Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SRR
Service Routing Redundancy

Routing basierte Cluster Server Redundanz

Amir Guindehi <amir@guindehi.ch>

Diplomarbeit DA-2002.30
Sommersemester 2002
Tutor: Ralph Keller
Supervisor: Prof. Dr. Bernhard Plattner

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Signature Not Verified

Amir Guindehi

Digitally signed by Amir Guindehi
DN: cn=Amir Guindehi, o=Digital Impact GmbH, c=US
Date: 2002.10.21 12:56:08 +02'00'

Abstract

Abstract (Deutsch)

Einführung

Heutzutage bilden Informationssysteme einen zentralen Faktor in Geschäftsprozessen. Um einen reibungslosen Ablauf dieser Geschäftsprozesse garantieren zu können, sind Organisationen darauf angewiesen, dass die komplexe und heterogene Infrastruktur wie auch die vielfältigen ineinander übergreifenden und voneinander abhängigen Netzwerkdienste jederzeit verfügbar sind. Informationssysteme sind jedoch einer Reihe von Problemen unterworfen wie Hard- und Softwarefehlern, unzuverlässige neue Softwareversionen, unvorsehbare Eventualereignisse (Brand, Zerstörung, Diebstal), Softwareabstürze durch Überlastung und viele andere mehr. Um zu vermeiden, dass Informationssysteme wegen eines einzigen Problems, eines sogennannten "*Single Point of Failures*", nicht mehr verfügbar sind, werden *Cluster Server* erstellt, welche auch beim Auftreten von Problemen weiterhin für Client-Applikationen verfügbar sind und den ihnen anvertrauten Dienst weiter erbringen.

Der Cluster stellt einen *virtuellen Server*, bestehend aus zwei oder mehr unabhängigen Servern, dar, der einen oder mehrere Dienste erbringt, welcher von Clients über das Netzwerk genutzt werden kann. Der virtuelle Cluster Server wird dabei über eine *virtuelle IP-Adresse* angesprochen, welche dem Server, welcher gerade die Anfragen beantworten soll, zugewiesen ist. Beim Ausfall des aktiven Servers des Clusters wird die virtuelle IP Adresse einem anderen Server innerhalb des Clusters zugewiesen, welcher dann die Kontrolle über den Cluster übernimmt, den Dienst neu selber erbringt und Anfragen von Clients beantwortet. Die Server des virtuellen Cluster Servers *erkennen automatisch den Ausfall* von andern Server des Clusters und der Dienst wird von einem der anderen vorhandenen Server transparent übernommen und auf diesem, im Zuge des sogenannten *Fail-Over,* neu gestartet.

Ziele

Ziel dieser Arbeit ist es einen solchen virtuellen Cluster Server zu realisieren. Der Cluster Server wird von Clients über eine virtuelle IP Adresse angespochen, welche jeweils dem aktiven Server innerhalb des virtuellen Cluster Servers zugeteilt wird. Es soll ein Protokoll entwickelt werden, welches je nach Zustand des Clusters die virtuelle IP Adresse zuteilt und es ermöglicht die entsprechenden Dienste im richtigen Moment zu starten, zu prüfen und zu stoppen.

Dabei sind unter anderem die folgenden Randbedingungen zu berücksichtigen:

- **Transparenz:** Die Client-Applikation, die den Service des Clusters verwendet, sollte beim Ausfall eines Servers möglichst nicht beeinträchtigt werden.
- **Fail-Over:** Gerade hohe Anforderungen an Redundanz und Ausfallsicherheit bedingten eine möglichst grosse physikalische Distanz zwischen den Teilnehmern des Cluster Server. **Wir wollen deshalb dass der virtuelle Cluster Server sich aus einer Menge von Servers bilden lassen, die nicht am selben physikalischen LAN angeschlossen sind!** Dies bedeutet dass sich die klassischen Methoden der Clusterbildung nicht anwenden lassen. Eine neue Methode zur Signalisierung der Server Zustände und der Zuordnung der virtuellen IP Adresse muss gefunden werden.
- **Keine Single Point of Failure**
- **Gruppen & Abhängigkeiten:** Es soll möglich sein aus diesen Diensten Dienstgruppen zu bilden um beispielsweise eine Disk-Gruppe zu bilden, die aus einem Dienst und einer physikalischen Ressource mit den zum Dienst zugehörigen Daten besteht. Innerhalb einer solchen Gruppe soll es möglich sein Dienstabhängigkeiten zu definieren.

Resultate

In dieser Arbeit wurde eine neue Methoden zur Zustandsermittlung und Verteilung der Dienste eines Cluster Servers betrachtet und es wurde im speziellen untersucht ob sich ein Flooding Algorithmus, wie ihn beispielsweise das Link State Protokoll OSPF, dazu verwenden lässt. Mit Hilfe von OSPF Opaque Link State Announcements (LSAs) wurde ein Signalisierungsprotokoll implementiert, dass es ermöglicht einen virtueller Cluster Server zu realisieren. Der Cluster Server besteht aus einem Unix Daemon (SRRD), der einen Webserver für die komfortable verteilte Konfiguration und die Kontrolle der Dienste und des Clusters implementiert. Der Daemon flutet dann, mit Hilfe des OSPFAPI's [16] und eines Zebra OSPF Daemons [17], die Service Zustände mittels OSPF LSAs ins Netz und zu den anderen am Cluster teilnehmenden Servers.

Der realisierte Cluster Server unterstützt sowohl Netzwerkdienste, mit einer oder mehreren IP Adressen, als auch lokale Ressourcen die physikalisch geteilt werden, wie beispielsweise SCSI Platten an einem SCSI Bus mit zwei Servers oder ein Netzwerk RAID-1 Mirror. Sowohl Dienst Gruppen, Dienst Abhängigkeiten wie auch kritische Dienste wurden implementiert. Des weiteren wurde als Beispiel einer physikalischen Ressource ein Netzwerk RAID-1 Mirror mit Hilfe von Enhanced Network Block Devices (ENBD) implementiert. Dieser RAID-1 Netzwerk Mirror kann dann hochverfügbar und verteilt die Daten für andere Dienste des Clusters zur Verfügung stellen. Bei Ausfall eines Servers kann ein anderer Server die Kontrolle über den Netzwerk RAID-1 Mirror übernehmen, gegebenfalls den Mirror neu spiegeln und den Dienst weiter erbringen. Diese Lösung hat sich hervorragend bewährt.

Da der Cluster Server OSPF als Signalisierungsprotokoll zwischen den Servern verwendet, wurde auch noch ein Link State Database Looking Glass, also ein Tool zum inspizieren der Link State Datenbank, implementiert. Ein Interface zum OSPF Routing Daemon zum inspizieren der Dienst Routing Tabellen ist ebenfalls im implementierten Webserver des Clusters enthalten. Um wenigstens "User Level Security" zu gewährleisten wurde, mit Hilfe der OpenSSL Bibliothek, auf Webserver Level Secure Socket Layers (SSL) implementiert. So ist möglich die Benutzer des Webservers mittels X509 Zertifikaten, die von einer Certificate Authority signiert sein können, zu authentisieren und zu authorisieren.

Weiterführende Arbeit

Es möglich, die Kommunikationsschicht weiter zu abstrahieren und zu ermöglichen dass Dienstzustandsänderungen nicht nur mittels OSPF kommuniziert werden, sondern dass ein beliebiger Kommunikationsstack eingesetzt werden kann, wie z.B. eines der bestehenden Heartbeat-Protokolle. Dies würde es ermöglichen dienstabhängig die Vorteile der verschiedenen Protokolle auszunutzen und so grösstmögliche Flexibilität zu bieten. Desweiteren wäre es möglich zu untersuchen wie sich der mittels des OSPF Protokolles realisierten Cluster Server in grossen heterogenen Netzen mit vielen Areas und Routern verhält wo die am Cluster teilnehmenden Server durch grosse Distanzen und somit durch viele Router getrennt sind.

Die Implementation des Service Rounting Redundancy Daemons ist "Single threaded". Beim Design des Daemons wurde davon ausgegangen, dass die Dienst Primitiven, die zum starten, stoppen und prüfen des Dienstes verwendet werden, eine kurze Ausführungszeit besitzen. Dies sprach für ein "Single threaded" Design des Redundanz Daemons. Es hat sich gezeigt dass die Implementation von so komplexen Diensten wie ein Netzwerk RAID-1 eine längere Ausführungszeit der Dienst Primitiven mit sich bringt. Dies wurde durch eine "Recall" Option bei den Dienst Primitiven berücksichtigt. Ein "Multi threaded" Design würde die Entwicklung von Dienst Primitiven vereinfachen.

Abstract (English)

Introduction

Today's information systems pays a central role in business processes. To guarantee trouble-free functioning of these business processes an organization depends on the continous availability of the complex and heterogeneous infrastructure as well on the manifold of interconnected and interdependent network services. Information systems are subjected to multiple problems such as hardware and software faults, unreliable new software versions, unpredictable and unforeseen natural events like fire, earthquake, destruction or theft, software crashes because of overload etc. To prevent information systems becoming unavailable because of a *"Single Point of Failure"*, Cluster Servers are built and deployed. These servers ensure that the service remain available to client applications in the event of any faults.

The cluster represents a *virtual Server* consisting of two or more independent servers called *cluster nodes* serving one or more services which can be used by clients over the network. The virtual cluster server gets addressed and accessed over a *virtual IP address* which is assigned to one of the cluster nodes. On failure of the active server of the cluster the Virtual IP-Address gets reassigned to one of the still available cluster nodes which in turn will provide the service and answers to request from clients. The cluster nodes of the cluster *automatically detect the failure* of a cluster node or a service. The service gets reassigned to another cluster node transparently and gets started on that new cluster node. This process if called *"Fail-Over"*.

Aims & Goals

The goal of this work is to design and implement such a virtual cluster server. The cluster server gets addressed with a virtual IP address by the clients. This address is assigned to the currently active cluster node of cluster server. A protocol will be designed, which assigns the virtual IP address to one of the cluster nodes dependant on the state of the cluster. This protocol will allow the starting, the termination and the checking of the entrusted services at the right instances.

To realize the aforementioned the protocol must fulfill the following criteria:

- **Transparency:** The client application using the service should not be affected by the failure of a cluster node.
- **Fail-Over:** To satisfy with the requirements of redundancy and reliability the maximum physical distance between the cluster nodes is desired.
  We want to build a virtual cluster server capable of using cluster nodes which are not connected to the same local area network (LAN)!
  This means that the classic methods to build clusters cannot be applied! A new method for signalization of the cluster node states and for the assignment of the virtual IP address has to be found.
- **No Single Point of Failures**
- **Service Group & Service Dependencies:** It should be possible to build service groups out of the services on a cluster. This allows for example to build a disk group consisting of a service and a physical resource with the data belonging to the service. Within such a service group it should be possible to define service dependencies among the services.

Results

In this diploma thesis new methods of service state detection and distribution in a cluster were investigated. Flooding-Algorithms like the Link State Protocol OSPF was explored for possible application to this problem. With the usage of OSPF Opaque Link State Announcements (LSAs) a signaling protocol was designed and implemented which allows to realize a virtual cluster server. The cluster server consists of a Unix daemon with the name Service Routing Redundancy Daemon (SRRD) which implements a web server for comfortable distributed configuration and controlling of the services and the cluster. The daemon floods the service states with help of the OSPFAPI [16] and a Zebra OSPF daemon [17] into the surrounding network and towards the other cluster nodes of the cluster.

The constructed cluster server supports network services with one or more IP addresses as well as for physically shared local resources, for example SCSI disks on a SCSI bus with two servers or a network RAID-1 mirror. Service groups, service dependencies as well as critical services are implemented. Furthermore as an example of a complex resource a network RAID-1 mirror using Enhanced Network Block Devices (ENBD) was implemented. This network RAID-1 mirror is capable of providing the data to the other services on a redundant, distributed and highly available basis. On failure of one of the active cluster nodes another available cluster node can take over the network RAID-1 mirror. This new cluster node will synchronize the mirror if needed and will continue to serve the associated service. This solution has performed admirably.

Since the cluster server uses OSPF as the signaling protocol between the cluster nodes, a Link State Database Looking Glass tool to inspect the Link State Database was implemented. An interface to the OSPF routing daemon and the Zebra routing daemon to inspect the routing tables is included in the implemented web server. To provide at least user level security the implemented web server supports Secure Socket Layer (SSL) . This allows to authorize and authenticate the users by X509 certificates possibly signed by a Certificate Authorities (CAs).

Further Work

The possible next stage of this work would be to abstract the communication layer further and to allow different communication stacks to be used for the service state detection and distribution such as using one of the existing heartbeat-protocols. This will allow the advantages of the different protocols to be tapped and consequently maximizing the service flexibility. In addition it would be interesting to determine the impact of wide heterogeneous networks with various routers, bridges or repeaters where routers are separated by large distances on the with the OSPF protocol realized cluster server.

The implementation of the Service Routing Redundancy Daemon (SRRD) is single threaded. In designing the daemon, it has been assumed that the implemented service primitives used to start, stop and check the services will have a short execution time. It showed that the implementation of complex services such as a network RAID-1 mirror implied a longer execution time of their service primitives. Provisions were made for this by implementing a “recall” option for the service primitives. A multi threaded design would simplify the development and the deployment of new service primitives.

Inhaltsverzeichnis

1. Einführung

1. 1. Motivation

1. 1. 1. Service Redundanz und Hochverfügbarkeit

Seit einiger Zeit nimmt die Zahl der Unternehmen und der Umfang, in dem diese das Internet geschäftlich nutzen, der sogenannte E-Commerce, dramatisch zu. Im Privatkundengeschäft, auch B2C genannt, wurden 1999 Umsätze von rund 20 Milliarden US-Dollar erzielt; für das Jahr 2004 wird eine Steigerung auf 185 Milliarden US-Dollar erwartet.

Diese Werte nehmen sich jedoch gering aus im Vergleich zu den Umsätzen, die im Handel zwischen Unternehmen, genannt B2B, erwartet werden. Es wird hierbei mit einer Steigerung der Umsätze von ca. 120 Mio. US-Dollar im Jahr 1999 auf etwa 2,7 bis 7,3 Billionen US-Dollar gerechnet; konservative Schätzungen gehen immerhin noch von rund 1,5 Billionen US-Dollar aus [1].

Im Rahmen dieser Zunahme der Geschäftstätigkeit im Internet wird es für Unternehmen immer wichtiger, rund um die Uhr online zu sein. Eine Unterbrechung der Internetanbindung kann zu großen finanziellen Verlusten führen. So berechnet Dell Computers, ein hauptsächlich im Online-Verkauf tätiges Unternehmen, einen Verlust von 580.000 US-Dollar pro Stunde bei Ausfall der Internetanbindung.

Einige Unternehmen wie Internet Service Provider (ISPs), Online-Händler oder Online-Banken, sind zu 100 Prozent auf das Internet angewiesen. Wenn ein Kunde auf die Web-Seite eines E-Commerce-Unternehmens surft, welche aber aufgrund eines Ausfalls nicht verfügbar ist, wird er zur Konkurrenz überwechseln; diese ist schließlich nur einen Mausklick entfernt. Für Unternehmen ist also eine durchgehende Internetpräsens nicht nur aus finanziellen Gründen, sondern auch im Interesse einer langfristigen Kundenbindung von grosser Wichtigkeit.

Bei einem Unternehmensnetzwerk, welches für die Anbindung ans Internet benötigt wird, passieren die Daten verschiedene Punkte wie Server, Firewalls, Router, Standleitungen oder ISPs. Diese bilden allesamt Schwachstellen, da nur durch den Ausfall eines einzigen dieser Punkte die Internetanbindung zusammenbricht. Da sich der Ausfall von Komponenten leider nicht vermeiden läßt, ist es notwendig, diese redundant auszuführen. Dadurch kann trotz eines Ausfalles die Internetanbindung aufrecht erhalten werden. Das Konzept der redundanten Ausführung von Komponenten wird Hochverfügbarkeit oder “High Availability” genannt.

Um auch auf den nicht unüblichen Ausfall eines ISP vorbereitet zu sein, muß das unternehmenseigene Netzwerk gleichzeitig an zwei oder mehr ISPs angebunden sein. So sind z.B. die Internet-Sites von Microsoft und Ebay bei drei verschiedenen ISPs angebunden [2].

Da aufgrund der Notwendigkeit von High Availability sämtliche kritischen Komponenten mindestens doppelt verfügbar sind, bietet es sich an, diese auch aktiv zu nutzen. Dadurch wird die Leistungsfähigkeit der Internetanbindung verdoppelt, wodurch die Aufwertung der Komponenten bei steigendem Datenverkehr hinausgezögert werden kann. Die Verteilung des Datenverkehrs auf mehrere gleichwertige Komponenten wird Lastverteilung oder “Load Balancing” genannt.

Bei der Planung einer Netzwerktopologie muß auch die zukünftige Entwicklung eines Unternehmens berücksichtigt werden. Bei einer Zunahme der E-Commerce Geschäftstätigkeit ist mit einem Anwachsen des Datenverkehrs zu rechnen, so dass einzelne Komponenten an ihre Leistungsgrenzen stoßen. Nötig ist also die Möglichkeit einer Anpassung der Netztopologie auf steigenden Datenverkehr, auch als Skalierung bezeichnet. Eine Skalierbarkeit ist durch den Einsatz von Load Balancing-Technologie jedoch grundsätzlich gegeben.

1. 1. 2. Einige Beispiele aus dem E-Commerce Markt

Weltweit ist der E-Commerce Markt im Umbruch. Das Internet eröffnet völlig neue Möglichkeiten, um Waren und Dienstleistungen zu vermarkten. Gleichzeitig verlieren klassische Methoden an Bedeutung. Wer jetzt den Anschluss behalten möchte, dehnt Vertrieb, Werbung und Marketing auf die elektronische Plattform aus.

Zu den Hauptprofiteuren des E-Commerce zählen die großen Kurier-Dienste, die Transport- und Logistik-Unternehmen, da schliesslich, wenn im Internet ein Produkt bestellt wird, dieses auch weltweit auslieferbar sein muss. Beispiele für solche Firmen sind:

FedEx	Federal Express, US-Lieferdienst, setzt in Europa den Short Message Service (SMS) zur Überwachung der Paketauslieferung ein
UPS	United Parcel Service, www.ups.com
Deutsche Post AG	Deutsche Post / Euro Express - Express Paket, www.deutschepost.de

Tabelle A: Einige erfolgreiche Kurier-Dienste

Einige Beispiele von "Erfolgreichen" im E-Commerce Geschäft, die sich zum Teil durch absolut ausgefeilte E-Commerce Strategien auszeichnen:

Amazon.com	Größter Online-Händler; warten allerdings immer noch auf “Break-even”. Mit “Z-Shop” vermietet Amazon.com seine Website an Online-Händler
Atrada Trading Network AG	Auktions- und Handelshaus im Internet, Nürnberg
CDNow	Der Online-Musikhandel mit ca. 9 Millionen individuellen Website-Besucher [Unique Visitors]
AOL	Internet-Service und Content-Betreiber mit dem Online-Dienst AOL-Bertelsmann und Online-Buch-Vertrieb "BOL", die internationale E-Commerce-Plattform von Bertelsmann
E*Trade	US Online-Brokerage-Pionier, Wertpapier-Anlagen über das Internet
E-Bay	Das weltweit grösste Internet-Auktions-Haus, versteigert Online [fast] alles

Tabelle B: Einige erfolgreiche Firmen im E-Commerce Geschäft

Evita	Der Internet-Marktplatz - Portal der Deutschen Post
Expedia.de	Microsoft, Virtuelles Reisebüro, 500 Fluglinien, Mietwagen, Hotels etc.
Primus Online	Auktions-Haus
Quelle Online	Versandhandel
Dell und Gateway	Computer Direktvertrieb über das Internet
Cisco	Der Telekommunikations- und Router-Spezialist
Consors	Online-Broker, Nürnberg
Comdirect Bank AG	Online-Broker, Quickborn
Yahoo	Web-Portal-Betreiber
Google	Web-Suchmaschine-Betreiber
Bank 24	Online-Ableger der Deutschen Bank
Ricardo.de	Internet-Auktionshaus, Online Versteigerungen

Tabelle B: Einige erfolgreiche Firmen im E-Commerce Geschäft

Jede dieser Firmen ist auf eine höchst zuverlässige Infrastruktur angewiesen und kommen deshalb nicht ohne redundante Diensterbringung aus.

1. 2. Gliederung dieses Berichtes

Der vorliegende Bericht ist in 9 weitere Kapitel gegliedert.

Im folgenden Kapitel 2 - “Problembeschreibung” wird eine genaue Problembeschreibung gegeben. Es erklärt die Probleme wie auch die Ziele eines Cluster-Servers.

In Kapitel 3 - “Definitionen” werden die nötigen Definitionen für den weiteren Text vorgestellt und detalliert besprochen.

Das Kapitel 4 - “Verwandte Arbeiten” beschreibt dann einige andere Cluster-Server Implementationen und deren Fähigkeiten.

Daraufhin wird in Kapitel 5 - “Design: Service Routing Redundanz” auf das Design unseres Cluster-Servers und der dazu benötigten Mechanismen auf theoretischer Ebene eingegangen.

In Kapitel 6 - “Implementation: Service Routing Redundanz” wird das besprochene Cluster-Server Design dann als Service Routing Redundancy Daemon Implementation vorgestellt und erklärt.

Das Kapitel 7 - “Resultate” bespricht danach die gewonnenen Resultate dieser Arbeit.

Danach wird in Kapitel 8 - “Demonstration” eine Demonstration des Implementierten Cluster-Servers und seiner Fähigkeiten gegeben. Viele Bildschirmfotos begleiten die Beschreibung der Demonstration und ermöglichen einen umfassenden Einblick über die Möglichkeiten des Redundanz Daemons im praktischen Einsatz.

Kapitel 9 - “Schlussfolgerungen und Ausblick” zieht dann Schlussfolgerungen aus den Resultaten der Arbeit und bietet einen Ausblick auf die Möglichkeiten, die sich über diese Arbeit hinaus ergeben.

In Kapitel 10 - “Anhang” sind dann die Anhänge zu finden. Mehrere Unterkapitel enthalten Kontaktinformationen zum Autor, Listen von Tabellen und Figuren wie auch die originale Problemstellung.

2. Problembeschreibung

Heutzutage bilden Informationssysteme einen zentralen Faktor in Geschäftsprozessen. Um einen reibungsfreien Ablauf dieser Geschäftsprozesse garantieren zu können, sind Organisationen darauf angewiesen, dass die komplexe und heterogene Infrastruktur,, wie auch die vielfältigen ineinander übergreifenden und voneinander abhängigen Netzwerkdienste jederzeit verfügbar sind. Informationssysteme sind jedoch einer Reihe von Problemen unterworfen, wie Hard- und Softwarefehlern, unzuverlässige neue Softwareversionen, unvorhersehbare Eventualereignisse (Brand, Zerstörung, Diebstal), Softwareabstürze durch Überlastung und viele andere mehr.

2. 1. Zuverlässige Diensterbringung mit einem Server

Eine vollständig zuverlässige Diensterbringung mit einem einzelnen Server ist undenkbar. Jede Komponente des Servers existiert nur einmal und ist bei einem Fehlerfall nicht ersetzbar. Somit stellt das gesamte Server System einen “Single Point of Failure” dar, selbst wenn die Netzwerkverbindungen zum Server mehrfach redundant ausgelegt sind!

Softwareseitig sieht es genauso aus. Es existiert nur ein einzelner Arbeitspeicher und es gibt keine Möglichkeit die Software des Systems in redundanter Form laufen zu lassen. Natürlich ist es möglich mit Scripten und Prüfsoftware dafür zu sorgen, dass die Software bei Fehlerbedingungen wieder gestartet, überwacht und im Fehlerfall alarmiert wird. Doch ein solches System kann sich nicht automatisch in jedem Fall retten, ist auf Hilfe von aussen angewiesen und seine Ausfallzeiten betragen Minuten oder gar Stunden.

Mit nur einem einzelnen Server ist somit eine vollständig zuverlässige Diensterbringung nicht denkbar, muss doch jede Komponente des Systems, sei es Hardware oder Software, in redundanter Form vorhanden sein.

2. 2. Zuverlässige Diensterbringung mit mehreren Servern

Um zu vermeiden, dass Informationssysteme wegen eines einzigen Problems, eines sogenannten “*Single Point of Failures*”, nicht mehr verfügbar sind, werden *Cluster-Server*[1] erstellt, welche auch beim Auftreten von Problemen weiterhin für Client-Applikationen verfügbar sind und den ihnen anvertrauten Dienst weiter erbringen.

Ein Cluster-Server besteht aus zwei oder mehrer unabhängigen Servern[2], die einen gemeinsamen Daten-Pool verwenden, um einen Dienst zu erbringen. Dieser Daten-Pool kann verschieden geartet sein und es gibt verschiedene Möglichkeiten einen gemeinsamen Daten-Pool zu realisieren, wobei alle diese Möglichkeiten eine weitere Synchronisation der Zugriffe der einzelnen Server auf die gemeinsam genutzte Ressource bedingen:

- Die Server können an einen gemeinsam genutzten Festplattenstapel angeschlossen werden. Die Festplatten sind mit einem gemeinsamen SCSI- oder Fibre-Channel-Bus verbunden, auf welchen alle Server Zugriff haben. Es greift immer nur ein Server auf die Festplatten zu und beim Ausfall eines Server kann ein anderer Server die Kontrolle über die Festplatten

1. Cluster-Server werden oft einfach “der Cluster” genannt
2. Cluster-Nodes genannt

übernehmen. Um eine (fast) vollständige Sicherheit zu garantieren, sollte der Festplattenstapel als Hardware-RAID[1]-System ausgelegt sein, um auch beim Ausfall einer Festplatten einen Erhalt der Daten zu garantieren.

- Der Daten-Pool kann als Network File System (NFS) Server realisiert werden, wobei der NFS Server natürlich zum Single Point of Failure wird, sollte er nicht ebenfalls durch einen Cluster Server realisiert werden. Der NFS Server sollte seine Daten ebenfalls auf einem Hardware-RAID speichern falls die Daten nicht durch eine andere Clusterlösung repliziert werden.

- Ein Server Cluster kann ein Distributed Filesystem (bspw. CODA) zur Verfügung stellen und auf diese Art einen replizierten und immer verfügbaren gemeinsamen Daten-Pool realisieren. Ausfälle von Servern werden von den Clients nicht bemerkt und müssen somit mit Ausnahme des Konfliktfalles bei Dateiänderungen nicht berücksichtigt werden.

- Mittels sogennanten *Network Block Devices* kann ein Netzwerk RAID-1 Mirror aus zwei oder mehr Network Block Devices[2] erstellt werden, der dann exklusiv von einem der Server des Clusters verwendet wird. Bei Ausfall dieses Servers kann ein anderer Server die Kontrolle über den Netzwerk RAID-1 Mirror übernehmen, gegebenfalls den Mirror neu spiegeln und den Dienst weiter erbringen. Diese Lösung wird in der vorliegenden Arbeit realisiert und hat sich hervorragend bewährt.

Der Cluster stellt einen *virtuellen Server* dar, der einen Dienst erbringt, welcher von Clients über das Netzwerk genutzt werden kann. Der virtuelle Cluster Server wird dabei über eine *virtuelle IP-Adresse* angesprochen, welche dem Server, welcher gerade die Anfragen beantworten soll, zugewiesen ist.

Beim Ausfall des aktiven Servers des Clusters wird die virtuelle IP-Adresse einem anderen Server innerhalb des Clusters zugewiesen, welcher dann die Kontrolle über den Cluster übernimmt, den Dienst neu selber erbringt und Anfragen von Clients beantwortet. Die Server des virtuellen Cluster Servers *erkennen automatisch den Ausfall* von andern Servern des Clusters und der Dienst wird von einem der anderen vorhandenen Server transparent übernommen und auf diesem im Zuge des sogennanten *Fail-Over* neu gestartet.

2. 3. Fail-Over Protokoll

Die Cluster-Nodes eines Cluster-Servers sind darauf angewiesen untereinander ein eigenes Protokoll zu sprechen, das es ihnen ermöglicht, den Zustand der anderen Cluster-Nodes, wie auch den Zustand des ihnen zugewiesenen Services, der vom Cluster-Server erbracht werden soll, zu überwachen und zu manipulieren.

Mittels dieses Fail-Over Protokolls bemerken die Cluster-Nodes, wenn ein Node ausfällt und seinen Service nicht mehr erbringen kann. Das Protokoll bietet den Cluster-Nodes die Möglichkeit, mittels eines Election-Verfahrens einen neuen Cluster-Node zu bestimmen, der die Funktion des ausgefallenen Nodes übernimmt.

1. Redundant Array of Inexpencive Disks
2. ENBD - Enhanced Network Block Devices für Linux implementiert Netzwerk Block Devices

Es ist Aufgabe des Fail-Over Protokolles genau diejenigen Ereignisse festzustellen, die auch wieder über Service Primitiven an die Services weitergegeben werden müssen. Dies muss natürlich in Koordination mit den anderen am Cluster-Server teilnehmenden Cluster-Nodes geschehen um so zu gewährleisten dass exclusiv immer nur ein Cluster-Node den Service erbringt.

Die nötigen Service Primitiven wurden in Kapitel 3 - “Service Primitiven” auf Seite 19 allgemein besprochen. In Kapitel 6 - “Service Primitiven” auf Seite 60 wird dann auf die Implementation der Service Primitiven im Rahmen des Service Rounting Redundancy Daemons eingegangen. Kapitel 6 - “Service Interface zur Shell:” auf Seite 60 geht dann noch ins Detail der Service Primitiven der im Service Routing Redundancy Daemon implementierten Netzwerk RAID-1 Arrays.

2. 4. Ziele eines Clusters

Die Ziele eines Cluster-Servers sind vielfältig:

- **Minimale Ausfallzeit:** Der Cluster soll die Ausfallzeit des von ihm erbrachten Services minimieren. Dies bedingt eine möglichst schnelle Detektion eines Fehlers und daraufhin ein möglichst schnelles Fail-Over zum nächsten funktionisfähigen Cluster-Node.
- **Transparenz:** Die Client Applikation, die den Service des Clusters verwendet, sollte beim Ausfall eines Cluster-Nodes, wie auch während des transienten Zustandes beim Wechsel zu einem anderen Node des virtuellen Clusters Servers, möglichst nicht beeinträchtigt werden.
- **Fail-Over:** Normalerweise sind die Nodes des Clusters auf dem gleichen LAN. Doch gerade hohe Anforderungen an Redundanz und Ausfallsicherheit bedingen eine möglichst grosse physikalische Distanz zwischen den Teilnehmern des Cluster Server.
  Wir wollen deshalb, dass der virtuelle Cluster Server sich aus einer Menge von Servern bilden lässt, die nicht am selben physikalischen LAN angeschlossen sind!
  Dies bedeutet im Einzelnen, dass sich die klassischen Methoden der Clusterbildung nicht anwenden lassen. Eine neue Methode zur Signalisierung der Server Zustände und der Zuordung der virtuellen IP-Adresse muss gefunden werden. Kapitel 5 - “Service Routing mit einem Link State Protokoll” auf Seite 38 geht genauer auf das Design dieses Signalisierungsprotokolles ein.
- **No Signle point of Failure:** Der Datenpfad des vom virtuellen Cluster Server erbrachten Dienstes soll keinen “Single Point of Failure” aufweisen!
- **Modularität:** Der virtuelle Cluster Server sollte so offen und flexibel wie möglich entworfen und implementiert sein, um ein möglichst breites Spektrum an Diensten unterstützen zu können.
- **Flexibilität:** Sowohl Netzwerkdienste als auch physikalische Ressourcen ohne (virtuelle) IP-Adresse, wie beispielsweise eine Disk, sollen unterstützt werden.
- **Gruppen:** Es soll möglich sein, aus diesen Diensten Dienstgruppen zu bilden, um beispielsweise eine Diskgruppe zu bilden, die aus einem Dienst und einer physikalischen Ressource mit den zum Dienst zugehörigen Daten besteht. Ebenfalls sollte es möglich sein, gewisse Dienste einer Gruppe als kritisch oder unkritisch für die gesamte Gruppe zu klassifizieren.
- **Abhängigkeiten:** Innerhalb einer solchen Gruppe soll es möglich sein, Dienstabhängigkeiten zu definieren die beim Start, Stop und Fail-Over der Dienste berücksichtigt werden.

In dieser Arbeit wurde einene lauffähige, flexible und innovative Software zur Erstellung eines virtuellen Cluster Servers entwickelt, die es erlaubt, einen Cluster Server aus mehreren einzelnen Server zu bilden. Der Cluster Server erlaubt es, Dienste zu starten, zu stoppen, zu übergeben und zu prüfen, wie auch automatisch im Fehlerfall ein Fail-Over auszuführen.

2. 5. Probleme eines Clusters

Ein Cluster ist ein Verbund von einzelnen Systemen. Das heisst: alle klassischen Probleme des Distributed Computing sind bei dem Design und der Implementation eines Cluster Servers zu berücksischtigen. Sowohl Single Points of Failures der Hardware und Software des Clusters, wie auch der, die Cluster-Nodes verbindenden Netzwerkkomponenten, wie Router und Switches, Kabel und Leitungen, als auch Elementarereignisse, die ganze Gebäude und Netzwerke lahmlegen können, sind zu berücksichtigen.

Wir werden in diesem Kapitel ausgewählte Probleme eines Cluster Servers besprechen.

