Hochschule für Technik und Wirtschaft Dresden

Fakultät Informatik/Mathematik

Bachelorarbeit

im Studiengang Allgemeine Informatik

Thema: Netzwerksimulation mittels Usermode-Linux

eingereicht von: Samuel Knobloch

eingereicht am: 13.07.2012

Betreuer: Prof. Dr.-Ing. Jörg Vogt

Inhaltsverzeichnis

Glossar

AS
AS - Autonomous System, ein eigenständiger Netzbereich 7, 30

COW
Copy-On-Write ist ein Modus zur Nutzung von Filesystem-Images durch Simulatoren 25

Daemon
Ein im Hintergrund laufender Systemdienst 4

Home-Verzeichnis
Das normale Arbeitsverzeichnis. Für User ist dies */home/<username>/*, für root */root/* 17

KVM
Kernel-based Virtual Machine, ist eine Linux-Kernel-Infrastruktur für Virtualisierung und seit Version 2.6.20 im Kernel vorhanden 15

MAC-Adresse
Die Hardware-Adresse eines Netzwerkadapters zur eindeutigen Identifizierung im Netzwerk 23, 24

Perl
Freie, plattformunabhängige Skriptsprache 21

QEMU
QEMU ist eine virtuelle Maschine zur Emulierung der gesamten Hardware eines Computers mittels KVM 15

sudo
Ein Befehl, mit dem ein normaler User Befehle als root ausführen kann 1

VNUML
Virtual Network User Mode Linux, Simulationsumgebung für virtuelle Netzwerke 21

XML
Extensible Markup Language, Auszeichnungssprache zur Darstellung von hierarchisch strukturierten Daten 21

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Netzwerke, insbesondere Rechnernetzwerke, sind ein wichtiger Bestandteil der heutigen Kommunikation. Die Vernetzung und Verbesserungen der Technik haben viele Dinge vereinfacht, beispielsweise das Versenden von Daten von einem Ort zum anderen. Dazu ist es notwendig, dass der Sender weiß, wohin er seine Daten schicken möchte und sicher sein kann, dass sie auch ankommen. Dies erfordert im Vorfeld einige Planung und Konzeption des Netzes.

Der erste Teil dieser Arbeit wird sich mit Lösungen und Protokollen für die Wegfindung in und zwischen Netzwerken, genannt Routing, beschäftigen. Eingegangen wird dabei auf verschiedene aktuelle Routing-Protokolle, deren Klassifizierung sowie die Bestimmgung der Güte von Routen anhand von Metriken.

Im zweiten Teil wird sich mit Netzwerk-Simulation auseinander gesetzt. Dabei wird darauf eingegangen, was eine Simulation ist und Lösungen zur Virtualisierung vorgestellt. Weiterhin werden zwei Netzwerk-Simulatoren für *User Mode Linux* näher vorgestellt und der Umgang anhand von Beispielen erklärt.

1.2 Bemerkungen

Im Rahmen dieser Arbeit wurde Debian, vergleichbar mit Ubuntu, als Host-System eingesetzt. Auf nicht-Debian-Systemen können Pfadangaben, Datei-Syntax sowie Tools zur Installation von weiteren Paketen variieren.
Desweiteren ist es an einigen Stellen erforderlich, als *root* oder mittels *sudo* Befehle auszuführen.

```
root@pc:/# <command>
user@pc:~$ sudo <command>
```

Listing 1.1: Als root ausführen

1.3 Aufgabenstellung

In Form von Praktikumsversuchen soll den Studenten der Informatikausbildung anhand einer Netzwerksimulation eine praktische Einführung in die aktuellen Netzwerktechnologien gegeben werden. Als Basis für die Netzwerksimulatoren soll Ubuntu-Linux dienen.

Teilaufgaben

- Übersicht über aktuelle Routingprotokolle
- Vergleich aktueller Netzwerksimulatoren auf Nutzbarkeit
- Auswahl eines geeigneten Netzwerksimulators
- Aufbau von für die Ausbildung geeigneter Netzkonfigurationen basierend auf dem ausgewählten Simulator
- Erstellung von passenden Praktikumsanleitungen /–lösungen
- Dokumentation aller relevanter Konfigurationsdaten

2 Routing

2.1 Überblick

In der Telekommunikation wird das Festlegen von Wegen für Nachrichtenübermittlungen in Netzwerken als Routing bezeichnet. Routing selbst bezeichnet dabei die Bestimmung des Gesamtweges für die Übermittlung durch das Netzwerk. Die Entscheidung an einzelnen Knoten, wohin diese ein Nachricht weitervermitteln, nennt man Forwarding.

Im Rahmen dieser Arbeit wird das paketvermittelnde Routing betrachtet. Bei diesem werden logisch adressierte Datenpakete aus ihrem Heimnetz heraus weitergeleitet bis hin zu ihren Zielnetzen. Auf dem Weg hin zum Ziel können mehrere unterschiedliche Zwischennetze liegen.

Je nach Art und Größe des zu verwaltendem Netzwerkes kommen verschiedene Routing-Verfahren zur Anwendung. Das *statische Routing* eignet sich besonders für kleine, übersichtliche Netzwerke, die sich einfach konfiguieren lassen. Für große Netzwerke mit komplexen Topologien oder Netzwerken, die häufigen Änderungen unterworfen sind, eignen sich Verfahren aus dem Bereich des *dynamischen* oder auch *adaptiv* genannten Routing. Auf diese beiden Verfahren und einige wichtige Routing-Protokolle wird in den nächsten Abschnitten genauer eingegangen.

Ein weiteres Verfahren ist das *Flooding*. Pakete werden bei diesem Verfahren via Broadcast über alle ausgehenden Leitungen an alle erreichbaren Ziele verteilt. Die Nachteile dieses Verfahrens sind viele Paketduplikate, ein hoher Aufwand bei dem Empfänger sowie eine starke Belastung der einzelnen Knoten. Dieses Verfahren arbeitet ohne *Routing-Tabelle*.

2.2 Metriken

In Netzwerken kann es vorkommen, dass Router verschiedene Wege zu dem gleichen Ziel kennen. Damit ein Routing-Protokoll aus den vorhandenen Wegen den besten und zu bevorzugenden auswählen kann, muss es jeden Weg qualitativ bewerten können. Metriken sind Parameter, welche zur Bewertung der verschiedenen Wege und deren Verbindungsqualität genutzt werden.

Eine Metrik ist ein numerisches Maß für die Güte einer Verbindung. Genutzt werden dabei unter anderem die folgenden Parameter:

- *Bandwidth* - Die Verbindung zum Ziel mit der höchsten verfügbaren Bandbreite wird bevorzugt.
- *Delay* - Gibt die Verzögerungszeit an, die ein Paket zum Durchlaufen des Pfades bis zum Ziel benötigt.
- *Hop count* - Gibt an, wieviele Knoten bis zum Ziel vorhanden sind. Der Weg mit den wenigsten Zwischenknoten wird dabei bevorzugt.
- *Load* - Gibt an, wie hoch die Auslastung einer Verbindung ist.
- *Reliability* - Gibt die Ausfallrate eines Links anhand der schon aufgetretenen Ausfälle und Übertragungsfehler an.

Die am häufigsten genutzten Parameter sind dabei die *Bandbreite*, *Hop count* und die *Verbindungsqualität*. Je nach genutzten Parametern wird entschieden, ob ein hoher oder ein niedriger Wert für eine gute Gesamtqualität einer Route steht.
Als Beispiel für einen Parameter, für den hohe Werte gut sind, sei die *Bandbreite* genannt. Bei den Parametern *Hop count* oder *Load* hingegen müssen die Werte gering sein, um für eine gute Qualität zu stehen.

2.3 Routing table und Forwarding table

Die *Routing-Tabelle* ist eine durch einen *Daemon* oder manuell erstellte Liste der im Gesamtnetz erreichbaren Ziele und kann durch den Informationsaustausch durch Routing-Protokollen aktualisiert werden. Ein Eintrag besteht aus der IP des Subnetzes, dem Gateway und einer Metrik für diese Route. Die Metrik muss nicht angegeben werden, wenn für ein Ziel nur eine Route existiert. Weiterhin kann als Information noch angegeben sein, welches Device für die Route genutzt werden soll. Nachfolgend ein Beispiel für eine Routing-Tabelle.

Network	Gateway	Subnet mask	Interface	Metric
127.0.0.0	127.0.0.1	255.0.0.0	127.0.0.1	1
192.168.5.0	192.168.5.26	255.255.255.0	192.168.5.26	10
192.168.5.100	127.0.0.1	255.255.255.255	127.0.0.1	10
192.168.5.255	192.168.5.26	255.255.255.255	192.168.5.26	10

Router können mehrere Routing-Tabellen besitzen. Dies ist der Fall, wenn auf einem Router mehrere verschiedene Routing-Protokolle zum Einsatz kommen. Jedes Protokoll erstellt dabei eine eigene Tabelle. Die Tabellen unterscheiden sich meist nur anhand der verwendeten Metriken. Da das Durchsuchen mehrerer Tabellen zur Wegfindung nicht sehr effizient ist, können diese zu einer kleineren kompakten Tabelle zusammengefasst werden. Meist wird dies durch einen Routing-Daemon übernommen. Diese Tabelle nennt man *forwarding table*. Sie ist optimiert für schnelles Durchsuchen und effizientes Updaten. Das Weiterleiten eines Paketes zum Ziel wird dann anhand der *forwarding table* entschieden.

2.4 Static Routing

Das statische Routing ist ein Verfahren zur Wegfindung durch Router in kleinen Netzwerken. Jeder Router besitzt dabei eine eigene, manuell verwaltete Routing-Tabelle, in die das automatische Einfügen von neuen Einträgen nur bedingt möglich ist, beispielsweise über *SNMP* (Simple Network Management Protocol). Im Fehlerfall, bei Änderung von IP-Adressen oder Netzerweiterungen müssen die Einträge der Routing-Tabellen jeweils von Hand angepasst werden.

2.4.1 Vorgehen

Für jeden Router müssen alle Einträge der Routing-Tabelle per Hand vorgenommen werden. Jede Zeile der Routing-Tabellen enthält hier ein erreichbares Ziel mit der Information, über welchen Ausgang es erreicht werden kann. Das gleiche Ziel kann dabei auch über verschiedene Ausgänge erreicht werden. Zusätzlich ist eine Gewichtung angebbar, welche die Wahrscheinlichkeit der Wahl dieser Route widerspiegelt.

