschreibung	Seite
/etc/network/interfaces	nein	Beinhaltet u.a. virtuelles Interface zur Administration der Knoten.	64
/etc/arpfake/arpfake.o	ja	Kernelobjekt Datei für arpfake	52; 66
/etc/arpfake/arpfake.conf	ja	Konfigurationsdatei von arpfake	53; 66
/etc/init.d/arpfake	ja	Init-Script für arpfake	66
/etc/preselect/ip_preselect.o	ja	Kernel Objekt Datei für Load Balancing	55; 57
/etc/preselect/passwordless.sh	ja	Script zur Einrichtung von passwordless Log-in zum/zu den Nachbarknoten	64
/etc/preselect/sync-etc.sh	ja	Script zur Synchronisation des /etc/ -Trees auf die Nachbarknoten	64; 64
/etc/preselect/preselect.conf	ja	Konfigurationsdatei mit einer Liste von Admin-IP-Adressen aller Knoten im Cluster. Wird von sync-etc.sh und passwordless.sh benötigt	64; 64
/etc/preselect/sync-exclude	ja	Liste von Dateien und Verzeichnissen in /etc/,	64

Dateiname	Ident	Kurzbeschreibung	Seite
		die nicht synchronisiert werden sollen. Wird von sync-etc.sh benötigt.	
/etc/heartbeat/resource.d/\ preselect	ja	Heartbeat Ressource zum starten/stoppen von Load Balancing	60; 67
/etc/heartbeat/ha.cf	nein	Konfigurationsdatei von Heartbeat für die jeweiligen Knoten.	69
/etc/heartbeat/haresources	ja	Konfigurationsdatei für alle Heartbeat Ressourcen im Zwei-Knoten-Setup	73
/var/lib/heartbeat/crm/cib.xml	ja	Konfigurationsdatei für alle Heartbeat Ressourcen im Mehr-Knoten-Setup	74
/etc/heartbeat/authkeys	ja	Enthält Shared Secret für Authentifikation der Heartbeat Messages.	72
/proc/net/ip_preselect/\ totalnodes	ja	Proc-Node zum Lesen/Ändern der Anzahl der Knoten im Cluster	58
/proc/net/ip_preselect/nodes	nein	Proc-Node zum Lesen/Ändern der virtuellen Knoten die lokal laufen.	58
/proc/net/ip_preselect/\ localips	ja	Proc-Node zum Lesen/Ändern jener IP-Adressen die von	58

Dateiname	Ident	Kurzbeschreibung	Seite
		ip_preselect immer zugelassen werden sollen.	
/proc/net/<interface-ip>	ja	Proc-Node zum Lesen/Ändern der Multicast MAC-Adresse die arp_arpfake verwenden soll.	54

Tabelle 10: Kurzreferenz für Dateien von Preselect und Arpfake.

Scripts und Module

/etc/heartbeat/resource.d/preselect

Parameter	Bedeutung
1. Parameter	Lokale Dienstnummer (beginnend mit 1, maximal 16)
2. Parameter	Gesamtanzahl von Diensten (beginnend mit 1, Maximal 16)
3. Parameter	Liste von IP-Adressen, die von Preselect nicht beachtet werden. (Maximal 16)
4. Parameter	Start: Starten des Dienstes Stop: Stoppen des Dienstes Status: 2. Parameter und 3. Parameter werden ignoriert. Ist der Dienst gestoppt gibt das Script „running“ aus, sonst „stopped“. Monitor: Muss aus Konformitätsgründen gegeben sein, wird jedoch ignoriert.
Exit-Code	Der Exit-Code ist 1 falls als 4. Parameter nicht start, stop, status oder monitor mitgegeben wurde. Der Exit-Code ist 0 falls die Ausführung erfolgreich war. Der Exit-Code ist -1 falls das Kernel-Modul ip_preselect.o benötigt, aber nicht vorhanden ist (die Parameter start).

Tabelle 11: Parameterbeschreibung von preselect.

/etc/init.d/arpfake

Parameter	Bedeutung
1. Parameter	Start: Starten des Dienstes Stop: Stoppen des Dienstes

Tabelle 12: Parameterbeschreibung von arpfake.

/etc/preselect/ip_preselect.o

Parameter	Bedeutung
totalnodes	Anzahl der Knoten, die sich im Cluster befinden. Minimal 1; Maximal 16
locaelnode	Index des lokal zu verwaltenden virtuellen Knoten. Minimal 1; Maximal Anzahl der Knoten die im Cluster sind.
ip_list	Liste von Destination IP-Adressen, bei denen die Vorauswahl von ip_preselect nicht erfolgen soll. Die Adressen sind mit einem Unterstrich getrennt.

Tabelle 13: Parameterbeschreibung von ip_preselect.o.