2. 5. 1. Service State Discovery

Jeder Cluster-Node eines Cluster-Servers ist darauf angewiesen, dass er den Zustand der anderen Cluster-Nodes kennt. Nur so ist garantiert, dass die Cluster-Nodes rechtzeitig den Ausfall eines anderen Cluster-Nodes erkennen und korrekt darauf reagieren können.

Einige Cluster-Server, die in Kapitel 4 - “Verwandte Arbeiten” besprochen werden, verwenden einen *Global Atomic Broadcast* um garantieren zu können, dass alle Cluster-Nodes immer den korrekten Zustand aller anderen Cluster-Nodes des Cluster-Servers kennen. Streng genommen ist ein Global Atomic Broadcast notwendig, um den gegenseitigen Ausschluss garantieren zu können.

Im Falle von Netzwerkdiensten wie Webservern und ähnlichem, ist eine solch strikte Garantie des gegenseitigen Ausschlusses nicht nötig und es ist möglich, mit Hilfe von weniger starken Prädikaten Aussagen über den Zustand des Clusters und der zugehörigen Cluter-Nodes zu machen. Da wir in dieser Arbeit einen Routing basierten Cluster Server entwickeln wollen und Routing in jedem Fall auf konvergierenden Algorithmen besteht, werden wir in unserem neuen Cluster Server darauf angewiesen sein, eine weniger starke Aussage über den gegenseitigen Ausschluss und die globale Sicht der Zustände der Cluster-Nodes zu machen.

In Kapitel 5 - “Service Discovery” auf Seite 29 wird auf das Service State Discovery im Rahmen des Service Routing Redundancy Daemons genauer eingegangen.

2. 5. 2. Service Fail-Over: Minimale Downtime

Das erklärte Ziel eines Cluster-Servers ist es, die Ausfallzeit des vom Cluster erbrachten Services zu minimieren. Um dies zu gewährleisten ist es erforderlich, ein zuverlässiges und möglichst schnelles Fail-Over zu implementieren.

Die Ausfallzeit des Services setzt sich zusammen aus der für das Fail-Over erforderlichen Zeit, plus der Zeit, die der Service braucht, um allfällige Service Zustände zu erfrischen, die durch intransparente Teile des Fail-Overs invalidiert wurden. Wenn beispielsweise ein Webserver mitten in einem Browser-Request versagt, so wird der Browser auch bei sofortigem Fail-Over auf einen zweiten Cluster-Node den Request wiederholen müssen, bevor er erfolgreich ausgeführt ist. Ein anderes Beispiel ist ein Samba-Server, bei dem sich der Client-Rechner zwar automatisch und transparent nach einem Fehlerfall wieder einloggt und der Client-User gar nichts davon bemerkt, bei dem Ganzen aber trotzdem natürlich eine gewisse Zeitspanne verstreicht.

Das folgende Kapitel 2 - “Transparenz” auf Seite 9 geht genauer auf die Transparenzanforderungen an den Cluster-Server ein und in Kapitel 5 - “Service Selektion” auf Seite 30 wird genauer auf das Design des Fail-Overs im Service Routing Redundancy Daemon eingegangen.

2. 5. 3. Transparenz

Bei einem Fail-Over des Services eines Cluster-Servers von einem Cluster-Node zu einem anderen gibt es eine transiente Übergangszeit, in der der Cluster-Server seinen Service eventuell, abhängig von der Methode des Fail-Over, nicht erbringen kann.

Es ist für einen Cluster-Server von eminenter Wichtigkeit, ein möglichst transparentes Fail-Over zu implementieren. In jeder der Fail-Over Methoden kann das Fail-Over atomar oder nicht-atomar implementiert sein.

Es existieren verschiedene Methoden ein transparentes Fail-Over zu ermöglichen. Die Verschiedenen Fail-Over Moethoden sind in der folgenden Tabelle zusammengefasst.

Fail-Over Methode	Layer[a]	Reichweite	Tempo	Atomar
MAC Address	Data Link (2)	LAN	< 1 sec	möglich
IP Address	Network (3)	LAN	< ~3 min[b]	möglich
Dynamic DNS	Application (7)	Weltweit	< ~5 min[c]	nein
IP Routing	Network (3)	Routing Domain	< 30 sec	nein

Table C: Fail-Over Methoden

a. Siehe [9] für einen Überblick über das klassische OSI Layer Modell
b. Abhängig vom ARP Cache der Router. Neue Cisco Router ermöglichen den ARP Cache zu löschen, indem ein spezielles ARP Paket gesendet wird
c. Abhängig vom Refresh Eintrag der DNS Zone

Gängige Cluster Lösungen verwenden die MAC Address Methode, da sie die schnellste Methode ist, ein Fail-Over zu implementieren, es gibt allerdings einige Open Source Implementationen die auch die IP Address Methode verwenden. Die Dynamic DNS Methode wird fast nie verwendet, da die Fail-Over Zeit bei dieser Methode sehr gross wird.

Leider hat die MAC Address wie auch die IP Address Methode einen entscheidenden Nachteil. Sie funktionieren beide nur im lokalen Netzwerk. Nun ist die Verteiltheit eines Clusters aber ebenfalls entscheidend für seine Zuverlässigkeit, wie in Kapitel 2 - “Räumliche Verteilung des Clusters” auf Seite 10 noch besprochen werden wird, man stelle sich nur ein Elementarereignis, wie ein überflutetes Kellergeschoss vor.

In dieser Arbeit wird eine neue Fail-Over Methode erarbeitet. Die neue Methode ist Routing basiert und ermöglicht ein Fail-Over über Netzwerkgrenzen hinweg. Da die neue Methode Routing basiert ist und jedes Routing Protokoll eine gewisse Konvergenz-Zeit besitzt, hat auch die neue Routing basierte Methode eine gewisse Konvergenz-Zeit und ist nicht atomar.

Eine weite Problematik steckt in den verwendeten Applikations-Protokollen. Man unterscheided zwischen Stateless und Statefull Protokollen.

Jedes Statefull Protokoll hat inherente Zustandsinformationen, die verhindern, dass ein neuer Cluster-Node, der an Stelle eines ausgefallenen Cluster-Nodes die Netzwerkverbindung übernehmen soll, ein transparentes Fail-Over machen kann. Ihm fehlen die inherenten Zustandsinformationen des entsprechenden Protokolles.

Doch so schlimm dies auch für das funktionieren eines Cluster scheint, ist es doch nicht. Viele Protokolle fangen auf Applikations-Layern solche Netzwerk-Fehler ab und spiegeln auf Applikations-Layern ein fehlerfreies Funktionionieren der Netzwerkverbindung vor. So erstellt ein SMB Client, der mit einem Samba-Server verbunden ist, eine unterbrochene Netzwerkverbindung automatisch und ohne Mitteilung an den Benutzer wieder neu.

2. 5. 4. Wartung von komplexen Systemen

Clustering ist eine komplexe, systemübergreifende Methode zur automatischen Redundanz, Verwaltung und Kontrolle. Es kann sehr verwirrend sein, verschiedene Services auf verschiedenen Systemen mit den selben oder ähnlichen Konfiguration zu verwalten und zu administrieren.

Deshalb ist es sehr wichtig, dass eine Cluster Lösung die Verwaltung und die Konfiguration der verschiedenen Services auf den einzelnen Cluster-Nodes auf eine übersichtliche und komfortable Art dem Benutzer präsentiert. Fast alle Cluster Lösungen besitzen eine Konfigurationsapplikation, die es dem Benutzer ermöglicht, die Services zu konfigurieren und zu kontrollieren.

In der in dieser Arbeit erarbeiteten Cluster Lösung wurde ein alle Cluster-Nodes übergreifender Webserver implementiert, der es dem Benutzer ermöglich, komfortabel und übersichtlich einen Überblick über die vorhandenen Services zu gewinnen, wie auch neue Services zu konfigurieren oder die bestehenden Services zu beeinflussen, sie zu starten, zu stoppen oder zu kontrollieren. Kapitel 6 - “Integrierter Webserver” auf Seite 59 geht genauer auf die Implementation des Webservers des Service Routing Redundancy Daemons ein.

Ein möglichst einfaches System ermöglicht es dem Cluster Betreiber auch sicherzustellen, dass keine menschlichen Fehler die Zuverlässigkeit des Cluster-Servers gefährden.

2. 5. 5. Räumliche Verteilung des Clusters

Gerade bei einem Cluster-Server ist die räumliche Verteilung von grosser Wichtigkeit. Alle Single Points of Failures sollen eliminiert werden, deshalb müssen auch Netzwerkverbindungen, Router und Switches mehrfach ausgelegt werden.

Da aber Elementarereignisse wie Feuer oder Wasserschäden nicht auszuschliessen sind, und diese ganze Stockwerke oder Gebäude bedrohen können, sind auch mehrfach ausgelegte Router und Switches keine Garantie gegen Ausfälle, stellen doch auch Gebäude und ähnliches Single Point of Failures dar, wie schon in Kapitel 3 - “Redundanz” auf Seite 14 ausgeführt worden ist.

2. 5. 6. Service Gruppen und Service Abhängigkeiten

Viele Services sind von anderen Services abhängig. Ein Webserver-Dienst ist meistens von einer Disk-Ressource, auf der sich die Daten des Webservers befinden, abhängig. Um solche Abhänigkeiten auch im Cluster modellieren zu können, ist der Benutzer darauf angewiesen, dass

die Cluster-Server Lösung solche Service-Gruppen unterstützt. Sowohl die gängigen kommerziellen Lösungen, wie auch die in dieser Arbeit vorgestellte Cluster-Lösung unterstützen Service Gruppen.

In Kapitel 5 - “Service Gruppen” auf Seite 37 wird das Design und in Kapitel 6 - “Implementation: Service Routing Redundanz” auf Seite 47 wird die Implementation der Service Gruppen und der Service Abhängigkeiten im Service Routing Redundancy Daemon besprochen.

3. Definitionen

3. 1. Hochverfügbarkeit und Redundanz

Hochverfügbar heisst möglichst viel verfügbar zu sein.

Die Größe, von der in diesem Zusammenhang am meisten die Rede ist, ist die der prozentualen Verfügbarkeit eines Dienstes (Wartungsfenster für die betroffenen Server sind hierbei ausgenommen).

Wie sonst im richtigen Leben ist auch hier niemals eine hundertprozentige Sicherheit gewährleistet, aber mit dem entsprechenden Aufwand an Hard- und Software lassen sich die Ausfallzeiten durchaus annehmbar klein halten, ist es doch so, dass wenn zwei Dienste redundant einen Dienst erbringen, sich die Ausfallwahrscheinlichkeit des Gesamtdienstes aus dem Produkt der Einzelausfallswahrscheinlichkeiten ergibt?

$$P_{total} = \prod_{i=1}^{N} P_{Dienst_i}$$

Figur 1: Totale Wahrscheinlichkeit bei Parallelschaltung

Da aber die Ausfallwahrscheinlichkeit eines Dienstes immer kleiner als 1 ist, folgt, dass die Gesamtausfallswahrscheinlichkeit mit zunehmender Anzahl Dienste sehr schnell sehr klein wird:

$$N \to \infty$$
$$P_{Dienst_i} < 1 \qquad \Rightarrow P_{total} << 1$$

Figur 2: Parallelschaltung vieler Elemente

Der Vollständigkeit halber sei hier noch gezeigt, wie sich die Ausfallswahrscheinlichkeit von voneinander abhängigen Diensten verhält. Diese Ausfallswahrscheinlichkeit entspricht derjenigen einer Gruppe von Diensten, von denen alle kritisch für die Funktion des von der Gruppe erbrachten Dienstes sind:

$$P_{total} = \sum_{i=1}^{N} P_{Dienst_i}$$

Figur 3: Totale Wahrscheinlichkeit bei Serieschaltung

Um die Qualität der Verfügbarkeit von Systemen kategorisieren zu können, hat man Fehlerklassen eingeführt, man spricht von “Number of 9s” [10].

Die folgende Tabelle vermittelt ein Gefühl für die tatsächlichen Ausfallzeiten, die hinter den prozentualen Verfügbarkeitsangaben stecken:

Prozentuale Verfügbarkeit	Grössenordung	Tatsächliche Ausfallzeit pro Jahr	Klasse
90 %	10^1	36.5 Tage	1
99 %	10^0	3.6 Tage	2
99.9 %	10^{-1}	8.76 Stunden	3
99.99 %	10^{-2}	52 Minuten	4
99.999 %	10^{-3}	5 Minuten	5
99.9999 %	10^{-4}	30 Sekunden	6
99.9999 %	10^{-5}	3 Sekunden	7

Table D: Prozentuale Verfügbarkeit

Aus der Tabelle ist klar ersichtlich, dass durch Einsatz genügend vieler redundanter Diensterbringer sich eine fast beliebige Ausfallswahrscheinlichkeit erreichen lässt.

Natürlich wächst der Koordinationsaufwand zwischen den einzelnen Diensterbringer und kann ab einem gewissen Punkt zum Flaschenhals werden, doch prinzipiell ist es möglich, die gewünschte Ausfallsicherheit durch Redundanz zu erhalten.

3. 1. 1. Wie erreicht man Hochverfügbarkeit?

Prinzipiell müssen einfach die “Signle Points of Failure” korrekt indentifiziert und anschließend eliminiert werden. Single Point of Failures (SPOFs) sind diejenigen Komponenten, deren Ausfall den Komplettausfall des gesamten Dienstes bedeuten würde. Aus der Sicht des Gesamtdienstes sind diese Komponenten kritische Komponenten! Diese Elimination der SPOFs beinhaltet ein redundantes Auslegen ihrer Funktionen. Im Falle eines Routers beispielsweise müsste ein zweiter redundante “Hot Standby” Router installiert werden, der bei Ausfall des ersten Routers einspringen klann.

Zum Anderen muß die Dienstverfügbarkeit bei Ausfall eines einzelnen Diensterbringers sichergestellt werden. Ob der konkrete ausfallende Diensterbringer in diesem Fall erreichbar ist oder nicht, spielt keine Rolle. Wichtig ist in diesem Fall, daß sich im Dienst Cluster ein anderer Diensterbringer befindet, der nahtlos dort weiterarbeiten kann, wo sein "Vorgänger" abgebrochen hat.

3. 1. 2. Redundanz

SPOFs können nur auf eine Art und Weise eliminiert werden: Man muss den SPOF redundant auslegen. Jede Komponente, ob Netzteil, Festplatte oder Netzwerkkarte sollte mit einem "Stell-

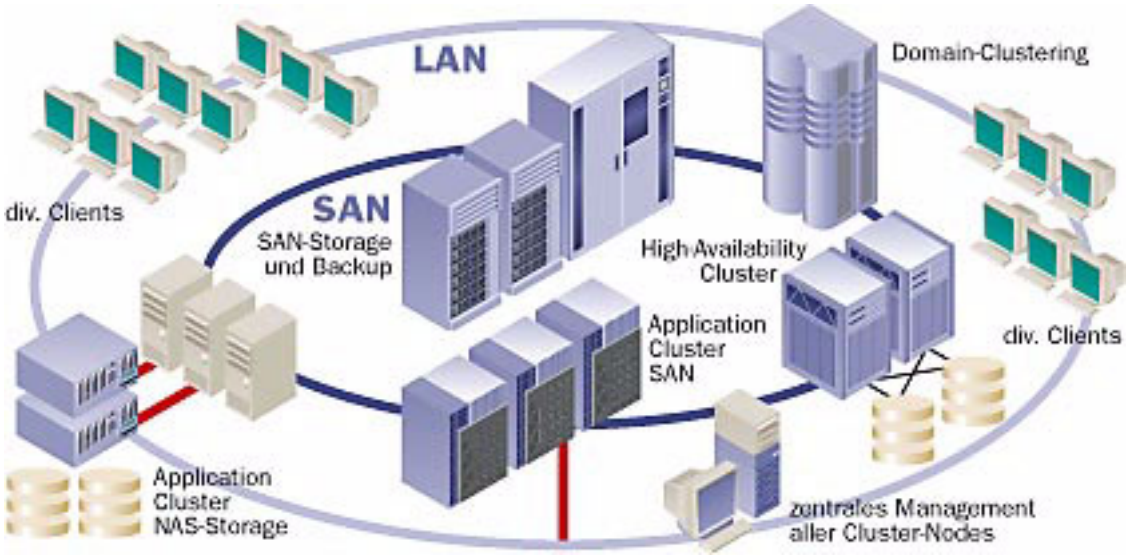
vertreter" abgesichert sein, der die Funktion der ausgefallenen Komponente wenn nötig übernimmt.

Damit ist der Redundanzen aber noch nicht genug, auch der Server als Ganzes sollte abgesichert sein. Konsequent im SPOF-Schema gedacht, stellt der Raum, das Gebäude oder sogar die Gegend, in der der oder die Diensterbringer sich befinden, wieder einen Single Point of Failure dar.

Um Ausfälle des gesamten Dienstes durch Elementarerignisse wie beispielsweise Gebäudebrand etc. auszuschließen, sollte der Diensterbringende Cluster auch möglichst räumlich weit verteilt sein, so dass Problemereignisse möglichst nur einen Teil der Diensterbringer tangieren können.

3. 2. Cluster and Cluster Node

Ein “Cluster” ist eigentlich nichts anderes als ein Verbund von Computern, die gemeinsam eine Aufgabe übernehmen. Im weitesten Sinn darf somit auch File Replication als Clustering Lösung gelten. Etwas enger gesehen versteht man unter Clustering jedoch einen integrierten Zusammenschluss von Computern, im Zusammenhang mit Clustering “Nodes” oder “Knoten” genannt, über dedizierte Verbindungen, oft mit gemeinsamem Zugriff auf Hochleistungs-Speichergeräte via SCSI (Nachteil: beschränkte Kabellänge) oder Fibre Channel (Nachteil: teuer).



Figur 4: Ein Cluster mit SAN[1] und NAS[2]

Mit echtem Clustering wird nicht nur der Dateizugriff via Failover garantiert, sondern auch die ständige Verfügbarkeit aller Anwendungen und Dienste, die auf dem Cluster laufen. Clustering kann zudem direkt vom Betriebssystem unterstützt werden; dies im Gegensatz zur File Replikation, die auf der Ebene des Dateisystems aufsetzt.

In der Welt der Midrange- und Unix-Systeme ist Clustering ein altbekanntes Verfahren, das nicht nur der Ausfallsicherheit dient, sondern auch als flexible Möglichkeit zur Skalierung der

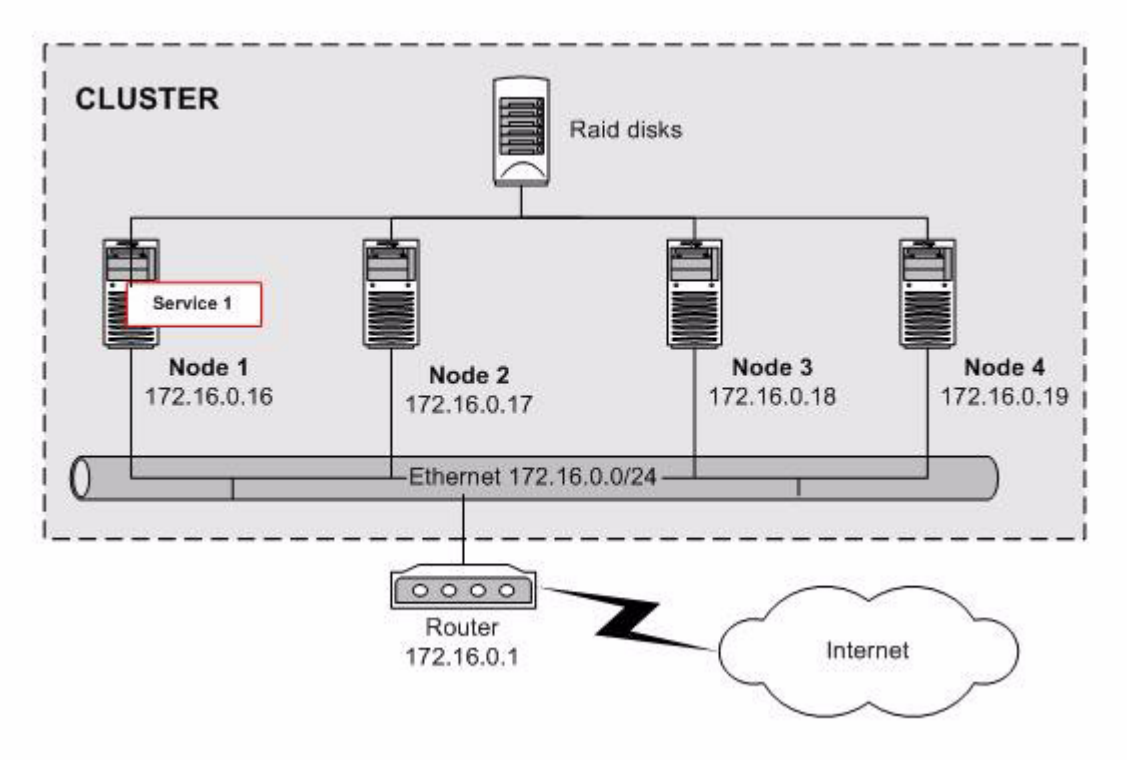
1. Storage Area Network
2. Network Attached Storage

System-Gesamtleistung genutzt wird - VAX-Cluster und Verbundinstallationen von zahlreichen Unix-Maschinen sind allseits bekannte Erscheinungen.

Clustering sollte im übrigen nicht mit einem anderen Verfahren zur Performance-Steigerung und Skalierung verwechselt werden: Beim Multiprocessing (unter NT als «SMP[1]» bekannt) handelt es sich um die parallele Nutzung mehrerer Prozessoren in ein und demselben Computer; Clustering dagegen ist ein Verbund mehrerer ganzer Computersysteme.

Ebenfalls anders geartet ist der parallele Betrieb der gleichen Anwendung auf mehreren Computersystemen, der in hochgradig zeitkritischen Bereichen, wie der Flugsicherung oder anderen Verkehrsleitsystemen vorkommt, und sofortige Umschaltung ermöglicht - hier kann jeder Systemausfall buchstäblich tödlich sein; die bei Clustern für den Failover-Vorgang benötigte Zeit von einigen wenigen Sekunden bis mehreren Minuten ist bereits nicht mehr akzeptabel.

Ein System aus verschiedenen aktiven Servern nennt man einen asymmetrischen Cluster, man kann dieses System aus verschiedenen Maschinen aufbauen.



Figur 5: Ein kaskadierter Cluster

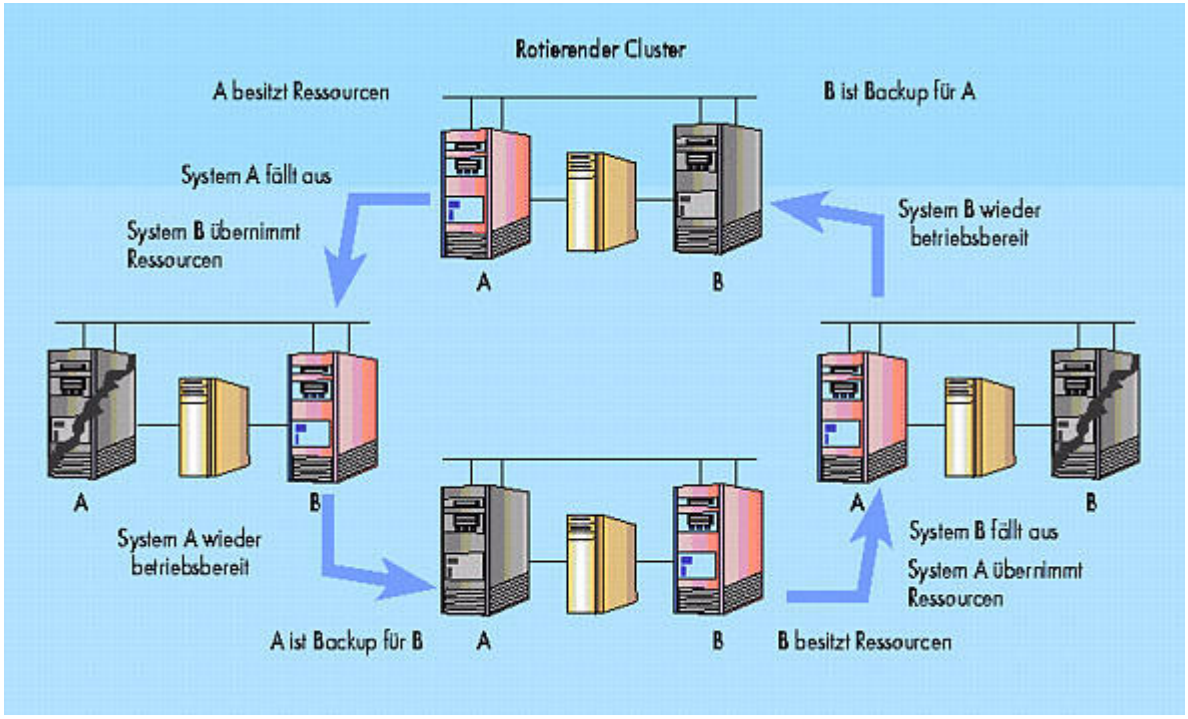
In der obigen Figur sind mehrere Cluster-Nodes in einem Cluster zu einem Gesamtsystem verbunden, die zum Einen auf ein gemeinsames RAID[2]-System (zwischen den beiden System) zugreifen, zum Anderen nach außen als ein Rechner erscheinen.

Die Kommunikation zwischen den Cluster-Nodes läuft über ein separates Netz, wie zum Beispiel Fast Ethernet. Bei einem asymmetrischen Cluster besitzen die Nodes unterschiedliche Prioritäten und können verschiedene Aufgaben erfüllen. Fällt das System aus, das die Ressour-

1. Symmetric Multi Processing
2. Redundant Array of Inexpensive Disks

cen zur Verfügung stellt, kann ein anderer Node diese Aufgabe kurzfristig übernehmen, aber nach der Rückkehr in den Cluster übernimmt er sie wieder. Ein solcher Cluster stellt also eine asymmetrische Lösung des Verfügbarkeitsproblemes dar.

Die zweite Möglichkeit ist ein symmetrischer Cluster.



Figur 6: Ein symmetrischer Cluster

Bei einem symmetrischen[1] Cluster sind die Nodes A und B *prinzipiell gleichberechtigt*! Das System, das die Ressourcen eines ausgefallenen Node übernimmt, behält sie so lange, bis es selbst ausfällt. Das andere System wird in der Zwischenzeit automatisch oder manuell wieder funktionsbereit gemacht. Aus diesem Grunde müssen die Leistungen der Nodes annähernd gleich sein.

1. Auch “rotierender” Cluster genannt

3. 3. Service

Ein Cluster erbringt einen oder mehrere “Services”. Ein solcher Service kann verschiedener Art sein. Komfortable Cluster Benutzeroberflächen unterscheiden eine Vielzahl von Services.

Prinzipiell lassen sich allerdings zwei Arten von Services unterscheiden:

- **Der Dienst**
  Ein unter einer IP-Adresse erbrachter Netzwerkdienst.
- **Die Ressource**
  Eine physikalische Ressource, wie beispielsweise ein Massenspeicher

Ein Service wird innerhalb eines Clusters vom Cluster gestartet und gestoppt. Der Cluster sorgt dafür, dass zu jeder Zeit immer nur ein Cluster-Node den Service startet.

3. 3. 1. Dienst

Ein Dienst enspricht einem Netzwerkdienst, wie beispielsweise einem Webserver, einem IMAP[1] oder POP[2] Server. Jeder dieser Dienste benutzt eine oder mehrere virtuelle IP-Adressen des Clusters.

Der Dienst ist darauf angewiesen, dass die von ihm verwendete IP Adresse vom Cluster so geroutet wird, dass Pakete, die mit der Dienst IP-Adresse adressiert sind, den Dienst erreichen.

3. 3. 2. Ressource

Eine Ressource benutzt im Gegensatz zu einem Dienst keine IP-Adresse. Bei physikalischen Ressourcen, wie beispielsweise Disks, RAID-Arrays, SAN oder NAS ist es von entscheidender Wichtigkeit, dass immer nur ein Cluster-Node gleichzeitig auf die Ressource zugreift. Der Cluster sorgt auch im Falle einer Ressource für den gegenseitigen Ausschluss, muss aber bei einer Ressource nicht auch noch für das Routing einer IP-Adresse sorgen.

Eine Ressource ist also ein Dienst ohne IP-Adresse.

3. 4. Shared-Disk / Shared-Nothing

Bei der Nutzung von Speichersystemen unterscheidet man zwischen den zwei Softwaremodellen “Shared-Disk” und “Shared-Nothing”.

Im Shared-Disk-Verfahren kann eine Softwareanwendung, die auf einem der Nodes im Cluster läuft, auf beliebige Disks an anderen Nodes zugreifen.Die Konsistenz der Daten wird softwareseitig durch Locking-Mechanismen gewährleistet. Ein Distributed Filesystem wie beispielsweise CODA implementiert solche Mechanismen, was allerdings sehr aufwändig[3] ist. Das Shared-Disk-Modell erlaubt Einsparungen bei der Peripherie und lässt auf einfache Weise die gemeinsame Datennutzung zu. Allerdings hat es einen gravierenden Nachteil: Die Locking-Protokolle beanspruchen mit steigender Anzahl Nodes im Cluster immer mehr Rechenleistung und wirken sich negativ auf die Gesamtperformance aus.

1. Verwendet das Internet Mail Access Protocoll
2. Verwendet das Post Office Protocoll
3. CODA ist eines der grössten Open Source Projekte (bezogen auf die Anzahl Code Zeilen)

Das Shared-Nothing-Verfahren ordnet jede Disk im Cluster einem bestimmten Node zu und erlaubt keine gemeinsame Dateinutzung. Selbst wenn mehrere Nodes physisch mit einem Speichergerät verbunden sind, bleibt dieses zu einem bestimmten Zeitpunkt nur genau einem Node zugeordnet - ein anderer Node kann erst darauf zugreifen, wenn der eigentlich zugeordnete Node ausfällt oder die Kontrolle explizit weiter reicht. Shared-Nothing-Installationen können zwar mehr kosten, weil mehr Peripherie benötigt wird, ermöglichen jedoch eine höhere Skalierbarkeit, weil bei zunehmender Node-Zahl der Locking-Aufwand wegfällt.

3. 5. Service Primitiven

Jeder Dienst oder jede Ressource, die vom Cluster in redundanter Form zur Verfügung gestellt werden soll, muss durch ein abstraktes Application Programmable Interface (API) gekapselt werden, damit der Cluster den Service kontrollieren kann.

Dieses API muss es dem Cluster ermöglichen:
- Den Service zu aktivieren
- Den Service zu deaktivieren
- Den Service zu starten
- Den Service zu stoppen
- Dem Service mitzuteilen, dass ein neuer Node hinzugekommen ist
- Dem Service mitzuteilen, dass ein Node aus dem Verbund ausgetreten ist
- Den Service zu prüfen
- Den Status des Service abzufragen
- Dem Service mitzuteilen, dass er gescheitert ist
- Den Service zu restarten
- Die Konfiguration des Services neu zu laden

In Kapitel 6 - “Service Primitiven” auf Seite 60 werden die implementierten Service Primitiven des Service Routing Redundance Daemons genauer beschrieben und in Kapitel 6 - “Service Interface zur Shell:” auf Seite 60 werden die Service Primitiven der in dieser Arbeit implementierten Netzwerk RAID-1 Mirror Ressource besprochen.

3. 6. Service Abhängigkeiten

Gerade bei komplizierteren Services ergeben sich unweigerlich Abhängigkeiten untereinander. Ein Service wie ein Webserver ist fast immer von einer bestimmten Ressource, wie beispielsweise seiner Daten-Disk oder einer Datenbank, abhängig, und kann seinen Dienst nicht erbringen, ohne dass die entsprechende Ressource auf dem selben Cluster Node verfügbar ist.

Solche Service Abhägigkeiten sind essentiell für einen modernen Cluster und die meisten Dienste könnten ohne Service Abhängigkeiten gar nicht implementiert werden!

3. 7. Service Gruppen

Hat man erst einmal erkannt, dass Service Abhängigkeiten innerhalb eines Clusters essentiell für den Betrieb des Clusters sind, sieht man schnell, dass solche Abhängigkeiten die Menge aller Services in Gruppen aufteilt. Solche Service Gruppen bestehen aus einer Menge von Ressourcen und Diensten, die zusammengehören und eventuell sogar von einander abhängig sind.

Ein Zeit-Dienst wie beispielsweise ein Network Time Protokoll (NTP) Server kann zu einer Service Gruppe gehören, weil eine möglichst genaue Zeit für das Funktionieren einer Menge von Diensten vorteilhaft ist, ist aber von keinem anderen Service abhängig und kein anderer Service ist von ihm abhängig (da es ja nur “vorteilhaft” aber nicht “notwendig” für den Betrieb des Dienstes ist). Der NTP Dienst gehört trotzdem zur Service Gruppe, damit gewährleistet ist, dass der Zeit-Dienst immer auf demjenigen Cluster Node läuft, auf dem auch die Service Gruppe ihren Dienst erbringt.

Von anderen Services abhängige Services gehören natürlich zusammen in eine Service Gruppe, damit sie immer zusammen auf einem Cluster Node gestartet werden und ihre Abhängigkeiten, wenn immer möglich, erfüllt sind. Allgemein gesehen entspricht die Service Gruppe einer Menge von Services die *zusammen einen neuen, gemeinsamen Dienst* erbringen. Durch die Gruppe garantiert der Cluster den Services einen gemeinsamen Node zur Diensterbringung.