Um eine neue Route hinzuzufügen, nutzt man den Console-Befehl *route*.
Für den einfachsten Fall benötigt man die IP-Adresse des Zielnetzwerkes, die *Subnet mask* und den Ausgang (Device), über den das Zielnetzwerk erreichbar ist.

```
route add -net 192.168.10.0 netmask 255.255.255.0 dev eth0
```

Listing 2.1: Statische Route hinzufügen

Eine weitere Möglichkeit ist das direkte Editieren der Konfigurationsdatei.

```
1  # Datei /etc/network/interfaces
2  auto eth0
3  iface eth0 inet static
4    address 192.168.5.26
5    netmask 255.255.255.0
6    network 192.168.5.0
7    broadcast 192.168.5.255
8
9    ## static route ##
10   post-up route add -net 192.168.10.0 netmask 255.255.255.0 gw
         192.168.5.1
11   pre-down route del -net 192.168.10.0 netmask 255.255.255.0 gw
         192.168.5.1
```

Listing 2.2: Hinzufügen über Konfigurationsdatei

Der *post-up*-Befehl fügt beim Aktivieren des Interface die Route zur Routing-Tabelle hinzu, der *pre-down*-Befehl entfernt diese beim Deaktivieren wieder. Neu ist hierbei der Parameter *gw*, über welchen man das Gateway angibt.

2.5 Dynamic Routing

Als *dynamisches* oder auch *adaptives Routing* bezeichnet man das Verfahren, welches zur Wegfindung und dem Informationsaustausch zwischen Routern in großen Netzwerken mit komplexen Topologien angewandt wird. Der Informationsaustausch zwischen Routen über erreichbare Zielen wird über *Routing-Protokolle* realisiert. Jeder Router entscheidet anhand der erhaltenen Informationen, welche Route zu einem bestimmten Ziel die von seiner Position im Netzwerk aus beste ist und fügt diese seiner Routing-Tabelle hinzu.
Ein Vorteil gegenüber dem *statischen Routing* ist, dass sich die Einträge der *Routing-Tabellen* automatisch anpassen, sobald es Änderungen im Netzwerk gibt. Fehler können somit schnell erkannt und umgangen werden. Ein über dynamisches Routing verwaltetes Netzwerk skaliert gut und ist aufgrund der Fehlererkennung und Fehlerbehebung sehr robust.

2.5.1 Einordnung der Protokolle

Die für dynamisches Routing genutzte Protokolle lassen sich kategorisieren. Protokolle können dabei auch mehreren Bereichen zugeordnet sein.
Die erste Unterscheidung wird zwischen *Classful* und *Classless Routing* getroffen und bezeichnet die Art der Adressierung von Subnetzen. Bei *Classful Routing*-Protokollen wird anhand der IP-Adresse die Netzklasse und dadurch das Subnetz unterschieden. Es gibt die folgenden Netzklassen zur Unterteilung in Bereiche beziehungsweise Subnetze:

Netzklasse	Startadresse	Endadresse
Class A	0.0.0.0	127.255.255.255
Class B	128.0.0.0	191.255.255.255
Class C	192.0.0.0	223.255.255.255
Class D	224.0.0.0	239.255.255.255
Class E	240.0.0.0	255.255.255.255

Das *Classful Routing* (CR) ist mittlerweile veraltet und wurde durch *Classless Inter-Domain-Routing* (CIDR) ersetzt. Während bei *Classful Routing* nur mit den Netzklassen gearbeitet wurde, teilt man bei CIDR den gesamten Adressraum in flexible Netze ein. Die Größe eines Netzwerkes ist nicht mehr nur durch die IP-Adresse ersichtlich, es wird eine *Subnet mask* benötigt.
Protokolle, die zu CR zählen, übermitteln keine *Subnet mask* in den Informationen zur Route. Zu CIDR zählende Protokolle übermitteln die *Subnet mask* zusammen mit der Adresse des Netzes. Fast alle aktuell genutzten Routing-Protokolle sind CIDR-Protokolle.

Netzklasse	Subnet mask	Beispiel CIDR-Notation
Class A	255.0.0.0	10.0.0.0/8
Class B	255.255.0.0	155.26.0.0/16
Class C	255.255.255.0	192.168.5.0/24

Eine zweite Unterscheidung ist anhand des Anwendungsbereiches der Protokolle möglich. Man unterscheidet zwischen netzintern (*Interior Gateway Protocol*, IGP) und netzextern oder netzverbindend (*Exterior Gateway Protocol*, EGP). IGP werden innerhalb von autonomen Netzen (AS) genutzt, EGP zur Kommunikation zwischen verschiedenen autonomen Netzen. Die meisten Routing-Protokolle zählen zur Klasse der IGP.

Weiterhin werden Protokolle anhand der eingesetzten Verfahren zur Ermittlung der besten Routen zu Zielen unterschieden. Eingeteilt werden Protokolle in die Verfahren *Distance Vector* und *Link-State*, auf diese wird in den nächsten Abschnitten näher eingegangen.

2.5.2 Distance Vector Protokolle

Distance Vector-Protokolle sind eine von zwei Hauptklassen von Routing-Protokollen. Sie nutzen zur Berechnung der kürzesten und besten Wege den *Bellman-Ford*-Algorithmus, aber auch der *Ford-Fulkerson*-Algorithmus findet Anwendung. Eine Route besteht hierbei aus den beiden Informationen Distanz und Richtung. Die Richtung ist der nächste Router auf dem Weg oder das Ausgangsinterface, die Distanz ist ein durch Metriken definierter Vektor, beispielsweise der *Hop count*.

Jeder Router sendet seine Routing-Informationen nur an seine direkten Nachbarn weiter und hat dadurch keine Kenntnisse über den Aufbau des Gesamtnetzes, sondern nur über seine direkte Umgebung. Bekommt ein Router von seinem Nachbarn Informationen über ein bekanntes Ziel, prüft und ersetzt dieser seine eigenen Informationen in der Routing-Tabelle, wenn die neue Route eine bessere ist als die aktuell vorhandene. Neue Routen werden direkt in die Routing-Tabelle eingetragen.

Das *Distance Vector*-Verfahren hat einige Nachteile und Schwächen.
Informationen, die ein Router von seinen Nachbarn erhält, sind von ihm nicht prüfbar und können schon nicht mehr aktuell sein, beispielsweise wenn ein entfernt liegender Knoten ausgefallen ist. Weiterhin dauert es einige Zeit, bis eine Information an alle Knoten im Netz verteilt wurde, es konvergiert nur sehr langsam. Da ständig viele Routing-Informationen zwischen den Routern ausgetauscht werden müssen, teilweise komplette Routing-Tabellen, gibt es einen recht hohen Overhead. Dieser Overhead kann die verfügbare Bandbreite des Netzes in nicht unbeträchtlichem Maße reduzieren.
Ein weiteres Problem ist die Möglichkeit der Schleifenbildung und das Verloren gehen von Paketen in diesen. Schleifen können entstehen, wenn ein Router ausfällt, seine direkten Nachbarn ihn somit nicht mehr erreichen können und den Eintrag aus ihren Routing-Tabellen löschen. Dies wird weitergegeben, aber gleichzeitig kann

ein Router existieren, welcher den ausgefallenen Knoten noch kennt und dies seinen Nachbarn immer wieder neu mitteilt. Dadurch wird gleichzeitig der Eintrag vom ausgefallenen Router aus den Tabellen gelöscht, im nächsten Schritt jedoch erneut hinzugefügt.

Routing-Protokolle nach diesem Verfahren eigenen sich beispielsweise für den Einsatz in einfachen Netzwerken ohne spezielle Hierarchien.

2.5.2.1 Routing Information Protocol

Das *Routing Information Protocol* (RIP) ist eines der ersten dynamischen Routing-Protokolle. Es eignet sich für kleine homogene Netzwerke und arbeitet mit dem *Bellman-Ford*-Algorithmus. Als Metrik verwendet RIP nur den *Hop count*.

Allgemeines Vorgehen

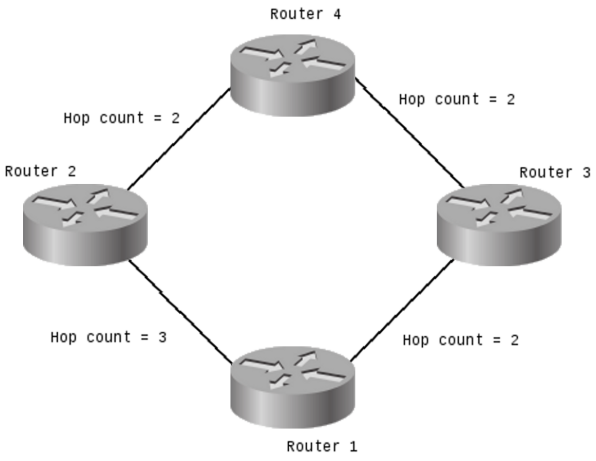
RIP verschickt nur die nötigsten Informationen, sogenannte Distanz-Tabellen, in Routing-Paketen. Durch den Austausch von Informationen lernt jeder Knoten im Netzwerk über Zeit alle möglichen Ziele kennen. Dabei wird wie folgt vorgegangen:

Jeder Knoten kennt nur seine Nachbarn und tauscht mit diesen Informationen aus. Dadurch erhält er weitere Einträge, die er seiner eigenen Routing-Tabelle hinzufügt. Existiert für eine erhaltene Information schon ein Eintrag in der Routing-Tabelle, so wird anhand der Metrik geprüft, welche Route die bessere zum Ziel ist. Je niedriger dabei der Wert, um so besser ist die Route. RIP kann für jedes Ziel nur einen Eintrag in der Routing-Tabelle besitzen, daher überschreiben bessere Routen die vorhandenen. Ein wesentlicher Nachteil von RIP ist die langsame Konvergenz, wenn ein Knoten im Netzwerk ausfällt.

Beispiel

Folgenes Beispiel verdeutlicht das Befüllen der Routing-Tabelle an Router 1.

Abbildung 2.1 Einfaches Netzwerkbeispiel für RIP



- **Schritt 1**: Router 1 kennt nur Router 2 und Router 3, Router 4 ist unbekannt.
  Distanz zu Router 2: 3 Hops
  Distanz zu Router 3: 2 Hops
- **Schritt 2**: Alle Router schicken ihren Nachbarn Updates.
- **Schritt 3**: Die Router prüfen die erhaltenen Updates mit ihrer Routing-Tabelle und fügen nicht vorhandene Einträge hinzu.
  Distanz Router 2: 3 Hops
  Distanz Router 3: 2 Hops
  Distanz Router 4: 4 Hops
  Distanz Router 4: 5 Hops
- **Schritt 4**: Da es zwei Wege zum Router 4 gibt, wird der bessere von beiden in die Routing-Tabelle übernommen und der andere Weg verworfen.
  Distanz Router 2: 3 Hops
  Distanz Router 3: 2 Hops
  Distanz Router 4: 4 Hops

Routing Information Protocol Version 1

Die Version 1 von RIP wurde 1988 veröffentlicht. Es ist ein *Classful Routing*-Protokoll und verschickt seine Routing-Updates via Broadcasting an alle benachbarten Router. Alle Subnetze, die RIP in Version 1 nutzen, müssen von der gleichen Größe sein. Weiterhin gibt es in dieser Version keine Möglichkeiten der Router-Authentisierung.