/etc/arpfake/arp_arpfake.o

Parameter	Bedeutung
mangle	0: Führt standardmäßig keine Arpfakes für neue Interfaces durch. 1: Führt standardmäßig Arpfakes für neue Interfaces durch.

Tabelle 14: Parameterbeschreibung von arp_arpfake.o.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Be05] Bellion, M. High Performance Packet Classification for Linux; online verfügbar: <http://www.hipac.org/documentation/nf-hipac-nfws2005.pdf> [Oktober 2005]

[BrJa02] Marowsky-Brée, L. -. J. Jaeger; Hochverfügbare Systeme unter Linux; online verfügbar: <http://www.suse.de/de/whitepapers/ha/> [05.Oktober.2005]

[CorRub05] Corbet, J. A. Rubini (2005); Linux Device Drivers; Verlag: O´Reilly; ISBN: 0-596-00590-3.

[Di05] Distributed Applications Support Team. Editor: Iperf; online verfügbar: <http://dast.nlanr.net/Projects/Iperf/>;

[Ed02] University of Edinburgh. Editor: Herve Eychenne; iptables manpage - administration tool for IPv4 packet filtering and NAT; online verfügbar: <http://unixhelp.ed.ac.uk/CGI/man-cgi?iptables+8>;

[Fl05] Villanustre, F. Loadbalancer-less clusters on Linux; online verfügbar: <http://flaviostechnotalk.com/wordpress/index.php/2005/06/12/loadbalancer-less-clusters-on-linux> [12. Juni 2005]

[Fsf05] Free Software Foundation. Editor: GNU Wget; online verfügbar: <http://www.gnu.org/software/wget/wget.html>;

[Ho03] Horman, S. Saru: Active-Active; online verfügbar: <http://www.ultramonkey.org/papers/active_active/active_active.shtml> [März 2003]

[Ie82] Internet Engineering Task Force. Editor: David Plumme; RFC826: An Ethernet Address Resolution Protocol; online verfügbar: <http://www.ietf.org/rfc/rfc0826.txt>;

[Ie84] Internet Engineering Task Force. Editor: C. Hornig; RFC894: "A Standard for the Transmission of IP Datagrams over Ethernet Networks"; online verfügbar: <http://www.ietf.org/rfc/rfc0894.txt>;

[Ie85] Internet Engineering Task Force. Editor: J. Postel; RFC959: "FILE TRANSFER PROTOCOL (FTP)"; online verfügbar: <http://www.ietf.org/rfc/rfc0959.txt>;

[Ie88] Internet Engineering Task Force. Editor: S. Deering; RCF1054: "Host Extensions for IP Multicasting"; online verfügbar: <http://www.ietf.org/rfc/rfc1054.txt>;

[Ie89a] Internet Engineering Task Force. Editor: S. Deering; RFC 1112: "Host Extensions for IP Multicasting"; online verfügbar: <http://www.ietf.org/rfc/rfc1112.txt>;

[Ie89b] Internet Engineering Task Force. Editor: R. Braden; RFC1122: "Requirements for Internet Hosts -- Communication Layers"; online verfügbar: <*http://www.ietf.org/rfc/rfc1122.txt*>;

[Ie96] Internet Engineering Task Force. Editor: C. Perkins; RFC2003: "IP Encapsulation within IP"; online verfügbar: <*http://www.ietf.org/rfc/rfc2003.txt*>;

[Ie97] Internet Engineering Task Force. Editor: W. Fenner; RFC2236: "Internet Group Management Protocol, Version 2"; online verfügbar: <*http://www.ietf.org/rfc/rfc2236.txt*>;

[Je00] Jenkins, R. A Hash Function for Hash Table Lookup; online verfügbar: <*http://burtleburtle.net/bob/hash/doobs.html*> [Juni 2000]

[KaPá05] Kadlecsik, J. -. G. Pásztor; Netfilter Performance Testing; online verfügbar: <*http://people.netfilter.org/kadlec/nftest.pdf*> 7; [13.Mai 2005]

[Le02] Leite, F. O. Load-Balancing HA Clusters with No Single Point of Failure; online verfügbar: <*http://www.linux-kongress.org/2002/papers/lk2002-leite.ps*> [September 2002]

[Li01] The Linux Kernel Archives. Editor: Jim Trocki; Manpage Mon; online verfügbar: <*http://www.kernel.org/software/mon/man/mon.html*>;

[Ma04] Marowsky-Brée, L. Heartbeat Deployment Tutorial; online verfügbar: <*http://www.linux-ha.org/_cache/TechnicalPapers__UKUUG-WinterConf-2004-SCRAT-Talk.pdf*> 10; [24. Februar 2004]

[Mau04] Maurer, W. (2004); Linux Kernelarchitektur - Konzepte, Strukturen und Algorithmen von Kernel 2.6; 1. Auflage; Verlag: Hanser; ISBN: 3-446-22566-8.