3. 8. Kritische und nicht-kritische Services einer Service Gruppe

Gewisse Services sind kritisch für den von der Service Gruppe erbrachten Dienst. Sollte einer dieser für die Gruppe kritischen Services seinen Dienst nicht erbringen können, kann die ganze Gruppe ihren Dienst nicht erbringen.

Ein Beispiel eines solchen Services ist eine Disk Ressource, auf der sich die Daten eines Dienstes, wie beispielsweise eines Webservers befinden. Der Webserver kann seinen Dienst, das Ausliefern von Webpages nicht erfüllen, wenn die Daten, die auf der Ressource aufbewahrt sind, nicht zur Verfügung stehen. Der Webserver-Dienst ist also von der Disk-Ressource abhängig. Ein anderer Dienst derselben Service Gruppe, beispielsweise ein NTP Zeit Server, gehört zwar ebenfalls zur selben Service Gruppe, da es erwünscht ist, dass die Zeit Stempel des Webservers eine korrekte Zeit tragen, ist aber für das Funktionieren der gesamten Service Gruppe nicht-kritisch und kann ausfallen, ohne dass der Betrieb der gesamten Service Gruppen in Frage gestellt würde.

Innerhalb einer Service Gruppe muss deshalb klar zwischen kritischen und nicht-kritischen Services der Gruppe unterschieden werden. In Kapitel 5 - “Kritische Service” auf Seite 38 wird auf das Design der kritischen Dienste einer Service Gruppe im Service Routing Redundancy Daemon eingegangen.

4. Verwandte Arbeiten

4. 1. Veritas Cluster Server

VERITAS Cluster Server (VCS) ist eine Unternehmenslösung, die auf hohe Verfügbarkeit und zur Minimierung von geplanten und ungeplanten Ausfallzeiten ausgerichtet ist. VCS soll die wachsenden und zunehmend anspruchsvolleren Verfügbarkeits-Anforderungen moderner internationaler Varianten des E-Business erfüllen. Die SAN-Strategie[1] eines Unternehmens kann durch den VERITAS Cluster Server ergänzt werden, da es den Clients auf mehrfache Weise Speicherzugriff ermöglicht; entweder direkt über Fibre Fabric oder auf Speicher-Pools über Fibre Switch.

Die wichtigsten Produktmerkmale:
- Unterstützt Solaris-, HP-UX- und Windows NT-Plattformen.
- Agenten für Oracle, Sybase, Informix Datenbanken und es werden laufend neue Agenten entwickelt.
- Kunden können mit dem Agent Development Kit für buchstäblich jeden Anwendungsdienst eigene Agenten zur Überwachung, Fehlersuche und Wiederherstellung entwickeln.
- Hohe Skalierbarkeit: unterstützt Cluster von 2 bis zu 32 Knoten.
- Unterstützt die Speichertypen führender Drittanbieter; fortlaufende Überprüfung durch das VERITAS-eigene Interoperability Lab (iLab) und die Storage Certification Suite, ein Selbsttest für Drittanbieter. Dadurch kann VERITAS Software neue Speicherlösungen unterstützen, sobald sie auf den Markt kommen.
- Die multi-thread-fähige Engine unterstützt ein mehrstufiges Fehlererkennungs-Schema
- Um eine flexible Überwachung und Ausfallübernahme zu erleichtern, können effektive, ressourcenbasierte Abhängigkeiten für Anwendungsdienste definiert werden.
- Konfigurierbare Policies erlauben es, dynamisch festzulegen, wo Recovery zur Optimierung der Ausfallübernahme erfolgen soll.
- Bietet stufenweises und automatisches Recovery nach aufeinanderfolgenden Ausfällen
- Ein hochleistungsfähiger Global Atomic Broadcast Mechanismus ermöglicht die Kommunikation zwischen Servern für zuverlässige Überwachung (Heartbeat) und Fehlermanagement.
- Lässt sich nahtlos in andere VERITAS Speichermanagement-Software integrieren, z.B. in VERITAS Volume Manager, VERITAS File System, VERITAS NetBackup und VERITAS Storage Replicator für Volume Manager.
- Die intuitive, JAVA-basierte Bedienoberflüche erleichtert die Cluster-Konfiguration: bis zu 256 Cluster mit je 32 Knoten können überwacht werden.
- Sofort im SAN mit mehreren, getesteten Konfigurationsmöglichkeiten einsetzbar.

4. 2. Windows Cluster Server

Die beiden aktuellen Hauptplayer am NT-Clustering-Markt sind der **Microsoft Cluster Server (MSCS)**, auch als “Wolfpack” bekannt, und der ursprünglich von Vinca entwickelte Co-Standby-Server, heute unter dem Label **Legato** erhältlich. **Lifekeeper von NCR** ist weniger geläufig, bietet jedoch interessante Möglichkeiten, wie stufenweises Cascading Failover über mehrere Server hinweg, sowie Recovery Kits zum automatisierten unterbruchsfreien Betrieb auf Anwendungsebene im Failover-Fall.

1. SAN ist eine Abkürzung für “Storage Area Network”

Daneben existieren auch einige High-end-Lösungen im sechsstelligen Kostenbereich, die für kleine und mittlere Betriebe allerdings kaum erschwinglich sind. Der **MSCS** ist gewissermassen kostenlos in der Enterprise Edition von Windows NT 4.0 enthalten, die laut Statistik nur etwa acht Prozent aller NT-Server-Installationen umfasst, und funktioniert ausschliesslich mit dieser Enterprise-Fassung. Anwender der Standard-Edition oder des Small-Business-Servers können den **Microsoft Cluster Server** ohne die kostspielige Enterprise-Edition-Lizenz nicht einsetzen. MSCS arbeitet im Shared-Nothing-Verfahren. Das Produkt setzt zertifizierte Hardware voraus, lässt sich also nicht mit beliebigen Server-PCs bzw. Speichergeräten einsetzen und dürfte damit für den breiten KMU-Einsatz von vornherein nicht in Frage kommen. Der klare Vorteil des Microsoft-Cluster-Servers: er wurde in Zusammenarbeit mit Cluster-erfahrenen Herstellern wie Tandem, Digital und IBM entwickelt, bietet eine offene Plattform für die Entwicklung Cluster-optimierter Anwendungen und stellt den De-facto-Industriestandard dar. In Zukunft dürfte eine steigende Zahl von Anwendungen speziell auf die Belange des MSCS hin entwickelt werden.

Der **Legato Co-Standby-Server** als Hauptkonkurrent des MSCS schlägt zwar mit etwas über 10'000 Franken zu Buche, arbeitet aber auch mit der Standard-Version von NT Server 4.0 und funktioniert auf beliebigen Server-PCs mit relativ bescheidenen Systemanforderungen. Dies bringt den Vorteil mit sich, dass bei der Installation eines Clusters bestehende ältere Server übernommen werden können; es müssen nicht zwei identische Hochleistungssysteme neu angeschafft werden. Der **Co-Standby-Server** unterstützt sowohl Shared-Disk- als auch Shared-Nothing-Konfigurationen. Scripts für gängige Server-Anwendungen werden mitgeliefert.

In der Schweiz verfügt die Legato-Lösung über eine beachtliche Basis. Vom Schweizer Distributor werden die hierzulande installierten Co-Standby-Server auf beachtliche 300 Installationen beziffert; dies sei etwa das Dreifache von Microsofts MSCS.

Verschiedene Hardware-Hersteller bieten fixfertige Zwei-Node-Cluster-Lösungen mit aller nötigen Hardware und Software an. Der “Cluster-in-a-Box” von Data General vereint zwei AVion-Server und ein RAID in einem komplett verkabelten Gehäuse mit vorinstalliertem NT, die Cluster-Administrationssoftware **ClusterX SE** und wahlweise **MSCS** oder **FirstWatch** sowie den Systemmanager NTerprise Manager.

Der ProLiant-Cluster von Compaq ist als Rackmount-Einheit ähnlich aufgebaut. Das Kernstück bilden zwei Proliant-1850R-Server und ein Fibre-Channel-Array; dazu kommen redundante Lüfter und Controller in Hot-Swap-Technik - ohne solche mehrfach vorhandenen Hardwarekomponenten macht ein Cluster ja eigentlich keinen Sinn; empfehlenswert ist zudem eine unterbruchsfreie Stromversorgung (USV).

4. 3. IPv4 und IPv6 Anycast

Im Jahr 1993 wurde ein Vorschlag für eine IPv4 Implementation von Anycast-Adressen als Request for Comment (RFC) [20] veröffentlicht. Später, wurde im Design von IPv6, in einem weiteren RFC [21] im Jahr 1998, ein spezieller, neuer IP Adressen Typ eingeführt, die IPv6 Anycast-Adresse.

Ein Paket, das an eine Anycast-Adresse geschickt wird, erreicht nicht-deterministisch einen der Server, die auf die Anycast-Adresse antworten. Dieser nicht-Determinismus breitet bei der Implementierung und Nutzung von zustandsbasierten Protokollen wie TCP grosse Probleme.

In der folgenden Tabelle wollen wir IPv4 Anycast mit IPv6 Anycast, das Anycast des neuen Internet Protokollvorschlags, vergleichen:

IPv4 Anycast	IPv6 Anycast
RFC 1546	RFC 2373
Separate Adress-Klasse	Unicast Adress-Block
Unterscheidbar von anderen Datagrammen	Nicht unterscheidbar von anderen Unicast Datagrammen
Paket ereicht im genau einen Server	Paket kann wählbar einen oder mehrere Server erreichen
Benötigt spezielles Routing	Benutzt Unicast Routing
Anycast-Adresse als Sender Adresse erlaubt	Anycast-Adresse als Sender Adresse nicht erlaubt

Tabelle E: Vergleich von IPv4 und IPv6 Anycast

Der neue Adressen-Typ soll Service Selection und Cluster Node Konfiguration vereinfachen. Es soll einfach sein Service Selection damit zu implementieren, da das ganze auf dem Netzwerk-Layer stattfindet. Die Service Selection wird mit Hilfe der Routing-Distanz entschieden.

Das Design hat aber die folgenden Probleme:
- Anycast *ist nicht-deterministisch*!
  Fragmentierte Pakete können unterschiedliche Server erreichen
- Anycast geht zwar auf das Service Selection Problem[1] ein und ermöglicht auch ein gewisses passives Service Discovery[2], unterstützt aber kein *proaktives* Service Discovery und Service Checking[3]!

Des weiteren mussten wir feststellen, dass mit [22] zwar eine IPv4 Implementation existiert, diese Implementation aber einen weltweiten Einsatz der Anycast-Adressen vorsieht, welcher dazu führt, dass das Routing der Anycast Adressen von den BGP Routern der Autonomen Systeme des Internets unterstützt werden muss. Dies erfordert Änderungen an den Internet Routern und im Border Gateway Protokoll (BGP) und ist uns in unserem Rahmen und Möglichkeiten nicht gegeben auf dieser Ebene Clustering zu betrachten. Dazu müssten mehrere Internet Router, die das Border Gateway Protokoll *BPG), ein Exterior Gateway Protokoll (EGP), sprechen können zur Verfügung stehen, und es müsste möglich sein, deren Konfiguration, wie auch deren BPG Implementation zu manipulieren.

In [23] wird ein sehr *interessantes Routing Protokoll für IPv6 Anycast Adressen* beschrieben. Es wird auch exzessiv evaluiert und leitet einige sehr interessante Resultate her, die die Performance der Ende-zu-Ende Paket Verzögerung des in beschriebenen neu entworfenen Routing Protokolles für IPv6 betreffen.

1. Das Kapitel 5 - “Service Selektion” auf Seite 30 geht genauer auf das Service Selection Problem ein.
2. Im Kapitel 5 - “Service Discovery” auf Seite 29 wird das Service Discovery genauer beschrieben.
3. Dies wird im Kapitel 5 - “Service Checking” auf Seite 44 weiter erläutert.

In [24] wird Gothic beschrieben, eine Architektur, die es erlaubt *sichere Multicast und Anycast Gruppen Zuganskontrolle* zu implementieren. Das Multicast-Problem braucht einen sicheren Multicast Mechanismus und das Anycast-Problem braucht eine sichere Methode, um Anycast Server Announcements austauschen zu können, ohne signifikante Probleme in Sicherheit und Vertraulichkeit einzuführen.

Des weiteren sieht diese Arbeit einen Einsatz des Clusters innerhalb einer Routing Domäne vor und *vereinfacht damit die Problematik auf diese einzelne Routing Domäne*. Dies führt dazu, dass eine Implementation ohne Änderungen an den Internet Routern und deren Protokollen auskommen kann. Leider scheint es, dass die Entwicklung und die Implementation von Anycast fähigen Routern sehr langsam voranschreitet und noch nicht das Experimentalstadium verlassen hat.

Anycast scheint ein sehr erfolgversprechendes Konzept zu sein, in dem allerdings, vor dem verbreiteten Einsatz, noch einige essentielle Probleme des Designs zu lösen sind, wie beispielsweise die Problematik der nicht-deterministischen Zustellung von Fragmenten an verschiedene Server. Im Moment ist Anycast leider nicht mehr als ein sehr interessantes Konzept mit einigen experimentellen Implementationen.

4. 4. Vergleich der verschiedenen Implementationen

Im folgenden wollen wir die verschiedenen Implementation miteinander vergleichen.

Wir werden ihre Vor- und Nachteile aufzeigen und erklären, weshalb wir einen neuen, anderen Ansatz in unserem Design, auf das weiter in Kapitel 5 - “Design: Service Routing Redundanz” auf Seite 27 eingegangen wird, verfolgen werden.

Die folgende Tabelle vergleicht die verschiedenen, existierenden Implementationen wie auch den, in dieser Arbeit entworfenen und von uns im Kapitel 6 - “Implementation: Service Routing Redundanz” auf Seite 47 beschriebenen Service Routing Redundancy Daemon (SRRD):

	Service Discovery	**Service Selection**	**Service Routing**	**Service Checking**
VCS	LAN, globales Clustering möglich, proprietäres Heartbeat Protokoll, Shared-Disk und Shared-Nothing	Unterstützt Gruppen, Abhängigkeiten, symmetrisches und asymmetrisches Fail-Over, GAB[a]	MAC-, IP-und DNS-Address Take-Over	proaktive oder passive Prüfung
MSCS	LAN, Windows basiert, entsprechend instabil, proprietäres Heartbeat Protokoll, Shared-Nothing	Unterstützt symmetrisches und asymmetrisches Fail-Over, GAB	MAC-, IP-und DNS-Address Take-Over	proaktive oder passive Prüfung
Life Keeper	LAN, Windows basiert, entsprechend instabil, proprietäres Heartbeat Protokoll, Recovery, Shared-Nothing	Unterstützt Gruppen, Abhängigkeiten, symmetrisches und asymmetrisches Fail-Over, GAB	MAC-, IP-und DNS-Address Take-Over	proaktive oder passive Prüfung

Tabelle F: Vergleich der existierenden Cluster Server Implementationen

	Service Discovery	Service Selection	Service Routing	Service Checking
Legato	LAN, globales Clustering möglich, proprietär, proprietäres Heartbeat Protokoll, Shared-Disk und Shared-Nothing	Unterstützt Gruppen, Abhängigkeiten, symmetrisches und asymmetrisches Fail-Over, GAB	MAC-, IP-und DNS-Address Take-Over	proaktive oder passive Prüfung
Anycast	Weltweit, muss von Internet Routern und BGP unterstützt werden	Unterstützt Gruppen, keine Abhängigkeiten, nur nichtdeterministische Verteilung, Routen Kosten basiert	Anycast Routing ist im experimental Stadium	passive Prüfung, Router basiert
SRRD	Routing Domäne weit, OSPF basiert, nicht proprietär, Recovery, Shared-Disk und Shared-Nothing, Open Source	Unterstützt Gruppen, Abhängigkeiten, symmetrisches und asymmetrisches Fail-Over, OSPF basiert	Routing Domäne weit, OSPF Protokoll basiert, nicht proprietär	proaktive oder passive Prüfung

Tabelle F: Vergleich der existierenden Cluster Server Implementationen

a. Es wird die Implementation eines Global Atomic Broadcast verwendet

Wie aus obiger Tabelle ersichtlich ist, benutzen alle existierenden Cluster Server Implementationen, ausser der Anycast Lösung und unserem Design, ein proprietäres Heartbeat-Protokoll. VCS und Legato unterstützen weltweites Clustering, doch dies geschieht soweit es dem Autor bekannt ist, mittels der DNS Take-Over, was eine lange Fail-Over Zeit mit sich bringt.

Wie ebenfalls der Tabelle zu entnehmen ist, verwenden alle Implementationen, ausser der Anycast Lösung und unser Design, nur die MAC-, IP- und die DNS-Take-Over Methode. Dies führt dazu, dass mit diese Cluster Lösungen nur Cluster auf dem gleichen lokalen Netzwerk oder weit verteilte Cluster, mit langsamen Fail-Over Zeiten, erstellt werden können.

Da die räumliche Verteilung der Cluster Node einen grossen Einfluss auf die Zuverlässigkeit und die Ausfallswahrseindlichkeit des gesammten Clusters hat, wollen wir eine neue Methode für das Service Fail-Overs entwickeln, die auch über Subnetzgrenzen hinweg, innerhalb der ganzen Routing Domäne, schnell, effizient und zuverlässig funktioniert.

Bei allen Cluster Lösungen, ausser Anycast und unserem SRRD, wurde ein Globaler Atomic Broadcast (GAB) implementiert, der garantiert, dass nie ein Service auf zwei Cluster Nodes gleichzeitig gestartet wird. Unser Design kommt in einer ersten Version ohne GAB aus, und kann deshalb nicht garantieren, dass keine Service Kollisionen geschehen, macht aber diese doch so unwahrscheindlich wie möglich. Wie im Kapitel 9 - “Schlussfolgerungen und Ausblick” auf Seite 81 erklärt wird, werden wir später versuchen GAB über unser Design zu legen und so GAB ebenfalls zu unterstützen. Ob dies möglich ist, werden zukünftige Arbeiten zeigen.

Nur VCS und Legato unterstützen Shared-Disk. Mit unserem Design ist es möglich externe, verteilte Dateisysteme[1] einzusetzen und so Shared-Disk auch zu unterstützen. Dies wird ver-

1. Wie beispielsweise das verteilte Open Source Dateisystem CODA.

mutlich auch mit einigen der anderen Produkte möglich sein. Alle besprochenen Implementationen unterstützen Shared-Nothing.

Sowohl Lifekeeper, wie auch unsere Redundanz Daemon Implementation, unterstützen Service Recovery beim Erreichen eines Fehlerzustandes. Lifekeeper ermöglicht dies mit sogenannten Recovery Kits. Unser Redundanz Daemon bietet eine spezielle Service Primitive, die in Kapitel 6 - “Service Primitiven” auf Seite 60 genauer besprochen wird, die es dem Service Implementator ermöglicht, im Fehlerfall zu versuchen das Fehlerereignis aktiv zu beheben, und erst wenn dies versagt, auf einen anderen Cluster Node zurückzugreifen.

Dadurch, dass unsere Implementation eines Cluster Servers Routing basiertes Service Routing macht, und dies mit Hilfe des vorhanden Routing Protokolles implementiert, wie in Kapitel 6 - “Implementation: Service Routing Redundanz” auf Seite 47 beschrieben wird, *ist es uns als einzige Implementation möglich, ein Routing Domänen weites Service Fail-Over zu realisieren.*

Wir verwenden auch als einzige Implementation eines Cluster Servers kein proprietäres Heartbeat-Protokoll für das Service Discovery. Dies führt zu einem vereinfachten Deployment in schon bestehenden IT Infrastrukturen.

5. Design: Service Routing Redundanz

5. 1. Routing basiertes Fail-Over Protokoll

Ein Cluster-Server hat eine einzige Aufgabe, nämlich die, einen Service so verfügbar wie möglich anzubieten. Dies beinhaltet dafür zu sorgen dass der Service garantiert auf einem der Cluster-Node des Clusters läuft wie auch des weiteren zu garantieren, dass Packete, die an den Service adressiert sind, auch den Service erreichen.

Die klassischen Methoden die Service-Erreichbarkeit auch bei Wechsel des Cluster-Nodes zu ermöglichen wurden im Kapitel 2 - “Transparenz” auf Seite 9 besprochen. Wie wir dort gesehen haben gehen die beiden gängigen Methoden davon aus dass sich alle Cluster-Nodes auf dem selben Local Area Network (LAN) Segment befinden, da beide Methoden[1] darauf basieren auf Level des ARP[2] Protokolles einzugreifen.

Auf einen ersten Blick scheint es auch dass das ARP Protokoll der richtige Ort sei um solche Adressauflösungen zu machen. Doch wenn man das genauer betrachtet, so stellt man fest das das ARP Protokoll nur dafür gedacht ist *lokale Adressen* aufzulösen, also Adressen aus dem lokalen LAN Segment!

Dies ist nun allerdings eine starke Einschränkung der Allgemeintheit! Wiso muss ein Cluster-Server sich vollständig auf dem selben lokalen LAN befinden? Sind nicht gerade die Anforderungen an einen Cluster so geartet dass eine Verteilung des Cluster auf mehrere Lokalitäten und auf mehrere LAN Segmente von grössten Vorteil wäre wie wir in Kapitel 2 - “Räumliche Verteilung des Clusters” auf Seite 10 besprochen haben?!

In diesem Kapitel 5 - “Design: Service Routing Redundanz” wollen wir eine neue Methode für das Service Routing Problem vorstellen. Wir stellen fest dass die Aufgabe des Service Routings sehr der allgemeinen Aufgabe des Routings gleicht und wir werden eine auf dem Routingprotokoll OSPF basierte Lösung des Service Routing Problemes entwerfen.

Eine weitere Aufgabe eines Clusters ist es sich über den aktuellen Zustand der Cluster-Nodes wie auch der auf den Cluster-Nodes laufenden Services zu verschaffen. Der Cluster ist darauf angewiesen dass er schnellstmöglich über Änderung der Zustände informiert wird damit der Cluster so schnell wie möglich auf diese Änderungen reagieren kann.

Die in Kapitel 4 - “Verwandte Arbeiten” vorgestellten Cluster-Server basieren alle auf einem Heartbeat-Protokoll. Das heisst die Cluster-Nodes haben aktive Verbindungen untereinander auf denen sie einen Herzschlag simulieren und so die anderen Cluster-Nodes ermöglicht ihr wohlergehen zu prüfen. Natürlich ist dies die direkteste und schnellste Methode um über Zustandsänderungen der anderen Cluster-Nodes informiert zu werden. Doch gerade das Heartbeat-Protokoll macht es auch kompliziert einen Cluster-Server zu konfigurieren!

Wir werden in den folgenden Kapitel zeigen dass Flooding-Protokolle einen sehr ändliches Ziel wie Heartbeat-Protokolle besitzen. Beide Protokolle lösen die Aufgabe Zustandsinformationen so schnell wie möglich zu verbreiten. Ein grosser Vorteil des Flooding-Protokolles ist es dass

1. Damit sind IP und MAC Address Take-Over gemeint
2. Address Resoulution Protocol

es in einer Routing-Domain schon vorhanden ist wenn ein modernes Routingprotokoll wie Open Shortest Path First (OSPF) eingesetzt wird, das ja selber schon auf einem Flooding-Protokoll basiert!

Gerade Service Zusandsänderungen können mit einem Flooding-Protokoll sehr schnell über weite Entfernungen mitgeteilt werden. Protokolle wie OSPF bieten des weiteren Möglichkeiten um mit Hilfe von “Virtual Links” auch sehr spezielle Topologien und Netze zu unterstützen. Auch spezielle Heartbeat-Netzwerke und ähnliches lassen sich somit mit Flooding-Protokollen modelieren.

5. 2. Überblick

Ein Cluster-Server Design muss die folgeden Teilprobleme besprechen und entwerfen:

- **Service Discovery**

  Dieses Teilproblem beinhaltet die ganze Heartbeat-Problematik, also die Problematik die Zustände der Cluster-Nodes wie auch der Services auf diesen Node schnellstmöglich und akurat in Erfahrungen zu bringen.

- **Service Selection**

  Nachdem der globale Zustand der Cluster-Nodes bestimmt ist, kann bebstimmt werden auf welchem Cluster-Node der Service als nächstes läuft, falls er den Node wechseln muss. Die ganze Problematik der Klassifikation von Services und Nodes wie auch ihrer Preferenzen und ihrer Zusammengehörigkeit sind Teil der Service Selection.

- **Service Routing**

  Läuft der Service auf einem Cluster-Node, muss dafür gesorgt werden dass er für alle anderen auch erreichbar ist.

- **Service Checking**

  Einen Service zu starten ist keine Garantie für seine Verfügbarkeit. Ein Service muss in regelmässigen Abständen geprüft werden und falls eine solche Prüfung ergibt dass seine Verfügbarkeit eingeschränkt ist, muss der Service gestoppt, als Fehlerquelle markiert, und auf einem neuen Cluster-Mode gestartet werden.

In den folgenden Kapiteln werden wir jedes einzelne dieser Teilprobleme betrachten und das von uns gewählte Design vorstellen. Wir werden Alternativen besprechen und aufzeigen wie eine vollständig auf Routing basierte Clusterlösung aussehen kann. Im Kapitel 6 - “Implementation: Service Routing Redundanz” werden wir dann den von uns implementierten Service Routing Redundancy Daemon besprechen, der alle in diesem Kapitel besprochenen Methoden in einem funktionsfähigen, einfach konfigurierbaren, sehr offenen und modularen Unix Daemon vereint.

5. 3. Service Discovery

Als erstes wollen wir besprechen, wie unser Cluster-Server die Zustände der Cluster-Nodes und der Services, die auf diesen laufen bestimmt und beobachtet. Bisherige Cluster Lösungen haben ein sogenanntes Heartbeat-Protokoll implementiert.

Ein Heartbeat-Protokoll ist ein Protokoll, das den Zustand verschiedener Server beobachtet und aktiv prüft, ob die Server verfügbar sind. Sollte ein Server nicht so verfügbar sein, wie es das Heartbeat-Protokoll erwartet, führt das Heartbeat-Protokoll gewisse Arbeiten aus. Sowohl die Definition der Verfügbarkeit, wie auch die Definition der ausgeführten Arbeiten bei Zustandswechsel der Server, sind frei wählbar.

Es gibt Open Source Implementationen von solchen Heartbeat-Protokollen, von denen wir im speziellen Heartbeat von Linux-HA [11] erwähnen möchten, bietet es doch all diese oben erwähnten Möglichkeiten.

Wir wollen hier allerdings die Frage aufwerfen, ob ein solches Heartbeat-Protokoll überhaupt notwendig ist, und ob es nicht andere Möglichkeiten gibt, solche Server und Service Zustandsinformationen zu verteilen.

Die Grundaufgabe ist einfach. Jeder Node eines Clusters muss wissen, welche anderen Nodes des Clusters für den einen gewissen Service verfügbar sind. Des Weiteren muss jeder Node so schnell wie möglich erfahren, wenn ein anderer Node seinen Zustand wechselt oder ausfällt.

Prinzipiell müssen also die folgenden Ereignisse und Informationen festgestellt werden:
- Das Starten und Stoppen des Services auf einem Node
- Das Hinzukommen eines Nodes und das Verlassen des Clusters eines Nodes
- Die Konfiguration jedes einzelnen momentan vorhandenen Nodes

Die Ereignisse und Informationen des zweiten und dritten Punktes werden benutzt, um zu jedem Zeitpunkt zu wissen, welche anderen Cluster-Nodes vorhanden sind, die eventuell den Service bei einem Ausfall übernehmen können. Diese Ereignisse und diese Informationen müssen so schnell wie möglich allen Nodes des Clusters mitgeteilt werden.

Bei genauer Betrachtung hat ein Flooding-Protokoll, wie es beispielsweise im Routing Protokoll OSPF eingesetzt wird, genau diese Eigenschaften! Genauer genommen haben alle *Link State Routing Protokolle* diese Eigenschaften, denn sie alle basieren auf einem Flooding-Protokoll, im Gegensatz zu Distance-Vector Routing Protokollen wie das Routing Information Protokoll (RIP), die keine Flooding-Protokolle verwenden.

Ein Flooding-Protokoll hat die Aufgabe solche Zustandsinformationen, im Fall von OSPF Routing Informationen, so schnell und so weit wie nur möglich zu verteilen. Es erledigt dies dadurch, dass es die zu verteilende Information an jeden erreichbaren Nachbarn weiterschickt, der dann ebenfalls das selbe tut. So wird das Netzwerk auf die schnellstmögliche Art und Weise mit der Information geflutet.

Um dies nicht ineffizient werden zu lassen, wird meistens eine Form von *intelligentem Flooding* eingesetzt. Intelligentes Flooding berücksichtigt die Topologie des Netzes und schickt beispielsweise keine Pakete mehr in die Richtung, aus der das ursprüngliche Paket empfangen wor-

den ist, mit der Annahme, dass die Information dort schon bekannt ist, ist sie doch gerade von dort her gekommen.

Auf diese Art und Weise verbreitet ein Flooding-Protokoll seine Informationen global im Netzwerk und jeder Knoten des Netzes kann sich daraus eine globale Sicht der vorhandenen Informationen ableiten. Ein Link State Routing Protokoll wie OSPF kann so aus der globalen Sicht der vorhandenen Links, Router und Netze den kürzesten Pfad zu jedem Knoten im Netz berechnen.

Doch welche Informationen sollen im Netzwerk verbreitet werden? In unserem Design, das für ein Flooding-Protokoll vorgesehen ist, wird nur noch echte Zustandsinformation verbreitet.

Diese Zustandsinformationen beinhalten:
- Welche Cluster-Nodes vorhanden sind
- Unter welcher Adresse und welchem Port der Webserver des Cluster-Nodes zu erreichen ist
- Welcher Cluster-Node welche Services anbieten kann
- In welchem Zustand die entsprechenden Services sind
- Failure-Codes falls der Service einen Failed-Zustand erreicht
- Wie die entsprechenden Services konfiguriert sind

Hat ein Cluster-Node alle diese Informationen des gesamten Cluster-Servers, kann er lokal, so wie in Kapitel 5 - “Service Selektion” auf Seite 30 besprochen wird, entscheiden, welche nächste Aktion, wie starten oder stoppen eines Services, von ihm erwartet wird.

Dieses Vorgehen, durch eine globale Sicht der Zustände einer Sache lokal eine Entscheidung zu treffen, ist allen Link State Routing Protokollen eigen und muss auch in unserem Cluster Design berücksichtigt werden, da wir die Cluster-Node Zustandsinformationen ebenfalls mit einem Flooding-Protokoll verteilen wollen.

Grerade weil Link State Routing Protokolle inherent schon einen Flooding Mechanismus besitzen, bietet es sich an, Service und Cluster-Node Zustände mit dem selben Protokoll zu verbreiten und so den Vorteil zu haben, kein weiteres Protokoll, ausser dem schon vorhandenen Routing Protokoll, unterstützen zu müssen!

Sowohl Routing Information für ganze Netze, wie auch Routing und Zustandsinformation für einen einzelnen Service, sind beides Zustandsinformationen, die so schnell wie möglich, so weit als möglich verbreitet werden sollen! *Prinzipiell ist zwischen den zwei Informationen kein Unterschied festzustellen und wir wollen im Rahmen unseres Cluster-Servers auch keinen Unterschied zwischen ihnen machen!*

5. 4. Service Selektion

Da unser Cluster-Server Design auf dem Flooding Mechanismus eines Link State Protokolles basieren soll, und da solche Protokolle darauf angewiesen sind, aus der globalen Sicht aller Zustände eines Systems eine lokale Entscheidung zu fällen, muss auch unser Fail-Over Protokoll fähig sein, aus der globalen Sicht der Zustände der einzelnen Cluster-Nodes und ihrer Services eine Entscheidung darüber zu fällen, was als Nächstes zu tun ist.

Wir werden eine ereignisgesteuerte Service Zustandsmaschine entwerfen, auf der sich die Services abbilden lassen. Die globale Sicht aller Service Zustände und aller Service Konfiguratio-

nen wird es unserer Zustandsmaschine dann ermöglichen, unter Beobachtung eines Ereignisses, wie eines Updates oder das Löschen einer Servicezustandsinformation, zu reagieren.

Im Folgenden gehen wir zuerst auf die Service Konfiguration, wie Service Preferenz und Service Typ, ein und betrachten dann die Zustandsmaschine und die verschiedenen Service Zustände.

5. 4. 1. Service Preferenzen

Wir wollen in unserem Cluster-Server Design verschiedene Arten von Fail-Over unterstützen. Sowohl assymmetrische wie symmetrische Fail-Over Varianten sollen möglich sein.

Um dies zu unterstützen hat jeder Service eine Service Präferenz, eine frei wählbare Zahl.

Ist ein Service auf zwei Cluster-Nodes mit der selben Präferenz konfiguriert, so spechen wir von symmetrisch konfigurierten Services, die im nächsten Kapitel genauer besprochen werden. Sind die Präferenzen nicht gleich, so sprechen wir von asymmetrisch konfigurierten Services, die wir im übernächsten Kapitel besprechen werden.

5. 4. 1. 1. Symmetrisch konfigurierte Services

Ist ein Service auf zwei oder mehr Cluster-Nodes mit der selben Präferenz konfiguriert, so sprechen wir von einem symmetrisch konfigurierten Service. Symmetrisch konfigurierte Services bevorzugen keinen der Cluster-Nodes.