Routing Information Protocol Version 2

RIP wurde weiterentwickelt und 1993 in Version 2 veröffentlicht. Diese Version ist im Gegensatz zu RIPv1 ein *Classless Inter-Domain-Routing*-Protokoll und unterstützt nun die Verwaltung von Subnetzen unterschiedlicher Größe. Hinzugefügt wurde eine *MD5-Authentisierung* zur Verbesserung der Sicherheit. RIPv2 versendet die Update-Informationen nicht mehr als Broadcast, sondern als Multicast. Eine weitere Neuerung ist das Markieren von Routen zur Unterscheidung, ob diese netzintern oder ausserhalb des Netzes über ein EGP erreichbar sind.

Routing Information Protocol Next Generation

Die neueste Version von RIP ist die Erweiterung *Next Generation* und wurde 1997 veröffentlicht. Sie erweitert RIPv2 um Unterstützung von IPv6 und bringt einige Neuerungen. RIPng nutzt zur Authentisierung nun *IPSec* und ist somit nicht mehr abwärtskompatibel zu RIPv1, weiterhin wurden die in Version 2 hinzugefügten Tags zur Markierung wieder entfernt.

2.5.2.2 Enhanced Interior Gateway Routing Protocol

Das *Enhanced Interior Gateway Routing Protocol* (EIGRP) ist die Weiterentwicklung des *Interior Gateway Routing Protocol* (IGRP), welches von Cisco als Verbesserung von RIPv1 entwickelt wurde. IGRP ist ein *Classful Routing*-Protokoll, hingegen die Weiterentwicklung EIGRP zu den *Classless Routing*-Protokollen zählt. EIGRP ist kein reines *Distance Vector*-Protokoll, sondern erweitert dieses Verfahren um Merkmale von *Link-State*-Protokollen. Die Vorteile daraus sind unter anderem eine schnellere Konvergenz des Netzes und die Verhinderung von Schleifen.

Dieses Protokoll nutzt drei verschiedene Tabellen zur Abspeicherung der Informationen. Die *Neighbor table* (Nachbarschafts-Tabelle) enthält Informationen über die benachbarten Router. Neue von benachbarten Routern empfangene Routen werden in der *Topology table* (Topologie-Tabelle) abgespeichert. Sie ist nicht die eigentliche Routing-Tabelle. Die *Routing table* enthält die eigentlichen Routen zu allen erreichbaren Knoten und wird über die Informationen aus der Topologie-Tabelle befüllt. EIGRP nutzt für jede Route sechs Metriken: verfügbare *Bandbreite*, die *Auslastung*, *Verzögerungszeit*, *Ausfallrate*, *MTU* und den *Hop count*.

Entgegen anderer *Distance Vector*-Protokolle arbeitet (E)IGRP nicht mit dem *Bellman-Ford*-Algorithmus, sondern dem *Diffusing Update*-Algorithmus (DUAL). Dieses Verfahren verhindert die Schleifenbildung durch Redundanz und passt sich dynamisch an Änderungen der Netzwerktopologie an. Dazu analysiert der Algorithmus die Einträge der Topologie-Tabelle und erstellt zu einem Ziel mehrere redundante Routen. Die Route mit den geringsten Kosten ist die Primärroute und wird hauptsächlich genutzt. Auf Sekundärrouten wird zugegriffen, wenn die Hauptroute ausgefallen oder überlastet ist.

2.5.2.3 Weitere Protokolle

Ein weiteres Vektor-Protokoll ist das *Border Gateway Protocol* (BGP). Es ist kein *Distance Vector*-Protokoll, sondern ein *Path-Vector*-Protokoll und zählt zu den *Classless Inter-Domain-Routing*-Protokollen. Dieses Protokoll verbindet autonome Systeme untereinander und ist aktuell das einzige noch eingesetzte EGP. Das BGP kann aber auch innerhalb eines Netzes genutzt werden. BGP nutzt zur Kommunikation mit seinen Nachbarn, sogenannten Border-Routern, vier Typen von Nachrichten, die mit einem Paket verschickt werden können.

- **OPEN**
  Diese Nachricht wird zum Aufbau einer Verbindung genutzt und muss vom antwortenden Router mit einem KEEPALIVE beantwortet werden.
- **KEEPALIVE**
  Zur Aufrechterhaltung und Bestätigung der Verbindung zwischen Routern wird die KEEPALIVE-Nachricht verschickt.
- **UPDATE**
  Durch eine UPDATE-Nachricht werden Änderungen der Pfade mitgeteilt.
- **NOTIFICATION**
  Um eine Verbindung zu beenden, nutzt man diesen Nachrichtentyp. Weiterhin können Fehler- und Statuscodes weitergegeben werden.

2.5.3 Link-State Protokolle

Die zweite Klasse von Routing-Protokollen sind die *Link-State*-Protokolle. Sie nutzen zur Berechnung den *Dijkstra*-Algorithmus. Im Gegensatz zu den meisten *Distance Vector*-Protokollen senden Router nur Informationen über die direkt an sie angeschlossenen Netzwerke und Devices weiter. Daraus lässt sich ein gutes Bild der Topologie des Netzes um den Router herum ableiten, weiterhin kann eine bessere Aussage darüber getroffen werden, welche Route zu einem bestimmten Ziel die beste ist.

Ein weiterer Vorteil von *Link-State*-Protokollen ist der geringe Overhead. Das bedeutet, Informationen werden zwischen Routern nur dann ausgetauscht, wenn es notwendig ist. Nach dem Aufbau des Netzes werden, insofern es keine Änderungen oder Fehlerfälle gibt, keine Pakete mit schon bekannten Informationen verschickt. *Distance-Vector*-Protokolle hingegen senden ihre Informationen in Intervallen.

Routing-Protokolle nach diesem Verfahren eignen sich für den Einsatz in großen, komplexen Netzwerken und dann, wenn ein schneller Netzaufbau erforderlich ist.

2.5.3.1 Open Shortest Path First Protocol

Das *Open Shortest Path First Protocol* (OSPF) ist ein Routing-Protokoll, welches das Link-State-Verfahren nutzt, zur Klasse der IRP gehört und daher nur zur Kommunikation innerhalb eines autonomen Systemes verwendet wird. Entwickelt wurde es als Ersatz für RIP und zählt zu den *Classless Routing Protocol*. Die erste Version von OSPF wurde 1988 veröffentlicht, 1998 folgte OSPFv2 und 2008 OSPFv3 mit Unterstützung von IPv6.

Einer der wesentlichen Vorteile von OSPF zu anderen Protokollen ist, dass Fehler im Netzwerk sehr schnell erkannt und behoben werden. Dies führt zur Vermeidung von Schleifen (*Loops*). OSPF skaliert besonders in großen Netzwerken sehr gut, bietet Möglichkeiten zur Lastverteilung auf Routen und der Authentisierung und Synchronisation der Routen eines Netzes über einen sogenannten designierten Router. Der *Designated Router* (DR) reduziert die Netzlast, da OSPF Routing-Informationen via *Flooding* verteilt und ist für die Generierung von Updates verantwortlich. Fällt der DR aus, springt ein Backup-Router (*Backup Designated Router*, BDR) ein und übernimmt die Aufgaben. Welche Router im jeweiligen Subnetz die Rollen von DR und BDR übernehmen, wird beim initialen *Hello* anhand der Priorität und IP aller Router entschieden. Im Normalfall ist die Priorität aller Router "*1*", daher werden die Router mit den höchsten IPs als DR und BDR gewählt. Das Prinzip hinter dem Einsatz von DR ist, dass jeder Router nicht mehr seine Informationen an alle Knoten im Netzwerk schickt (*Flooding*), sondern nur noch an den DR. Dieser sendet Routing-Informationen via Multicast nur an bestimmte Router und entlastet dadurch das Gesamt-Netz.

Zur Kommunikation und dem Informationsaustausch besitzt OSPF fünf verschiedene Paket-Typen, welche kurz erläutert werden.

- **Hello**
  Dieser Typ wird genutzt, um eine Verbindung mit anderen Routern aufzubauen und zu verwalten.
- **Database description (DBD)**
  Dieses Paket enthält eine verkürzte Liste der *Link-state Datenbank* des sendenden Routers und wird vom empfangenden Router zur Prüfung mit der eigenen Datenbank verwendet.
- **Link-state request (LSR)**
  Mit diesem Typ können weitere Informationen über einen bestimmten Eintrag der DBD angefordert werden.
- **Link-state update (LSU)**
  Dieses Paket übermittelt durch LSR angeforderte Informationen an den fragenden Router.
- **Link-state acknowledgment (LSAck)**
  Um den Erhalt von Informationen zu bestätigen, nutzt man diesen Paket-Typ.

2.5.3.2 Intermediate System to Intermediate System

Intermediate System to Intermediate System (IS-IS) ist ein *Link-State*-Protokoll und nutzt zur Ermittlung des besten Pfades ebenso wie OSPF den *Dijkstra*-Algorithmus. Von der Funktionsweise ist IS-IS ähnlich zu OSPF. Es nutzt Hello-Pakete, um sich mit anderen Routern im Netzwerk bekannt zu machen und unterstützt die Authentisierung von Routing-Updates. Ursprünglich wurde IS-IS nicht für die Nutzung mit dem *Internet Protocol* entwickelt, sondern für *Connectionless Network Protocol* (CLNP). Erst durch Erweiterungen wurde die Nutzung mit IPv4 und IPv6 möglich.

Es gibt drei Arten von IS-IS-Routern. Die sogenannten *Level 1*-Router sind für die Kommunikation und den Informationsaustausch innerhalb eines autonomen Systems verantwortlich und können nur mit anderen *Level 1*-Routern kommunizieren. Die *Level 2*-Router sind für die Verbindung zwischen verschiedenen autonomen Systemen zuständig und können auch nur mit Routern des gleichen Level kommunizieren. Die dritte Art ist die Kombination aus beiden Leveln, *Level 1-2*-Router. Diese werden an Knoten genutzt, die Informationen zwischen autonomen Systemen und dem eigenen Netz vermitteln.

3 Netzwerk-Simulation

3.1 Simulation und Simulator

Die Simulation von Netzen ist ein wichtiger Bestandteil im Bereich der Kommunikation und Netzwerktechnologie. Mit ihrer Hilfe können komplexe Topologien nachgebildet und ihr Verhalten untersucht werden.

Definition Simulation

> Die Simulation von Rechnernetzen ist ein Mittel zur schnellen und kostengünstigen Untersuchung und Bewertung von Protokollen und somit ein unersetzliches Werkzeug für die Netzwerkforschung. Während analytische Betrachtungen häufig mit der Komplexität der Szenarien und Feldversuche mit dem Hardware-Aufwand und den damit verbundenen Kosten kämpfen, kann durch Simulation der Parameterraum hinsichtlich Netzwerktopologien, Kommunikationsmustern und Abhängigkeiten zu anderen Protokollen effizient erforscht werden. [Fü; Li10]

Die Wikipedia definiert Simulation allgemein:

> Die Simulation oder Simulierung ist eine Vorgehensweise zur Analyse von Systemen, die für die theoretische oder formelmäßige Behandlung zu komplex sind. [Wika]

Anwendung findet Netzwerk-Simulation beispielsweise bei der Planung von komplett vernetzten Gebäuden oder Gebieten, um Fehler vermeiden zu können.

