

Seminar User Mode Linux

RIP Network Laboratory

Andreas Klöber

09.12.2005

Inhaltsverzeichnis

1 Grundlagen zum Routing	2
1.1 Routing-Protokolle	2
1.2 Routing-Algorithmen	2
2 RIP (Routing Information Protokoll)	3
2.1 Funktionsweise	3
2.2 Ausfall eines Links	5
2.3 Der Bouncing-Effekt	7
2.4 Counting to Infinity	7
2.5 Split Horizon und Triggered Updates	8
2.6 Nachrichtenformat	8
3 Routing mit RIP unter Linux	9
3.1 Konfiguration und Start von Zebra	10
3.2 Konfiguration und Start von ripd	11
4 Beispielszenario für RIP in VNUML	12
4.1 Simuliertes Netzwerk	12
4.2 Start der Simulation	12
4.3 Start von Zebra und ripd	13
4.4 Ausfall einer Verbindung	14
A VNUML-Konfigurationsdatei	17

1 Grundlagen zum Routing

Das Internet wird durch eine Ansammlung von **autonomen Systemen** gebildet. Ein autonomes System kann beispielsweise ein Internet-Provider mit all seinen Kunden oder eine Universität mit all ihren Anschlüssen sein. Diese autonomen Systeme sind wiederum miteinander verbunden und formen das Internet. Damit ein Teilnehmer eines autonomen Systems Daten an einen Teilnehmer eines anderen autonomen Systems senden kann, müssen die Datenpakete (evtl. über weitere autonome Systeme) zielgerichtet weitergeleitet werden. Dies bezeichnet man als **Routing** und die dafür zuständigen Geräte als **Router**.

Ein Router besitzt mindestens 2 Anschlüsse zu verschiedenen Netzwerken und leitet ankommende Datenpakete an das dem Ziel am nächstliegende Netzwerk weiter. Um dies leisten zu können verfügen Router über eine interne **Routing-Tabelle**. Sie enthält Einträge darüber, welches Ziel über welches Netzwerk (bzw. Interface) am schnellsten zu erreichen ist. Die Art und Weise, wie diese Routing-Tabellen aufgebaut werden, wird in verschiedenen **Routing-Protokollen** definiert.

1.1 Routing-Protokolle

Die Routing-Protokolle lassen sich anhand ihrer Anwendungsgebiete in zwei Gruppen unterteilen:

- *Interior-Gateway-Protokolle (IGP)*
Sie werden zum Routing innerhalb autonomer Systeme genutzt. Zu ihnen zählen beispielsweise: RIP, IGRP, EIGRP, OSPF und ISIS.
- *Exterior-Gateway-Protokolle (EGP)*
Sie werden zum Routing zwischen verschiedenen autonomen Systemen, die über sogenannte Backbones verbunden sind, verwendet. Hier findet vor allem das Border-Gateway-Protokoll (BGP) Verwendung.

Aufgabe der Routing-Protokolle ist das Ausfüllen der Routing-Tabellen, die die Informationen zum Erreichen verschiedener Zielnetzwerke speichern. Diese Informationen bestehen für ein gegebenes Zielnetzwerk aus dem „next-Hop“, also dem nächsten Router, der dem Ziel am nächsten ist. Dieser Router schaut dann in seiner Routing-Tabelle nach und leitet das Datenpaket ebenfalls entsprechend weiter. Dies wiederholt sich bis das Datenpaket im Zielnetzwerk angekommen ist.

1.2 Routing-Algorithmen

Die Routing-Protokolle lassen sich auch anhand der Algorithmen charakterisieren, die sie benutzen, um die Routing-Tabellen möglichst effizient zu berechnen. Die Art des eingesetzten Algorithmus entscheidet auch über das Verhalten des Protokolls. Entscheidend sind hier:

- Benötigte Zeit bis alle Routing-Tabellen vollständig sind. Dies wird auch als Konvergenz bezeichnet.
- Rechenintensivität des Algorithmus
- Reaktionsverhalten zum Beispiel beim Ausfall eines Routers oder bei Änderungen in der Netzwerktopologie.

Die beiden bedeutendsten Algorithmen sind der *Distance-Vector-Algorithmus*, der beispielsweise bei RIP eingesetzt wird, und der *Link-State-Algorithmus*, wie er in OSPF eingesetzt wird.

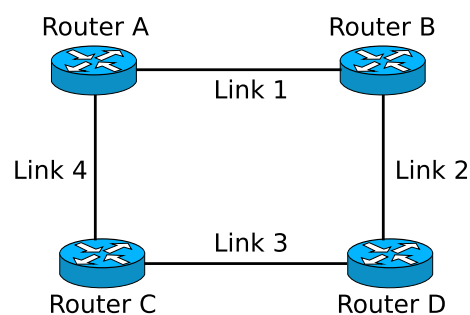
Der Distance-Vector-Algorithmus, wie er in RIP eingesetzt wird, ist einfach zu implementieren und reicht für kleine Netzwerktopologien mit geringen Ausfallwahrscheinlichkeiten aus. Für grössere Netzwerke ist er jedoch aufgrund seiner Trägheit in Bezug auf Änderungen im Netzwerk und den daraus entstehenden inkonsistenten Zuständen bis zur erneuten Konvergenz (z.B. Schleifen) eher ungeeignet. Hier eignen sich ausgereiftere, allerdings auch komplizierte Algorithmen, wie Link-State (OSPF) besser.