Symmetrische Services haben das folgende Verhalten:
- Haben ein oder mehrere Cluster-Nodes die selbe Präferenz und ist kein Cluster-Node verfügbar, der eine höhere Präferenz hat, so wird der Service auf dem Cluster-Node gestartet, der die höchste IP-Adresse hat, falls er noch nicht auf einem der beiden Cluster-Nodes läuft.
- Sollte der Service schon auf einem der Cluster-Nodes am laufen sein, so wird nichts getan.

Dieses Verhalten führt dazu, dass der Service den Cluster-Node *so selten wie möglich und so oft wie nötig* wechselt. Wenn ein Cluster-Node versagt, so wechselt der Service auf einen der anderen Cluster-Nodes mit gleicher Präferenz, der Service bleibt aber dann auch dort, wenn der Cluster-Node, der versagt hat, wieder verfügbar ist.

5. 4. 1. 2. Asymmetrisch konfigurierte Services

Ist ein Service auf zwei oder mehr Cluster-Nodes mit unterschiedlicher Präferenz konfiguriert, so sprechen wir von einem asymmetrisch konfigurierten Service. Bei asymmetrisch konfigurierten Services wird derjenige Cluster-Node mit der höchsten Präferenz bevorzugt, solange er verfügbar ist.

Asymmetrische Service haben das folgende Verhalten:
- Ein Cluster-Node, der einen Service mit einer tieferen Präferenz konfiguriert hat, als ein anderer Cluster-Node, wird den Service nie starten, solange der Cluster-Node mit der höheren Präferenz verfügbar ist.

Dieses Verhalten führt dazu, dass der Service in einer genau bestimmten Reihenfolge von Cluster-Node zu Cluster-Node fällt, wenn die Cluster-Nodes mit höherer Preferenz ausfallen. Al-

lerdings führt dieses Verhalten zu eigentlich überflüssigen Fail-Over's, da der Service sofort auf einen Cluster-Node mit höherer Präferenz zurückfällt, wenn dieser wieder neu verfügbar ist.

Ein solches Verhalten kann aber auch erwünscht sein, wenn ein Service auf einem bestimmten Cluster-Node eine bessere oder grössere Leistung erbringen kann, als auf einem anderen Cluster-Node. Dann ist es erwüscht dass der Service bevorzugt auf dem leistungsfähigeren Cluster-Node läuft, auch wenn dies ein zusätzliches Fail-Over bei erneuter Verfügbarkeit des leistungsfähigeren Cluster-Nodes bedingt.

5. 4. 2. Service Typen

Unser Cluster-Server Design soll verschiedene Typen von Services unterstützen. Einige der im Kapitel 4 - "Verwandte Arbeiten" vorgestellten Cluster-Server Lösungen unterstützen eine Vielzahl von Service Typen. Diese Cluster-Server unterscheiden unter den vielen Service Typen, um dem Benutzer ein möglichst vereinfachtes Benutzerinterface präsentieren zu können.

Betrachtet man die Service Typen genauer, so erkennt man, dass es im Prinzip nur zwei verschiedene Service Typen gibt, diese beiden Typen aber in einer Vielzahl von Variationen vorkommen. Die verschiedenen Variationen unterscheiden sich nicht in ihrer Funktion, sie abstrahieren aber eine Vielzahl von unterschiedlichen Services.

Dies mag für ein komerzielles Produkt vorteilhaft sein, für unser Design wollen wir aber nur die zwei essentiellen Grundtypen Dienst und Ressource betrachten. Weitere Variationen können direkt im Dienst oder der Ressource selber implementiert und gekapselt werden, und sollen uns hier nicht weiter beschäftigen.

In Kapitel 6 - "Ressource: Netzwerk RAID-1 Array" wird die Implementation einer komplexen Ressource im Detail besprochen und es zeigt, wie weitere Variationen von Resourcen und Diensten bei Bedarf implementiert werden können.

5. 4. 2. 1. Dienst

Ein Dienst ist ein Service, der eine oder mehrere IP-Adressen zum Betrieb braucht. Das heisst, ein Dienst ist ein Netzwerksdienst, der einen oder mehrere SAP's[1] anbietet. Diese SAP's können ein oder mehrere Ports auf einer oder mehreren IP-Adressen darstellen.

Ein solcher Dienst ist darauf angewiesen, dass vor seinem Start ein oder mehrere Interfaces richtig konfiguriert sind, damit er seine SAP's auch an die Interfaces binden kann. Natürlich muss nach einem Stop des Dienstes auch dafür gesorgt werden, dass die entsprechenden Interfaces wieder verschwinden, damit der Dienst auf einem anderen Cluster-Node auch von dem, den Dienst stoppenden Cluster-Server aus, erreichbar ist. Kapitel 5 - "Interface Alias" auf Seite 39 geht genauer auf die Interfaces und Kapitel 5 - "Dynamisches Routing von Interfaces" auf Seite 39 geht genauer auf das Service Routing der Service IP-Adressen ein.

5. 4. 2. 2. Ressource

Eine Ressource ist ein Dienst ohne IP-Adresse. Das heisst, dass eine Ressource keinen SAP besitzt und deshalb auch kein Netzwerk Dienst sein kann. Eine Ressource stellt eine unter den Clu-

1. Service Access Points

ster-Nodes eines Cluster-Servers gemeinsam genutzte physikalische Ressource dar. Die Ressource wird in den meisten Fällen von Diensten genutzt, die auf dem selben Cluster-Server konfiguriert sind. Meistens werden Datenspeicher wie Disks als Ressourcen auf dem Cluster abgebildet.

Eine Ressource wird in unserem Design genau gleich behandelt wie ein Dienst. Eine Ressource erfährt genau die gleichen Erreignisse wie ein Dienst. Der einzige Unterschied ist, dass für eine Ressource kein Interface vorhanden sein muss, und dass kein Service Routing nötig ist. Eine Ressource wird immer lokal von Diensten auf dem Cluster-Node verwendet.

Um sicherzustellen, dass Dienste auch die nötigen Ressourcen für ihren Betrieb auf dem Cluster-Node, auf dem sie laufen sollen, vorfinden, werden wir in Kapitel 5 - “Service Gruppen” auf Seite 37 das Konzept der Service Gruppe und der Service Abhängigkeiten einführen, deren Implementation im Rahmen des Service Routing Redundancy Daemons wir dann im Kapitel 6 - “Implementation: Service Routing Redundanz” auf Seite 47 genauer besprechen werden.

5. 4. 3. Service Zustände

Eine der Informationen, die unser Cluster-Server Design vorsieht unter den Cluster-Nodes auszutauschen, ist der Service Zustand. Der Service Zustand drückt den Laufzeit-Zustand des Services aus und beinhaltet auch den Zustand der Service Konfiguration. Jeder Zustand unseres ereignisgesteuerten Zustandsautomaten besitzt somit eine Bezeichnung der Form

```
<Konfigurations Zustand> / <Service Zustand>
```

Mögliche Zustände für die Konfigurationen sind:
- disabled
- inactive
- injected
- active
- deactivated
- failed

Mögliche Zustände für den Service sind:
- stopped
- starting
- started
- stopping
- failed

Wobei nur Zustände der Form

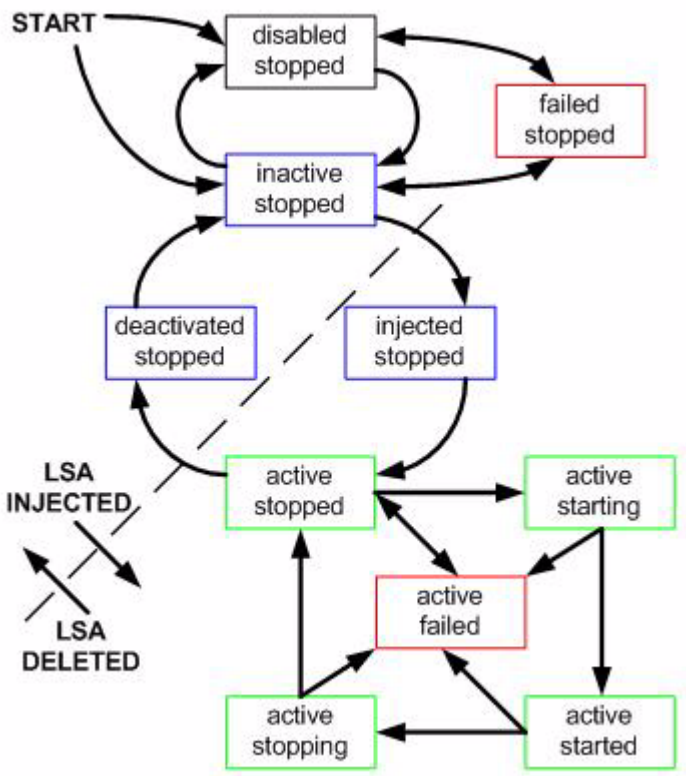
```
active / <Service Zustand>
```

aktiv ins Netz geflutet und verteilt werden, wie auch die folgende Figur 7: ”Inaktive und aktive Zustände der Service-Zustandsmaschine” auf Seite 34 zeigt.

5. 4. 3. 1. Übersicht

Als Erstes wollen wir uns einen Überblick über die Gesamte, im Design des Cluster-Servers verwendete, Zustandmaschine verschaffen.

Die folgende Figur zeigt die vollständigen Zustandsübergänge des entworfenen Zustandsautomaten.



Figur 7: Inaktive und aktive Zustände der Service-Zustandsmaschine

Prinzipiell gibt es zwei Arten von Zuständen, namentlich Active und Inaktive. Ein aktiver Zustand bedeutet, dass der Zustand auch ins Netz geflutet wird und als Zustandsinformation den anderen Cluster-Nodes zur Verfügung steht.

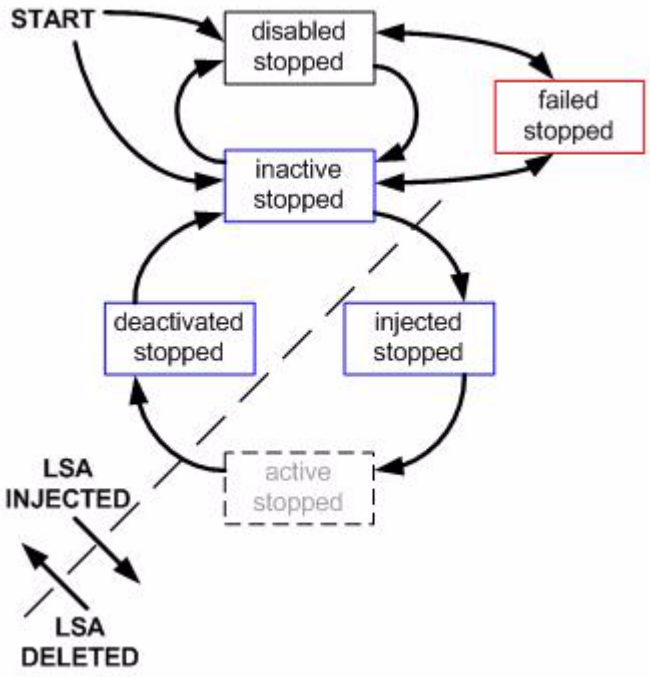
Wir werden dieses Zustandsübergansdiagramm mit Hilfe eines *ereignisgesteuerten Zustandsautomaten* implementieren. Als Ereignisse werden Service Zustandsinformations-Updates und Löschung, Zustandsänderung des verwendeten Routing Daemons wie auch Benutzerinputs über den Konfigurations-Webserver angesehen.

Die Zustandsmaschine startet entweder im Zustand **disabled/stopped** oder im Zustand **inactive/stopped**, je nachdem ob der Service im Moment disabled, also abgeschaltet, konfiguriert ist oder nicht.

Im Folgenden werden wir zuerst die inaktiven Zustände besprechen, um dann im nächsten Kapitel auf die aktiven Zustände, in denen der Service auch wirklich gestarted werden kann, einzugehen

5. 4. 3. 2. Inaktive Zustände

In der folgenden Figur sind die inaktiven Service Zustände des Zustandsautomaten dargestellt.



Figur 8: Inaktive Zustände der Service-Zustandsmaschine

Ein Service befindet sich als Erstes im Zustand **inactive/stopped**. Services in diesem Zustand sind bereit aktiviert zu werden, sind aber noch inaktiv. Ein Services in diesem Zustand kann disabled werden und fällt dann in den Zustand **disabled/stopped**. In diesem Zustand ist der Service explizit abgeschaltet und kann gar nicht versehentlich aktiviert werden.

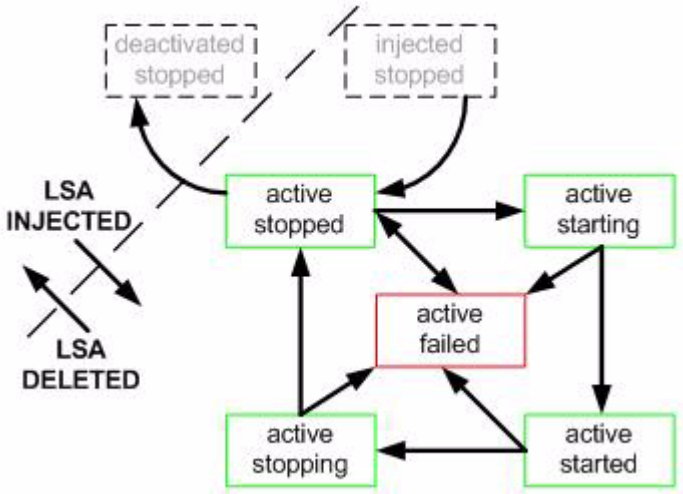
Wird ein Service aktiviert, so fällt er als Erstes in den Zustand injected/stopped und verweilt dort solange seine Zustandsinformationen zwar schon dem Subsystem zum Fluten durch das Netz übergeben wurde, aber sein eigenes Echo dieses Flutens noch nicht gehört wurde. Solange dies der Fall ist, muss er somit davon ausgehen, dass auch die anderen Cluster-Nodes noch nichts von diesen Zustandsinformationen wissen.

Sobald dann dieses Echo des Flutprozesses gehört wurde, geht der Service in den Zustand **active/stopped** über und wird zum aktiven Service, der den anderen Cluster-Nodes bekannt ist.

Sollte dieses Einfügen der Zustandsinformationen in die globale Zustandsinformationsdatenbank scheitern, so geht der Service in den Zustand **failed/stopped** über, aus dem der Service nur nach einer Reaktivierung wieder in einen der inaktiven Zustände wechseln kann.

5. 4. 3. 3. Aktive Zustände

Wurde die Zustandsinformation eines Services erfolgreich emitiert, so erreicht ein Service einen aktiven Zustand. Solange ein Service sich in einem aktiven Zustand befindet, sind auch die anderen Cluster-Nodes des Cluster-Servers über den Zustand informiert und jeder Cluster-Node, der einen aktiven Service besitzt, muss damit rechnen, dass von ihm erwartet werden kann, dass er den Service startet.



Figur 9: Aktive Zustände der Service-Zustandsmaschine

Ein frisch aktivierter Service wird als Erstes in den Zustand **active/stopped** geraten. In diesem Zustand ist den anderen Cluster-Nodes der Service auf dem aktuellen Cluster-Node bekannt.

Die anderen Cluster-Nodes verlassen sich darauf, dass der aktuelle Cluster-Node die Service-Zustände der anderen Cluster-Nodes beobachtet und bemerkt, wenn er zu demjenigen Cluster-Node mit der höchsten Präferenz wird.

Die anderen Cluster-Nodes sind darauf angewiesen, dass ein Cluster-Node, der einen Service in aktivem Zustand hat, auch wirklich bereit ist, den Service zu starten, wenn ein anderer Cluster-Node versagt.

Hat ein Cluster-Node aus der globalen Sicht der Service-Zustände bestimmt, dass er den Service starten muss, so ändert sich der Service Zustand in **active/starting**. Natürlich wird diese Zustandsänderung, da sie eine Zustandsänderung eines aktiven Services betrifft, auch wieder ins Netz geflutet.

Sollte ein anderer Cluster-Node den Service im Zustand **active/started** haben, so wird er seinen Service auf der Stelle stoppen, also einen Zustandsübergang in **active/stopping** machen, um nicht mit dem startenden Cluster-Node zu kollidieren. Startende Services werden so lange im Zustand **active/starting** verharren, bis alle anderen Cluster-Nodes den Service gestoppt haben.

Ein Cluster-Node, der das Echo seines eigenen Zustandsüberganges in den Zustand **active/starting** hört, testet, ob alles für einen Service Start in Ordung ist.

Dazu gehört das Sicherstellen dass:
- Alle anderen Cluster-Node den Service im Zustand active/stopped haben
- Dass alle Service Abhänigkeiten, die in Kapitel 5 - “Service Gruppen” auf Seite 37 und Kapitel 5 - “Service Abhängigkeiten” auf Seite 38 genauer erläutert werden, dieses Services erfüllt sind

Sind alle diese Start-Bedingungen erfüllt, wird der Service auf diesem Cluster-Node gestartet. Der Service hat die Möglichkeit den Startvorgang zu verzögern, falls er selber auch noch Zeit braucht, um Startvorbereitungen zu erledigen. Hat der Service seinen Start vollzogen, wird er dies signalisieren und der Service-Zustand ändert sich in **active/started**. Dieser Zustandsübergang wird natürlich auch gleich wieder ins Netz geflutet.

Soll ein gestarteter Service im Zustand **active/started** gestoppt werden, so wird er den Zustandsübergang in den Zustand **active/stopping** machen.

Ein Cluster-Node, der das Echo seines eigenen Zustandsüberganges in den Zustand **active/stopping** hört, testet, ob für einen Service Stop alles in Ordunung ist.

Dazu gehört das Sicherstellen dass:
- Dass alle Service Abhängigkeiten, die in Kapitel 5 - “Service Gruppen” auf Seite 37 und Kapitel 5 - “Service Abhängigkeiten” auf Seite 38 genauer erläutert werden, dieses Services erfüllt sind. Dies beinhaltet das Stoppen aller Services einer Service Gruppe, falls der Service kritisch für den Betrieb der Service Gruppe ist, wie in Kapitel 5 - “Kritische Service” auf Seite 38 noch genauer erläutert wird.

Sind alle diese Stop-Bedingungen erfüllt, wird der Service auf diesem Cluster-Node gestoppt. Der Service hat die Möglichkeit, den Stopvorgang zu verzögern, falls er selber auch noch Zeit braucht um Stopvorbereitungen zu erledigen. Hat der Service seinen Stop vollzogen, wird er dies signalisieren und der Service-Zustand ändert sich in **active/stopped**. Dieser Zustandsübergang wird natürlich auch gleich wieder ins Netz geflutet und kann Auslöser sein, dass ein anderer Cluster-Node den Service bei sich startet.

Falls einer der Schritte des Startes oder des Stops eines Service schief geht, so wird der Service in den Zustand **active/failed** gesetzt und verharrt dort solange, bis er reaktiviert wird. Natürlich wird auch dieser Zustandsübergang sofort ins Netz geflutet, ist doch auch **active/failed** ein aktiver Service Zustand.

5. 4. 3. 4. Service Gruppen

Damit Dienste über ihre Daten auch in hochverfügbarer Form verfügen können, und damit Services die keine IP-Adressen benutzen implementiert werden können, haben wir in unserem Cluster-Server Design Ressourcen vorgesehen.

Damit auch garantiert werden kann, dass ein Dienst, die für seine Dienstleistung nötigen Ressourcen zur Verfügung hat, sind Service Gruppen im Design des Cluster-Servers vorgesehen. Service Gruppen bilden auch die Grundlage für Service Abhängigkeiten, die wir im folgenden Kapitel genauer besprechen

5. 4. 3. 5. Service Abhängigkeiten

Ist ein Service von einem anderen Service, das kann ein Dienst oder eine Ressource sein, abhängig, so garantiert der Cluster-Server, dass der abhängige Service erst nach dem anderen Service gestartet wird, und dass der abhängige Service immer vor dem anderen Service gestoppt wird.

Ein abhängiger Dienst kann also davon ausgehen, dass, wenn er gestartet wird, alle seine Abhängigkeiten erfüllt sind. Sind die Abhängigkeiten eines Services nicht erfüllt, so wird er im Zustand **active/starting** verharren, bis seine Abhängigkeiten erfüllt sind, oder sein Start abgebrochen wird.

Genau dasselbe in umgekehrter Form geschieht beim Stoppen eines Services, von dem ein oder mehrere andere Services abhängig sind.

Zuerst werden alle abhängigen Services, seien das Dienst oder Ressourcen gestoppt, und der, das ganze auslösende Service, verharrt in gestarteter Form im Zustand **active/stopping**. Erst wenn alle abhängigen Services gestoppt sind, wird auch dieser Service gestoppt und er verlässt den Zustand active/stopping und geht ebenfalls über in den Zustand **active/stopped**.

Dieses Verhalten garantiert, dass die Abhängikeiten eines Service garantiert erfüllt sind, wenn ein Service gestartet wird.

5. 4. 3. 6. Kritische Service

Gewisse Services einer Service Gruppe können als kritisch für die Gruppe markiert werden.

Dienste und Ressourcen einer Service Gruppe werden gemeinsam gestoppt, sobald ein als kritisch gekennzeichneter Service gestoppt wird oder versagt. Dies garantiert, dass die ganze Service Gruppe einen Cluster-Node verlässt, sobald ein kritischer Service der Gruppe auf einem der Cluster-Nodes versagt.

So kann eine Service Gruppe so konfiguriert werden, dass, wenn ihr die Grundlage für ihren Betrieb entzogen wird, beispielsweise eine Daten enthaltende Ressource versagt, und dies die ganze Service Gruppe beeinträchtigt, die ganze Gruppe gestoppt wird, den Cluster-Node verlässt und auf einem anderen Cluster-Node, nach Start der ganzen Service Gruppe dort, den Service der Service Gruppe wieder erbringt.

5. 5. Service Routing mit einem Link State Protokoll

Service Routing ist Aufgabe des Routing Protokolles. Dies wird allerdings in den im Kapitel 4 - “Verwandte Arbeiten” vorgestellten Cluster-Server Lösungen nicht so gesehen. Die klassischen Methoden für das Service Routing im Cluster, die in Kapitel 2 - “Transparenz” auf Seite 9 besprochen wurden, verwenden andere Methoden des Service Routings im LAN. Da sich diese Cluster-Server alle auf dem selben LAN befinden, haben sie auch recht, dass sie diese Methoden, wie MAC und IP Address Take-Over, verwenden, sind dies doch die effizientesten und schnellsten Methoden im lokalen Netzwerk.

Sobald allerdings nicht-lokale Cluster betrachtet werden, was wir in dieser Arbeit tun wollen, versagen die klassischen Methoden des Fail-Overs bzw. des Service Routings, da sich diese

Methoden nicht über Subnetzgrenzen hinweg anwenden lassen.

Wir bedienen uns einer anderen Methode, die im Grunde nicht neu ist, aber eigentlich noch nie irgendwo automatisiert worden ist. “Von Hand” wurde ähnliches sicher schon oft gemacht, ähnelt es doch im Prinzip dem aufsetzen eines Point-to-Point Links zwischen zwei Systemen, von denen eines am dynamischen Routing teilnimmt.

Wir werden im folgenden Kapitel 5 - “Interface Alias” auf Seite 39 zuerst genauer auf die Voraussetzungen eingehen, dass ein Dienst, denn nur für Dienste ist es nötig IP-Adressen zu routen, überhaupt gestartet werden kann, um dann in Kapitel 5 - “Dynamisches Routing von Interfaces” auf Seite 39 genauer zu betrachten, wie wir es anstellen, dass auch andere Systeme den Dienst über das Netzwerk erreichen können.

5. 5. 1. Interface Alias

Damit ein Dienst, Ressourcen brauchen keine IP-Adressen, starten kann, muss die IP-Adresse, unter der er laufen soll,und unter der er ereichbar sein soll, als Interface existieren. Um dies zu erreichen, behelfen wir uns eines erlaubten Tricks.

Wir konfigurieren ein Interface Alias, auf dem der Service laufen soll und konfigurieren das Interface Alias mit der IP-Adresse des Dienstes und einer Netzmaske von 255.255.255.255.

“Von Hand” würde ein solches Kommando etwa so aussehen:

```
ifconfig eth0:0 172.16.15.1 netmask 255.255.255.255
```

Das heisst: wir spiegeln dem System vor, wir hätten einen Point-to-Point Link zu einem Server mit der Dienst IP-Adresse. Da das Interface aber ein Interface Alias und kein echtes Interface ist, werden wir die IP-Pakete für die Dienst IP-Adresse akzeptieren und wie ein normales Interface Alias verarbeiten.

Dies bedeutet, dass ein Dienst ohne Probleme starten und auf das Interface Alias mit der IP-Adresse des Dienstes binden kann.

Der Dienst kann also ganz normal starten. Es ist für ihn kein Unterschied festzustellen, wenn er so auf einem Iterface Alias gestartet wird.

Ist ein Dienst so gestartet, kann er vom lokalen System aus erreicht werden und Clients auf dem lokalen System können ab dann mit dem Dienst arbeiten. Damit Clients auf anderen Systemen die Dienst IP-Adresse auch erreichen können, muss die IP-Adresse auch noch gerouted werden, was wir im folgenden Kapitel besprechen.

5. 5. 2. Dynamisches Routing von Interfaces

Ist ein Dienst einmal auf einem Interface Alias gestartet, so ist er vom lokalen System erreichbar, doch alle anderen Systeme wissen vorest einmal nicht, wie sie diese IP-Adresse erreichen können.

Um dem abzuhelfen, muss der vorhandene Routing Daemon umkonfiguriert werden, so dass er das neue Interface Alias als gültiges zu routendes Interface sieht.

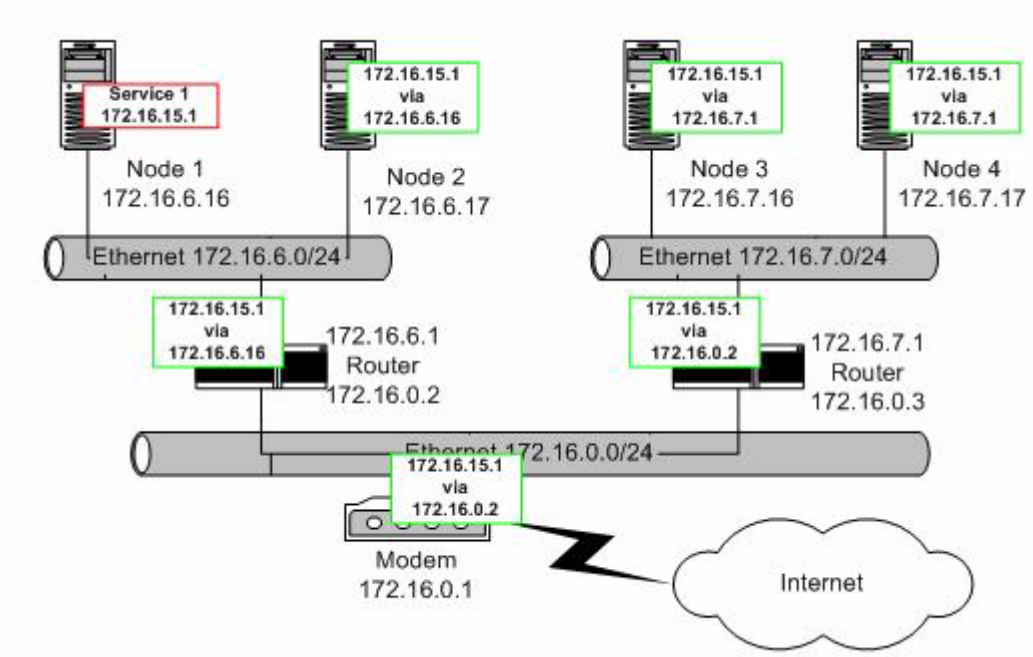
Sobald dies gesehen ist, wird der Routing Algorithmus dafür sorgen, dass alle anderen Router wissen, wie die Dienst IP-Adresse zu erreichen ist.

Da das Interface eine Netzmaske von 255.255.255.255 besitzt, **wird die Dienst IP-Adresse als Host-Route** angesehen, und es wird, je nach Routing Algorythmus, eine Host Route im ganzen Netzwerk installiert, die dafür sorgt, dass alle Router, und dadurch auch alle die Router verwendenden Clients, die Dienst IP-Adresse erreichen können.

Wird, wie in unserem Design, das Routing Protokoll OSPF verwendet, so kann durch dessen kurze Konvergenzzeit bei Routingänderungen eine sehr kurze Fail-Over Zeitspanne garantiert werden. In Kapitel 2 - “Transparenz” auf Seite 9 werden die verschiedenen Zeiten der Address-Take-Over der verschiedenen Methoden, miteinender verglichen und betrachtet.

Im Folgenden wollen wir die Routing Tabellen der Router und Cluster-Nodes eines hypothetischen Cluster-Servers betrachten, wenn der Dienst zuerst auf dem ersten, dann auf dem zweiten und zuletzt auf dem vierten Cluster-Node des Cluster-Servers läuft. Im betrachteteten Beispielsnetzwerk läuft das Routing Protokoll OSPF auf allen Routern und auf den Cluster-Nodes.

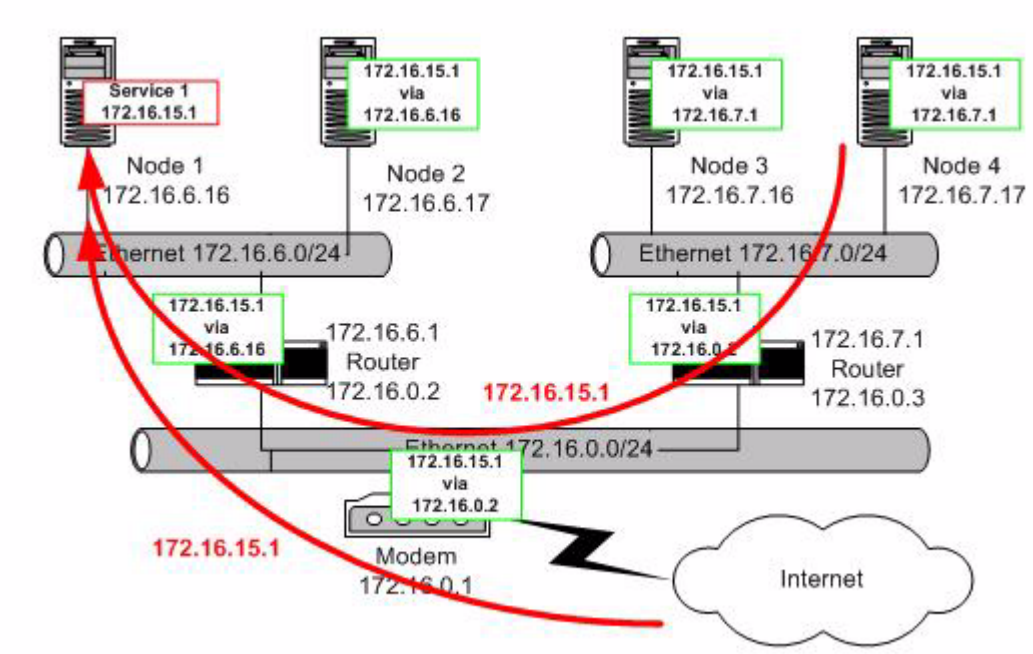
Die folgende Figur zeigt den Dienst, wenn er auf dem Cluster-Node 1 läuft:



Figur 10: Dienst auf Node 1 des Clusters

Wie man sieht, haben alle anderen Cluster-Nodes Routing Tabellen für die Dienst IP-Adresse 172.16.15.1, die auf den Cluster-Node 1 zeigen. Pakete von jeder möglichen Quelle nehmen den kürzesten Weg zum Dienst, wie es auch nicht anders von einem richtig funktionierenden Routing Protokoll zu erwarten ist.