Definition Netzwerk-Simulator

> A network simulator is a piece of software or hardware that predicts the behavior of a network, without an actual network being present. A network simulator is a software program that imitates the working of a computer network. In simulators, the computer network is typically modelled with devices, traffic etc. and the performance is analysed. [Wikb]

Ein Simulator ist also eine Hardware oder Software, welche das Verhalten von Netzwerken nachbilden kann und zur Analyse von beispielsweise dem Verhalten von Routing-Protokollen genutzt wird. Nicht zu verwechseln ist dies mit einem *Emulator*, welcher die Funktionalität eines Systems nachbildet.

3.2 Netzwerk-Simulatoren im User Mode Linux

Im Rahmen dieser Arbeit wird sich mit den beiden Simulatoren *Netkit* und *Virtual Networks over linuX* (VNX) auseinandergesetzt. Netzwerk-Simulatoren können Komponenten wie Router oder Switches simulieren und damit verschiedene Netzaufbauten und Szenarien realisieren. Der Vorteil gegenüber dem tatsächlichen Aufbau eines Netzwerkes ist in erster Linie ein Kostenfaktor, denn schon während der Simulation können beispielsweise Planungsfehler im Netzwerk gefunden und behoben werden. Auch ist testbar, ob sich ein bestimmtes Routing-Protokoll für den Einsatz in diesem Netz eignet. In einem realen Netzwerk würden solche Versuche und Fehler Zeitaufwand zur Behebung und eventuelle Mehrkosten für weitere Komponenten bedeuten.

3.2.1 Virtualisierung

Um eine Umgebung oder ein System zu simulieren bedarf es der Virtualisierung. Man unterscheidet dabei verschiedene Ansätze. Bei *Hardware Emulation* wird versucht, eine bestimmte Hardware oder ein System nachzubilden. Die *Hardware Virtualization* geht einen anderen Weg. Sie erlaubt die Aufteilung von Ressourcen und den Zugriff auf die Hardware eines Systemes für Gastsysteme. Emuliert werden hierbei nur system-kritische Zugriffe wie System-Calls.

Im Rahmen dieser Arbeit werden auf *Hardware Virtualization* aufbauende Netzwerk-Simulatoren genutzt. Es gibt verschiedene Möglichkeiten und Lösungen zur Virtualisierung wie *Xen*, *VirtualBox*, *KVM* oder *UML*.

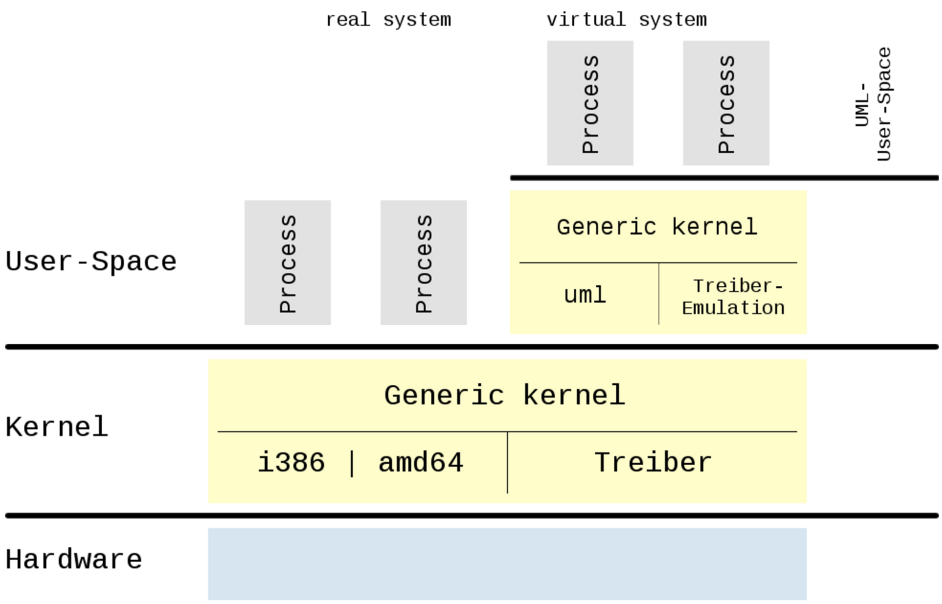
3.2.1.1 Kernel-based Virtual Machine

Eine Möglichkeit der Virtualisierung ist *Kernel-based Virtual Machine*. KVM ist eine Erweiterung des Linux-Kernels um Module, welche Infrastrukturen zur Emulation bereitstellen. Um KVM nutzen zu können, muss der eingesetzte Prozessor Erweiterungen für Virtualisierung implementiert haben. KVM selbst emuliert nicht. Für die Emulation von virtuellen Maschinen sorgt die Software Quick EMUlator, kurz QEMU. Sie kann die komplette Hardware eines PC emulieren und bietet unter Linux, BSD und Mac OS X auch Unterstützung für UML.

3.2.1.2 User Mode Linux

UML ist eine weitere Möglichkeit der Virtualisierung welche es ermöglicht, einen Linux-Kernel als normalen Prozess zu starten. Der Prozess läuft im User-Space getrennt vom eigentlichen Kernel und erhält vom Host-System Ressourcen zugewiesen. Der normale Aufbau besteht, vereinfacht dargestellt, aus der Hardware-Schicht, darauf aufsetzend dem Kernel oder Kernel-Space und dann dem User-Space. Ein UML-Prozess im User-Space fügt sich selbst eine weitere Ebene hinzu, seinen eigenen User-Space. Folgende Grafik veranschaulicht dies.

Abbildung 3.1 Vereinfachte Darstellung der Ebenen



UML-Kernel werden gebaut wie normale Kernel und lassen sich von diesen daher kaum unterscheiden. Der UML-Kernel bietet eigene Scheduler und Verwaltung von Speicher, Prozessen und Dateisystemen. Das Host-System sorgt nur für die Weitergabe der realen Ressourcen an das Gast-System.

3.2.2 Netkit

Netkit ist ein auf *Open Source*-Software basierender Netzwerk-Emulator. Mit relativ wenig Aufwand lassen sich verschiedene Netzwerk-Strukturen aufbauen und simulieren. Das simulierte Netzwerk ist dabei unabhängig vom Netzwerk des Host-Systems.

Netkit nutzt UML und simuliert jedes Netzwerk-Device als eine eigene virtuelle Maschine, welches über einen virtuellen Hub des Host-Systems mit anderen VMs verbunden ist. Theoretisch kann der virtuelle Hub auch genutzt werden, um auf externe Netzwerke zuzugreifen, beispielsweise für eine Internetverbindung. Die Einrichtung der virtuellen Maschinen und Hubs erfolgt automatisch durch Netkit-Skripte. Gesteuert und konfiguriert werden die virtuellen Maschinen über Konfigurationsdateien. Es werden eine Vielzahl von Tools zur Überwachung und Analyse des Netzwerkes mitgeliefert und unter anderem folgende Routing-Protokolle unterstützt:

- BGP
- RIP
- OSPF

3.2.2.1 Installation

Die Installation von Netkit gestaltet sich recht unkompliziert. Netkit ist in drei Pakete aufgeteilt:

- **Netkit Core**
  Das Kernpaket von Netkit. Es enthält das Nutzer-Interface und die Dokumentation.
- **Netkit Filesystem**
  Enthält das Dateisystem mit vorinstallierten Tools und Diensten. Virtuelle Maschinen werden mit einer Kopie von diesem Dateisystem erstellt.
- **Netkit Kernel**
  Dieses Paket enthält den modifizierten Kernel zur Simulation einer virtuellen Maschine.

Diese Pakete müssen von Netkit [Net] heruntergeladen werden und im gleichen Ordner liegen. Für diese Arbeit werden Core und Kernel in Version 2.8 sowie das Filesystem in Version 5.2 verwendet. Die Archive werden mit folgenden Befehlen entpackt:

```
user@pc:~$ tar -xjSf netkit-2.8.tar.bz2
user@pc:~$ tar -xjSf netkit-filesystem-F5.2.tar.bz2
user@pc:~$ tar -xjSf netkit-kernel-K2.8.tar.bz2
```

Listing 3.1: Entpacken von Netkit

Im nächsten Schritt müssen Umgebungsvariablen für Netkit exportiert werden. Die beste Möglichkeit ist, dies über die Konfigurationsdatei *.bashrc* im Home-Verzeichnis zu tun. Neue Befehle fügt man zumeinst am Ende der Datei ein. Der erste Export enthält den absoluten Pfad zum Netkit-Verzeichnis. Der zweite Export fügt den Pfad zu den Manpages (Dokumentation) hinzu und der letzte Export fügt die Netkit-Scripte der globalen Pfadvariable hinzu.

```
1 export NETKIT_HOME=/path/to/netkit
2 export MANPATH=:$NETKIT_HOME/man
3 export PATH=$NETKIT_HOME/bin:$PATH
```

Listing 3.2: Umgebungsvariablen exportiern

Die Konfiguration muss nun neu geladen werden.

```
user@pc:~$ source ~/.bashrc
```

Listing 3.3: .bashrc neu laden

Zur Überprüfung wechselt man nun in das Verzeichnis von Netkit und führt ein Script aus. Werden keine Fehler ausgegeben, war die Installation erfolgreich.

```
user@pc:~$ cd /path/to/netkit
user@pc:~$ ./check_configuration.sh
```

Listing 3.4: Installation prüfen

3.2.2.2 Konfiguration

Die Konfiguration der VMs von Netkit erfolgt über Textdateien. Jede VM kann dabei als Router, als Host oder als Switch konfiguriert werden. Netkit bietet zwei Gruppen von Befehlen zur Steuerung. Die erste Gruppe bilden die Befehle zur Steuerung einer einzelnen VM.

- **vstart**
  Startet eine neue virtuelle Maschine
- **vlist**
  Listet alle laufenden virtuelle Maschinen auf
- **vconfig**
  Fügt laufenden virtuelle Maschinen Netzwerk-Inferfaces hinzu
- **vhalt**
  Fährt eine virtuelle Maschine herunter
- **vcrash**
  Lässt eine virtuelle Maschine abstürzen
- **vclean**
  Beendet alle Netkit-Prozesse, wird auch *panic command* genannt

Die zweite Gruppe sind Befehle zur Steuerung eines gesamten Szenarios, genannt Lab. Ein Lab umfasst die Simulation eines Netzwerkes mit allen vorkonfigurierten VMs.