2 RIP (Routing Information Protokoll)

Das *Routing-Information-Protokoll (RIP)* wurde zuerst in frühen UNIX-Systemen eingesetzt. Es existiert in zwei Versionen, wobei in der zweiten Version einige Verbesserungen vorgenommen wurden, die in der ersten Version noch nicht vorhanden waren. Hierzu zählt u.a. die Möglichkeit RIP auch in Subnetzen einzusetzen.

2.1 Funktionsweise

Um die Erklärung zu vereinfachen wird im folgenden Beispiel nicht zwischen Hosts und Routern bzw. Subnetzen und Links unterschieden. Folgendes Beispielnetz sei gegeben:



Die Initialisierung beginnt mit dem sogenannten *Cold Start*. In diesem Zustand haben alle Router nur Informationen über sich selber. Die initiale Routingtabelle für Router A sieht dann beispielsweise wie folgt aus:

Von A nach	Link	Kosten
A	lokal	0

Die Tabelle enthält nur einen Eintrag über den Router selbst, da er noch nichts über die anderen Router weiss. Die Kosten belaufen sich auf 0, da er es ja selbst ist, was nochmals durch den Link „lokal“repräsentiert wird. Die Tabellen der anderen Router sehen äquivalent aus.

Aus dieser Tabelle generiert der Router einen Distanz-Vektor, mit genau diesem Eintrag:

$$A = 0$$

Diesen schickt er mittels *Broadcast* an alle seine Nachbarn, also B und C. Router B beispielsweise erhält dann auf Link 1 den Distanz-Vektor von A. Da er den soeben passierten Link mitberücksichtigen muss, erhält er alle Distanzen um 1, woraus der Distanz-Vektor $A = 1$ entsteht. Diesen vergleicht er mit seiner Tabelle und da diese noch keinen Eintrag über Router A enthält fügt er hinzu, dass Router A über Link 1 mit den Kosten 1 zu erreichen ist. Seine Tabelle sieht dann folgendermassen aus:

Von B nach	Link	Kosten
B	lokal	0
A	1	1

Bei Router C führt dies zu folgender äquivalenter Tabelle:

Von C nach	Link	Kosten
C	lokal	0
A	4	1

Da sich sowohl bei Router B, als auch bei Router C die Tabellen geändert haben, schicken diese ihre aktualisierten Tabellen wiederum per broadcast an alle Nachbarn. Dies führt dazu, dass Router A von B den Distanz-Vektor $B = 0; A = 1$ über Link 1 erhält. Nachdem er die Werte erhöht hat, vergleicht er die Werte ($B = 1; A = 2$) mit seiner Tabelle und fügt die Informationen für B in seine Tabelle ein. Das Gleiche passiert mit dem Distanz-Vektor von Router C. Anschliessend sieht die Tabelle so aus:

Von A nach	Link	Kosten
A	lokal	0
B	1	1
C	4	1

Die Distanz-Vektoren von Router B und C kommen natürlich auch bei Router D an, was dort zu folgender Tabelle führt:

Von D nach	Link	Kosten
D	lokal	0
B	2	1
C	3	1
A	3	2

Der Linkeintrag zum Router A kann auch 2 sein, wenn die Tabelle von Router B zuerst angekommen wäre. Hier ist beides möglich, da der Weg von D nach A, egal ob über B oder C, gleich lang ist. Router D schickt jetzt wiederum seine Tabelle an Router B und C:

Von C nach	Link	Kosten	Von B nach	Link	Kosten
C	lokal	0	B	lokal	0
A	4	1	A	1	1
B	3	2	C	2	2
D	3	1	D	2	1

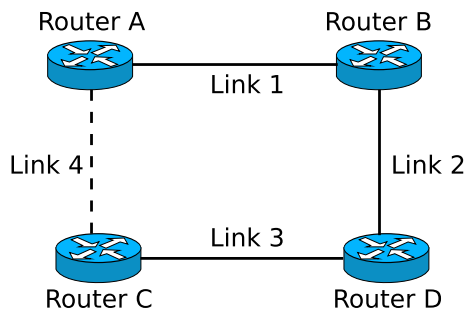
Router B und C senden darauf ihre Tabellen über broadcast, was zu einer letzten Änderung in der Tabelle von Router A führt:

Von A nach	Link	Kosten
A	lokal	0
B	1	1
C	4	1
D	4	2

Der Link zu Ziel D kann hier auch wieder 1 betragen, wenn die Tabelle von Router B zuerst angekommen wäre. Diese Tabelle wird zwar jetzt auch nochmal per broadcast an Router B und C geschickt, doch da dies zu keinen Änderungen in deren Tabellen führt, werden keine weiteren broadcasts durchgeführt. Alle Routing-Tabellen sind jetzt vollständig ausgefüllt und jeder Router ist von jedem anderen Router erreichbar. Diesen Zustand bezeichnet man als Konvergenz. Trotzdem senden die Router auch weiterhin alle 30 Sekunden ihre Tabellen, um zu signalisieren, dass sie noch erreichbar sind, was jedoch im konvergenten Zustand zu keinen Ketten-broadcasts führt, da sich in den Tabellen nichts mehr ändert.