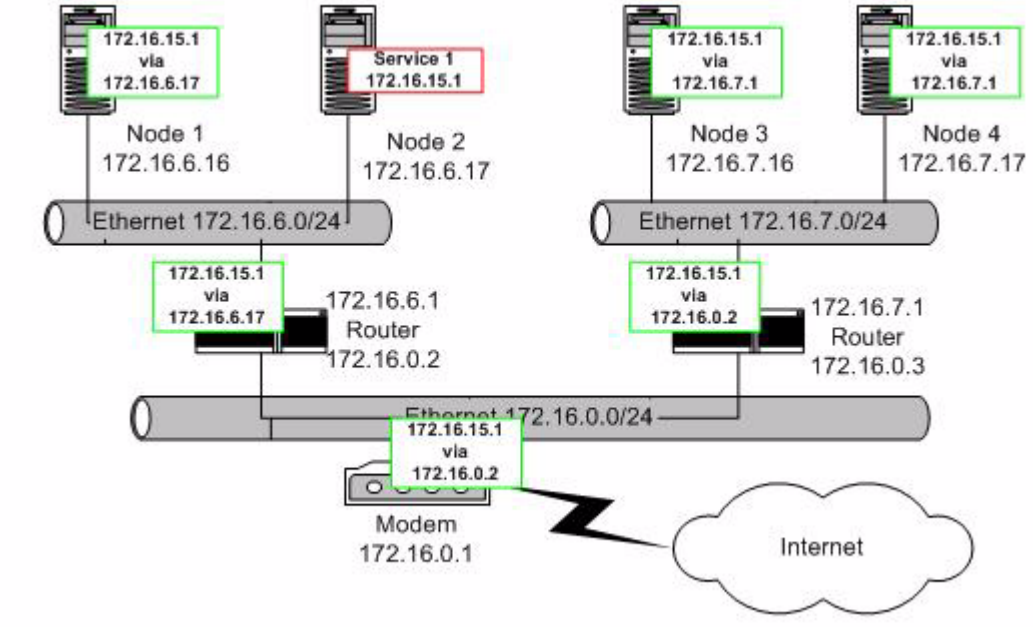
Betrachten wir den Datenfluss im Netzwerk für die Dienst IP-Adresse, so sieht er folgendermassen aus:



Figur 11: Datenfluss bei Dienst auf Node 1 des Clusters

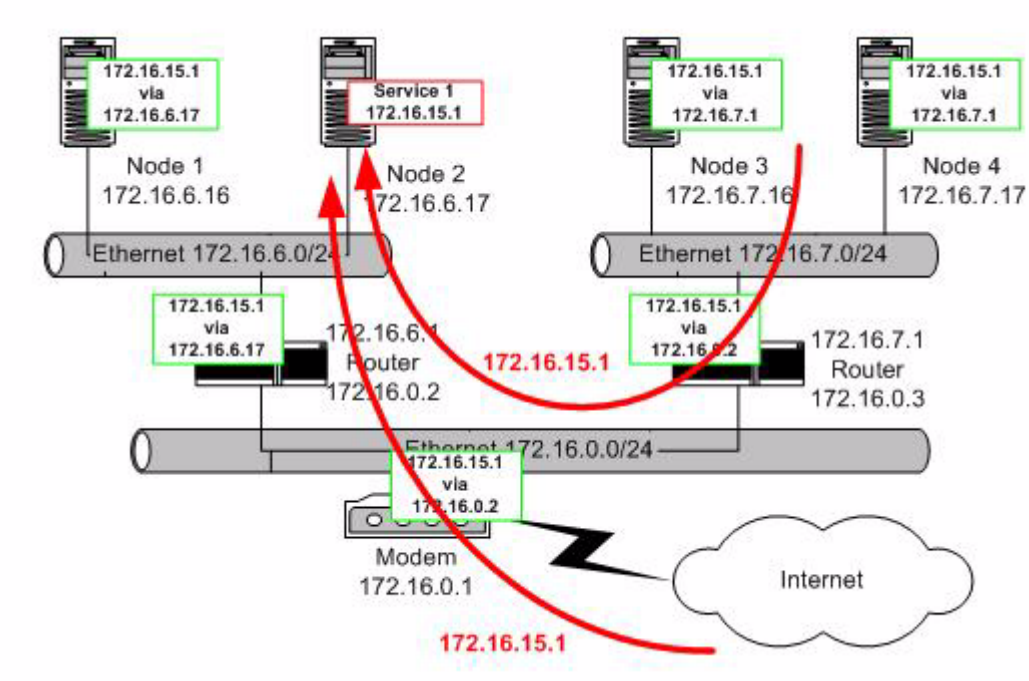
Wenn der Dienst jetzt gewollt oder ungewollt den Cluster Node 1 verlassen muss, so wird er auf einem anderen Cluster-Node neu gestartet.

Im folgenden Beispiel ist dies Cluster-Node 2:



Figur 12: Dienst auf Node 2 des Clusters

Wie wir auf der folgenden, den Datenfluss zur Dienst IP-Adresse hervorhebenden, Figur sehen, ist auch jetzt das Routing optimal:

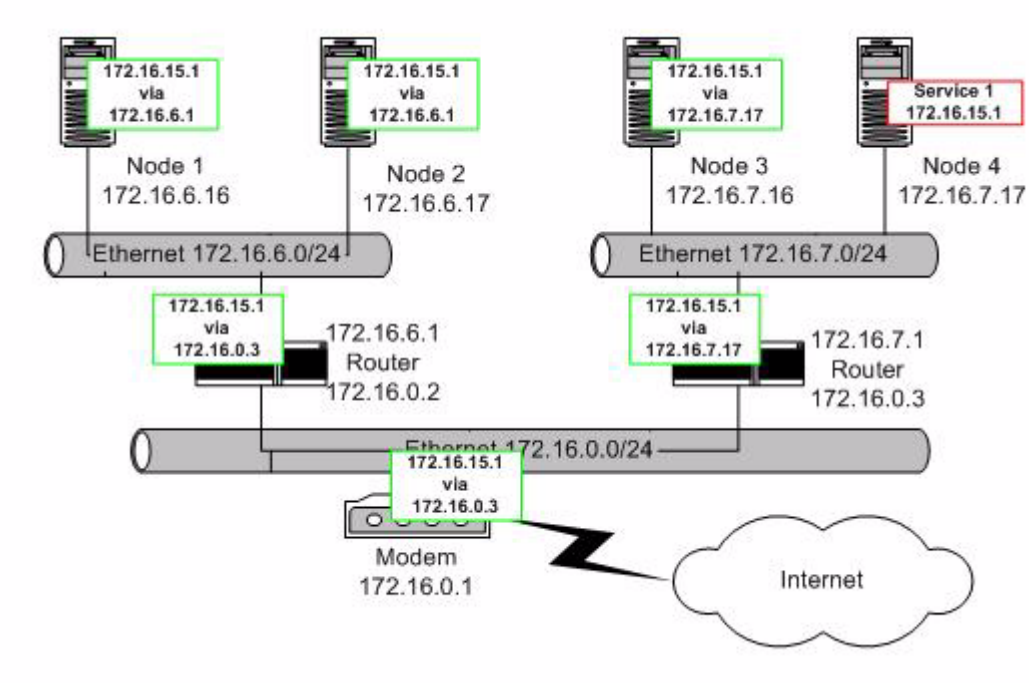


Figur 13: Datenfluss bei Dienst auf Node 2 des Clusters

Dieses Fail-Over hätte allerdings eine der klassischen in Kapitel 2 - “Transparenz” auf Seite 9 besprochenen Methoden effizienter und mit kleiner Fail-Over Zeit lösen können.

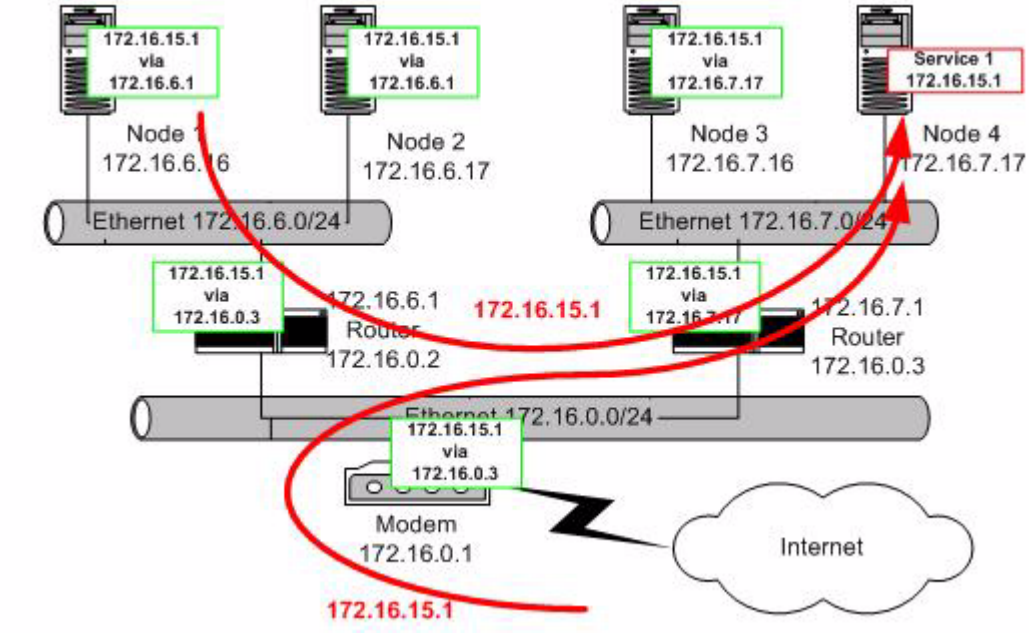
Wir wollen im letzten Beispiel ein Fail-Over zeigen, das die klassischen Methoden nicht meistern können, und zwar das Fail-Over auf dem Cluster-Node 4. Dieser Cluster-Node ist nicht auf dem selben Subnetz wie die Nodes 1 und 2.

Wie wir in der folgenden Figur sehen, hat die von uns entworfene Fail-Over Methode keine Probleme ein Fail-Over auf einem anderen Subnetz, in diesem Falle 172.16.7.0/24 auszuführen:



Figur 14: Dienst auf Node 4 des Clusters

Auch in diesem Subnetz-übergreifenden Fall wird sich das Routing nach einer kurzen Konvergenzzeit wieder optimal darstellen:



Figur 15: Datenfluss bei Dienst auf Node 4 des Clusters

5. 5. 3. Bemerkungen

Wir möchten einige Bemerkungen zu dem im obigen Text vorgestellten Fail-Over Verfahren machen.

Da die neue Fail-Over Methode auf Routing Änderungen basiert, ist die Fail-Over Zeit abhängig vom Abstand des Clients zum Server, wie auch abhängig von der Konvergenzzeit des verwendeten Link State Routing Protokolles. Ein Routing Protokoll wie OSPF, das im vorliegenden Cluster-Server Design verwendet wird, hat eine sehr kurze Konvergenzzeit, was sich vorteilhaft für die Fail-Over Zeiten des Cluster-Servers bemerkbar macht. Kapitel 7 - “Fail-Over Messungen: Minimum/Maximum/Mittel” auf Seite 67 wird einige Messresultate zu den Fail-Over Zeiten der neuen Methode darlegen.

Des Weiteren soll bemerkt werden, dass die vorgestellte Methode eine Route und somit ein Routing Table Eintrag pro konfiguriertem Service eines Cluster-Servers verwendet. Bei grossen Clustern und vielen Services kann dies sehr viele Host-Routen im Netzwerk ergeben.

Um das Netzwerk nicht mit allzu grossen Routing Tabellen zu belasten, sollte das verwendete Routing Protokoll, wie zum Beispiel OSPF, Routen Aggregation unterstützten.

Dies ermöglicht im Falle von OSPF, durch Aufteilen der Routing Domain in Areas, und durch Sumarization der Routen in einem OSPF Summary Record, dass die nicht zum Cluster gehörenden Areas nur eine einzige Netz-Route zu sehen kriegen, da die vielen Host-Routen im Summary Record durch eine Netzwerk-Route ersetzt werden!

5. 6. Service Checking

Laufende Services müssen in Betrieb geprüft werden. Das folgende Kapitel bespricht diese Prüfung, und das darauf folgende Kapitel bespricht was geschieht, wenn ein solcher Services diesen Test nicht besteht.

5. 6. 1. Periodischer Service Check

Um einen einwandfreien Betrieb des Services garantieren zu können, muss ein Cluster-Node die auf ihm laufenden Services kontrollieren. Da dies für jeden Service unterschiedlich zu geschehen hat, muss dieser Prozess auch gekapselt werden, wie schon der Start- und Stop-Prozess eines Services.

Unser Cluster-Server Design bietet mit der Schnittstelle zum Service dem Implementator des Services die Möglichkeit, völlig frei den Service zu prüfen und so zu garantieren, dass der Service die ihm gestellte Aufgabe zu vollster Zufriedenheit erfüllt oder andernfalls ein anderer Cluster-Node seine Funktion übernimmt.

Immer nachdem ein Service geprüft wurde, wird die Service Zustandsinformation im Netzwerk erfrischt und mit einem neuen Service Check Stamp, einem Prüfstempel der letzten Prüfzeit versehen. Auf diese Art und Weise ist es einem anderen Cluster-Node möglich zu verfolgen, wann der letzte Service Check erfolgt ist.

5. 6. 2. Service Failure States

Falls ein Service den Check nicht besteht so bietet unser Design dem Service die Möglichkeit in einen aktiven Zustand **active/failed** zu fallen.

Um in diesem Zustand einen Rückschluss darauf zu geben, wieso ein Service in den Fehlerzustand geraten ist, besitzt der Service Zustand, der ins Netz geflutet wird,auch einen Fehler-Code, der verwendet wird, um den anderen Cluster-Nodes mitzuteilen, was schief gegangen ist.

Jede Änderung dieses Fehler-Codes wird natürlich auch wieder ins Netzwerk geflutet.

Es gibt eine Vielzahl solcher Fehler-Codes und es würde den Umfang dieses Werkes sprengen, sie hier alle wiederzugeben. Interessierte Leser können in den Source die im Kapitel 10 - “SRR Sourcen” auf Seite 89 erwähnt werden, und deren URL dort auch angegeben sind, die verschiedenen Fehler-Codes betrachten.

6. Implementation: Service Routing Redundanz

Wir wollen den in Kapitel 5 - “Design: Service Routing Redundanz” entworfenen Cluster-Server mit Hilfe des OSPF Routing Protokolls implementieren.

Dazu werden wir im folgenden Kapitel 6 - “Das OSPF Routing Protokoll” zuerst kurz auf das OSPF Protokoll eingehen, um dann im Kapitel 6 - “OSPF Erweiterung: Opaque Link State Announcements” die Erweiterungen von OSPF zu besprechen.

Das darauf folgende Kapitel 6 - “Zebra Routing Daemon” auf Seite 49 geht kurz auf den Routing Daemon Zebra ein. Zebra ist ein moderner Routing Daemon, der die Routing Protokolle RIP, OSPF und ihre IPv6 Brüder beherrscht.

Zebra ist sehr modular aufgebaut und hat durch seine Modularität und durch seine modernen Features wie Opaque-LSA ermöglicht ein OSPFAPI Framework zu entwickeln, das es externen Applikationen erlaubt, die Opaque-LSA Features des OSPF Daemons auch zu nutzen. Das Kapitel 6 - “OSPFAPI Module” auf Seite 51 geht genauer auf die OSPFAPI Schnittstelle ein und beschreibt, wie wir dieses API verwenden, um die Opaque-LSA Features des OSPF Routing Daemons vom SRRD Unix Daemon aus nutzen können, um damit die Service Zustände des Cluster-Servers im Netzwerk zu verbreiten.

6. 1. Das OSPF Routing Protokoll

Das OSPF Routing Protokoll [4] ist ein sehr erfolgreiches Link State Routing Protokoll. Es ist ein klassisches *Interior Gateway Protokoll*[1] und wird seit langem erfolgreich in kleineren, mittleren und auch grossen Netzen eingesetzt.

OSPF teilt die Routing Domain in sogenannte *Areas* auf, die alle mit einer besonderen Area, der *Backbone Area*, oder auch Area 0, verbunden sind. Areas sind in sich geschlossene Bereiche der Routing Area und Inter-Area Datenverkehr wird im Normalfall über die Backbone Area geroutet. Gleichzeitig reduzieren die Areas die Grösse der Routing Tabelle, da Details über eine Area einer anderen verschlossen bleiben, da die Routing Informationen an der Grenze der Areas auch noch Sumarized, also aggregiert werden können.

Durch die Randbedingung, dass jede Area an die Backbone Area grenzen muss, ergibt sich, dass innerhalb der Backbone Area, also zwischen den Areas mittels eines Distance Vector Algorithmus gearbeitet werden kann, der ja erfahrungsgemäss mit sternförmigen Netzwerk-Topologien keinerlei Probleme hat.

OSPF abstrahiert die vorhandene Netzwerk-Topologie und verpackt diese Informationen in sogenannte Link State Announcement Pakete, die dann ins Netz geflutet werden.

Jeder OSPF Router, der solche LSAs hört, verschickt sie wieder an alle seine Nachbarn. Auf diese Art und Weise wird das Netzwerk geflutet. Natürlich wird dies in intelligenter Form ge-

1. Ein Interior Gateway Protokoll (IGP) ist das Gegenteil eines Exterior Gateway Protokolls wie beispielsweise EGP oder das Border Gateway Protokoll (BGP). IGP Protokolle, wie RIP oder OSPF, werden für das Routing innerhalb eines autonomen Systemes verwendet, EGP Protokolle hingegen werden als Routing Protokolle zwischen den autonomen Systemen angewandt.

macht, um das Ganze nicht ineffizient zu machen. Das RFC [4] gibt genauer über den Flooding-Algorithmus von OSPF Auskunft.

OSPF besitzt auch ein ausgeklügeltes Verfahren, um seine Nachbar Router zu erkennen und zu beobachten, das Neighbor Discovery genannt wird. Ein sogenanntes Hello-Protokoll sorgt dafür, dass ein OSPF Router immer weiss, wer seine Nachbarn sind, und auch erkennt, wenn ein solcher Nachbar nicht mehr verfügbar ist.

Dank des Hello-Protokolls bemerkt ein OSPF Router nach spätestens 30 Sekunden, wenn ein anderer Router nicht mehr zur Verfügung steht.

Des Weiteren ist das OSPF Protokoll darauf ausgelegt, dass OSPF um neue Link State Announcement (LSA) Typen erweitert werden kann. Dies ist auch geschehen, denn OSPF hat im Orginal nur die LSA Typen 1 bis 5 gekannt, die Weiteren sind erst später dazugekommen.

Die 11 verschienden LSA Typen sind:

Typ	Name	Verwendung
1	**Router-LSA**	Router Representation
2	**Network-LSA**	Netzwerk Segement Representation
3	**Summary-LSA**	Implementieren hierarchisches Routing zwischen den verschiedenen Areas der Routing Domain
4	**ASBR-Summary-LSA**	
5	**AS-External-LSA**	
6	**Group-Member-LSA**	Werden für MOSPF[a] verwendet
7	**AS-NSSA-LSA**	Not So Stubby Areas, eine OSPF Erweiterung
8	**External-Attributes-LSA**	Verwendet um BGP Pfad Informationen durch OSPF Domains zu transportieren
9	**Opaque-Link-LSA**	Werden für Opaque Link State Announcment verwendet. Das heisst mit diesen LSA können Opaque Daten versendet werden
10	**Opaque-Area-LSA**	
11	**Opaque-AS-LSA**	

Tabelle G: OSPF LSA Typen mit Erklärung

a. Multicast OSPF

Wie aus der Tabelle ersichtlich, sind die Typen 9, 10 und 11 für Opaque Link State Announcements reserviert.

Im folgden Kapitel 6 - “OSPF Erweiterung: Opaque Link State Announcements” werden wir diese LSAs genauer betrachten, und wir werden sehen, dass die Opaque LSAs genau das Hilfsmittel sind, das wird brauchen, um unsere Service Zustände, die wir in Kapitel 5 - “Service Selektion” besprochen haben, im Netzwerk zu verbreiten.

6. 2. OSPF Erweiterung: Opaque Link State Announcements

Das OSPF Protokoll lässt sich erweitern. Es ist vorgesehen, dass neue LSA Type eingeführt werden können, und eine der Erweiterungen, die im Laufe der Zeit eingebaut wurden, sind *Opaque Link State Announcements*.

Sie ermöglichen einem OSPF Modul, wie beispielsweise MPLS-TE[1], mit Hilfe des OSPF Protokolls eigene Daten, eben “Opaque Daten”, im Netz zu verteilen und zu empfangen.
Den OSPF Daemons ist es egal, was in den opaquen LSA Paketen ist, und die opaquen Daten werden einfach als opaquer Datenblock am Ende des klassischen LSA Headers angefügt.

Natürlich können opaque Daten nur von dem selben Modul wieder verstanden werden, das sie auch versendet hat!

Damit verschiedene Module verschiedene opaque Daten versenden können, haben opaque LSA eine sehr speziell aufgebaute LSA Id, von der der eine Teil als Opaque-Typ und der zweite Teil als Opaque-Id verwendet wird, wie die folgende Tabelle zeigt.

Bit 31 - 24	Bit 23 - 16	Bit 15 - 8	Bit 7 - 0
Opaque Type	Opaque ID		

Tabelle H: Aufbau der LSA ID eines Opaque LSA Pakets

Module können sich für gewisse Opaque-Typen registrieren und werden dann vom OSPF Daemon aufgerufen, wenn solche opaquen LSAs vom OSPF Daemon gesehen werden.

Ein Modul kann so sichergehen, vom OSPF Daemon immer dann benachrichtig zu werden, wenn die opaquen Daten in der globalen LSA Datenbank mit neuen Daten erfrischt oder wenn sie gelöscht werden.

Durch Einfügen und Löschen von eigenen opaquen LSAs vom entsprechenden Typ, kann das Modul dann eigene opaque Daten im Netzwerk verbreiten.

Wir wollen uns in der Implementation des Service Routing Redundancy Daemons diese Möglichkei, opaque Daten im Netzwerk zu verteilen, zu Nutze machen und sie dazu verwenden, die Service Zustände der einzelnen Services, wie auch der einzelnen SRRD Unix Daemone im Netz zu verbreiten und unter den einzelnen SRRD’s auszutauschen.

6. 3. Zebra Routing Daemon

Der Zebra Routing Daemon ist sehr modular aufgebaut. Seine verschiedenen Protokoll-Module sind als separate Unix Prozesse implementiert und kommunizieren über TCP Verbindungen untereinander, um in Kooperation den Routing Dienst zu erbringen.

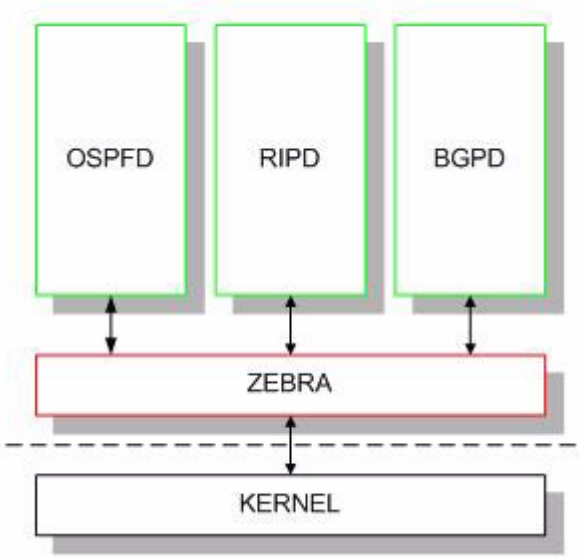
Das Zebra Routing Daemon Paket besteht aus dem **zebra** Daemon, der für alle statischen Routen zuständig ist, und der mit dem Kernel kommuniziert, um die entsprechenden Routen in den Routing Tabellen des Kernels zu installieren.

1. Multi Protokoll Label Switching mit Traffic Engeneering

Alle anderen Protokoll-Module kommunizieren mit dem Zebra Daemon falls sie eine Route installieren wollen, oder wenn sie eine Route dynamisch entfernen wollen.

Die weiteren Protokoll-Module sind der **ripd** und der **bgpd.** Diese beiden Daemone implementieren RIP und BGP. Ein weiterer Daemon mit dem Namen **ospfd** implementiert OSPF.

Die folgende Figur zeigt den modularen Aufbau des Zebra Routing Daemon Paketes:



Figur 16: Modularer Aufbau von Zebra

Jedes Protokoll-Modul wie ripd, bgpd und ospfd öffnet bei entsprechender Konfiguration einen TCP Port, über den das entsprechende VTY[1] des entsprechenden Protokoll-Daemons angesprochen werden kann. Über diesen Port kann mit einer TCP Verbindung, wie beispielsweise Telnet, eine Verbindung zum Kommandozeilen Interpreter des entsprechenden Protokoll-Moduls erstellt werden. Dort kann mit einfachen Kommandos der Zustand des Protokoll-Moduls, wie auch die Konfiguration eingesehen und verändert werden.

Wir werden die Möglichkeiten,die uns die einzelnen VTYs bieten, in unserem Service Routing Redundancy Daemon nutzen, um darüber Interface Aliases zu installieren und deinstallieren, wie auch den OSPF Daemon zu veranlassen, eine Route zum entprechenden Interface zu erstellen oder diese Route wieder zu löschen, wie es das Design des Cluster-Servers in Kapitel 5 - “Service Routing mit einem Link State Protokoll” auf Seite 38 bespricht.

1. Virtual TTY - Eine kleines Kommandozeilen Shell, mit dem die internen Zustände des Protokoll-Moduls, wie auch seine Konfiguration eingesehen und verändert werden können.

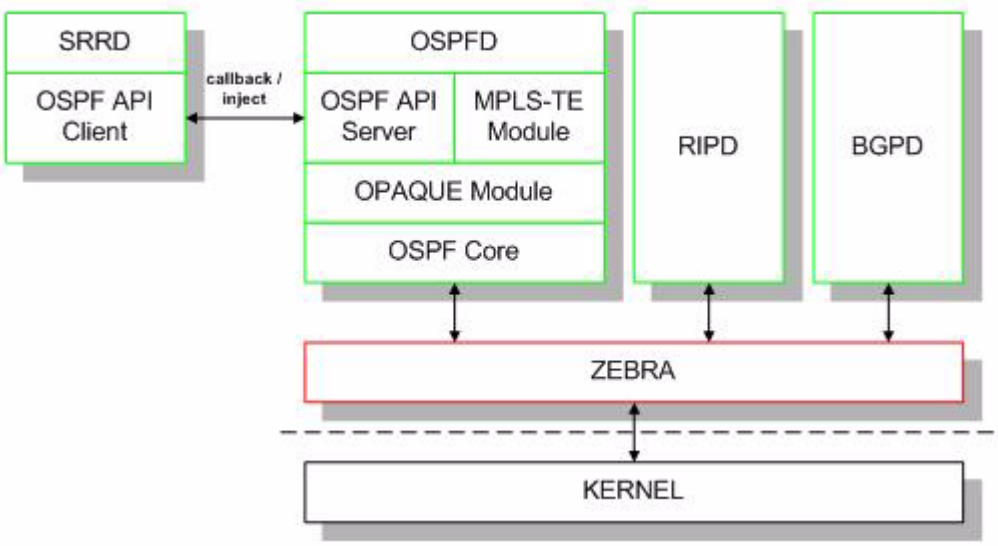
6. 4. OSPFAPI Module

Das OSPFAPI Modul nützt den modularen Aufbau des OSPF Daemons aus. Der OSPF Daemon ist speziell so aufgebaut, dass es möglich ist, durch neue Module seine Funktionalität zu erweitern und so Module einzufügen, die die speziellen Opaque LSA Features des OSPF Daemons nutzen.

Um es auch externen Applikation zu ermöglichen, die Opaque LSA Features des OSPF Daemons zu nutzen, klinkt sich das OSPFAPI als spezielles OSPF Modul in den OSPF Daemon ein und ermöglicht es Client Applikationen, durch verwenden einer kleinen Client-Bibliothek, Verbindung zum OSPFAPI Modul im OSPF Daemon aufzunehmen. Dadurch wird es einer externen Applikation ermöglicht, sich für spezielle Opaque-Typen zu registrieren und dann opaque Daten zu emitieren und zu empfangen.

Die OSPFAPI Schnittstelle ermöglicht es der Client-Applikation natürlich auch, sich mit der LSA Datenbank des ensprechenden OSPF Daemons zu synchronisieren und somit eine vollständige Kopie aller im Moment bekannten LSA im Netz zu bekommen.

In der folgenden Figur wird der modulare Aufbau des OSPFAPI und des OSPF Daemons genauer illustriert:



Figur 17: Modularer Aufbau des OSPFAPI

6. 4. 1. Dynamische Registrierung von neuen Opaque Types

Eine OSPFAPI Client Applikation muss sich beim OSPF Daemon für einen speziellen Opaque-Typ registrieren, bevor sie opaque LSAs von diesem Type emitieren kann.

Hat sich eine Applikation für einen bestimmten Typ registiert, so wird sie kurze Zeit später einen Ready-Callback für den ensprechenden Typ bekommen. Ist dies geschehen, kann die Applikation neue LSA der entsprechenden Typen emitieren und auch wieder löschen. Ändert sich eine Information in der globalen LSA Datenbank, so wird eine Update- oder eine Delete-Callback aufgerufen, wie dies im folgenden Kapitel 6 - “Update / Delete Callback” auf Seite 52 beschieben wird.

6. 4. 2. Update / Delete Callback

Wenn irgend jemand die LSA Informationen in der globalen LSA Datenbank auffrischt, verändert oder löscht, informiert das OSPFAPI die Client-Applikation über einen Callback.

Werden Daten erfrischt oder neu in die globale Datenbank eingefügt, so wird in einer OSPFAPI Client Applikation ein *Update-Callback* aufgerufen, der es der Client Applikation ermöglicht, sich über die LSA Daten auf dem Laufenden zu halten.

Werden Daten in der globalen LSA Datenbank gelöscht, so erfährt eine OSPFAPI Client Applikation dies durch einen *Delete-Callback*.

6. 4. 3. Callback für Zustandsänderungen

Das OSPFAPI Framework ermöglicht es der Client Applikation, sich auch über Zustandsänderungen der Interface-Zustandsmaschine und der Netzwerk-Zustandsmaschine des OSPF Daemons auf dem Laufenden zu halten.

Dazu kann die Client Applikation einen ISM-Change-Callback und einen NSM-Change-Callback definieren, die dann vom OSPF Daemon aufgerufen werden, wenn sich der Zustand der entsprechenden Zustandsmaschine ändert.

6. 5. Der Service Redundancy Daemon

Der Service Routing Redundancy Daemon implementiert das ganze in Kapitel 5 - “Design: Service Routing Redundanz” auf Seite 27 besprochene Design des Cluster-Servers als einzelner Unix Daemon.

Der SRRD Daemon besteht aus vielen einzelnen Modulen, von denen jedes eine spezielle Aufgabe besitzt. Kapitel 6 - “Übersicht der Implementation” auf Seite 53 wird einen Überblick über den Aufbau des Service Redundancy Daemons bieten und wird auf den detaillierten Aufbau des Daemons eingehen.

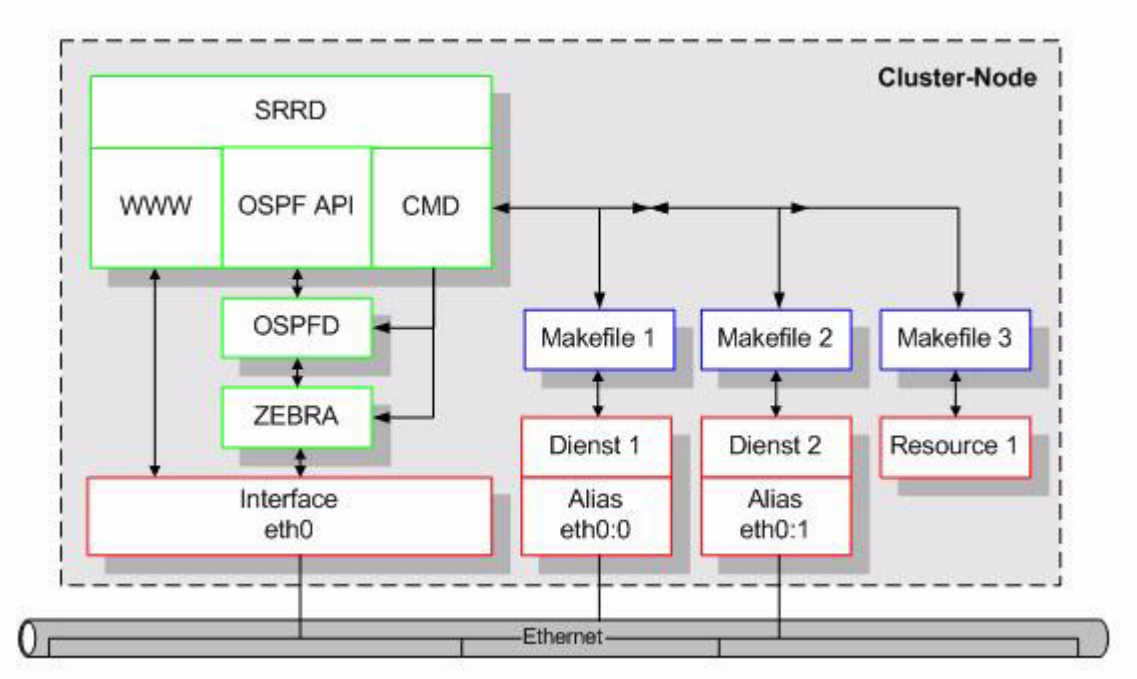
Da der Service Routing Redundancy Daemon ein sehr grosses Software Projekt war und ist, er besteht zum jetztigen Zeitpunkt aus 17’000 C Code Zeilen, können wir in diesem Text nur einige spezielle Teile seiner Implementation besprechen.

Sollte der geneigte Leser weitere Informationen oder genauere Implementationsdetails wünschen, so kann der Source-Code des ganzen Projektes an dem in Kapitel 10 - “SRR Sourcen” auf Seite 89 beschriebenen Ort bezogen werden.

6. 5. 1. Übersicht der Implementation

Im Folgenden wollen wir einen Überblick über die Implementation des Service Routing Redundancy Unix Daemons gewinnen. Er implementiert den gesamten Cluster-Server, der im Kapitel 5 - “Design: Service Routing Redundanz” entworfen wurde.

Der SRRD wurde sehr modular aufgebaut. Ganz grob gesprochen besteht der Daemon aus drei verschieden Teilen, dem Webserver Modul, das das Benutzerinterface implementiert, dem OSPFAPI Modul, das die Kommunikation mit der Link State Datenbank des OSPF Daemons kontrolliert und einem Kommando Modul, das sowohl den Routing Daemons Kommandos gibt, wie auch mit Hilfe der Shell und dem Unix Kommando “make” Service Access Points[1] (SAPs) für Services implementiert, wie die folgende Figur genauer zeigt:



Figur 18: Modularer Überblick über einen SRRD Cluster-Node

Auf dem oben gezeigten Cluster-Node sind drei Unix Daemone gestartet. Das Zebra Routing Paket bestehend aus **zebra** und **ospfd** Daemonen implementieren das dynamische Routing.