- **lstart**
  Startet ein Lab
- **lhalt**
  Beendet ein Lab
- **lcrash**
  Lässt ein komplettes Lab abstürzen
- **lclean**
  Entfernt temporäre Dateien eines Labs
- **linfo**
  Zeigt Informationen über ein Lab ohne es zu starten
- **ltest**
  Führt Tests für ein Lab aus

Labs werden über verschiedene Dateien und Verzeichnisse konfiguriert und gesteuert. Für jede virtuelle Maschine wird innerhalb des Lab-Verzeichnisses ein Unterordner angelegt. In diesem können Konfigurationsdateien abgelegt werden, welche beim Start des Lab automatisch umgesetzt werden.

lab.conf

Dies ist die globale Konfigurations-Datei für das Lab. Sie enthält die Netzwerk-Topologie des Lab und spezielle Parameter für virtuelle Maschinen. Folgendes Beispiel ist der Netkit-Einführung entnommen. Es gibt drei virtuelle Maschinen, eine mit Zuweisung von Speicher und zwei verschiedene Subnetze. Die zweite Maschine besitzt dabei zwei Interfaces, um die Subnetze miteinander zu verbinden.

```
1 pc1[0]=A
2
3 r1[0]=A
4 r1[1]=B
5 r1[mem]=256
6
7 pc2[0]=B
```

Listing 3.5: Beispiel lab.conf

Unterverzeichnisse

Für jedes Unterverzeichnis wird eine virtuelle Maschine mit dem Namen des Verzeichnisses gestartet. Dateien, die in einem Unterverzeichnis *vm_name* abgelegt werden, bindet Netkit beim Start der jeweiligen virtuelle Maschine in deren Root-Verzeichnis ein. Dies wird beispielsweise zur Konfiguration von Routing-Daemon oder anderen speziellen Einstellungen genutzt.

**.startup und *.shutdown*

Diese beiden Dateien können für jede virtuelle Maschine angelegt werden und beinhalten Befehle, die innerhalb der virtuelle Maschine ausgeführt werden. Die Dateien erhalten als Präfix den Namen der virtuellen Maschine, beispielsweise *pc1* oder *r1*. Folgendes Beispiel konfiguriert und aktiviert für einen Router zwei Netzwerk-Interfaces in jeweils verschiedenen Subnetzen.

```
1 ifconfig eth0 192.168.1.30 netmask 255.255.255.0 up
2 ifconfig eth1 192.168.1.94 netmask 255.255.255.224 up
```

Listing 3.6: Beispiel r1.startup

Möchte man global für alle virtuellen Maschinen einen bestimmten Befehl ausführen, legt man dazu eine Datei *shared.startup* beziehungsweise *shared.shutdown* an. Folgendes Beispiel fügt allen virtuellen Maschinen eine vorkonfigurierte *hosts*-Datei hinzu.

```
1 cp /hostlab/hosts /etc/hosts
```

Listing 3.7: Beispiel shared.startup

lab.dep

Möchte man virtuelle Maschinen in einer bestimmten Reihenfolge starten, kann man dies über die Datei *lab.dep* realisieren. Folgendes Beispiel legt fest, dass *pc3* erst starten kann, wenn *pc1* und *pc2* schon laufen.

```
1 pc3: pc1 pc2
```

Listing 3.8: Beispiel lab.dep

Erstellen einer Netzwerk-Topologie-Map mit "linfo"

Dieses Kommando zeigt, in der Console ausgeführt, Informationen über das aktuelle Lab an. Mit einem angehangenen Parameter wird eine PS-Datei mit der Netzwerk-Topologie erstellt. Dazu muss auf dem System die Zeichenbibliothek *GraphViz* installiert sein.

```
# GraphViz als root installieren
root@pc:/# aptitude install graphviz
# Topologie-Map erstellen
user@pc:~/path/to/lab$ linfo -m topology.ps
user@pc:~/path/to/lab$ evince topology.ps
```

Listing 3.9: linfo und GraphViz

3.2.3 Virtual Networks over linuX

Eine weitere Simulationsumgebung für Netzwerke ist *VNX* [Vnxa]. Dieses Umgebung ist die Weiterentwicklung des eingestellten Vorgängerprojektes VNUML. Wie auch Netkit baut VNX auf UML auf und bietet daher grundsätzlich die gleichen Möglichkeiten. Ein wesentlicher Vorteil zu Netkit ist jedoch, dass die komplette Konfiguration einer Simulation in einer Datei mittels XML gelöst wird. Ein Parser liest die XML-Beschreibungsdatei der Simulation ein und generiert alle virtuellen Maschinen und Interfaces. Durch die Nutzung von XML ist die Konfiguration größerer Netzwerke mit VNX einfacher und übersichtlicher.
Wie auch Netkit bietet VNX Unterstützung für gängige Routing-Protokolle.

3.2.3.1 Installation

Für die Installation von VNX sind einige Vorarbeiten notwendig. Es werden einige Perl-Module zum Parsen von XML-Dateien, Netzwerk-Klassen und der Fehlerbehandlung sowie einige Netzwerk-Systemtools benötigt. Auf dem Host-System müssen diese Pakete und Abhängigkeiten (*Dependencies*) sowie Virtualisierungskomponenten installiert werden. Dazu werden *root*-Rechte benötigt

```
root@pc:/# aptitude install qemu-kvm libvirt-bin vlan xterm
    bridge-utils screen virt-manager virt-viewer libxml-checker-
    perl libxml-parser-perl libnetaddr-ip-perl libnet-pcap-perl
    libnet-ipv6addr-perl liberror-perl libexception-class-perl uml
    -utilities libxml-libxml-perl libterm-readline-perl-perl
    libnet-telnet-perl libnet-ip-perl libreadonly-perl libmath-
    round-perl libappconfig-perl libdbi-perl graphviz genisoimage
    gnome-terminal tree libio-pty-perl libsys-virt-perl libfile-
    homedir-perl curl w3m
```

Listing 3.10: Abhängigkeiten-Installation für VNX

Nach der Installation müssen noch einige Anpassungen an Konfigurationsdateien gemacht werden. Das VNX-Wiki bietet hierzu eine gute Anleitung [Vnxc] für Ubuntu, welche auch auf Debian anwendbar ist. Ist dies abgeschlossen, kann VNX heruntergeladen [Vnxb] und installiert werden. Im Rahmen dieser Arbeit wird das Latest-Package in Version 2.0b.2262 verwendet. Nun wechselt man in das Verzeichnis, in welches das heruntergeladene Archiv gespeichert wurde und führt die folgenden Befehle aus.

```
user@pc:~$ tar -xSf vnx-latest.tgz
root@pc:/home/user/# cd vnx-*
root@pc:/home/user/vnx-*/# ./install_vnx
```

Listing 3.11: VNX installieren

Nach Abschluss der Installation steht das Command *vnx* zur Verfügung.

3.2.3.2 Das Kommando vnx

- **Schalter -f <xml>**
  Wird bei fast jedem Aufruf benötigt und übergibt die XML-Datei der zu nutzenden Simulation.
- **Schalter -t**
  Alternativ: - -create
  Mit diesem Schalter wird die Simulation erstellt.
- **Schalter -x <cmd_seq>**
  Alternativ: - -execute
  Führt die im XML vordefinierte cmd_seq der Simulation aus.
- **Schalter -d**
  Alternativ: - -shutdown
  Beendet die laufende Simulation
- **Schalter -P**
  Alternativ: - -destroy
  Beendet die laufende Simulation und löscht alle erzeugten Dateien, beispielsweise Filesystem-Images.
- **Schalter -v | -vv | -vvv**
  Aktiviert den Verbose-Mode und gibt detailierte Informationen zu jedem ausgeführten Schritt aus.
- **- -show-map**
  Erzeugt eine Topologie-Grafik der Simulation im PNG-Format.

Eine komplette Liste aller unterstützten Schalter und Parameter kann mit dem folgenden Aufruf ausgegeben werden.

```
user@pc:~$ vnx --help
# Alternativ:
user@pc:~$ vnx -h
```

Listing 3.12: VNX-Hilfe aufrufen

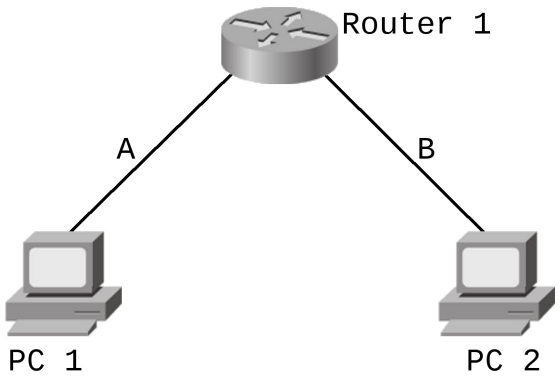
3.2.3.3 Konfiguration

Jede Simulation wird in einer Datei über XML-Baumstrukturen aufgebaut und vollständig mit allen Parametern konfiguriert. Je nach gewählten Komponenten und Konfigurationen ist es notwendig, die Simulation als *root* zu erstellen und zu verwalten, was im allgemeinen auch empfohlen wird.
Anhand des nächsten Beispieles wird der Grundaufbau einer Simulation erklärt.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd
       ">
3    <global>
4      <version>2.0</version>
5      <simulation_name>Beispiel-VNX-Simulation</simulation_name>
6      <automac/>
7      <vm_mgmt type="none" />
8      <vm_defaults>
9        <filesystem type="cow">/path/to/filesystem</filesystem>
10       <kernel>/path/to/kernel</kernel>
11       <console id="0">xterm</console>
12       <console id="1">xterm</console>
13       <console id="2">xterm</console>
14     </vm_defaults>
15   </global>
16   <net name="A" mode="uml_switch" />
17   <net name="B" mode="uml_switch" />
18   <vm name="pc1" type="uml">
19     <if id="1" net="A">
20       <ipv4>192.168.0.10</ipv4>
21     </if>
22     <route type="ipv4" gw="192.168.0.20">default</route>
23   </vm>
24   <vm name="r1" type="uml">
25     <if id="1" net="A">
26       <ipv4>192.168.0.20</ipv4>
27     </if>
28     <if id="2" net="B">
29       <ipv4>192.168.5.20</ipv4>
30     </if>
31   </vm>
32   <vm name="pc2" type="uml">
33     <if id="1" net="B">
34       <ipv4>192.168.5.10</ipv4>
35     </if>
36     <route type="ipv4" gw="192.168.5.20">default</route>
37   </vm>
38 </vnx>
```

Listing 3.13: XML der VNX-Beispiel-Simulation

Abbildung 3.2 Topologie der VNX-Beispiel-Simulation



Der erste Bereich beinhaltet das **global**-Tag. Es enthält globale Definitionen und Einstellungen für VMs. **version** enthält die verwendete VNX-Version, **simulation_name** den Namen der Simulation ohne Leerzeichen. Das Tag **automac** gibt an, dass die MAC-Adressen der Interfaces automatisch generiert werden. Über den **vm_mgmt**-Tag werden Einstellungen für das Interface einer VM vergeben, welches direkt vom Host-System ansprechbar ist. Der Parameter *type* gibt im Beispiel an, dass kein solches Interface vorhanden ist. Mit dem Tag **vm_defaults** legt man global für alle VM geltende Parameter fest. Notwendig sind die Angaben von dem zu nutzenden **kernel** und dem **filesystem** mit jeweils absoluter Pfadangabe, falls die Parameter nicht für jede VM separat definiert wurden.

Da VMs standardmäßig ohne Ein-/Ausgabe-Kanäle gestartet werden, kann man das Tag **console** angeben. Als Parameter wird die *id* der VM angegeben, für die Ein-/Ausgabe aktiviert werden soll. Als Standardwert wird *xterm* übergeben, wobei die verwendete Console über das **xterm**-Tag auch angepasst werden kann. Weitere Tags sind **ssh_version** oder **ssh_key**.

Mit dem Tag **net** definiert und benennt ein neues Subnetz. Damit können Interfaces bestimmten Netzen zugeordnet werden. Weitere Parameter sind *name* zur Vergabe eines Namens für dieses Netz und *mode*. Wenn der Parameter *mode* nicht gesetzt auf auf *virtual_bridge* steht, erstellt VNX über das System-Tool *brctl* eine virtuelle Bridge für den Aufbau des Netzes. Dies erfordert *root*-Rechte. Ist der Parameter *uml_switch*, wird der gleichnamige Daemon eingesetzt. Die Weiterleitung von Paketen zwischen den virtuellen Maschinen erfolgt dann über UNIX-Sockets.

Für jede virtuelle Maschine folgt nun ein Block **vm** mit den Parameter *name* und *type* zur Definition. Für jedes Interface einer VM wird ein **if**-Tag angelegt mit der *id* des Interfaces und dem zugehörigen Subnet *net*. Über das **ipv4**-Tag wird eine IPv4-Adresse zugewiesen, analog dazu das Tag **ipv6**. Fehlt im *global*-Bereich das *automac*-Tag, kann über **mac** eine MAC-Adresse angegeben werden. Über das **route**-Tag kann der VM weiterhin eine statische Route mit *type* und *gw*-Adresse definiert werden. Dieses Tag ist optional.

Der VM kann man über die Tags **filesystem** und **kernel** die Nutzung modifizierter Versionen übergeben. Diese Einstellungen überschreiben die des *global*-Tags.

Ein weiteres Tag ist **exec** mit den Parametern *seq* und *type*. Für *seq* gibt man eine Sequenz, beispielsweise *start* oder *stop* an. Der Parameter *type* wird mit *verbatim* für fast alle Befehle gesetzt.