2.2 Ausfall eines Links

Wenn es zwischen zwei Routern zum Ausfall eines Links kommt, so merken die beiden betroffenen Router dies erst, wenn die regelmässigen broadcasts auf diesem Link für länger als 30 Sekunden ausbleiben. Im folgenden Beispiel ist der Link 4 zwischen Router A und C ausgefallen:



Sobald Router A und C den Ausfall bemerkt haben, setzen sie die Kosten für alle Wege, die über diesen Link laufen, auf unendlich (*infinity*):

Von A nach	Link	Kosten	Von C nach	Link	Kosten
A	lokal	0	C	lokal	0
B	1	1	A	4	inf
C	4	inf	B	3	2
D	4	inf	D	3	1

Wenn Router A dann seinen Distanz-Vektor broadcastet, führt dieser in Router B zu keiner Änderung, da die Wege von A nach C bzw. D zwar unendliche Kosten haben, doch Router B erreicht diese über Link 2.

Anders sieht es bei Router D aus. Wenn er den Distanz-Vektor von Router C empfängt, setzt er seine Kosten für A auf unendlich. Dies liegt daran, dass sein schnellster Weg nach A über Link 3 geht, doch genau von diesem Link bekommt er jetzt die Information, dass Router A über diesen Link nicht mehr zu erreichen ist. Seine Tabelle sieht demnach wie folgt aus:

Von D nach	Link	Kosten
D	lokal	0
B	2	1
C	3	1
A	3	inf

Da sich die Tabelle von Router D geändert hat, macht er einen Broadcast. Dieser bewirkt allerdings keine Änderung in den Tabellen von Router C und B. Obwohl sich die Routing-Tabellen in einem inkonsistenten Zustand befindet, stellt sich keine Konvergenz ein. Erst wenn Router B wieder seinen periodischen Broadcast macht, treten die erforderlichen Änderungen in Kraft. Zuerst werden die Wege in den Tabellen von Router A und D korrigiert:

Von A nach	Link	Kosten	Von D nach	Link	Kosten
A	lokal	0	D	lokal	0
B	1	1	B	2	1
C	1	3	C	3	1
D	1	2	A	2	2

Anschliessend wird durch den Broadcast von Router D die Tabelle von Router C korrigiert:

Von C nach	Link	Kosten
C	lokal	0
A	3	3
B	3	2
D	3	1

Jetzt befinden sich alle Routing-Tabellen wieder in einem konsistenten Zustand und Konvergenz ist eingetreten. Der ausgefallenen Link 4 wird durch die Links

1, 2 und 3 überbrückt. Allerdings wird hier auch eine der Schwachstellen von RIP deutlich:

Es dauert durchaus etwas länger bis sich nach dem Ausfall eines Links erneut Konvergenz einstellt. In diesem Fall musste beispielsweise zweimal gewartet werden bis die periodischen Broadcasts eine Änderung in den Routing-Tabellen propagierten.

2.3 Der Bouncing-Effekt

Neben der trägen Anpassung von RIP an Änderungen in der Netzwerktopologie gibt es noch zwei weitere unerwünschte Effekte, die hierbei auftreten. Der erste wird als *Bouncing-Effekt* bezeichnet (*bouncing* = hüpfen). Er lässt sich gut an dem letzten Beispiel erklären. Sobald Router C den Ausfall von Link 4 bemerkt hat, sieht seine Routing-Tabelle folgendermassen aus:

Von C nach	Link	Kosten
C	lokal	0
A	4	inf
B	3	2
D	3	1

Wenn Router C diese Tabelle sofort per broadcast weitersendet, entspricht das obigem Verlauf. Wenn jedoch Router D vorher seinen periodischen broadcast macht, ändern sich die Routing-Tabellen ungünstig:

Von C nach	Link	Kosten	Von D nach	Link	Kosten
C	lokal	0	D	lokal	0
A	3	3	B	2	1
B	3	2	C	3	1
D	3	1	A	3	2

Bei näherem Hinsehen erkennt man, dass sich eine Schleife gebildet hat:

Wenn ein Paket bei Router C mit dem Ziel A ankommt, so wird es über Link 3 an Router D weitergeleitet. Dieser jedoch leitet das Paket wieder über Link 3 zurück an Router C. Das Paket fängt also an, zwischen den Routern C und D zu pendeln, bis seine TTL vorbei ist. Dies bezeichnet man als Bouncing-Effekt. Wenn jetzt Router C seine Tabelle wieder broadcastet ändert Router D die Distanz zu Router A auf 4. Beim nächsten Broadcast von Router B merkt Router D dann erst, dass der Weg über Link 2 kürzer ist und ändert dies in seiner Tabelle. Nach einem weiteren broadcast von Router D berichtigt schliesslich auch Router C seine Tabelle und das Netzwerk befindet sich wieder in einem konsistenten Zustand. Auch hier dauert es also durchaus einige Broadcasts bis alle Änderungen durch das Netzwerk propagiert wurden und das Netz konvergiert.