[Mi03] Microsoft Corporation. Editor: Network Load Balancing (NLB) Planning and Architecture Topics; online verfügbar: <*http://www.microsoft.com/technet/prodtechnol/windowsserver2003/planning/nlb.mspx*>;

[Qu04] Quade, J. Linux-Treiber entwickeln; Proc-Filesystem; online verfügbar: <*http://ezs.kr.hsnr.de/TreiberBuch/html/index.html*> [21. Dezember 2004]

[Ro00] Robertson, A. The Open Cluster Framework (OCF); online verfügbar: <*http://www.opencf.org/documents/HAFramework.pdf*> [13.August 2002]

[Ro04] Robertson, A. High-Level IBM overview of HA on Linux; online verfügbar: <*http://linux-ha.org/_cache/Talks__LinuxHAv3-7.ppt*> 9; [September 2004]

[Ro05a] Robertson, A. Linux-HA Release 2 Hands-On Tutorial; online verfügbar: <*http://linux-ha.org/_cache/Talks__LWCESF2005_tutslides.ppt*> [7. August 2005]

[Ro05b] Robertson, A. Overview of Linux-HA Release 2; online verfügbar: <*http://linux-ha.org/_cache/Talks__HA-Academy-2005.sxi*> [4. Oktober 2005]

[RoSt04] Roth, T. A. Stieglecker; Heartbeat Netfilter-Dokumentation-mit-HOWTO; online verfügbar: <*http://www.fim.uni-linz.ac.at>;* [5. Oktober 2004]

[RuWe02] Russel, R. -. H. Welte; Linux netfilter Hacking HOWTO; online verfügbar: <*http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-4.html#ss4.3*> [2.Juli 2002]

[We00] Welte, H. The journey of a packet through the linux 2.4 network stack; online verfügbar: <*ftp://ftp.gnumonks.org/pub/doc/packet-journey-2.4.ps.gz*> [14.Oktober 2000]

[We02] Welte, H. HA for netfilter based firewalls; online verfügbar: <*http://www.linux.org.uk/~ajh/ols2002_proceedings.pdf.gz*> S.565ff; [26. Juni 2002]

[We05] Welte, H. Netfilter High Availability; online verfügbar: <*http://people.netfilter.org/~hidden/nfws-2005-ctsync_slides.pdf*> S.20; [10 März 2005]

[Weis02] Weiser, F. Linux Clustering; online verfügbar: <*http://www.ordix.de/onews2/2_2002/artikel/sundn_3.php*> [12.10.2005]

[Wes97] Wessels, D. ICP and the Squid Web Cache; online verfügbar: <*http://citeseer.ist.psu.edu/cache/papers/cs/1907/http:zSzzSzircache.nlanr.netzSz~wesselszSzPaperszSzicp-squid.pdf/wessels97icp.pdf*> [13.August 1997]

[Wil01] Willms, G. (2001); C++ Das Grundlagenbuch; 2. Auflage; Verlag: Data Becker Verlag; ISBN: 3-8158-1437-5.

[Zh00] Zhang, W. Linux Virtual Server for Scalable Network Services; online verfügbar: <*http://www.linuxvirtualserver.org/ols/lvs.pdf*> [22.Juli 2000]

[ZhSh04] Zhang, W. -. S. Jin; Creating Linux Virtual Servers; online verfügbar: <*http://www.linuxvirtualserver.org/clvs.ps.gz*> [06.Oktober 2004]

Lebenslauf

Alexander Stieglecker

Liedlgutweg 16
4400 Steyr
Email: alex_za@gmx.net
Familienstand: ledig

Ausbildung:

1984 – 1988	Volksschule Steyr Tabor
1988 – 1992	Musikhauptschule Steyr
1992 – 1994	Fachschule für Elektronik
1994 – 1998	Höhere Abteilung für Elektronik – Ausbildungszweig Technische Informatik. Reifeprüfung 1998
1993/08	Ferialpraxis bei „Schmied und Pachler Kabel-TV“
1996/07	Ferialpraxis bei „Schmied und Pachler Kabel-TV“
1998/08 – 1999/04	Ableistung des Grundwehrdienstes in St. Pölten/Sprazern.
1999/04 – 1999/06	Motorenmontage bei BMW-Steyr.
1999 – 2005	Studium für Technische Informatik an der Johannes Kepler Universität in Linz.

Sprachkenntnisse: Deutsch (Muttersprache);
Englisch (Sehr gut)

Hobbies: Linux, Snowboarden, Krafttraining, Segeln, Mountainbiken, Reisen.

Steyr, März 2006

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Magisterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Steyr, März 2006

.....

Unterschrift