```
1 <exec seq="start" type="verbatim">/usr/local/bin/test.sh</exec>
```

Listing 3.14: Beispiel exec-Tag

Weitere Tags und Parameter sind im VNUML-Wiki [Vnu] aufgeführt und erklärt.

3.2.4 Weitere Netzwerk-Simulatoren

Ein weiterer recht häufig genutzter Simulator ist *ns-2*. Er ist in *C++* geschrieben und basiert auf einer Simulations-Bibliothek sowie einem Simulations-Scheduler. Die Simulationsszenarien für *ns-2* werden in der Skriptsprache *Object-oriented Tcl* (OTcl) geschrieben. Dies ist eine an *C++* angelehnte Sprache und bietet für jedes Objekt aus *C++* eine OTcl-Entsprechung. Um beide Sprache miteinander zu verbinden gibt es die Tcl-Erweiterung *Tclcl*. ns-2 bietet Unterstützung für alle gängigen Routing-Protokolle.

Als weitere, ebenso auf UML setzende Netzwerk-Simulatoren sind *Cloonix-net* oder *Marionnet* zu nennen.

3.3 Vergleich und Zusammenfassung

Die Simulatoren kann man nach verschiedenen Aspekten vergleichen. Einige dieser Aspekte sind *Funktionalität*, *Erweiterbarkeit*, *Ressourcenbedarf* sowie die *Erlernbarkeit* und die *Dokumentation* des Simulators.

Funktionalität

Netkit und VNX bieten aufgrund der Nutzung eines angepassten Kernels und Filesystems prinzipiell die gleiche Funktionalität. Je nach vorgenommenen Anpassungen an Filesystem-Images können sich verfügbare Funktionen auch ändern. VNX bietet weiterhin Unterstützung für Virtualisierung mittels KVM.

Erweiterbarkeit

Es ist bei beiden Simulatoren möglich, einen eigenen Kernel zu erstellen und zu nutzen, sowie auch ein eigenes, speziell angepasstes Filesystem. VNX bietet für die Erstellung von Filesystem-Images mehrere Anleitungen für verschiedene Systeme und auch Unterstützung für KVM-Images. Netkit nutzt nur UML-Images.

Ressourcenbedarf

Beide Simulatoren benötigen für eine VM nur wenige Ressourcen wie Arbeitsspeicher, sind jedoch anpassbar. Den meisten Platz nehmen die Filesystem-Images ein.

Netkit arbeitet dabei im Copy-On-Write-Modus (COW). Dies bedeutet, für jede VM wird eine Kopie des Original-Filesystems angelegt, in welche Änderungen innerhalb der VM geschrieben werden. Nach dem Beenden der VM wird die Kopie wieder entfernt. VNX bietet die Möglichkeit, ohne COW zu arbeiten.

Erlernbarkeit und Handhabung

Eine Voraussetzung sind Kenntnisses im Umgang mit Linux-Bordmitteln und der Console. Die Konfiguration von Daemon oder Mitloggen von Daten ist in Manpages oder Online-Wikis dokumentiert und meist verständlich. Für VNX ist es notwendig, den Umgang mit XML zu beherrschen und sich mit den verfügbaren Tags und Attributen auseinanderzusetzen.

Da VNX mit XML arbeitet, kann die Einarbeitungszeit etwas höher liegen als die für Netkit notwendige.

Dokumentation

VNX bietet einige wenige Beispiele, jedoch ist der größte Teil aus dem Vorgängerprojekt VNUML weitestgehend übernehmbar. Installationsvorgang, die Erstellung eigener Kernel und eines Filesystems sowie die XML-Referenz sind ausführlich und leicht verständlich.

Netkit bietet hingegen sehr viele ausführliche und auch praxisnahe Beispiele, die mit jeweils eigenen Foliensätzen sehr gut dokumentiert sind. Auch hier ist die Erstellung von Kernel und Filesystem ausführlich dokumentiert.

Auswahl eines Simulators

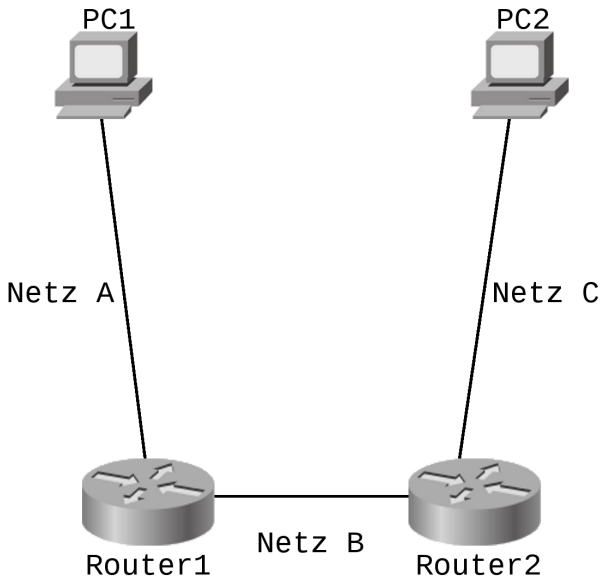
Aufgrund der Installation ohne die Erfordernis von root-Rechten, vielen gut dokumentierten Beispielen und der Konfigurierbarkeit der VMs direkt über Dateien im Dateisystem wird für die Anwendungsbeispiele *Netkit* gewählt.

4 Anwendungsbeispiele

4.1 Netkit - Static Routing

In diesem Beispiel wird der Netzwerkaufbau über statischer Routen erklärt. Gegeben ist die folgende Netzwerk-Topologie, bestehend aus drei Subnetzen.

Abbildung 4.1 Topologie - Netkit: Static Routing



Zu Beginn werden die notwendigen Verzeichnisse erstellt und Konfigurationen der einzelnen VM vorgenommen. Die IP-Adressen der Subnetze A und C sind dabei frei aus einem /24er-Netz wählbar, für Subnetz B wurde ein /30er-Netz mit zwei verfügbaren Adressen gewählt.

```
user@pc:~/lab$ mkdir -p netkit-static/pc1 netkit-static/pc2
   netkit-static/router1 netkit-static/router2
user@pc:~/lab$ cd netkit-static
```

Listing 4.1: Verzeichnisse - Netkit: Static Routing

```
# Datei lab.conf
pc1[0]=A
pc2[0]=C

router1[0]=A
router1[1]=B

router2[0]=B
router2[1]=C

# Datei pc1.startup
ifconfig eth0 192.168.50.10 netmask 255.255.255.0 broadcast 192.
    168.50.255 up

# Datei pc2.startup
ifconfig eth0 192.168.150.10 netmask 255.255.255.0 broadcast 192.
    168.150.255 up

# Datei router1.startup
ifconfig eth0 192.168.50.20 netmask 255.255.255.0 broadcast 192.
    168.50.255 up
ifconfig eth1 192.168.100.1 netmask 255.255.255.252 broadcast 192
    .168.100.3 up

# Datei router2.startup
ifconfig eth0 192.168.100.2 netmask 255.255.255.252 broadcast 192
    .168.100.3 up
ifconfig eth1 192.168.150.20 netmask 255.255.255.0 broadcast 192.
    168.150.255 up
```

Listing 4.2: Konfiguration - Netkit: Static Routing

Mit *lstart* wird die Simulation erstellt und die vier VMs nacheinander gestartet. Nun kann man beispielsweise in *pc2* testen, welche Knoten erreichbar sind.

```
pc2:~# ping 192.168.150.20 -c 4
pc2:~# ping 192.168.100.2 -c 4
```

Listing 4.3: Erreichbarkeit - Netkit: Static Routing

Der erste Befehl liefert eine erfolgreiche Antwort, der zweite eine Fehlermeldung *connect: Network is unreachable*. Ein Blick auf die Routing-Tabelle (Befehle *route*) von *pc2* liefert eine Antwort auf die Meldung. *pc2* weiß nicht, über welchen Knoten er sein eigenes Netz verlassen und andere Netze erreichen kann. Mit folgendem Befehl fügt man beiden PC jeweils eine Standard-Route hinzu.

```
pc1:~# route add default gw 192.168.50.20
pc2:~# route add default gw 192.168.150.20
```

Listing 4.4: PC-Routen - Netkit: Static Routing

Wiederholt man den Ping-Versuch, bekommt man nun eine positive Antwort. Der nächste Schritt ist, *pc1* über *router1* zu erreichen. Pingt man *router1-eth1* an, erhält man keine Antwort. Die Pings kommen dennoch an, wie folgende Befehle zeigen.