2.4 Counting to Infinity

Das zweite Problem wird als *Counting to Infinity* bezeichnet und kann an dem vorigen Zustand erklärt werden. Wenn Link 2 auch noch ausfällt, bevor Router

Broadcasten kann (was ja bewirkte, dass der Pfad zu Router A umgeleitet und die Schleife aufgebrochen wurde), so führen Router C und D weiter abwechselnd Broadcasts durch und die Kosten für A werden in zweier Schritten hochgezählt. Dies liegt daran, dass beide Router meinen, Router A wäre über den zwischen ihnen liegenden Link 3 zu erreichen.

Da die beiden Router C und D jetzt allerdings vom restlichen Netz getrennt sind, stellt sich nicht wie beim Bouncing-Effekt nach mehreren Broadcasts Konvergenz ein. Im Gegenteil, beide Router zählen die Kosten für A bis ins Unendliche hoch. Dies lässt sich nur vermeiden, in dem man dem Infinity-Wert eine feste Zahl zuordnet, so dass nur bis zu diesem Wert hochgezählt wird. Allerdings beschränkt dies natürlich auch die Netzgröße, da weiter entfernte Router nie erreicht werden können, weil ihre Kosten „infinity“ getragen. Bei RIP beträgt infinity meistens 16 und für die normalen Kosten stehen 0 bis 15 zur Verfügung.

2.5 Split Horizon und Triggered Updates

Um die Eigenschaften von RIP zu verbessern wurden verschiedenen Verbesserungen implementiert:

Bei *Split Horizon* liegt folgende Überlegung zugrunde: Wenn ein Router A Pakete zum Router C über Router B schickt, macht es keinen Sinn beim Mitteilen der Routing-Tabelle an B die Entfernung zu Router C mitzusenden, da der kürzeste Weg eh über Router B führt. Um diese Technik umzusetzen, werden in den Distanz-Vektoren auf den verschiedenen Links einfach alle Einträge weggelassen, die sowieso über diesen Link führen. In einer als „split horizon with poisonous reverse“ bezeichneten, verbesserten Form werden die Einträge nicht weggelassen, sondern ihre Kosten auf infinity gesetzt. Dies bewirkt, dass alle Schleifen, die aus zwei Knoten bestehen sofort eliminiert werden.

Triggered Update dient dazu, die Änderungen im Netz schneller zu verarbeiten. Dazu werden die Routing-Tabellen bei Änderungen sofort via Broadcast weitergesendet, ohne auf den Timer für die periodischen Broadcasts zu warten. Damit würde beispielsweise, bei dem Beispiel zum Bouncing-Effekt, die Wahrscheinlichkeit, dass der Broadcast von Router D vor dem von Router C einsetzt, erheblich minimiert.

2.6 Nachrichtenformat

Die folgende Grafik zeigt das Nachrichtenformat von RIP Version 1:

0		7		15		31
Befehl		Version		null		
Adressfamilie				null		
IP-Adresse						
null						
null						
Metrik						

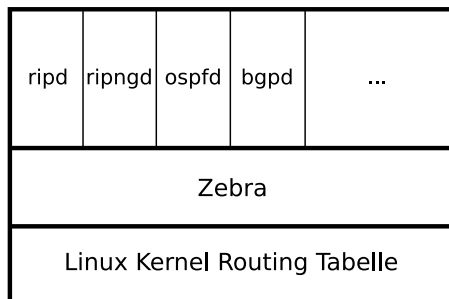
Die Nachrichten von RIP beginnen mit einem Befehlsbezeichner. Hier spielen eigentlich nur das *request*-Kommando mit einem Wert von 1 und das *response*-Kommando mit einem Wert von 2 eine Rolle. Anschliessend folgt ein Feld für die Version des RIP-Protokolls, die entweder 1 oder 2 betragen kann. Die folgenden 2 Bytes werden mit Nullen aufgefüllt.

Nun folgt eine Liste mit den Adress-Metrik-Paaren, dem eigentlichen Distanz-Vektor. Jeder Eintrag besteht aus 20 Bytes. Die ersten 2 Bytes sind für einen Adressfamilien-Bezeichner reserviert. In der Praxis wird hier allerdings nur die 2 für das IP-Protokoll verwendet. Andere Bezeichner werden ignoriert. Es folgen 2 Bytes mit Nullen. Dann stehen 4 Bytes für die IP-Adresse des Ziels zur Verfügung, gefolgt von wiederum 8 Bytes mit Nullen. Die letzten 4 Bytes jedes Eintrags enthalten die Kosten oder Metrik. Trotz dem 32-bit-Wertebereich werden hier nur die Werte 0 bis 16 genutzt, wobei 16 für infinity steht.

Das normale Verhalten des RIP-Protokolls besteht in dem Senden von *response*-Nachrichten. Nur beim Start eines Routers sendet dieser eine *request*-Message, um nicht auf den nächsten periodischen Broadcast der Nachbar-Router warten zu müssen.