Der **srrd** Daemon verwendet sowohl den **ospfd** wie auch den **zebra** Daemon um einen Cluster-Node zu implementieren und so Teil eines aus mehreren srrd Daemonen bestehenden Cluster-Server zu werden.

Die Schnittstelle zu den Services, bestehend aus Ressourcen und Diensten, wird über eine Makefile API implementiert. Vorherbestimmte Makefile-Targets, die Service Primitiven, werden vom srrd Daemon zum richtigen Zeitpunkt aufgerufen, um den Service zu starten oder zu stoppen. Das Kapitel 6 - “Service Interface zur Shell:” beschreibt die verschiedenen Service Primitiven die den SAP zu den einzelen Diensten darstellen.

1. Der Ort, an dem die Service Primitiven des Services, aufgerufen werden können

Wie in der obigen Figur 18: “Modularer Überblick über einen SRRD Cluster-Node” auf Seite 53 zu sehen ist, kann ein **srrd** Daemon viele verschiedene Services kontrollieren und mit Hilfe verschiedener Makefiles verschiedene Services starten und stoppen. Die Makefile geben dem Daemon auch die Möglichkeit den Service zu checken und seinen Status abzufragen.

Diese Makefile-Targets entsprechen den als Service Primitiven bezeichneten SAPs der Services. Das Kapitel 3 - “Service Primitiven” auf Seite 19 hat die Service Primitiven eingeführt und in Kapitel 6 - “Service Interface zur Shell:” auf Seite 60 werden die einzelen Service Primitiven besprochen. Dasselbe Kapitel geht auch auf die Rückgabewerte derselben Service Primitiven ein und bespricht auch, welche Möglichkeiten für eine solche Service Primitive in einem Makefile bestehen, um den Service Rountig Redundancy Daemon beeinflussen zu können.

Der **srrd** Daemon implementiert einen eigenen Webserver über den der Cluster-Node kontrolliert, wie auch der Status des Clusters eingesehen und manipuliert werden kann. Der Webserver unterstützt SSL und kann mit einer Liste von Certificate Authorities (CAs) und Zertifikaten so konfiguriert werden, dass nur eine geschlossene Benutzergruppe den Webserver einsehen kann.

Man beachte: Die Kommunikation unter der **srrd** Daemonen, wie auch die Kommunikaton mit den Routing Daemonen, ist ungesichert! Lediglich die Benutzerschnittstelle über dem Webserver ist mit SSL gesichert. Dies wurde im Hinblick auf die Einfachheit, mit der ein Eindringling Unheil stiften könnte, implementiert, so dass doch Zugang zu einem der einzelnen Systeme erlangt werden müsste, um das System über das LAN zu kompromitieren.

Wie in Figur 18: “Modularer Überblick über einen SRRD Cluster-Node” auf Seite 53 auch zu sehen ist, wird für jeden Dienst ein Interface Alias erstellt, solange der Dienst auf dem System läuft. Eine Ressource ohne IP-Adresse braucht kein Interface Alias, wie auch zu sehen ist. Die Interface Alias werden vom Kommando Modul des **srrd** Daemons mit Hilfe des Routing Daemons **zebra** und seines VTYs[1] erstellt und gelöscht.

Dies garantiert, dass beide Routing Daemons die Interface Alias sehen und kennen. Um dann dafür zu sorgen, dass auch Service Routen im Netzwerk installiert werden, kann das Kommando Modul des **srrd** Daemons über das VTY des **ospfd** Daemons den OSPF Routing Daemon dazu veranlassen, das Interface Alias ins Netzwerk zu propagieren. Andere OSPF Routing Daemone im Netzwerk installieren dann die Host Route zum Interface Alias.

Mehrere solche **srrd** Daemone erkennen sich im Netzwerk und können mit Hilfe der gegenseitig ausgetauschten Service Zustandsinformationen kooperativ einen Cluster-Server bilden. Die Webserver der einzelnen **srrd** Daemone erkennen sich ebenfalls gegenseitig und bilden ein vermaschtes Netzwerk aus Universal Resource Locations (URLs), die gemeinsam ebenfalls kooperativ einen Gesamtwebserver bilden, mit dem sich der gesamte Cluster-Server beobachten, kontrollieren, konfigurieren und manipulieren lässt.

1. Virtual TTY

Die gesamte Benutzerinteraktion findet über den Webserver statt. Services werden auf dem Webserver konfiguriert. Der **srrd** Daemon sorgt für die persistente Speicherung der lokalen Konfigurationsinformation eines Cluster-Nodes. Beim Start eines srrd Daemons wird die letzte Konfiguration wieder geladen und auch wieder zu den anderen srrd Daemonen propagiert. So existiert schlussendlich eine auf alle Cluster-Nodes verteilte Konfiguration die in ihrer Gesamtheit die Cluster-Server Konfiguration ausmacht. Der Webserver verweist den Benutzer jeweils auf denjenigen Webserver, dessen srrd Daemon konfiguriert werden soll.

Der Benutzer hat dann die Möglichkeit, die Services zu aktivieren, zu starten und zu manipulieren. Es existieren vielfältige Möglichkeiten einen Service zu beeinflussen.

Des Weiteren wurde eine Visualisierung der Link State Datenbank implementiert, die es ermöglicht den Zustand der Datenbank, wie auch den Inhalt der einzelnen Link State Announcment einzusehen.

Ein Kommando Interface zu den beiden Routing Daemonen wurde auch implementiert und ermöglicht es, sowohl die Routing Tabellen, wie auch jedes beliebige bekannte Kommando des Benuzterinterfaces der Routing Daemonen zu verwenden. So kann ein Benuzter die vollständige Übersicht über den Zustand der Services, des Service Routings, wie auch der Link State Datenbank gewinnen.

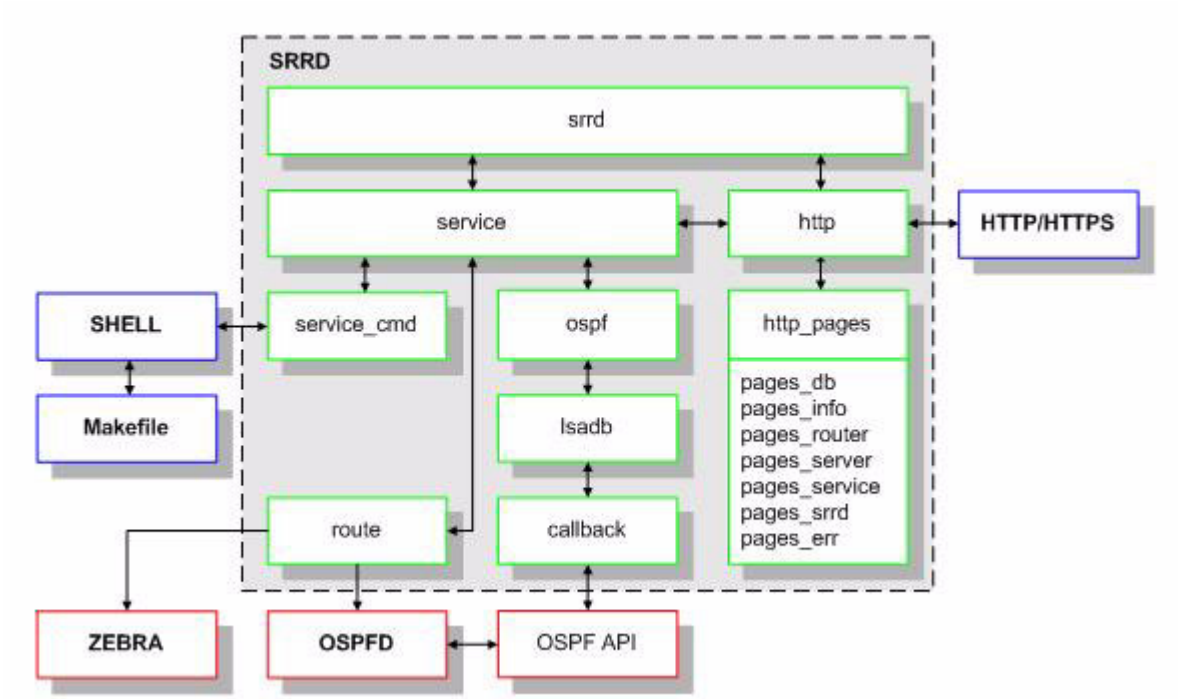
Die Benutzerschnittstelle zum Status der Services ermöglicht es, jeden Service, ob lokal auf diesem Webserver oder auf einem anderen, nicht lokalen Webserver, zu kontrollieren, seinen Zustand zu sehen, wie auch den laufenden Prozess selbst zu prüfen. Sowohl die Service Gruppen, wie auch kritische Services sind zu sehen und ihr Zustand, wie auch der Gruppen-Gesamtzustand, wird visualisiert. Versagende Services oder Service Gruppen werden hervorgehoben und es ist dem Benutzer möglich einzuschreiten.

Das folgende Kapitel geht auf den detaillierten Aufbau des Service Routing Redundancy Daemons ein, und beschreibt dessen modularen Aufbau. Kapitel 6 - “Integrierter Webserver” geht genauer auf die Möglichkeiten des Daemon Webservers ein.

6. 5. 2. Detallierter Aufbau der Implementation

Im Folgenden wollen wir einen einzelen **srrd** Daemon eines einzelnen Cluster-Nodes betrachten. Wir wollen seinen modularen Aufbau besprechen und kurz auf jedes einzelne Modul eingehen und beschreiben, was die Funktion und Aufgabe des entsprechenden Moduls ist.

Wie in der folgendem Figur zu sehen ist, besteht der eigentliche Unix Daemon aus einem Hauptmodul mit dem Namen **srrd**. Es enthält die main() Funktion des ganzen Daemons und seine Aufgabe ist die Initialisierung der einzelnen Teilmodule, wie auch die Hauptschleife des Daemons zu implementieren.



Figur 19: Modulare Details eines SRRD Unix Daemons

Das Hauptmodul wiederum verwendet zwei weitere Module mit den Namen **service** und **http** um jeweils Webserver und Cluster-Node zu implementieren. Natürlich verwenden die beiden Module sich auch gegenseitig um Aktionen, die der Benutzer auf dem Webserver vornimmt, gemeinsam zu erledigen.

Das **http** Modul verwendet das **http_pages** Modul und die vielen Module mit dem Prefix **pages_** um in ihrer Gesamtheit den Webserver des Daemons zu bilden. **http_pages** enthält viele Hilfsfunktionen, um die Webserver Seiten zu generieren und manipulieren. Eine im Modul **http** definierte Liste von erlaubten URLs, und die sie generierenden Funktionen, wird mittels Funktions-Pointern implementiert. Über sie kann das **http** Modul die richtige Generator-Funktion der ensprechenden URL anspringen.

In den einzelnen Modulen mit dem Prefix **pages_** sind dann die jeweiligen Webserver Seiten wie folgt implementiert:

- **pages_db** implementiert die Link State Datenbank Seite
- **pages_info** implementiert die Author und Programm Informationsseiten
- **pages_router** implemenentiert das Router Interface

- **pages_server** implementiert den Server Shutdown
- **pages_service** implementiert die Service Seite
- **pages_srrd** implementiert das root des Webservers
- **pages_err** implementiert die verschiedenen Fehler Webserver Seiten

Das **http** Modul implementiert sowohl HTTP wie auch HTTPS. Das Module ist fähig, das HTTP 1.0 Protokolle zu sprechen. Es ist keine vollständige HTTP 1.0 Implementation, sondern implemenentiert nur den Teil des Protokolles, der auch von dem Modul verwendet wird. Das HTTPS Protokoll wird mit Hilfe der Open Source Bibliothek **openssl** implementiert und ermöglich den Einsatz von Secure Socket Layer (SSL) zur sicheren authorisierten und authentisierten Kommunikation mit dem Cluster Server Webserver Benutzer.

Der eigentliche Cluster-Node wird durch das Modul **service** implementiert. Die vollständige Service Zustandsmaschine ist darin enthalten. Um über die durch das OSPF Protokoll ausgetauschten Service Zustandsinformationen auf dem Laufenden zu sein, verwendet das **service** Module das **ospf** Modul, das die Kommunikation mit der LSA Datenbank des Service Redundancy Daemons abstahiert. Das **ospf** Modul verwendet wiederum das **lsadb** Modul, das die LSA Datenbank des Service Routing Redundancy Daemons implementiert. Die vom OSPFAPI benötigten Callback-Funktionen sind im Modul **callback** implementiert. Diese Funktionen werden vom OSPFAPI aufgerufen, wenn Zustandsänderungen in der globalen Link State Datenbank geschehen.

Das **service** Modul verwendet das **route** Modul, wenn Interface Aliases und Interface Alias Routen erstellt oder gelöscht werden müssen. Das **route** Module öffnet und pflegt zwei TCP Verbindungen zu den beiden VTYs der beiden Routing Daemone zebra und ospfd. Diese Verbindungen sind persistent über die ganze Laufzeit des SRRD. Sie werden zur Initialisationszeit des Moduls geöffnet und werden erst bei Beendigung des Unix Daemons wieder geschlosssen.

Als Letztes verwendet das **service** Modul das **service_cmd** Modul, um die Service Primitiven der einzelnen konfigurierten Services aufzurufen und abzufragen. Das **service_cmd** Modul verwendet mehrere Pipes, wie auch das Unix Kommando fork(), um in einem Child-Prozess das Makefile auszuführen und seine Ausgaben wieder zu parsen. Über diese Schnittstelle werden die einzelnen Services gestartet, gestoppt und geprüft.

Im folgenden Kapitel wollen wir kurz auf die im Service Routing Redundancy Daemon implementierte Zustandsmaschine eingehen, um dann in Kapitel 6 - “Aufbau der neuen LSA's” auf Seite 58 die neuen opaquen Link State Announcments, die der Service Routing Redundancy Daemon einführt, genauer zu betrachten.

6. 5. 3. Zustandsmaschine: LSA Ereignis

Das Service Modul des Service Redundancy Daemons implementiert die gesamte in Kapitel 5 - “Service Zustände” auf Seite 33 entworfene ereignisgesteuerte Zustandsmaschine.

Im Kontext unseres Cluster-Servers entsprechen Ereignisse der Änderung eines unserer opaquen Cluster LSAs in der globalen Datenbank. Ein OSPFAPI Client sieht seine eigenen Änderungen in der globalen LSA Datenbank selber ebenfalls. Dies ermöglicht es, einen echt ereignisgesteuerten Zustandsautomaten zu implementieren, der lokal, rein aus seiner Sicht der globalen Zustandsänderungen, anhand der als Ereignisse gesehenen LSA Änderungen der globalen Datenbank, reagiert.

6. 5. 4. Aufbau der neuen LSA's

Der Service Routing Redundancy Daemon verwendet die LSA Typen 9, 10 und 11, also Opaque-Link, Opaque-Area und Opaque-AS LSAs, um die Service Zustandsinformationen im Netzwerk zu verbreiten.

Der Service Routing Redundancy Daemon verwendet opaquen LSAs mit dem Opaque-Typ 222. Das bedeutet, dass die emitierten LSAs alle eine LSA-Id der Form 222.X.Y.Z haben.

Es werden zwei verschiedene Typen von opaquen LSAs ins Netzwerk propagiert. Wir werden diese beiden Typen in den beiden folgenden Kapiteln im Detail besprechen.

6. 5. 4. 1. Service Announcment LSA

Jeder auf dem Cluster-Node konfigurierte Service, der in einem aktiven Zustand ist, wie in Kapitel 5 - “Aktive Zustände” auf Seite 36 beschrieben, wird im Netzwerk als Service LSA verbreitet.

Ein Service LSA ist ein LSA vom Typ 9, 10 oder 11, je nachdem welcher Cluster-Radius bei der Service Konfiguration verwendet wurde. Diese drei Typen entsprechen den drei opaquen LSA Typen die OSPF in seiner Erweiterung definiert. Der Service LSA besteht aus dem normalen LSA Header, gefolgt von den opaquen Daten.

Die opaquen Daten eines Service LSAs werden in der Implementation durch eine C Struktur repräsentiert, die folgendermassen aussieht:

```
struct service_opaque_s
{
  char name[MAX_SERVICENAME];
  char group[MAX_SERVICENAME];
  char dependency[MAX_SERVICENAME];
  u_char pref;
  u_char state;
  u_char flags;
  u_char failure;
  u_char group_critical;
  u_int16_t check_intervall;
  time_t check_stamp;
  struct in_addr addr[MAX_SERVICEADDRS];
};
```

Wie man sieht, sind darin alle nötigen Service-Konfigurationsinformationen, wie auch die Zustandsinformationen der Zustandsmaschine enthalten. Die Strukturkomponenten sollten durch ihre Namen selbsterklärend sein.

Der Primary-Key eines Services ist sein Name. Zwei Services, die auf zwei verschiedenen Cluster-Nodes mit dem selben Service-Namen konfiguriert werden, bilden zusammen einen Cluster-Server für diesen Service. So können aus mehreren Cluster-Nodes viele verschiedene Cluster-Server gebildet werden. Ein Cluster-Node kann so viele verschiedene Services gleichzeitig in unterschiedlicher Cluster-Node Gruppierung gleichzeitig betreiben.

Die Felder name, group, dependency, addr und pref werden aus der Konfiguration des entsprechenden Cluster-Nodes Services ausgelesen. Die restlichen Felder stellen den Zustand des Services auf einem Cluster-Node dar.

Alle Service LSAs werden mit einer Opaque-Id die der Service-Id entspricht versendet. Die Service-Id ist eine von 0 aufsteigende Zahl, die die einzelnen Services auf der Maschine identifiziert und eindeutig für einen Service auf einem einzelnen Cluster-Node ist.

Service LSAs haben also eine LSA-Id der Form: **222.0.0.<service id>**

6. 5. 4. 2. Daemon Announcment LSA

Der Service Routing Redundancy Daemon verwendet einen zweiten opaquen LSA Typ. Dieser zweite LSA Typ wird verwendet, damit verschiedene Service Routing Redundancy Daemons sich gegenseitig erkennen und wissen, auf welchen Adressen und Ports die entsprechenden Webserver der anderen am Cluster beteiligten Service Routing Redundancy Daemons zu erreichen sind.

Die opaquen Daten werden wieder durch eine C Struktur in der Implementation repräsentiert:

```
struct service_router_opaque_s
{
  struct in_addr addr;
  u_int16_t port;
};
```

Wie man sieht enthält die Struktur lediglich IP-Adresse und Port des Webservers eines SRRD Daemons.

Alle Daemon LSAs werden mit einer Opaque-ID von 256 versendet. Daemon LSAs haben also eine LSA-Id der Form: **222.0.1.0**

6. 5. 5. Integrierter Webserver

Der Service Routing Redundancy Daemon implementiert einen eigenen Webserver. Der Webserver ermöglicht es einem Benutzer, alle den Cluster-Server betreffenden Informationen einzusehen und den Cluster-Server fast beliebig zu beeinflussen.

Folgende Webserver Seiten werden angeboten:
- Eine Konfigurations Überblick Seite
- Eine Seite um neue Service zu definieren
- Eine Link State Datenbank Überblick Seite
- Eine Service Zustandsübersicht Seite
- Eine Kommandointerface Seite zu den Routing Daemons
- Eine Service Status Seite für jeden laufenden Service

Das Kapitel 8 - “Demonstration” auf Seite 69 zeigt den Webserver und seine Seiten ausführlich.

6. 6. Service Interface zur Shell:

Im Folgenden wollen wir das Interface zur Shell, das die einzelnen Service startet und stoppt, genauer betrachten.

Um eine grösstmögliche Abstraktion und gleichzeitig ein möglichst flexibles Interface zu verwirklichen, wurde ein etwas spezieller Ansatz gewählt.

Jeder Dienst wird durch ein Makefile gestartet, das mit dem Unix Kommando “make” interpretiert wird. Makefiles bieten die spezielle Möglichkeit sehr einfache “Targets” zu definieren, die dann mit einem einfachen “make <target>” ausgeführt werden können. Ein solches Target kann aus beliebigen Shell Kommandos zusammengesetzt und von weiteren Targets abhängig sein. Jeder Service Primitive ist einem speziellen Makefile Target mit gleichem Namen zugeordnet. Die Targets entsprechen den Service Primitiven mit gleichem Namen.

Durch diesen Aufbau ist es möglich, jeden Dienst durch eine beliebige Folge von Kommandos zu starten und auch beliebig komplizierte Service Primitiven zu implementieren, da wieder beliebige Skripte, Interpreter und Aehnliches aufgerufen werden können.

Im Folgenden besprechen wir zuerst die verschiedenen Service Primitiven in Kapitel 6 - “Service Primitiven”, um dann in Kapitel 6 - “Resultate von Service Primitiven” die Rückgabewerte der einzelnen Service Primitiven zu betrachten.

6. 6. 1. Service Primitiven

Jedes Service Makefile muss 8 verschiedene Service Primitiven unterstützen. Es muss nicht jede Service Primitive implementiert werden! Es steht einem Benuzter frei, Makefile mit Service Primitiven zu implementieren, die fix immer einen korrekten Rückgabewert zurückgeben. Es muss aber jedes Target für alle 8 Service Primitiven im Makefile vorhanden sein.

Die Service Primitiven lassen sich in 4 Klassen aufteilen:
- **Aktivierungs-Klasse:** activate, deactivate
- **Start-Klasse:** start, stop, reload, restart
- **Informations-Klasse:** join, leave
- **Debug-Klasse:** failed, status, check

Die folgende Tabelle zeigt die verschiedenen Service Primitiven und ihre Aufgabe:

Service Primitive	Zeitpunkt des Aufrufs	Aufgabe
activate	Service soll aktiviert werden	Vorbereiten für einen allfälligen Start
deactivate	Service soll deaktiviert werden	Service vollständig stoppen
start	Service soll gestartet werden	Start des Services
stop	Service soll gestoppt werden	Stop des Services

Tabelle I: Die 8 Service Primitiven

Service Primitive	Zeitpunkt des Aufrufs	Aufgabe
reload	Service soll reinitialisiert werden	Viele Dienste können mit einem Signal ihre Konfigration neu parsen oder ähnliches (Bsp: kill -HUP <pid>)
restart	Service soll gestoppt und neu gestartet werden	Service soll gestoppt und neu gestartet werden
join	Gleicher Service wurde auf anderem Cluster-Node aktiviert	Service wird über den neuen Cluster-Node Partner informiert
leave	Gleicher Service wurde auf anderem Cluster-Node deaktiviert	Service wird über das Ausscheiden eines Cluster-Node Partners zu informieren
failed	Service ist im Fehlerzustand auf diesem Cluster-Node	Alarmieren, beispielsweise durch SMS Versand oder EMail Benachrichtigung
status	Benutzerabfrage auf dem Webserver	Echter Prozesszustand zu raportieren, Speicherverbrauch, Cache Zustand oder Aehnliches
check	In regelmässigen, konfigurierbaren Intervallen	Überprüfung der vollständigen Funktionsfähigkeit des Services

Tabelle I: Die 8 Service Primitiven

Ein einfacher, normaler Dienst, wie ein Webserver oder eine Datenbank, brauchen sich nicht um die für sie nicht interessanten Service Primitiven wie activate / deactivate und join / leave zu kümmern und können sie ignorieren, indem die Targets in den entsprechenden Dienst Makefiles leer gelassen werden.

Komplizierte Services, wie zum Beispiel die von uns implementierte Netzwerk RAID-1 Disk brauchen die Service Primitiven activate und deactivate, weil sie auch in noch nicht gestartetem Zustand gewisse laufende Prozesse brauchen, damit andere Cluster-Nodes den Service bei sich starten können. Genauso ist es mit den beiden Service Primitiven join und leave, die bei einer Netzwerk RAID-1 Disk dazu verwendet werden, weitere neue Cluster-Nodes in die Replikation der Disk einzubeziehen oder wieder auszuschliessen, wenn sie nicht mehr verfügbar sind.

Das Kapitel 6 - “Ressource: Netzwerk RAID-1 Array” bespricht die implementierte Netzwerk RAID-1 Ressource im Detail.

6. 6. 2. Resultate von Service Primitiven

Jede Service Primitive gibt dem aufrufenden Service Routing Redundancy Daemon ein Resultat zurück, das aussagt, ob der Aufruf der Service Primitiven erfolgreich war, nicht erfolgreich war, oder noch gar nicht beendet wurde. Sollte der Aufruf noch gar nicht beendet sein, so braucht der Service mehr Zeit, und es wird zu einem späteren Zeitpunkt evaluiert werden, was das Resultat des Aufrufs war. Dies schafft die Möglichkeit, auch Service Primitiven zu schreiben, die zu lange dauern, so dass normalerweise der Service Routing Redundancy Daemon zu lange aufgehalten und so seine Funktion beeinträchtigt würde.

6. 7. Service Checking

Um ein einwandfreies Funktionieren eines Services garantieren zu können, hat jeder Service die Möglichkeit, über den Service Primitive “check” sich selbst in seiner Funktionsfähigkeit zu prüfen.

Dienste, wie zum Beispiel ein Webserver, haben die Möglichkeit, durch das echte Verbinden auf den Port 80 und dem Request einer Webseite zu prüfen, ob sie voll funktionsfähig sind.

Eine Datenbank könnte eine bestimmte Datenbankabfrage ausführen, um zu überprüfen, ob eine Datenbank aus der Sicht des Benutzers voll funktionsfähig ist.

Je komplizierter und ausführlicher diese Implementierten Checks sind, desto akurater kann ein Cluster-Node den Zustand eines Services abschätzen und korrekt darauf reagieren.

6. 8. Ressource: Netzwerk RAID-1 Array

In diesem Kapitel wollen wir uns einen kleinen Überblick über die als Ressource im Service Routing Redundancy Daemon implementierte Netzwerk RAID-1 Disk verschaffen.

Um die volle Funktionsfähigkeit des implementierten Cluster-Servers demonstrieren zu können, war es nötig, wenigstens eine Form einer verteilten Datenspeicherung anzubieten. Den einfachsten Weg, einfach ein exotisches, verteiltes Dateisystem zu verwenden, war dem Autor der Arbeit zu unflexiebel und zu aufwändig in der Verwaltung und im Unterhalt desselben. Eine SCSI-Lösung mit zwei Servern an einem Bus ist ebenfalls sehr schwerfällig und ebenfalls recht speziell.

Um trotzdem eine verteilte Datenspeicherung zu implementieren und auch, um einen neuen Ansatz, der rein in Software, ohne spezielle exotische Dateisysteme auskommt, vorzuführen, wurde in dieser Arbeit ein Netzwerk RAID-1 mit Hilfe von Netzwerk Block Devices[1] für Linux implementiert..

Netzwerk Block Devices sind ganz normale Devices, die auch ganz normal für eine RAID Mirror Konfiguration verwendet werden können. Das spezielle an NBDs ist, dass sie Disks oder Files auf entfernten Remote-Systemen repräsentieren. Wird auf ein solches NBD-Device geschrieben, so schickt das System die zu schreibenden Blöcke an das Remote-System, das die Blöcke dann dort auf die Disk schreibt. So ist es möglich eine Disk an einem weit entfernten System so zu benutzen, als ob sie eine lokal am System angeschlossene Disk wäre.

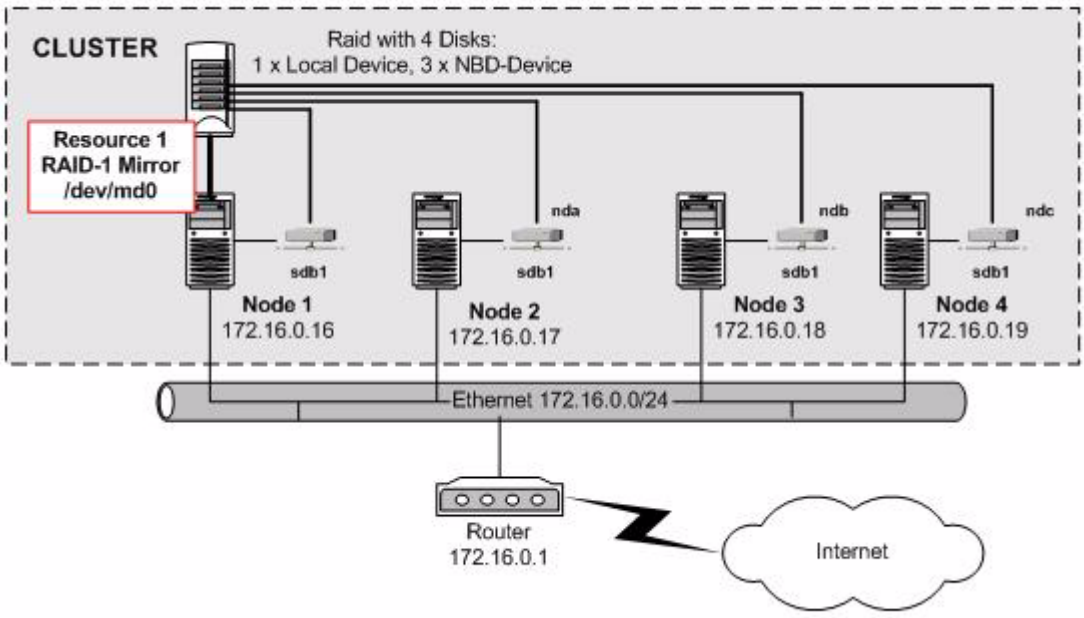
Um nun mit diesen Netzwerk Block Devices einen verteilten redundanten Datenspeicher zu bilden, verwenden wir das RAID Subsystem dazu, Daten, die auf dem Cluster-Node, der die Disk gerade verwenden darf und sie gestartet hat, geschrieben werden, zu den anderen, am Cluster beteiligten Cluster-Nodes und deren Disk zu replizieren.

Das RAID Subsystem hat also immer eine lokale Disk und mehrere (die maximale Anzahl entspricht der Cluster-Node Anzahl minus 1) Netzwerk Block Device Disk im Array. Scheidet ein Cluster-Node aus, so wird das Netzwerk Block Device, das zu ihm gehört als “Failed” im Array

1. NBD oder ENBD für Enhanced Network Block Devices, eine Linux Implementation

markiert und sollte es wieder hinzukommen, so wird es neu mittels “raidhotadd” wieder zum Array zugefügt und eine neue Plattenspiegelung zu ihm erfolgt.

In der folgenden Figur zeigen wir einen Cluster mit einer Netzwerk RAID-1 Ressource auf 4 Cluster-Nodes, die in ihrer Gesamtheit einen Cluster-Server bilden:



Figur 20: Cluster-Server mit einer Netzwerk RAID-1 Ressource

Die implementierte Netzwerk RAID-1 Ressource ist in sich selbst fast wieder ein kleines Projekt. Sie besteht aus über 2000 Zeilen Perl Code, die die Service Primitiven für die Netzwerk RAID-1 Disk implementieren. Die Service Primitiven machen exzessiven Gebrauch der Unix Kommandos des RAID Subsystems, wie auch der Network Block Devices.

Das RAID-1 Array wird on-the-fly erstellt und die Netzwerk Block Device Daemone werden bei Bedarf gestartet und gestoppt.

Die Netzwerk RAID-1 Ressource verwendet alle in Kapitel 6 - “Service Primitiven” auf Seite 60 beschriebenen Service Primitiven.

Jeder Cluster-Node hat schon zur Zeit seiner Aktivierung einen NBD Server gestartet, damit ein anderer Cluster-Node, der den neuen, frisch aktivierten Cluster-Node sieht und die Ressource gestartet hat, auch ein korrektes Join ausführeren kann, das wiederum beinhaltet einen NBD Client bei sich zu starten und ein “raidhotadd” des entsprechenden neuen NBD Device auszuführen.

Sollte ein Cluster-Node ausscheiden, so wird durch die Leave Service Primitive das NBD Device dieses Cluster-Nodes als Failed markiert und scheidet so aus dem RAID-1 Array aus. Danach wird bei einem Deactivate, des herunterfahrenden Cluster-Nodes, der bei der Aktivierung gestartete NBD Server wieder gestoppt.

Sollte ein Cluster-Node vom aktivierten Zustand aus die Ressource starten, so wird er zuerst den eigenen NBD Server seiner lokalen Disk stoppen und dann NBD Clients für jeden verfügbaren Cluster-Node des Cluster-Servers starten. Danach bildet der startende Cluster-Node aus den vorhandenen Netzwork Block Devices einen RAID-1 Mirror.
Vielfältige Kontrollmechanismen gegen Datenverlust und gegen die unabsichtliche Spiegelung in die falsche Richtung wurden implementiert.

Beispielsweise werden RAID Mirrors mit “Tags” markiert, die etwas über den letzten Zugriff und die “freshness” der Daten aussagen. So kann garantiert werden, dass auch bei speziellen Situationen wie, beispielsweise wenn die letzten beiden Cluster-Node abgeschaltet, aber in einer anderen Reihenfolge wieder eingeschaltet werden. Der Cluster-Server wird dann mit dem starten der Ressource so lange warten, bis auch die frischeste Disk im Netzwerk RAID-1 Array vorhanden ist, bevor er das Array startet.

Es würde zu weit führen alle Details der Netzwerk RAID-1 Disk hier besprechen zu wollen. Das Perl Script der Ressource ist in der Source Distribution enthalten und kann frei eingesehen und weiterverwendet werden.