```
pc2:~# ping 192.168.100.1

router1:~# tcpdump -i eth1
listening on eth1, link-type EN10MB (Ethernet), capture size 96
   byte
11:44:32.810687 IP 192.168.150.10 > 192.168.100.1: ICMP echo
   request, id 4098, seq 17, length 64
1 packets captured
1 packets received by filter
0 packets dropped by kerne
```

Listing 4.5: Erreichbarkeit router1 - Netkit: Static Routing

Der Grund, weshalb *router1* nicht antwortet ist, dass er das Netzwerk von *pc2* nicht kennt. Man muss *router1* mitteilen, über welches Gateway das Netz von *pc2* erreichbar ist und *router2*, über welches Gateway er das Netz von *pc1* erreichen kann.

```
router1:~# route add -net 192.168.150.0/24 gw 192.168.100.2 dev
   eth1

router2:~# route add -net 192.168.50.0/24 gw 192.168.100.1 dev
   eth0
```

Listing 4.6: Routen - Netkit: Static Routing

Jetzt kann man von *pc2* aus *pc1* erreichen.

```
pc2:~# traceroute 192.168.50.10
traceroute to 192.168.50.10 (192.168.50.10), 64 hops max, 40 byte
    packets
 1  192.168.150.20 (192.168.150.20)  11 ms  1 ms  0 ms
 2  192.168.100.1 (192.168.100.1)  11 ms  1 ms  1 ms
 3  192.168.50.10 (192.168.50.10)  11 ms  1 ms  1 ms
```

Listing 4.7: Traceroute - Netkit: Static Routing

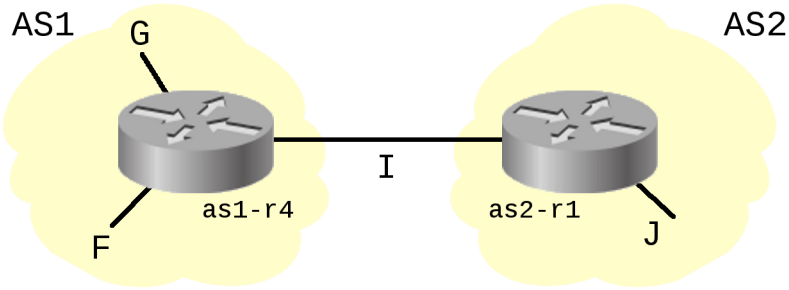
4.2 Netkit - RIP und BGP

Dieses Beispiel ist unterteilt in drei Versuche. Im ersten Versuch wird eine Verbindung zwischen zwei unterschiedlichen AS über BGP aufgebaut. Der zweite Versuch fügt AS1 ein Netz mit weiteren Router hinzu, welche ihre Routing-Informationen über RIP austauschen. Im dritten Versuch wird der Ausfall eines Interfaces simuliert.

4.2.1 Teil 1: BGP

Gegeben ist die folgende Netzwerk-Topologie. Jede von einem Router abgehende Verbindung ist ein eigenes Subnetz.

Abbildung 4.2 Topologie - Netkit: BGP



Die einzelnen Subnetze und Interfaces der Router sind nach folgender Tabelle konfiguiert. Die jeweiligen Dateien *lab.conf*, *as1-r4.startup* sowie *as2-r1.startup* sind entsprechend zu erstellen und die Interface-Konfigurationen anzulegen.

Router-Interface	Netz	IP in CIDR-Notation
as1-r4 eth0	F	150.50.100.10/30
as1-r4 eth1	G	150.50.100.13/30
as1-r4 eth2	I	100.0.0.1/30
as2-r1 eth0	I	100.0.0.2/30
as2-r1 eth1	J	200.50.10.1/24

Neu hinzu kommen für dieses Beispiel Konfigurationsdateien für den Routing-Daemon *zebra/quagga* [Qua] und den eingesetzten Routing-Protokoll-Daemon. Dazu müssen folgende Verzeichnisse angelegt werden.

```
user@pc:~/lab$ mkdir -p as1-r4/etc/zebra
user@pc:~/lab$ mkdir -p as2-r1/etc/zebra
```

Listing 4.8: Verzeichnisse - Netkit: BGP

In den erstellten Ordnern werden nun die Dateien *daemons* und *bgpd.conf* angelegt. Die Datei *daemons* enthält die zu startenden Routing-Daemon, die jeweilige

bgpd.conf enthält Konfigurationen und Informationen über eigene Subnetze und direkte Nachbarn sowie Log- und Debug-Parameter.

```
# Datei daemons
zebra=yes
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
ldpd=no

# Datei bgpd.conf fuer a1-r4
!
hostname bgpd
password zebra
enable password zebra
!
router bgp 1
network 150.50.0.0/16
neighbor 100.0.0.2 remote-as 2
neighbor 100.0.0.2 description Router as2-r1
!
log file /var/log/zebra/bgpd.log
!
debug bgp
debug bgp events
debug bgp filters
debug bgp fsm
debug bgp keepalives
debug bgp updates
!

# Datei bgpd.conf fuer a2-r1
!
hostname bgpd
password zebra
enable password zebra
!
router bgp 2
network 200.50.0.0/16
neighbor 100.0.0.1 remote-as 1
neighbor 100.0.0.1 description Router as1-r4
!
...
# log- und debug-Parameter wie oben
```

Listing 4.9: Konfiguration - Netkit: BGP

Über den Parameter *network* wird das Subnetz des jeweiligen AS angegeben. Die *neighbor*-Parameter enthalten Informationen über benachbarte BGP-Router wie die IP und den Namen. Weitere mögliche Parameter sind Filter oder Default-Routen.

Das Lab kann nun gestartet werden. Auf *as1-r4* wird nun der Daemon gestartet und eine Überwachung der Routing-Tabelle ausgeführt.

```
# Routing-Daemon starten
as1-r4:~# /etc/init.d/zebra start

# Ueberwachen der Routing-Tabelle auf neue Eintraege
as1-r4:+# watch -n1 route
```

Listing 4.10: Verzeichnisse - Netkit: BGP

In *as2-r1* wird nun auch der Daemon gestartet. Steht in der Routing-Tabelle von *as1-r4* das neu verfügbare Subnetz, kann man die Quagga-Console *vtysh* öffnen und detaillierte Informationen wie zur Routing-Tabelle ansehen.

```
# Routing-Daemon starten
as2-r1:~# vtysh
as2-r1# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - ISIS, B - BGP, > - selected route,
       * - FIB route

C>* 100.0.0.0/30 is directly connected, eth0
C>* 127.0.0.0/8 is directly connected, lo
B>* 150.50.0.0/16 [20/0] via 100.0.0.1, eth0, 00:01:43
C>* 200.50.10.0/24 is directly connected, eth1
```

Listing 4.11: Verzeichnisse - Netkit: BGP

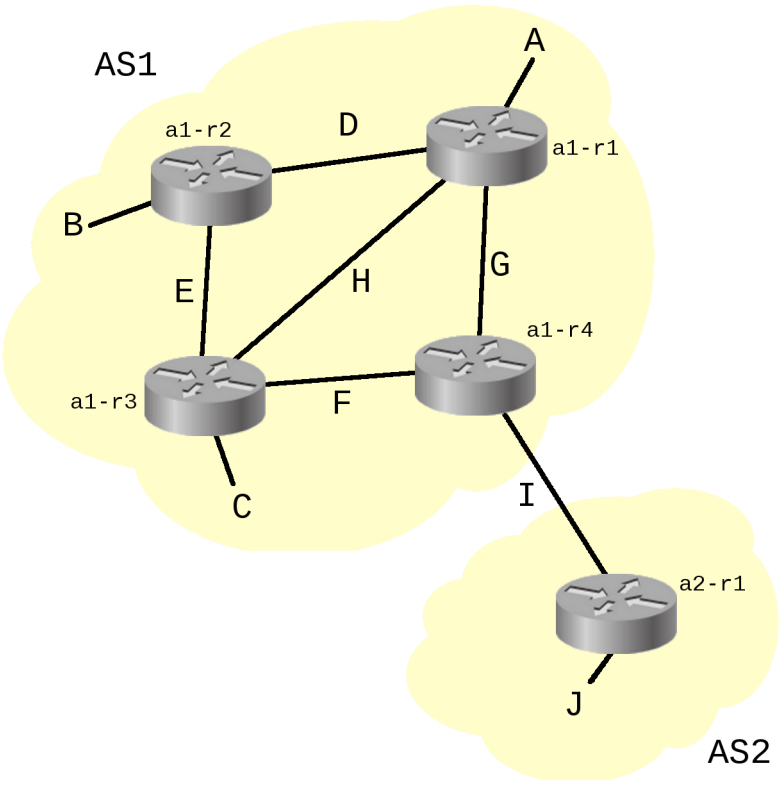
Der vorletzte Eintrag zeigt ein neues Subnetz an, erreichbar über *as1-r4*. Weitere interessante Befehle für BGP-Interfaces sind beispielsweise *show ip bgp neighbor* oder *show ip bgp summary*. Der erste Befehl gibt detaillierte Informationen über sämtliche verbundene Nachbarn aus und der zweite eine kurze Übersicht.

4.2.2 Teil 2: Zusammenspiel von RIP und BGP

Jetzt wird AS1 um drei Router erweitert. Diese Router sollen untereinander und mit *as1-r4* über RIP kommunizieren und müssen dafür konfiguriert werden. Weiterhin sollen alle Router aus AS1 mit AS2 kommunizieren können.

Zu Beginn werden die noch fehlenden Verzeichnisse für *as1-r1*, *as1-r2* und *as1-r3* angelegt. Die startup-Dateien sind anhand der Tabelle zu konfigurieren und die Datei *lab.conf* zu erweitern. Folgende Grafik zeigt die neue Topologie des Netzwerkes.

Abbildung 4.3 Topologie - Netkit: RIP und BGP



Router-Interface	Netz	IP in CIDR-Notation
a1-r1 eth0	A	150.50.10.1/24
a1-r1 eth1	D	150.50.100.1/30
a1-r1 eth2	G	150.50.100.14/30
a1-r1 eth3	H	150.50.100.17/30
a1-r2 eth0	B	150.50.20.1/24
a1-r2 eth1	D	150.50.100.2/30
a1-r2 eth2	E	150.50.100.5/30
a1-r3 eth0	C	150.50.30.1/24
a1-r3 eth1	E	150.50.100.6/30
a1-r3 eth2	F	150.50.100.9/30
a1-r3 eth3	H	150.50.100.18/30
a1-r4 eth0	F	150.50.100.10/30
a1-r4 eth1	G	150.50.100.13/30
a1-r4 eth2	I	100.0.0.1/30
a2-r1 eth0	I	100.0.0.2/30
a2-r1 eth0	J	200.50.10.1/24

Die Datei *daemons* wird für jeden Router in AS1 angepasst. Weiterhin erhält jeder Router in AS1 eine Konfigurationsdatei *ripd.conf* für den RIP-Daemon mit folgendem Inhalt.

```
# Anpassung in daemons
zebra=yes
bgpd=no
ripd=yes

# Datei ripd.conf fuer alle Router in as1
!
hostname ripd
password zebra
enable password zebra
!
router rip
version 2
network 150.50.0.0/16
redistribute connected
!
log file /var/log/zebra/ripd.log
```

Listing 4.12: Konfigurationen - Netkit: RIP und BGP

In *ripd.conf* wird das zu verwaltende Subnetz angegeben und mit *redistribute connected* die Propagation von Routen aktiviert. Den *startup*-Dateien aller AS1-Router kann man nun noch den Start-Eintrag für den Daemon hinzufügen.