3 Routing mit RIP unter Linux

Für das Routing im Allgemeinen steht eine komplette Routingsuite namens *Quagga* zur Verfügung, die aus dem ehemaligen *Zebra*-Projekt hervorgegangen ist. Der Vorteil von Quagga ist die flexible Integration verschiedenster Routing-Protokolle:



Die Schnittstelle zur Routing-Tabelle des Kernels bildet der Zebra-Prozess. Unter ihm sind die Prozesse angeordnet, die die verschiedenen Routing-Protokolle implementieren: ripd (RIP), ripngd (RIP-next-generation), ospfd (OSPF), bgpd (BGP) u.s.w. Diese Architektur ermöglicht es einerseits, Unterstützung für neue Routing-Protokolle hinzuzufügen. Andererseits können auf diese Art und Weise mehrere Routing-Protokolle auf einem Router betrieben werden, etwa wenn sich dieser an der Grenze zwischen IGP- und EGP-Protokollen befindet. Zebra koordiniert dabei das Zusammenspiel der einzelnen Protokolle. Die einzelnen Routing-Prozesse holen sich die aktuellen Routing-Informationen über Zebra

vom Kernel. Neu gewonnene Routing-Informationen (durch Kommunikation mit Nachbarroutern über das jeweilige Protokoll) bringen sie wieder über Zebra in die Routing-Tabelle des Kernels ein.

3.1 Konfiguration und Start von Zebra

Bevor die einzelnen Routing Prozesse gestartet werden können, muss Zebra gestartet werden, um die Verbindung zur Routing-Tabelle des Kernels herzustellen. Zebra wird über eine Datei namens `zebra.conf` konfiguriert:

```
1  ! -*- zebra -*-
2  !
3  ! zebra sample configuration file
4  !
5  ! $Id: zebra.conf.sample,v 1.1.1.1 2002/12/13 20:15:30 paul Exp $
6  !
7  hostname Router
8  password xxxx
9  enable password xxxx
10 log file zebra.log
```

Zeilen, die mit einem Ausrufezeichen beginnen werden als Kommentare interpretiert. Hier wurde der Hostname des Routers angegeben, sowie ein Passwort für den Zugang zu Zebra, dass darüberhinaus noch aktiviert wurde. Eine Log-Datei erleichtert die Fehlersuche.

Man kann ebenfalls statische Routen definieren, Interfaces einrichten, etc. Dies ist allerdings nur in komplizierteren Szenarien nötig.

```
root:~# zebra -f config_file -d
```

Dies startet Zebra im Hintergrund. Wenn eine Standardkonfigurationsdatei verwendet wird, kann die Pfadangabe auch weggelassen werden. Nachdem Zebra gestartet ist, kann man eine `telnet`-Verbindung über Port 2601 aufbauen und Zebra nach Eingeben des Passworts interaktiv konfigurieren:

```
root:~# telnet localhost 2601
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is quagga (version 0.96.4).
Copyright 1996-2002 Kunihiro Ishiguro.
```

```
User Access Verification
```

```
Password:
Router>
```

Die möglichen Befehle kann man sich mit `list` ausgeben lassen. Alle vorgenommenen Einstellungen können anschliessend in der Konfigurationsdatei gespeichert werden, womit ein erneutes Einstellen von Zebra beim nächsten Start entfällt.

3.2 Konfiguration und Start von ripd

Genau wie Zebra, wird auch der RIP-Dämon `ripd` über eine Konfigurationsdatei namens `ripd.conf` gesteuert. Sie könnte zum Beispiel wie folgt aussehen:

```
1  ! -*- rip -*-
2  !
3  ! RIPd sample configuration file
4  !
5  ! $Id: ripd.conf.sample,v 1.1.1.1 2002/12/13 20:15:30 paul Exp $
6  !
7  hostname Router
8  password xxxx
9  router rip
10 network 10.0.1.0/24
11 network 10.0.2.0/24
12 network eth0
13 log file ripd.log
```

Nachdem wieder der Hostname und das Passwort für die telnet-Verbindung angegeben wurden, müssen noch die Netzwerke angegeben werden, auf denen RIP aktiviert werden soll. In diesem Fall wird RIP für die Netzwerke 10.0.1.0 und 10.0.2.0 mit der Netzmaske 255.255.255.0 und für das an das Interface `eth0` angeschlossene Netz aktiviert. Auch hier ist das Angeben einer Log-Datei hilfreich.

Gestartet wird der `ripd` ähnlich, wie Zebra:

```
root:~# ripd -f config_file -d
```

Auch hier kann die Konfigurationsdatei weggelassen werden. Eine telnet-Verbindung zu `ripd` wird über den Port 2602 aufgebaut:

```
root:~# telnet localhost 2602
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is quagga (version 0.96.4).
Copyright 1996-2002 Kunihiro Ishiguro.
```

```
User Access Verification
```

```
Password:
Router>
```

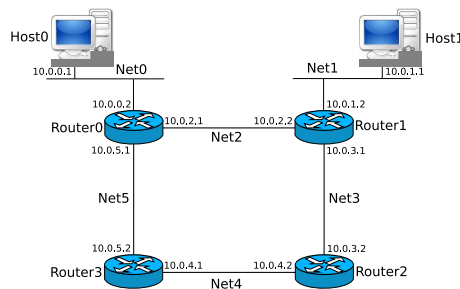
Die Konfiguration gestaltet sich ähnlich der von Zebra und auch hier können die vorgenommenen Änderungen in der Konfigurationsdatei gespeichert werden. Sobald `ripd` gestartet wurde, werden Routinginformationen in den Netzwerken, für die RIP aktiviert wurde, ausgetauscht. Die Routing-Tabellen benachbarter Router werden an den betreffenden Interfaces auf Port 520 empfangen und verarbeitet. Dabei wird das in Abschnitt 2.6 beschriebene Nachrichtenformat verwendet.