Es sei bemerkt, dass diese Implementation der Netzwerk RAID-1 Disk nur ein Prototyp darstellt. Jeder, der Daten auf ihr speichert, tut dies auf eigene Gefahr!

7. Resultate

In diesem Kapitel wollen wir die erreichten Ziele dieser Arbeit besprechen. Wir wollen zeigen, welche theoretischen und praktischen Resultate unsere Arbeit erarbeitet hat und wir besprechen die Fähigkeiten und Leistungen des implementierten Cluster-Servers.

7. 1. Theoretische Resultate

Wir haben im Zuge dieser Arbeit eine neue Methode entwickelt, wie Cluster-Server ihre Service und Cluster-Node Zustandsinformationen austauschen können. Ein grosser Vorteil der neuen Methode ist, dass sie kein neues oder proprietäres Protokoll verwendet, sondern sich eine Fähigkeit des OSPF Protokolles zu Nutze macht, opaque Daten in sogenannten Opaque LSAs verbreiten zu können.

Des Weiteren haben wir das Konzept des Cluster-Servers über Subnetzgrenzen hinweg erweitert, so dass es möglich ist, Cluster-Nodes zu konzipieren, die über mehrere Subnetze hinweg kooperativ einen Cluster-Server erstellen. Dazu mussten wir eine neue Methodik für das Service Routing einführen, das den Anforderungen, auch über Subnetzgrenzen hinweg zu funktionieren, genügte. Dadurch, dass wir echtes Routing einsetzen und Host-Routen dazu verwenden, einzelne Dienste erreichen zu können, ist es möglich, die Services über weite Strecken innerhalb einer Routing Domäne zu routen.

Diese theoretischen Resultate haben sich in der Implementation des Service Routing Redundancy Daemons bewährt, der auf genau diesen theoretischen Erkenntnissen aufbaut und sie in einer praktischen und funktionsfähigen Implementation vereint.

7. 2. Zebra's OSPF Routing Daemon

Im Zuge dieser Arbeit wurde das Opaque-Modul des Zebra OSPF Routing Daemons für dynamische Informationen verwendet, was bedeuet, dass dieser Teil des Routing Daemons sehr stark benutzt wurde.

Es zeigte sich, dass unser Service Routing Redundancy Daemon eine der ersten Applikationen ist, die sich dieser Fähigkeit des OSPF Routing Daemons zu Nutze macht. Dies führte dazu, dass wir sehr viele unerkannte Fehler des OSPF Daemons entdecken durften.

Da das erklärte Ziel eines Cluster-Servers Zuverlässigkeit ist, blieb uns nichts anderes übrig, als im Zuge unserer Arbeit auch die Fehler des OSPF Daemons zu beheben.

Wir konnten alle Fehler entfernen, die ein zuverlässiges Funktionieren des Service Routing Redundancy Daemons gefährdeten und haben eine Reihe von Patches produziert, die wir dem Entwickler des Opaque-Moduls des OSPF Daemons, Masahiko Endo [19], zukommen liessen.

Masahiko Endo wird die Korrekturen dem CVS Repository des Zebra Routing Daemons zufügen. Wir möchten ihm an dieser Stelle für seine Unterstützung und seine wertvollen ASCII-Zeichnungen danken. Er war eine grosse Hilfe bei der Fehlersuche im Zebra OSPF Daemon.

7. 3. OSPFAPI - Interface to OSPFD

Der Service Routing Redundancy Daemon verwendet das an der ETH Zürich entwickelte OSPFAPI [16]. Das OSPFAPI besteht aus einem Patch zum Zebra OSPF Routing Daemon, der einen OSPFAPI Server Kern in den OSPF Daemon als OSPF Modul einfügt, und einer Client Bibliothek für die OSPFAPI Client Entwicklung.

Es hat sich gezeigt, dass OSPFAPI, bis auf kleine Probleme, die wir im Zuge der Entwicklung des Service Routing Redundancy Daemon entdeckt und behoben haben, perfekt seiner Aufgabe gewachsen war. Die Kommunikation zwischen Redundanz Daemon und OSPF Daemon war von Anfang an sehr zuverlässig und bereitete wenige bis gar keine Probleme.

7. 4. Service Routing Redundancy Daemon

Unsere theoretischen Erkenntnisse wurden im Service Routing Redundancy Daemon, einem Unix Daemon, implementiert. Die theoretischen Überlegungen haben sich bewährt und es zeigte sich, dass die Implementation eines Cluster-Servers, mit dem in Kapitel 5 - “Design: Service Routing Redundanz” auf Seite 27 entworfenen Design, zu einem voll funktionsfähigen und mit den meisten gängigen Fähigkeiten ausgestatteten Cluster-Server geführt hat.

Die Implementation wurde zu einem sehr grossen Projekt, das, mit inzwischen etwas über 17’000 Zeilen C-Code für den Redundanz Daemon und etwa 2500 Zeilen Perl-Code für die Netzwerk RAID-1 Disk, implementiert wurde.

Es zeigte sich, dass die anfänglichen Befürchtungen, es könnte durch die nicht vorhandene Atomizität des implementierten Protokolles zu Problemen kommen, sich nicht bewahrheitet haben. Der Cluster-Server arbeitet zuverlässig und erbringt die gewünschte Redundanz.

Da der Zebra OSPF Daemon anfänglich sehr unzuverlässig war, wurde der Service Routing Redundancy Daemon so entwickelt, dass er auch damit umgehen kann. Sollte sich ein OSPF Daemon verabschieden, so stoppt der Service Rounting Redundancy Daemon alle Service und wartet bis der OSPF Daemon wieder verfügbar ist. Ist dies der Fall, so verhält sich der Redundanz Daemon so, als ob er neu gestartet worden wäre und meldet sich als verfügbarer Cluster-Node neu bei den anderen verfügbaren Cluster-Nodes an.

Es zeigte sich, dass mit dem Service Routing Redundancy Daemon alle gängigen Ressourcen und Dienste implementiert werden können. Sowohl ein Webserver wie auch eine hoch komplexe Ressource, eine Netzwerk RAID-1 Disk, wurden erfolgreich implementiert. Weitere Services werden folgen, werden aber keinerlei Herausforderungen an die Implementierung mehr stellen.

Der implementierte Webserver hat sich auch bewährt, hat er doch das komplexe, verteilte Konfigurierungsproblem eines Cluster-Servers auf ein Minimum an Aufwand und ein Maximum an Komfort vereinfacht.

Ebenfalls bewährt hat sich die Möglichkeit, die LSA Datenbank direkt im Redundanz Daemon einsehen zu können und so Fehlverhalten und falsch verbeitete Service Zustände gerade innerhalb des Webservers einsehen zu können.

Das implementierte Routing Daemon Kommando Interface erlaubt die vollständige Einsicht in die Zustände der beiden Routing Daemone, was sowohl für das Verständins der Konfiguration und der Routing Tabellen von grossem Vorteil war und ist, als es auch dem Benutzer ermöglicht, das Service Routing direkt an der Quelle zu beobachten.

Die Service Zustandsvisualisierung zeigt sowohl Service Gruppen Zugehörigkeiten, wie auch Service Abhängigkeiten. Alle Service Zustände sind auf einer einzigen Webserver Seite einsehbar und die Services können von dort aus manipuliert werden. Diese zentrale Kontrolle der Service, kombiniert mit der Möglichkeit die Webserver mit SSL, Certifacte Authorities und Zertifikaten zu konfigurieren, erlauben komfortabel die Service zu kontrollieren, zu sichern und zu verwalten.

7. 5. Fail-Over Messungen: Minimum/Maximum/Mittel

Im Folgenden wollen wir ganz kurz auf einige kleine Messungen eingehen.

Wir wollten wissen, wie lange ein Dienst nicht verfügbar ist, wenn er von einem Cluster-Node zu einem anderen wechseln muss.

Des Weiteren interessierte uns, wie sich ein solcher Dienst verhält, wenn er in einer Service Gruppe mit einer Netzwerk RAID-1 Ressource ist, von der er abhängig ist.

Eine dritte Messung soll zeigen wie sich ein Dienst beim Versagen eines OSPF Routing Daemons verhält und wie lange der Dienst in diesem Fall nicht verfügbar ist.

Die folgende Tabelle J zeigt die gefundenen Resultate. Die Messungen wurden vom Border-Router des Cluster Node Netzes aus gemacht. Das heisst dass dieser Router einen einzelnen Hop von jedem Cluster Node entfernt ist. Dies ist der Normalfall für einen Cluster, bei dem alle Cluster Nodes an einem einzelnen lokalen Netzwerk angeschlossen sind.

Fail-Over	Minimum	Maximum	Mittel[a]
A[b] unabhängig	1 sec	9 sec	5 sec
A abhängig von D[c]	17 sec	22 sec	21 sec
A unabhängig, OSPFD crash	4 sec	8 sec	5 sec

Tabelle J: Vergleich der Messungen von verschiedenen Fail-Over Situationen

a. Arithmetisches Mittel aus allen Messwerten
b. Apache Webserver Dienst auf 3 symmetrisch konfigurierten Cluster Nodes
c. Netzwerk RAID-1 Disk (512MB) auf 3 symmetrisch konfigurierten Cluster Nodes

Wie man sehen kann liegen die Mittelwerte der Messungen zwischen 5 und 21 Sekunden. Eine Downtime von solcher Grösse ist für viele Dienst-Arten akzeptabel. Dass ein Webserver für 5 Sekunden nicht verfügbar ist kann von Client-Applikationen eigentlich nicht bemerkt werden.

Auch abhängige Dienste haben eine akzeptable Downtime!

Wie man sieht, erhöht sich die Zeit zwar stark, von 5 auf 21 Sekunden, doch dies ist darauf zurückzuführen, dass die Service Primitiven der Netzwerk RAID-1 Disk sehr komplex und zeitaufwendig sind und weniger darauf, dass die Services von einander abhängig sind.

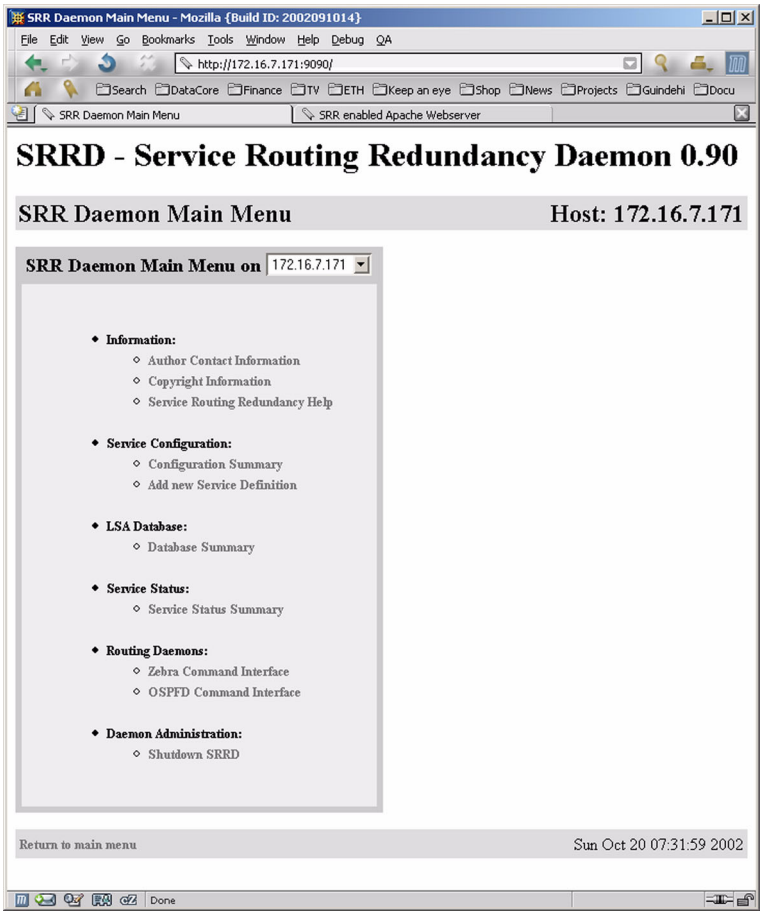
Weitere Tests mit anderen Ressourcen, die weniger aufwendige Service Primitiven besitzen, werden zeigen, dass Services mit nicht so komplexen Service Primitiven, auch wenn sie voneinander abhängig sind, eine kürzere Downtime besitzen, als wenn sie von der hoch komplexen Netzwerk RAID-1 Disk abhängig sind.

8. Demonstration

Im folgenden wollen wir den Service Routing Redundancy Daemons demonstrieren. Wir haben einem Test-Cluster aus drei Cluster Nodes konfiguriert und aufgesetzt. Die drei Cluster Nodes sind genau gleich konfiguriert. Auf jedem Node ist sowohl eine Apache Webserver Dienst wie auch eine Netzwerk RAID-1 Ressource installiert.

8. 1. Service Routing Redundancy Daemon

Die folgende Figur zeigt das Hauptmenü des Service Routing Redundancy Daemon auf dem Cluster Node mit der IP Adresse 172.16.7.171:



Figur 21: Das Hauptmenü eines SRRD Cluster Node Webservers

Jede Seite des Webservers zeigt in der oberen rechten Ecke den aktuellen Cluster Node, der im Moment manipuliert wird. Das Pop-Up mit der IP Adresse gibt dem Benutzer die Möglichkeit zwischen den verschieden bekannten Cluster Nodes hin und her zu schalten. Dabei wird der Benutzer auf den Webserver des entsprechenden Cluster Nodes verwiesen.

Am unteren, linken Rand aller Seiten wird immer die am meisten gebrauchten Menüpunkte eingeblendet. In der rechten unteren Ecke ist auf jeder Seite die Generierungszeit der Webserver Seite angeben.

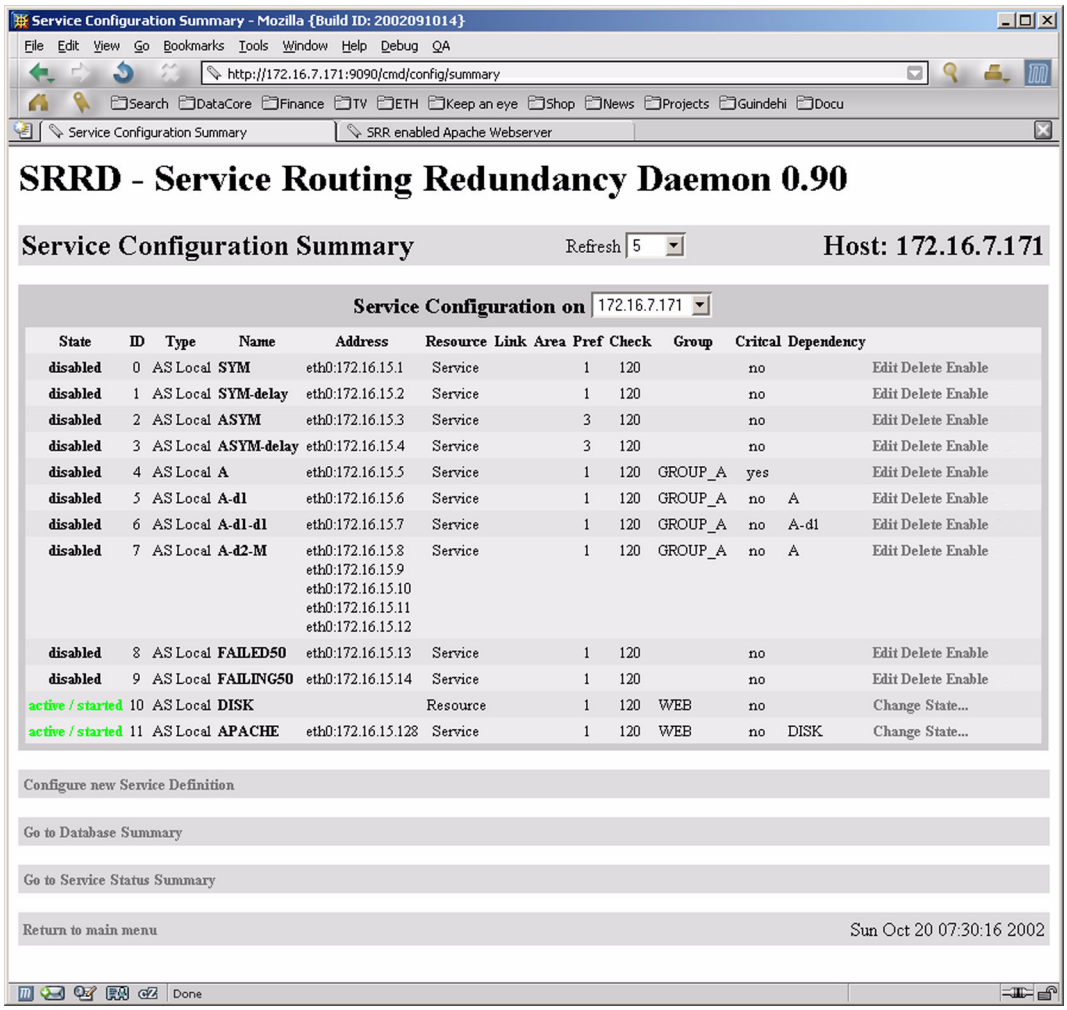
Zum Teil ermöglicht eine Seite einen automatischen Refresh. Dieses Pop-Up wird gegebenfalls neben dem Pop-Up des Cluster Nodes erscheinen.

Wie in Figur 21: “Das Hauptmenü eines SRRD Cluster Node Webservers” auf Seite 69 zu sehen ist, bietet das Hauptmenü die folgenden Optionen:

- **Information:** Hier sind Autor Kontakt Informationen und die Hilfe Seiten zu finden.
- **Service Configuration:** Hier können neue Service konfiguriert werden, wie auch die schon konfigurierten Services und eingesehen und gegebenfalls verändert werden. Im Kapitel 8 - “SRRD Service Konfiguration” auf Seite 71 wird dies gezeigt.
- **LSA Database:** Auf dieser Seite kann die Link State Database eingesehen werden. Alle LSA Typen werden dort, mitsamt ihren opaquen Daten, visualisiert. Wie dies aussieht zeigt das Kapitel 8 - “SRRD LSA Datenbank” auf Seite 72
- **Service Status:** Hier kann der globale Cluster Zustand eingesehen werden. Alle Service Zustände werden visualisiert und der Cluster kann vom Benutzer beeinflusst werden. Das Kapitel 8 - “SRRD Service Status Überblick” auf Seite 73 zeigt dies genauer.
- **Routing Daemons:** Diese beiden Menüpunkte ermöglichen es dem Benutzer alle, von den Routing Daemons verstandenen Kommandos, auszuführen und die Resultate dieser Kommandos einzusehen. Im Kapitel 8 - “SRRD Zebra/OSPF Kommando Interface” auf Seite 76 wird dies demonstriert.
- **Daemon Administration:** Mit diesem Menüpunkt kann der Cluster Node heruntergefahren werden. Alle Services werden dabei gestoppt und deaktiviert.

8. 1. 1. SRRD Service Konfiguration

Die Konfigurationsübersicht des Service Routing Redundancy Daemons sieht wie folgt aus:



Figur 22: Service Konfigurationsmenü eines SRRD Cluster Nodes

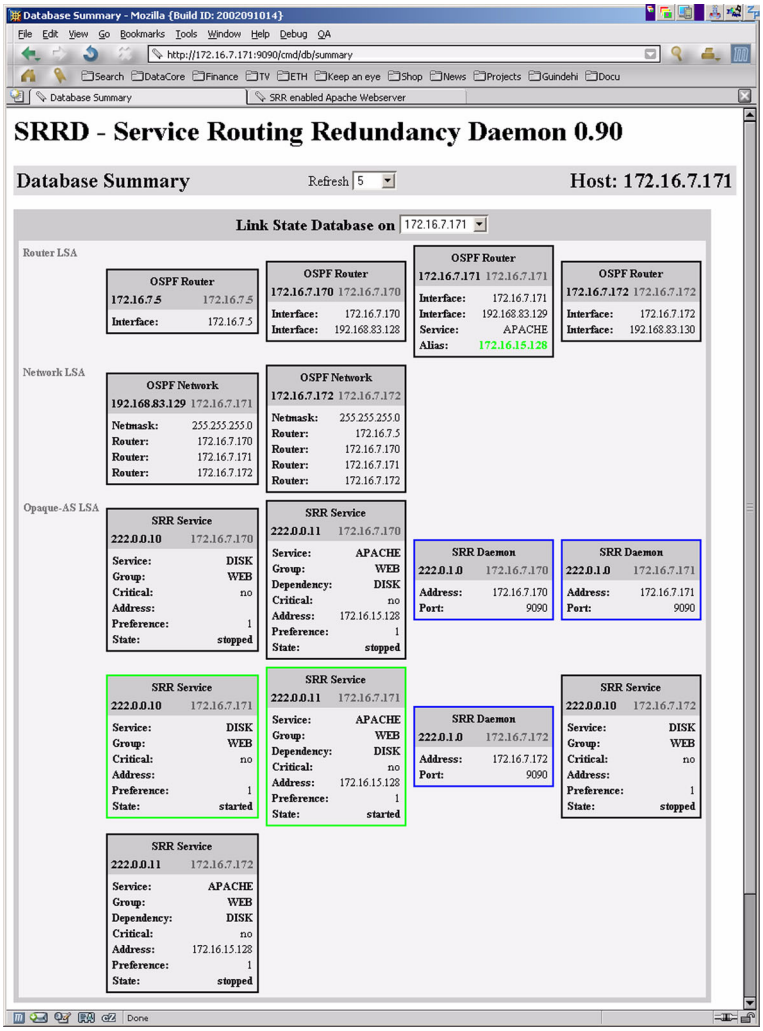
Wie zu sehen ist kann auch hier wieder der Cluster Node, wie auch die Refresh-Zeit, mit Pop-Ups ausgewählt werden. Die Übersicht zeigt Zustand, ID, Typ, Name, IP Adresse(n), Service-Typ, Link, Area, Präferenz, Prüf-Intervall, Gruppe, Service Abhängigkeit wie auch ob der Service kritisch für die Service Gruppe ist.

Ganz am rechten Rand sind die für jeden konfigurierten Service möglichen Kommandos als Links enthalten. Sie ermöglichen das Verändern, das Löschen wie auch das Enablen eines Services. Ist einer der Services in einem aktiven Zustand, so kann seine Konfiguration nicht verändert werden. Ein Service in einem aktiven Zustand kann nur von der Service Status Überblick Seite, die in Kapitel 8 - “SRRD Service Status Überblick” auf Seite 73 genauer beschrieben wird, manipuliert werden.

Mit dem Menüpunkt “Configure new Service Definition” ist es möglich neue Services zu konfigurieren.

8. 1. 2. SRRD LSA Datenbank

Auf dieser Webserver Seite wird die LSA Datenbank visualisiert. Die folgende Figur zeigt den Zustand der LSA Datenbank, wenn der Cluster mit allen drei Cluster Nodes aktiv ist.



Figur 23: LSA Database Zustand eines SRRD Cluster Nodes

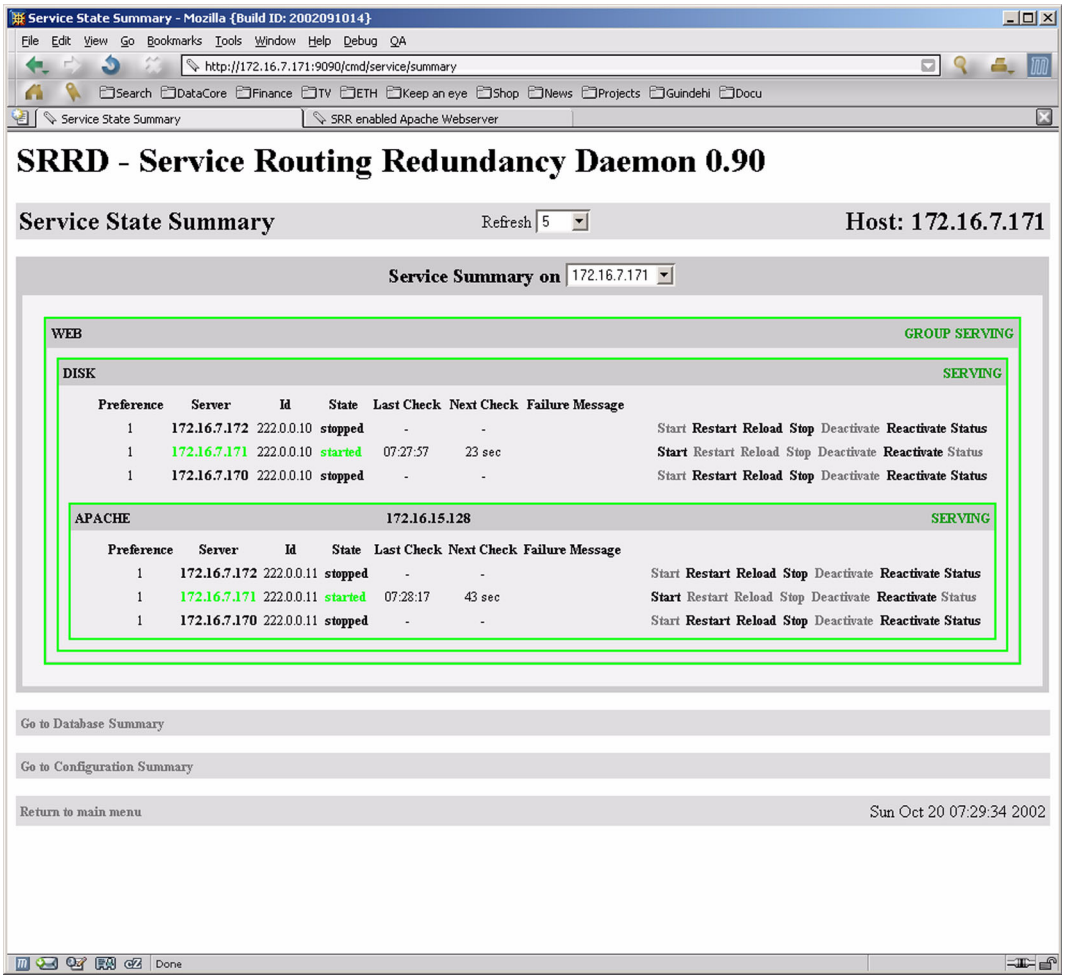
In der obigen Figur sind die auf dem lokalen Cluster Node bekannten, von den vorhandenen OSPF Routern verbreiteten, Router- und Netzwerk LSAs dargestellt. Wie zu sehen ist, sind momentan 4 Router verfügbar, von denen einer die Apache Webserver Dienst IP Adresse als Interface Alias installiert hat. Wie auch zu sehen ist, sind die Router an zwei verschiedene Netzwerke angeschlossen. Die Netzwerk LSAs zeigen die auf den Netzwerken vorhanden Router Adressen an.

Etwas weiter unten sind die Opaque-AS LSAs zu sehen, die von den Redundanz Daemons ins Netz geflutet werden. Wie man sehen kann, sind momentan drei verschiedene Cluster Nodes aktiv und jeder von ihnen hat die beiden Services DISK und APACHE konfiguriert. Auf dem Cluster Node mit der IP Adresse 172.16.7.171 sind die beiden Services gestartet. Sollte dieser Cluster Node ausfallen, so sind zwei weitere vorhanden, die die Services starten können.

Wie ebenfalls zu sehen ist sind die beiden Services in der Service Gruppe WEB und der APACHE Dienst ist von der Ressource DISK abhängig. Alle Services haben die selbe Service Präferenzen konfiguriert.

8. 1. 3. SRRD Service Status Überblick

Hat ein Cluster aktive Service Zustände, so kann auf der Service Status Überblicksseite der Cluster Zustand eingesehen werden:



Figur 24: Service Zustand eines SRRD Clusters

Wie in der obigen Figur zu sehen ist, besteht der Cluster aus drei symmetrisch konfigurierten Cluster Nodes. Auf diesem Cluster ist eine Service Gruppe WEB konfiguriert. Diese Service Gruppe enthält zwei Services, eine Ressource, eine Netzwerk RAID-1 Disk, mit dem Namen DISK und einen Dienst, ein Apache Webserver, mit dem Namen APACHE.

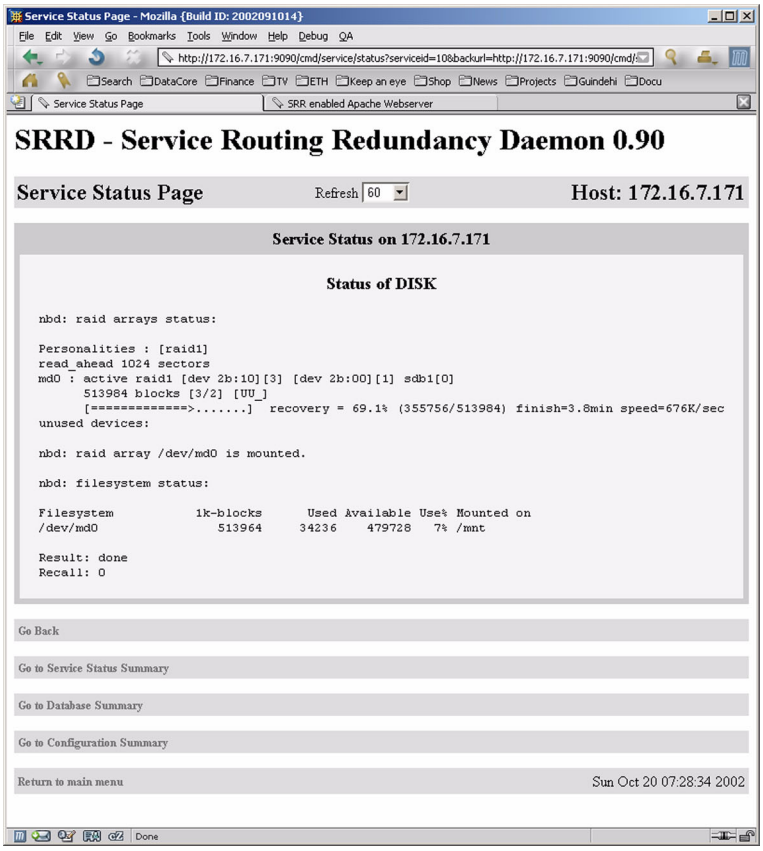
Beide Services sind momentan auf dem Cluster Node mit der IP Adresse 172.16.7.171 gestartet. Die beiden anderen Cluster Nodes werden bei Ausfall dieses Nodes die Services übernehmen und so deren Redundanz gewährleisten.

Sowohl Service Präferenz, Cluster Node IP, LSA ID, Service Zustand wie auch der Zeitpunkt der letzten Funktionsprüfung des Services werden angezeigt. Sollte einer der Services versagen, so wird die zugehörige Fehlermeldung angegeben. Auf der rechten Seite sind dann die mögli-

chen Kommandos als Links verzeichnet, die es dem Benutzer ermöglichen den Cluster zu manipulieren.

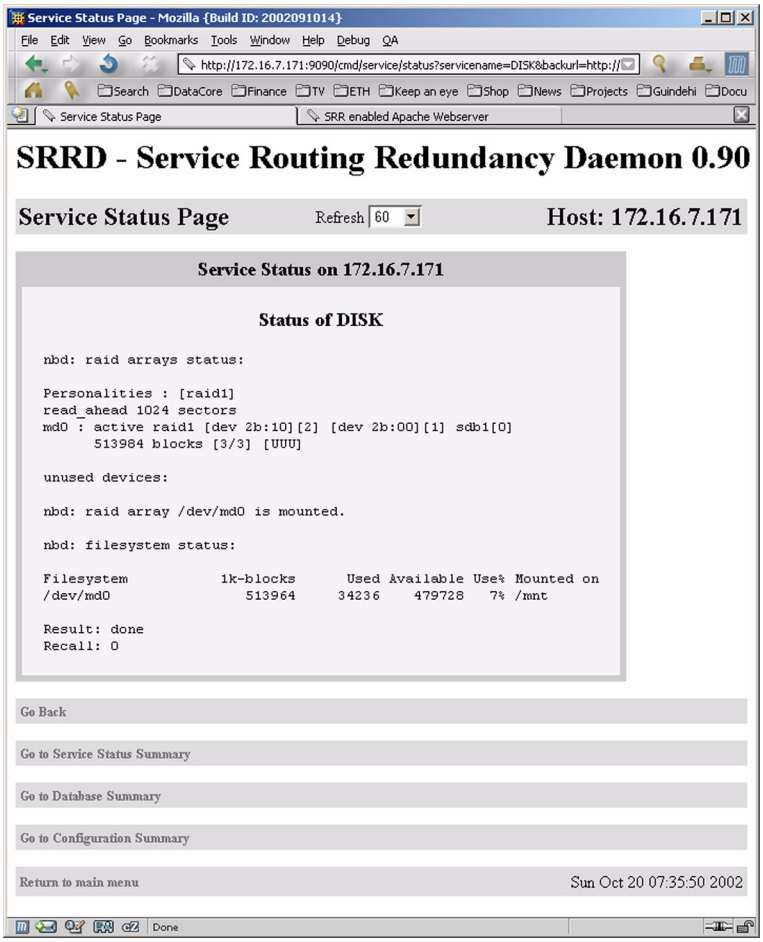
Auch hier kann sowohl der Cluster-Node wie auch die Refresh-Zeit der Seite mit den beiden Pop-Ups eingestellt werden.