```
# Einfuegen in alle .startup aus as1
/etc/init.d/zebra start
```

Listing 4.13: Daemon Autostart - Netkit: RIP und BGP

Das Lab wird gestartet und auf *a1-r2* die Überwachung der Routing-Tabelle ausgeführt.

```
as1-r2:+# watch -n1 route
```

Listing 4.14: Überwachung Routing-Tabelle - Netkit: RIP und BGP

Man kann beobachten, wie sich die Tabelle mit allen erreichbaren Adressen aus AS1 füllt. Im nächsten Schritt wird der Routing-Daemon auf *as2-r1* gestartet sowie in *as1-r4* Anpassungen in der *daemons* vorgenommen und auch hier danach der Daemon neu gestartet.

```
# Anpassung in /etc/zebra/daemons fuer as1-r4
zebra=yes
bgpd=yes
ripd=yes
```

Listing 4.15: Aktivierung von BGP - Netkit: RIP und BGP

Sieht man sich nun die Routing-Tabelle von *as1-r4* an, befinden sich in dieser nun auch die erreichbaren Netze aus AS2. Diese erscheinen jedoch nicht in der Ausgabe von *as1-r2*. Der Grund dafür ist, dass man Routern, welche mit verschiedenen Protokollen arbeiten, mitteilen muss, dass die Routen aller Protokolle weiter verteilt werden sollen. Für dieses Beispiel bedeutet dies, dass über BGP erhaltene Routen von RIP mitverteilt werden sollen. Die folgende Anpassung in der *ripd.conf* von *as1-r4* behebt das Problem.

```
1 # Einfuegen unter redistribute connected
2 redistribute bgp
```

Listing 4.16: Weitergabe von BGP-Routen - Netkit: RIP und BGP

Nach einem Neustart des Daemon werden die erreichbaren Netze von AS2 über RIP an alle Router in AS1 verteilt. Die vollständige Routing-Tabelle von *as1-r2* sieht nun aus wie folgt.

```
as2-r2:~# vtysh
as1-r2# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - ISIS, B - BGP, > - selected route,
       * - FIB route

R>* 100.0.0.0/30 [120/3] via 150.50.100.6, eth2, 00:13:22
C>* 127.0.0.0/8 is directly connected, lo
R>* 150.50.10.0/24 [120/2] via 150.50.100.1, eth1, 00:13:56
C>* 150.50.20.0/24 is directly connected, eth0
R>* 150.50.30.0/24 [120/2] via 150.50.100.6, eth2, 00:13:37
C>* 150.50.100.0/30 is directly connected, eth1
C>* 150.50.100.4/30 is directly connected, eth2
R>* 150.50.100.8/30 [120/2] via 150.50.100.6, eth2, 00:13:37
R>* 150.50.100.12/30 [120/2] via 150.50.100.1, eth1, 00:13:56
R>* 150.50.100.16/30 [120/2] via 150.50.100.1, eth1, 00:13:56
R>* 200.50.0.0/16 [120/3] via 150.50.100.6, eth2, 00:09:4
```

Listing 4.17: Routing-Tabelle - Netkit: RIP und BGP

Dieses Beispiel ist beliebig um weitere Router oder PC in AS1 und AS2 erweiterbar. Soll das Subnetz in AS2 auch über RIP kommunizieren und die in AS1 verfügbaren Netze kennen, müssen Anpassungen analog zu *as1-r4* vorgenommen werden. Um ein neues AS zu verbinden, müssen die *bgpd.conf* der als Border-Router fungiernden Router entsprechend konfiguriert werden.

4.2.3 Teil 3: Ausfall von Interfaces

Nun wird im Netzwerk aus Teil2 der Ausfall von zwei Interfaces simuliert. Ziel ist es zu zeigen, dass RIP in solchen Fällen nur sehr langsam konvergiert. Der Ausgangsrouter für dieses Beispiel ist *as1-r2*, das zu erreichene Zielnetz liegt in AS2.
Zu Beginn wird geprüft, ob und über welche Routen AS2 erreichbar ist.

```
as1-r2:~# traceroute 200.50.10.1
traceroute to 200.50.10.1 (200.50.10.1), 64 hops max, 40 byte
   packets
 1  150.50.100.6 (150.50.100.6)  1 ms  0 ms  1 ms
 2  150.50.100.10 (150.50.100.10)  1 ms  1 ms  1 ms
 3  200.50.10.1 (200.50.10.1)  1 ms  1 ms  1 ms
```

Listing 4.18: Traceroute AS2 - Netkit: RIP und BGP

Nun werden zwei Interfaces deaktiviert. Um einen möglichst langen Weg zu simulieren, werden die Interfaces **eth2** von *as1-r1* sowie **eth1** von *as1-r3* gewählt.

```
as1-r1:~# ifconfig eth2 down
as1-r3:~# ifconfig eth1 down
```

Listing 4.19: Interfaces deaktivieren - Netkit: RIP und BGP

Ein erneutes *traceroute* auf *as1-r2* führt zunächst zu folgendem Ergebnis:

```
as1-r2:~# traceroute 200.50.10.1
traceroute to 200.50.10.1 (200.50.10.1), 64 hops max, 40 byte
   packets
 1  * 150.50.100.5 (150.50.100.5)  40 ms !H
```

Listing 4.20: AS2 unerreichbar - Netkit: RIP und BGP

Das Flag *!H* besagt, dass der Host nicht erreichbar ist. RIP besitzt drei Timer, *update*, *timeout* und *garbage*. Der Update-Timer läuft aller 30s und sendet dann die aktuelle Routing-Tabelle an seine Nachbarn. Der Timeout für eine Route beträgt 180s. Ist die Route länger als diese Zeit nicht erreichbar, wird sie als ungültig markiert, verbleibt aber bis zum nächsten Update noch in der Routing-Tabelle. Als ungültig markierte Routen werden durch den Garbage-Timer nach 120s endgültig entfernt. Dementsprechend lange dauert es, bis AS2 von *as1-r2* wieder erreichbar ist.

```
as1-r2:~# traceroute 200.50.10.1
traceroute to 200.50.10.1 (200.50.10.1), 64 hops max, 40 byte
   packets
 1  150.50.100.1 (150.50.100.1)  11 ms  1 ms  0 ms
 2  150.50.100.18 (150.50.100.18)  10 ms  1 ms  1 ms
 3  150.50.100.10 (150.50.100.10)  11 ms  1 ms  1 ms
 4  200.50.10.1 (200.50.10.1)  1 ms  1 ms  1 ms
```

Listing 4.21: AS2 erreichbar - Netkit: RIP und BGP

Die neue Route führt nun über *as1-r1* zu *as1-r3* und erst dann über *as1-r4* zu AS2.

Im Quagga-Wiki [Qua] finden sich weitere Informationen zur Konfiguration und Anpassung der Routing-Daemon.

Quellenverzeichnis

[Bau07] Rainer Baumann. *A Survey on Routing Metrics*. 2007. URL: http://rainer.baumann.info/public/tik262.pdf (besucht am 25. 05. 2012).

[Cid] *CIDR - Classless Inter-Domain Routing*. URL: http://compnetworking.about.com/od/workingwithipaddresses/a/cidr_notation.htm (besucht am 28. 05. 2012).

[For10] Fortinet. *FortiOS Dynamic Routing Guide*. 2010. URL: http://docs.fortinet.com/fgt/handbook/fortigate-dynamic-routing-40-mr1.pdf (besucht am 28. 05. 2012).

[Fü] Holger Füssler. *Vorlesung: Simulation von Rechnernetzen*. URL: http://pi4.informatik.uni-mannheim.de/pi4.data/content/courses/2005-ss/netsim/ (besucht am 06. 06. 2012).

[Joh08] Allan Johnson. *Routing Protocols and Concepts - CCNA Exploration Labs and Study Guide*. 2008. URL: http://www.fruned.com/dwn/Routing-Protocols-and-Concepts-CCNA-Exploration-Labs-and-Study-Guide.pdf (besucht am 25. 05. 2012).

[Keu05] Tim Keupen. *User-Mode Linux*. 2005. URL: http://userpages.uni-koblenz.de/~vnuml/docs/vnuml/uml.pdf (besucht am 08. 07. 2012).

[Koc05] Tino Koch. »Implementation und Simulation von RIP-MTI«. Diplomarbeit. Universität Koblenz-Landau, 25. Apr. 2005. URL: http://www.uni-koblenz.de/vnuml/docs/rip/Diplomarbeit_TKoch.pdf (besucht am 06. 06. 2012).

[Li10] Ji Li. »Konzeption und Implementierung von Netzwerksimulationskomponenten für die online-Plattform des CANDY-Frameworks«. Diplomarbeit. Technische Universität Dresden, 4. Nov. 2010. URL: http://www.rn.inf.tu-dresden.de/uploads/Studentische_Arbeiten/Diplomarbeit_Li_Ji.pdf (besucht am 23. 06. 2012).

[LS06] Anatol Lemke und Arash Sarkohi. *Werkzeuge zur Netzwerksimulation*. 2006. URL: http://nsl.csie.nctu.edu.tw/NCTUnsReferences/Anatol%2520Lemke,%2520Arash%2520Sarkohi%2520-%2520Werkzeuge%2520zur%2520Netzwerksimulation.pdf (besucht am 08. 07. 2012).

[Net] *Download Official - Netkit Wiki*. URL: http://wiki.netkit.org/index.php/Download_Official (besucht am 23. 06. 2012).

[Qua] *Quagga Software Routing Suite*. URL: http://www.nongnu.org/quagga/docs/docs-info.html (besucht am 06. 07. 2012).

[Vnu] *Reference 1.8 - VNUML-WIKI*. URL: http://neweb.dit.upm.es/vnumlwiki/index.php/Reference (besucht am 29. 06. 2012).

[Vnxa] *VNX*. URL: http://web.dit.upm.es/vnxwiki/index.php/Main_Page (besucht am 29. 06. 2012).

[Vnxb] *VNX Download Latest*. URL: http://vnx.dit.upm.es/vnx/vnx-latest.tgz (besucht am 29. 06. 2012).

[Vnxc] *Vnx-install-ubuntu2 - VNX*. URL: http://web.dit.upm.es/vnxwiki/index.php/Vnx-install-ubuntu2 (besucht am 29. 06. 2012).

[Wika] *Simulation – Wikipedia*. URL: http://de.wikipedia.org/wiki/Simulation (besucht am 12. 06. 2012).

[Wikb] *Simulation – Wikipedia*. URL: http://en.wikipedia.org/wiki/Network_simulation (besucht am 12. 06. 2012).

Verzeichnis der Listings

Selbstständigkeitserklärung

Ich versichere, dass ich die *Bachelorarbeit* selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

_____

Unterschrift