4 Beispielszenario für RIP in VNUML

Für das folgende Beispielszenario zur Demonstration von RIP wird VNUML 1.6 benutzt, um ein Netzwerk mit Hosts und Routern zu simulieren. Die Router haben zu Beginn nur Informationen über die direkt angeschlossenen Netze. Erst nach dem Starten von Zebra und `ripd` werden die Routing-Tabellen vollständig gefüllt und alle Netze sind erreichbar.

4.1 Simuliertes Netzwerk

Das der Simulation zugrundeliegende Netzwerk sieht wie folgt aus:



Es besteht aus 4 Routern, an die 2 Hosts angeschlossen sind. Auf mehr Hosts wurde lediglich aus Performancegründen verzichtet. Die Simulation kann jedoch leicht erweitert werden.

Die VNUML-Konfigurationsdatei für obiges Szenario befindet sich in Anhang A. Die einzigen Routing-Informationen sind die default-Einträge für die beiden Hosts. Die Router verfügen über keinerlei Informationen. Auf den Routern muss `<forwarding type="ip"/>` eingeschaltet werden, um Routing zu aktivieren.

4.2 Start der Simulation

Die Simulation wird gestartet mit:

```
user:~$ vnumlparser.pl -t rip.xml
```

Jetzt werden die virtuellen Maschinen gebootet, wobei für `Host0`, `Host1`, `Router0` und `Router1` ein Terminalfenster geöffnet wird.

Nachdem das Szenario komplett gestartet ist, kann man sich bei `Host0` anmelden und versuchen `Host1` mit der IP-Adresse `10.1.1.1` zu pinggen. Dies schlägt natürlich fehl:

```
Host0:~# ping 10.0.1.1 -c1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Net Unreachable

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Der Grund hierfür ist, dass der ping-request durch den default-Eintrag zwar bis zu Router0 (10.0.0.2) vordringt, dieser jedoch nichts über den Weg zu Net1 (10.0.1.0) weiss. Man sieht dies, wenn man sich in Router0 einloggt und sich dessen Routing-Tabelle ausgeben lässt:

```
Router0:~# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
10.0.5.0         *               255.255.255.0  U        0      0      0 eth3
10.0.0.0         *               255.255.255.0  U        0      0      0 eth1
192.168.3.0     *               255.255.255.0  U        0      0      0 eth0
10.0.2.0         *               255.255.255.0  U        0      0      0 eth2
```

Diese enthält nach dem Start nur die Einträge über die direkt angeschlossenen Netze 10.0.0.0, 10.0.2.0 und 10.0.5.0. Der Eintrag für das Netz 192.168.3.0 kann ignoriert werden, da es sich hierbei nur um das Management-Interface der virtuellen Maschine handelt, dass nichts mit dem simulierten Netzwerk zu tun hat.

4.3 Start von Zebra und ripd

Damit RIP in Aktion tritt müssen Zebra und ripd auf den Routern gestartet werden. Hierbei spielt die Konfigurationsdatei des ripd eine entscheidende Rolle:

```
1  hostname ripd
2  password xxxx
3  !
4  router rip
5  network 10.0.2.0/24
6  network 10.0.3.0/24
7  network 10.0.4.0/24
8  network 10.0.5.0/24
9  redistribute connected
10 !
11 log stdout
```

In den Zeilen 5 bis 8 werden die Netze angegeben, auf denen RIP aktiviert werden soll. Auf den zugehörigen Interfaces werden dann RIP-Nachrichten empfangen und gesendet. Wichtig ist jedoch auch Zeile 9. Sie bewirkt, dass auch Informationen über angeschlossene Netze weitergegeben werden sollen, auf denen RIP nicht aktiviert ist. Dies wäre für Router0 beispielsweise Net0. Für dieses Netz ist RIP deaktiviert (es werden also keine RIP-Nachrichten von diesem Netz empfangen und in dieses Netz gesendet), im Distanzvektor von Router0 wird Net0 aber mitgeschickt, damit die anderen Router wissen, dass dieses Netz über Router0 zu erreichen ist.

Mit

```
user:~$ vnumlparser.pl -x ripd@rip.xml
```

werden Zebra und `ripd` auf den Routern gestartet. Beim Start von RIP wird nicht erst auf die periodischen Updates gewartet. Es werden durch `request`-Nachrichten direkt die Informationen von den Nachbarroutern angefordert. Dadurch stellt sich die Konvergenz sehr schnell ein. So ist schon nach sehr kurzer Zeit die Routing-Tabelle von `Router0` komplett mit den Informationen ausgefüllt, um alle Netze zu erreichen:

```
Router0:~# route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
10.0.4.0         10.0.5.2        255.255.255.0   UG    2     0     0 eth3
10.0.5.0         *               255.255.255.0   U     0     0     0 eth3
10.0.0.0         *               255.255.255.0   U     0     0     0 eth1
192.168.3.0     *               255.255.255.0   U     0     0     0 eth0
10.0.1.0         10.0.2.2        255.255.255.0   UG    2     0     0 eth2
10.0.2.0         *               255.255.255.0   U     0     0     0 eth2
10.0.3.0         10.0.2.2        255.255.255.0   UG    2     0     0 eth2
```

Nun funktioniert auch der `ping` von `Host0` zu `Host1`:

```
Host0:~# ping 10.0.1.1 -c1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=62 time=101 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 101.515/101.515/101.515/0.000 ms
```

4.4 Ausfall einer Verbindung

Um den Ausfall einer Verbindung zu simulieren und das Verhalten von RIP daraufhin zu untersuchen kann man einfach ein Interface deaktivieren:

```
Router0:~# ifconfig eth2 down
```

Hier wurde Interface 2 von Router 0 deaktiviert, um die Verbindung zwischen Router 0 und Router 1 zu trennen.

Wenn man daraufhin die Routing-Tabelle von `Router0` betrachtet, fällt auf, dass die Einträge nahezu direkt angepasst wurden und der Weg nach `Net1` jetzt über `Router3` (10.0.5.2) führt:

```
Router0:~# route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
10.0.4.0         10.0.5.2        255.255.255.0   UG    2     0     0 eth3
10.0.5.0         *               255.255.255.0   U     0     0     0 eth3
10.0.0.0         *               255.255.255.0   U     0     0     0 eth1
192.168.3.0     *               255.255.255.0   U     0     0     0 eth0
10.0.1.0         10.0.5.2        255.255.255.0   UG    4     0     0 eth3
10.0.2.0         10.0.5.2        255.255.255.0   UG    4     0     0 eth3
10.0.3.0         10.0.5.2        255.255.255.0   UG    3     0     0 eth3
```

Trotzdem schlägt der `ping` von `Host0` zu `Host1` fehl:

```

Host0:~# ping 10.0.1.1 -c1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

Dies liegt daran, dass Router0 den Ausfall des Interfaces zwar direkt erkannt hat, Router1 jedoch geht immer noch davon aus, dass die Verbindung intakt ist. So gelangt der ping-request zwar bis zu Host1, der ping-reply allerdings nimmt nach wie vor den Weg über Net2 und kommt dadurch nicht an. Erst wenn die periodischen Broadcasts von Router0 über Net2 ausbleiben und ein Timeout einsetzt, erkennt er den Ausfall der Verbindung.

Dies kann man nachvollziehen, wenn man sich unter Router1 über telnet beim ripd anmeldet, um die RIP-Informationen abzurufen:

```

Router1:~# telnet localhost 2602
...
ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

      Network      Next Hop      Metric From      Tag Time
R(n) 10.0.0.0/24   10.0.2.1      2 10.0.2.1       0 02:06
C(r) 10.0.1.0/24   0.0.0.0       1 self           0
C(i) 10.0.2.0/24   0.0.0.0       1 self           0
C(i) 10.0.3.0/24   0.0.0.0       1 self           0
R(n) 10.0.4.0/24   10.0.3.2      2 10.0.3.2       0 02:18
R(n) 10.0.5.0/24   10.0.2.1      2 10.0.2.1       0 02:06
C(r) 192.168.3.0/24 0.0.0.0       1 self           0

```

In der ersten Zeile der Tabelle erkennt man, dass als next-Hop zu Net0 immer noch Router0 (10.0.2.1) eingetragen ist. Erst in 2 Minuten und 6 Sekunden, wie man der letzten Spalte entnehmen kann, wird ein Ausfall der Verbindung an diesem Interface angenommen. Wenn dieser Timeout eintritt, werden die Kosten für alle Verbindungen über dieses Interface mit 16 (infinity) beziffert:

```

ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

      Network      Next Hop      Metric From      Tag Time
R(n) 10.0.0.0/24   10.0.2.1     16 10.0.2.1       0 02:00
C(r) 10.0.1.0/24   0.0.0.0       1 self           0
C(i) 10.0.2.0/24   0.0.0.0       1 self           0
C(i) 10.0.3.0/24   0.0.0.0       1 self           0
R(n) 10.0.4.0/24   10.0.3.2      2 10.0.3.2       0 02:50
R(n) 10.0.5.0/24   10.0.2.1     16 10.0.2.1       0 02:00
C(r) 192.168.3.0/24 0.0.0.0       1 self           0

```

Jetzt kann durch ein periodisches Update von Router2 eine neue Route zum Net0 in der Routing-Tabelle von Router1 eingetragen:

```

ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

      Network      Next Hop      Metric From      Tag Time
R(n) 10.0.0.0/24   10.0.3.2      4 10.0.3.2       0 02:36
C(r) 10.0.1.0/24   0.0.0.0       1 self            0
C(i) 10.0.2.0/24   0.0.0.0       1 self            0
C(i) 10.0.3.0/24   0.0.0.0       1 self            0
R(n) 10.0.4.0/24   10.0.3.2      2 10.0.3.2       0 02:36
R(n) 10.0.5.0/24   10.0.3.2      3 10.0.3.2       0 02:36
C(r) 192.168.3.0/24 0.0.0.0       1 self            0