Wir wollen im folgenden nun den Status der Ressource DISK auf dem Cluster Node 172.16.7.171 betrachten, kurz nachdem der Cluster Node 172.16.7.172 gestartet wurde. Natürlich musste die Netzwerk RAID-1 Ressource den neu dazugekommenen Cluster Node dem RAID-1 Array zufügen. Wie in der folgenden Figur zu sehen ist, hat das RAID-1 Array gerade das Netzwerk Block Device des neuen Nodes dem RAID neu hinzugefügt und ist dabei den Mirror neu zu spiegeln:



Figur 25: Zustand der RAID-1 Netzwerk Ressource bei der Synchronisation

Mit Hilfe dieser RAID-1 Mirrors wird jeder Schreibzugriff auf die Ressource automatisch auf alle vorhanden Cluster Nodes gespiegelt. Ist der RAID-1 Netzwerk Mirror des neu hinzugekommenen Cluster Nodes synchronisiert, so zeigt dies die Service Status Seite wie folgt:



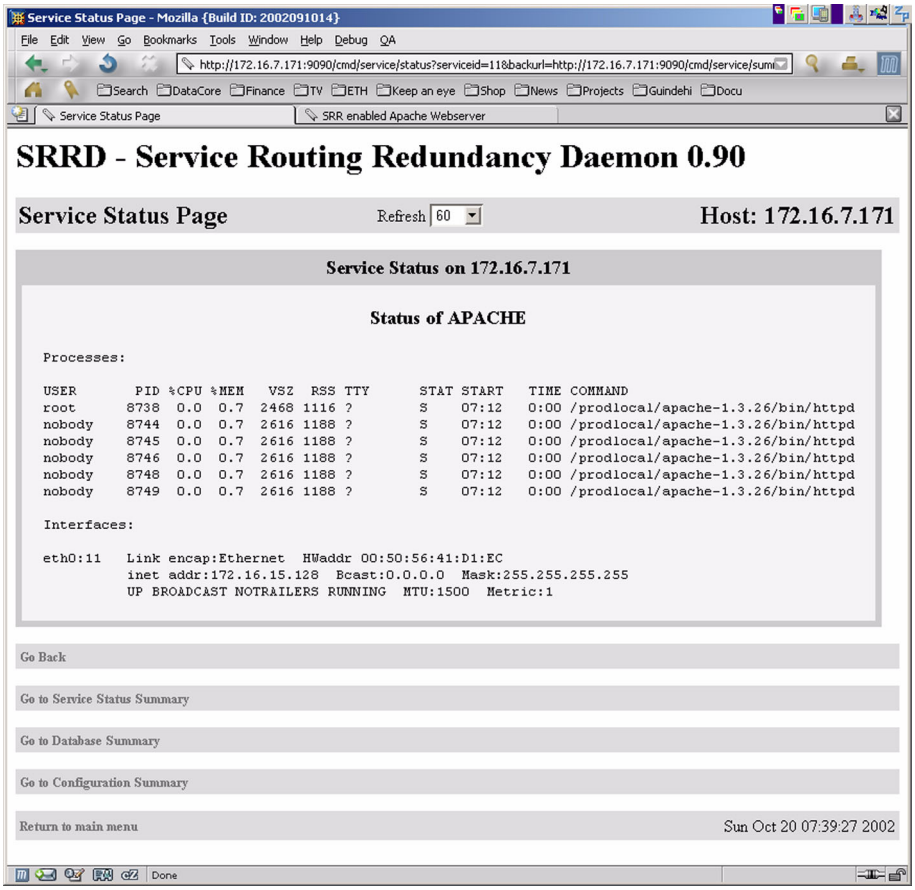
Figur 26: Zustand der RAID-1 Netzwerk Ressource nach der Synchronisation

Wie zu sehen ist sind alle drei Devices des RAID-1 dem RAID-1 Array zugefügt worden und die drei Mirror sind “up and running”, was mit [UUU] angezeigt wird.

Wir wollen im weiteren noch den Service Zustand des Apache Webserver Dienstes betrachten.

Dieser Dienst verwendet die IP Adresse 172.16.15.128 um einen Apache Webserver zu starten und seinen Service Access Point (SAP) auf den Port 80 des Interface Aliases mit dieser IP Adresse zu binden.

Die Service Status Seite des Webservers zeigt laufenden Apache Prozesse, wie auch das extra für den Webserver erstellte Interface Alias mit der Adresse 172.16.15.128. Die spezielle Point-to-Point Netzmaske 255.255.255.255 ist ebenfalls in der Figur 27: “Service Status Seite eines Apache Webserver Dienstes” auf Seite 76 zu sehen.



Figur 27: Service Status Seite eines Apache Webserver Dienstes

8. 1. 4. SRRD Zebra/OSPF Kommando Interface

Sowohl der Zebra Routing Daemon wie auch der OSPF Routing Daemon kann mit Hilfe des Webservers des Service Routing Redundancy Daemons kontrolliert werden.

Jedes Kommando, das auf den VTYs der beiden Routing Daemon möglich, ist kann auch vom Webserver aus ausgeführt werden.

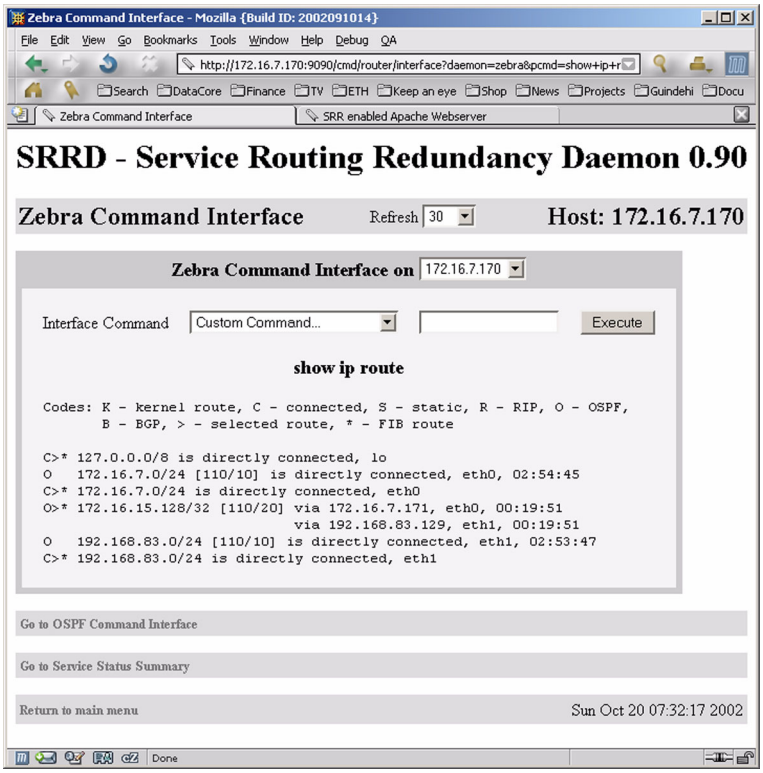
Im folgenden wollen wir zuerst die Routing Tabelle des Zebra Daemons auf dem Cluster Node 172.16.7.170 einsehen. Dieser Routing Daemon ist dafür zuständig im Kernel die entsprechenden Routen zu installieren. Auch der OSPF Daemon ist auf den Zebra Daemon angewiesen um Routen installieren zu können.

Der Zebra Routing Daemon markiert jede installierte Route mit einem Buchstaben: “C” steht für direkt, lokal erreichbare Routen und “O” steht für vom OSPF Routing Daemon installierte Routen.

Da die beiden Services momentan auf dem Cluster Node mit der Adresse 172.16.7.171 laufen, müssen alle Router im Netzwerk, wie auch alle anderen Cluster Nodes eine Route für den Apache Webserver Dienst zum Cluster Node 172.16.7.171 besitzen.

Das dazu auf dem VTY des Zebra Daemons auszuführende Kommando heisst “show ip route”. Es veranlasst den Zebra Daemon alle von ihm installierten, wie auch die nicht von ihm installierten Routen anzuzeigen.

Wie in der folgenden Figur zu sehen ist, ist diese Host-Route mit der Netzmaske 255.255.255.255 vom OSPF Routing Daemon nach dem Start des Dienstes auf dem Cluster Node 172.16.7.171 installiert worden:



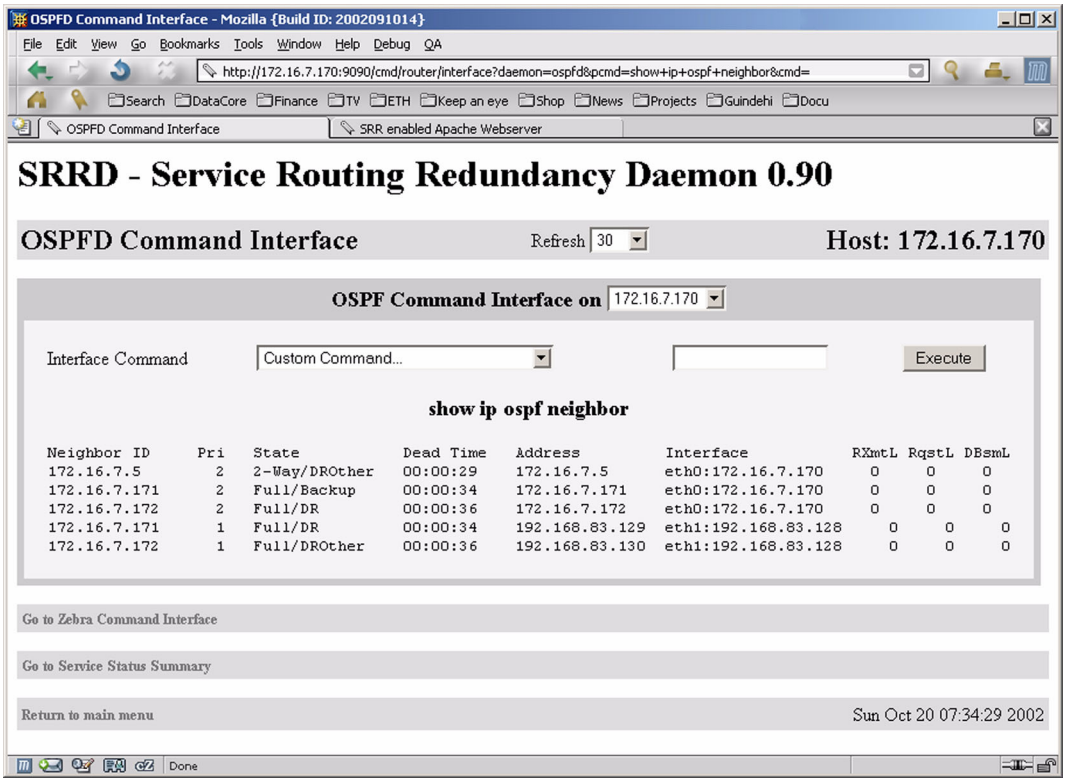
Figur 28: Zebra Kommando Interface eines Cluster Nodes - show ip route

Nun wollen wir als nächstes nachschauen, welche OSPF Nachbar Router vom OSPF Daemon gesehen werden. Es ist oft so, dass Fehlkonfigurationen in Firewalls und ähnlichem dazu führen, dass zwei OSPF Daemone sich gegenseitig oder auch einseitig nicht sehen.

Mit Hilfe des OSPF VTY Kommandos “show ip ospf neighbor” kann der Zustand der OSPF Neighbor Routers eingesehen werden.

Die folgende Figur 29: “OSPF Kommando Interface eines Cluster Nodes - show ip ospf neighbor” auf Seite 78 zeigt diese Router Nachbarn und ihre Zustände.

Wie zu sehen ist sind 5 Router direkte Nachbarn des Cluster Nodes 172.16.7.170, wobei dies aber nur 3 wirklich verschiedene Router sind, wie anhand der Nachbar-ID zu erkennen ist. Zwei dieser Router 172.16.7.171 und 172.16.7.172 sind jeweils über zwei verschiedene Interfaces erreichbar und desshalb zwei mal verzeichnet. Sowohl die “Dead Time” wie auch die Adresse des Routers auf diesem Netzwerk sind verzeichnet.



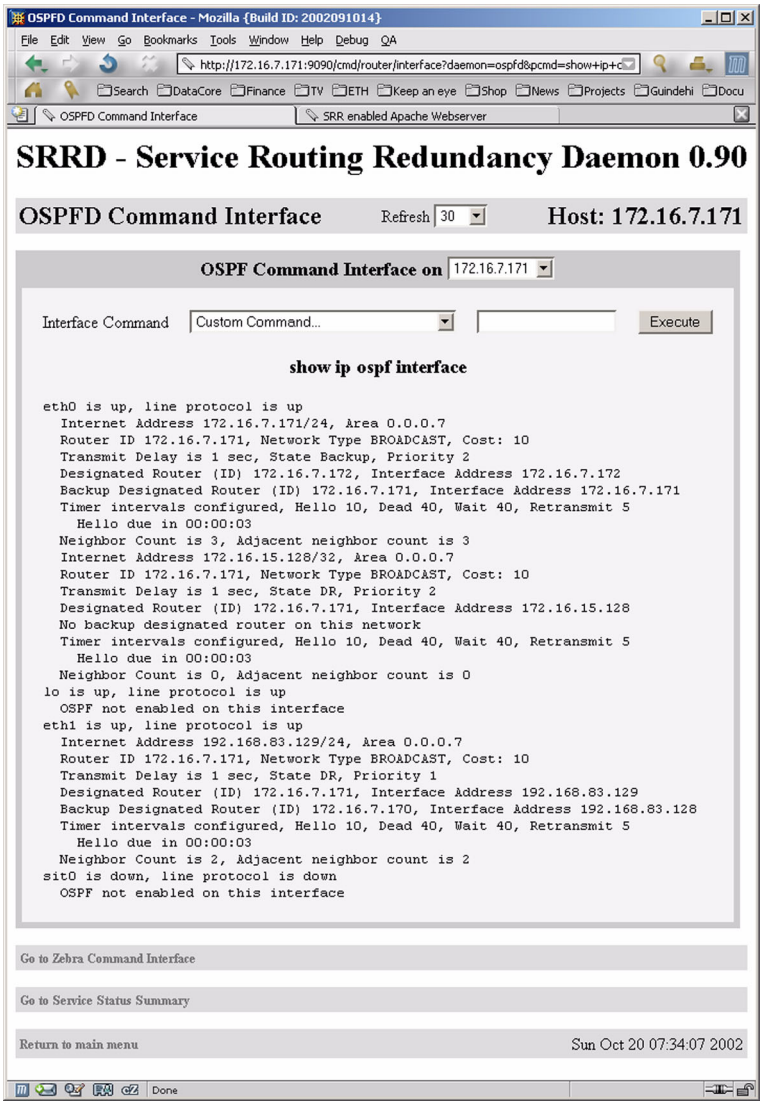
Figur 29: OSPF Kommando Interface eines Cluster Nodes - show ip ospf neighbor

Nun wollen wir noch die dem OSPF Daemon bekannten Interfaces auf dem Cluster Node 172.16.7.171 einsehen, auf dem ja momentan die beiden Services laufen.

Da der APACHE Dienst auf diesem Cluster Node gestartet ist, muss natürlich ein Interface Alias installiert sein, das mit der IP Adresse des Dienstes konfiguriert ist und eine Netzmaske von 255.255.255.255 hat.

Natürlich sind auch die physikalischen Interfaces sichtbar. Die vollständige OSPF Protokoll Konfiguration des Interfaces ist einsehbar. Sowohl “Designated Router” wie auch “Backup Designated Router” sind sichtbar. Die Anzahl der direkten OSPF Router Nachbarn wie auch die Zeit bis zum nächsten erwarteten OSPF Hello Paket sind verzeichnet.

Wie man in der Figur 30: “OSPF Kommando Interface eines Cluster Nodes - show ip ospf interface” auf Seite 79 sieht ist das Interface Alias für den Apache Webserver Dienst als Interface Alias mit der Adresse 172.16.15.128/32 auf dem physikalischen Interface eth0 installiert.



Figur 30: OSPF Kommando Interface eines Cluster Nodes - show ip ospf interface

9. Schlussfolgerungen und Ausblick

Es hat sich gezeigt, dass Service Routing eine mögliche und auch praktische Methode ist, um Cluster-Server zu bilden. Ein grosser Vorteil dieser Methodik liegt in darin, dass sie Subnetzgrenzen übergreifend funktioneirt und so ermöglicht, Cluster-Server einzusetzen, die mehrere Subnetze, ja ganze Routing Domänen umfassen können.

Es hat sich des Weiteren gezeigt, dass mit dem OSPF Protokoll sowohl Service wie auch Cluster-Node Zustände effektiv, schnell und zuverlässig im Netzwerk verteilt werden können.

Das entworfene Cluster-Server Design aus Kapitel 5 - “Design: Service Routing Redundanz” auf Seite 27 hat sich in der Praxis bewährt und die resultierende Implementation ist stabil und so zuverlässig, wie es von einem Prototypen zu erwarten ist.

Als Resultat dieser Stablitätsbetrachtungen des OSPF Daemons ist ein rubusterer Routing Daemon hervorgegangen, der auch mit stark dynamischen opaquen Daten zuverlässig und stabil funktioniert.

Als nächstes wäre eine Überarbeitung der OSPF Abstraktionsschicht des Service Routing Redundancy Daemon von Nöten. Es wäre sehr interessant, die ganze Kommunikationsschicht der Service Routing Redundanz Daemone vollständig zu abstrahieren und zu verallgemeinern, so dass nicht nur Service Discovery mit Hilfe des OSPF Protokolles möglich ist, sondern wahlweise das OSPF Protokoll oder andere proprietäre Methoden zum Zuge kommen.

Des weiteren wäre es sehr interessant eine Form von Global Atomic Broadcast (GAP) im implementierten Service Discovery des Redundanz Daemons einzubauen. Diese würde uns ermöglichen den globalen gegenseitigen Ausschluss der Services zu garantieren, was uns mit der vorliegenden Implementation nicht möglich ist.

Der Redundanz Daemon ist sehr zuverlässig. Weitere Service zu implementieren und zu testen ist eine der nächsten Aufgaben, die zu erledigen sind. Ein echter Einsatz in einem Produktions-Environment wäre höchst interessant und wird auch vorbereitet. Wir erhoffen uns Aussagen über die Belastung des Netzwerkes durch die Cluster Tätigkeit und mehr Zuverlässigkeitsaussagen über den Redundanz Daemon.

Die entworfene und implementierte Netzwerk RAID-1 Disk hat sich ebenfalls bewährt und funktioniert zuverlässig unter Normalbedingungen. Im Weiteren können zusätzliche Zuverlässigkeitstests und ein zusätzlicher Langzeit-Test die Sicherheit der Netzwerk RAID-1 Disk erhöhen.

Für die nächste Version des Redundanz Daemons wäre vorzusehen, die ganze LSA Datenbank-Verwaltung neu zu entwerfen und die gewonnenen Erkentnisse über die Anforderungen an die Datenstrukturen einfliessen zu lassen.

Der Autor glaubt, dass durch eine geschickte Gliederung und Kombination der Konfiguration und der Laufzeitdaten eines Services eine sehr kompakte und einfach zu verwendende Implementation einer Abstraktionsschicht zur LSA Datenbank möglich wäre. Dies und einige geschickt entworfene Datenbank-Query Funktionen könnten den komplizierten Umgang mit den sehr komplexen Konfigurations- und Laufzeitstatusinformationsstrukturen des Redundanz Daemons sehr vereinfachen.

Diese Arbeit hat gezeigt, dass es möglich ist, Service Zustandsinformationen mit Hilfe des Routing Protokolles zu verteilen und auch wieder zu empfangen. So ist es möglich, rein mit Hilfe des Routing Protokolles redundantes Service Routing zu implementieren, was uns auch erfolgreich gelungen ist. Diese Erfolge werfen die Frage auf, ob es nicht wirklich ein natürlicher und vernünftiger Ort ist, Service Discovery und Service Routing zu implementieren. Die bisherigen Ansätze haben dazu immer ein proprietäres Protokoll implementiert. Es fragt sich, ob ein modernes Routing Protokoll, hier im speziellen OSPF, nicht auch Service Discovery und Service Routing implementieren sollte, ist doch Routing die Hauptaufgabe des Routing Protokolles.

Wir werden den Service Routing Redundancy Daemon als Open Source verfügbar machen und weiter entwickeln. Seine bisher implementierten Fähigkeiten sehen sehr erfolgversprechend aus.

Wir hoffen mit unserer Arbeit einen wertvollen Beitrag zur gängigen Cluster-Server Methotik geleistet zu haben.

10. Anhang

10. A. Offizielle Problemstellung

Aufgabenstellung:	Prof. Dr. Bernhard Plattner, Ralph Keller
Titel:	Routing-basierte Cluster Server Redundanz
Beginn der Arbeit:	17. Juni 2002
Abgabetermin:	17. Oktober 2002
Betreuung:	Ralph Keller
Arbeitsplatz:	ETZ G69
Umgebung:	Linux, C
Art der Arbeit:	50% Protokolldesign, 50% Implementation

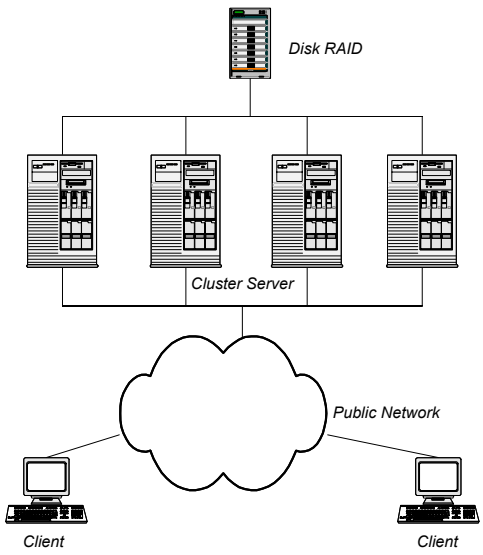
10. A. 1. Einleitung

Heutzutage bilden Informationssysteme einen zentralen Faktor in Geschäftsprozessen. Um einen reibungsfreien Ablauf garantieren zu können, sind Organisationen darauf angewiesen, dass Netzwerkdienste jederzeit verfügbar sind. Informationssysteme sind jedoch einer Reihe von Problemen unterworfen, wie Hard- und Softwarefehlern, unzuverlässige neue Softwareversionen, unvorhersehbare Eventualereignisse (Brand, Zerstörung), Softwareabstürze durch Überlastung und vielen mehr.

Um zu vermeiden, dass Informationssysteme wegen eines einzigen Problems (“Single Point of Failure”) nicht mehr zur Verfügung stehen, werden dafür sogenannte Cluster Server erstellt, welche auch beim Auftreten von Problemen weiterhin für Client-Applikationen verfügbar sind.

10. A. 2. Cluster Server

Ein Cluster-Server besteht aus zwei oder mehreren unabhängigen Servern, welche normalerweise an einen gemeinsam genutzten Festplattenstapel angeschlossen sind. Die Festplatten sind mit einem gemeinsamen SCSI- oder Fibre-Channel-Bus verbunden, auf welchen alle Server Zugriff haben. Im Normalfall greift nur ein Server auf die Festplatten zu, beim Ausfall eines Servers kann jedoch ein anderer Server die Kontrolle über die Festplatten übernehmen. Um eine vollständige Sicherheit zu garantieren, sollte der Festplattenstapel als Hardware-RAID-System ausgelegt sein, um auch beim Ausfall einer Festplatte einen Erhalt der Daten zu garantieren.



Der Cluster stellt einen oder mehrere virtuellen Server dar, welcher von Clients über das Netzwerk genutzt werden können. Der virtuelle Cluster Server wird dabei über eine virtuelle IP-Adresse angesprochen, welche dem Server, welcher gerade die Anfragen beantworten soll, zugewiesen wird. Beim Ausfall des aktiven Cluster Servers wird die virtuelle IP Adresse einem anderen Server innerhalb des Clusters zugewiesen, welcher dann die Kontrolle über den Cluster übernimmt und Anfragen von Clients beantwortet. Die Cluster Servers erkennen automatisch den Ausfall von andern Cluster Servern und der Dienst wird von einem anderen vorhandenen Server transparent übernommen und auf diesem neu gestartet (“Fail Over”).

10. A. 3. Aufgabenstellung

In dieser Aufgabe geht es darum, ein Framework zu entwickeln, welches erlaubt, mehrere Server zu einem Cluster Server zusammenzufassen. Der Cluster Server wird von Clients über eine virtuelle IP-Adresse angesprochen, welche jeweils dem aktiven Server zugeteilt wird. Dabei geht es darum, ein Protokoll zu Entwickeln, welches je nach Zustand innerhalb des Cluster Servers die virtuelle IP-Adresse einem anderen Server zuteilt. Dabei sind folgende Funktionalitäten notwendig:

1. Client-Applikationen sollten beim Ausfall eines Servers möglichst nicht beeinträchtigt werden, d.h. der Wechsel von einem Server zu einem anderen Server sollte für die Applikation transparent geschehen .
2. Der Cluster Server lässt sich aus einer Menge von Servern bilden, welche nicht am selben physikalischen LAN angeschlossen sind.
3. Das Framework soll keinen “Single Point of Failure” aufweisen. Insbesondere ist zu vermeiden, dass sämtlicher Verkehr durch einen einzigen Router geleitet werden muss.

10. A. 3. a. Ziel

Ziel dieser Arbeit ist ein lauffähiges, flexibles Framework, welches es erlaubt, einen Cluster Server aus mehreren Servern zu bilden.

Demonstrieren Sie die Funktionsweise ihres Frameworks, indem Sie zeigen, wie eine Client-Applikation trotz Ausfall eines Servers weiter funktioniert, indem ihr Framework automatisch auf einen entsprechenden anderen Server ausweicht.

10. A. 3. b. Vorgehen

- Lesen Sie sich in die Literatur zum Thema ein. Machen Sie sich zuerst mit bestehenden Cluster Server Systemen vertraut ([1], [2], [3], [4], [5], [6], [7]) und studieren Sie deren Funktionsweise. Charakterisieren Sie jede der bestehenden Lösungen anhand von Vor- und Nachteilen.
- Erstellen Sie sich einen Zeitplan.
- Skizzieren Sie eine Lösung, welche unseren gestellten Anforderungen genügt. Wichtig dabei ist in diesem Ansatz, dass kein “Single Point of Failure” besteht.
- Definieren Sie ein Protokoll, wie die verschiedenen Server miteinander kommunizieren sollten.
- Implementieren Sie die notwendigen Komponenten unter Linux.
- Demonstrieren Sie die Lauffähigkeit ihres System, indem Sie zeigen, wie eine Client-Applikation trotz Ausfall eines Servers transparent weiter arbeiten kann.
- Auf eine klare und ausführliche Dokumentation wird besonders Wert gelegt. Es wird empfohlen, diese laufend nachzuführen und insbesondere die entwickelten Konzepte ausführlich schriftlich festzuhalten.

10. A. 4. Bemerkungen

- Mit dem Betreuer sind wöchentliche Sitzungen zu vereinbaren. In diesen Sitzungen soll mündlich über den Fortgang der Arbeit berichtet und anstehende Probleme diskutiert werden.
- Es ist ein Zeitplan für den Ablauf der Arbeit vorzulegen und mit dem Betreuer abzustimmen.
- Am Ende des zweiten Monats der Arbeit soll ein kurzer schriftlicher Zwischenbericht abgegeben werden, der über den Stand der Arbeit Auskunft gibt.
- Die Dokumentation ist vorzugsweise mit dem Textverarbeitungsprogramm FrameMaker oder Latex zu erstellen.

10. A. 5. Ergebnisse der Arbeit

Neben einem mündlichen Vortrag von 30 Minuten Dauer im Rahmen des Fachseminars Kommunikationssysteme sind die folgenden schriftlichen Unterlagen abzugeben:

- Ein Bericht (in 3 Exemplaren). Dieser enthält eine Darstellung der Problematik, eine Beschreibung der untersuchten Entwurfsalternativen, eine Begründung für die getroffenen Entwurfsentscheidungen, sowie eine Liste der gelösten und ungelösten Probleme. Eine kritische Würdigung der gestellten Aufgabe und des vereinbarten Zeitplanes runden den Bericht ab.
- Ein Handbuch zum fertigen System bestehend aus Systemübersicht und Implementationsbeschreibung,
- Eine Sammlung aller zum System gehörenden Software.
- Eine englischsprachige Zusammenfassung von 1 bis 2 Seiten, die einem Aussenstehenden einen schnellen Überblick über die Arbeit gestattet. Die Zusammenfassung ist wie folgt zu gliedern: (1) Introduction, (2) Aims & Goals, (3) Results, (4) Further Work.

10. A. 6. Literatur

[1] DEC True Cluster,
http://www.compac.com

[2] Cisco Virtual Router Redundancy Protocol (VRRP), RFC 2338,
http://www.faqs.org/rfcs/rfc2338.html

[3] Linux Virtual Server Project (LVS),
http://www.linuxvirtualserver.org/

[4] Linux HA project,
http://linux-ha.org/

[5] Microsoft Cluster Server,
http://www.microsoft.com/ntserver/ProductInfo/Enterprise/clustering/ClustArchit.asp

[6] SafeKit, White Paper,
http://www.evidian.com/safekit

[7] VERITAS Cluster Server,
http://www.veritas.com/us/products/clusterserver/

10. B. SRR Sourcen

Die Sourcen des Service Routing Redundancy Frameworks, wie auch dessen Vortrags-Presentation und Dokumentation, können auf der folgenden Website bezogen werden:

http://srr.digital-impact.ch/

Für allfällige Fragen und Anregungen bezüglich des Service Routing Redundancy Frameworks ist der Autor unter den folgenden E-mail Adressen erreichbar:

amir@digital-impact.ch

und

amir@guindehi.ch

10. C. Liste aller Figuren

10. D. Liste aller Tabellen

10. E. Referenzen

[1] Jürgen Kuri, c't - Magazin für Computertechnik, 7/2000,
“Report E-Commerce”,

[2] Microsoft.com: C&W, Uunet, Sprint; Ebay.com,
Exodus, Internap, Uunet

[3] Diplomarbeit Frank Baumgart,
“Skalierbare Hochverfügbarkeitslösungen mit Lastverteilung für E-Commerce”,
http://yeti.homeip.net/~frank/diplomarbeit/inhalt.html

[4] Internet FAQ Consortium, Feb 2000,
“RFC 1247 - OSPF Version 2”,
http://www.faqs.org/rfcs/rfc1247.html

[5] Internet FAQ Consortium, Feb 2000,
“Using HSRP for Fault-Tolerant IP Routing”
http://www.faqs.org/rfcs/rfc2281.html

[6] Lars Marowsky-Brée, Jana Jaeger, SuSE Withepapers,
“Hochverfügbare Systeme unter Linux”,
http://www.suse.de/de/whitepapers/ha/

[7] InfoWeek - Shoppers' Advisor: PC Guide [11/1999],
“Clustering & Load Balancing”,
http://www.infoweek.ch/archive/a_single.cfm?artikelx=3422

[8] Veritas,
“Veritas Cluster Server (VCS)”,
http://www.tim.de/Datenblattverzeichnis/Veritas/V-ClusterServer.pdf

[9] Network Essential Notes,
“OSI Model Layers”,
http://www.geocities.com/SiliconValley/Monitor/3131/ne/osimodel.html

[10] Enrique Vargas, Sun VluePrints OnLine,
“High Availability Fundamentals”,
http://www.sun.com/blueprints

[11] Heartbeat, Linux-HA, OpenSource,
“Heatbeat Package - A Heartbeat Protocol Implementation”,
http://www.linux-ha.org/download/

[12] Linux Virtual Server,
“Linux Virtual Server Project”,
http://www.linuxvirtualserver.org

[13] High Availabillity,
“Linux HA project”,
http://linux-ha.org

[14] Windows Clustering,
“Microsoft Cluster Server”,
http://www.microsoft.com/ntserver/ProductInfo/Enterprise/clustering/ClustArchit.asp

[15] White Paper,
“SafeKit”,
http://www.evidian.com/safekit

[16] OSPFAPI, Ralph Keller, ETH Zürich,
“An API for the Zebra OSPF daemon”,
http://www.tik.ee.ethz.ch/~keller/ospfapi

[17] Zebra Project,
“The Zebra Routing Daemon”,
http://zebra.org

[18] Internet FAQ Consortium,
“Host Anycasting Service”,
http://www.faqs.org/rfcs/rfc1546.html

[19] Masahiko Endo,
“Zebra OSPF Opaque Module”,
mailto:endo@suri.co.jp

[20] Internet FAQ Consortium, Nov 1993,
“RFC 1546 - Host Anycasting Service”,
http://www.faqs.org/rfcs/rfc1546.html

[21] Internet FAQ Consortium, Jul 1998,
“RFC 2373 - IP Version 6 Addressing Architecture”,
http://www.faqs.org/rfcs/rfc2373.html

[22] Diana Katabi, John Wroclawski, Laboratory for Computer Science, MIT
“A Framework for Global IP-Anycast (GIA)”,
http://www.ana.lcs.mit.edu/dina/pub/Katabi-350.pdf

[23] Dong Xuan, Weijia Jia, IEEE Distributed Systems, Vol. 11, No. 6, Jun 2000,
“A Routing Protocol for Anycast Messages”,
http://anyserver.cityu.edu.hk/weijia/publication/IEEE_TPDS_Routing_Anycast1.pdf

[24] Paul Judge, Mostafa Ammar, College of Computing, Georgia Institute of Technology,
“Gothic: A Group Access Control Architecture for Secure Multicast and Anycast”,
http://citeseer.nj.nec.com/judge02gothic.html