```

Das Netz befindet sich nun erneut im Zustand der Konvergenz und Net1 ist von Host0 wieder zu erreichen:

```

Host0:~# ping 10.0.1.1 -c1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=60 time=2.52 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.522/2.522/2.522/0.000 ms

```

Allerdings wurden bei diesem simulierten Linkausfall auch die Nachteile von RIP deutlich. Es dauerte durchaus einige Minuten bis das Netz wieder in einem konsistenten Zustand war. Man kann diese Zeit zwar verkürzen, indem man den Timeout herabsetzt, doch dies führt wiederum zu einem höheren Verkehr, da dann die periodischen Broadcasts ebenfalls öfters durchgeführt werden müssen.

A VNUML-Konfigurationsdatei

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
3 <vnuml>
4   <global>
5     <version>1.6</version>
6     <simulation_name>rip</simulation_name>
7     <ssh_version>1</ssh_version>
8     <ssh_key>~/ssh/identity.pub</ssh_key>
9     <automac/>
10    <vm_mgmt type="net" network="192.168.3.0" mask="24">
11      <mgmt_net sock="/var/run/uml-utilities/uml_switch.ctl" hostip="192.168.3.1"/>
12    </vm_mgmt>
13    <default_filesystem type="cow">
14      /usr/local/share/vnuml/filesystems/root_fs
15    </default_filesystem>
16    <default_kernel>/usr/local/share/vnuml/kernels/linux</default_kernel>
17  </global>
18
19  <net name="Net0" mode="uml_switch" />
20  <net name="Net1" mode="uml_switch" />
21  <net name="Net2" mode="uml_switch" />
22  <net name="Net3" mode="uml_switch" />
23  <net name="Net4" mode="uml_switch" />
24  <net name="Net5" mode="uml_switch" />
25
26  <vm name="Host0">
27    <boot>
28      <con0>xterm</con0>
29      <xterm>gnome-terminal,-t,-x</xterm>
30    </boot>
31    <if id="1" net="Net0">
32      <ipv4>10.0.0.1</ipv4>
33    </if>
34    <route type="inet" gw="10.0.0.2">default</route>
35  </vm>
36  <vm name="Host1">
37    <boot>
38      <con0>xterm</con0>
39      <xterm>gnome-terminal,-t,-x</xterm>
40    </boot>
41    <if id="1" net="Net1">
42      <ipv4>10.0.1.1</ipv4>
43    </if>
44    <route type="inet" gw="10.0.1.2">default</route>
45  </vm>
46
47  <vm name="Router0">
48    <boot>
49      <con0>xterm</con0>
50      <xterm>gnome-terminal,-t,-x</xterm>
51    </boot>
52    <if id="1" net="Net0">
53      <ipv4>10.0.0.2</ipv4>
54    </if>
55    <if id="2" net="Net2">
```

```

56     <ipv4>10.0.2.1</ipv4>
57 </if>
58 <if id="3" net="Net5">
59     <ipv4>10.0.5.1</ipv4>
60 </if>
61 <forwarding type="ip"/>
62 <exec seq="ripd" type="verbatim">zebra -d</exec>
63 <exec seq="ripd" type="verbatim">ripd -d</exec>
64 </vm>
65 <vm name="Router1">
66 <boot>
67     <con0>xterm</con0>
68     <xterm>gnome-terminal,-t,-x</xterm>
69 </boot>
70 <if id="1" net="Net1">
71     <ipv4>10.0.1.2</ipv4>
72 </if>
73 <if id="2" net="Net2">
74     <ipv4>10.0.2.2</ipv4>
75 </if>
76 <if id="3" net="Net3">
77     <ipv4>10.0.3.1</ipv4>
78 </if>
79 <forwarding type="ip"/>
80 <exec seq="ripd" type="verbatim">zebra -d</exec>
81 <exec seq="ripd" type="verbatim">ripd -d</exec>
82 </vm>
83 <vm name="Router2">
84 <if id="1" net="Net3">
85     <ipv4>10.0.3.2</ipv4>
86 </if>
87 <if id="2" net="Net4">
88     <ipv4>10.0.4.2</ipv4>
89 </if>
90 <forwarding type="ip"/>
91 <exec seq="ripd" type="verbatim">zebra -d</exec>
92 <exec seq="ripd" type="verbatim">ripd -d</exec>
93 </vm>
94 <vm name="Router3">
95 <if id="1" net="Net5">
96     <ipv4>10.0.5.2</ipv4>
97 </if>
98 <if id="2" net="Net4">
99     <ipv4>10.0.4.1</ipv4>
100 </if>
101 <forwarding type="ip"/>
102 <exec seq="ripd" type="verbatim">zebra -d</exec>
103 <exec seq="ripd" type="verbatim">ripd -d</exec>
104 </vm>
105 </vnuml>

```