

**Simulation des
Routing Information Protocol
und des
Routing Information Protocol
with Minimal Topology Information
unter VNUML**

**Studienarbeit
AG Rechnernetze und Rechnerarchitektur
SS 2005**

Leif Thorvald Franker
franker@uni-koblenz.de
<http://www.uni-koblenz.de/~franker/>

Universität Koblenz – Landau
Abteilung Koblenz

Prof. Dr. Christoph Steigner
Dipl. -Inf. Harald Dickel

25.08.2005

1.	Einleitung.....	5
2.	Routingprotokolle	6
2.1	RIP – Routing Information Protocol.....	6
2.1.1	Split Horizon.....	8
2.1.2	Split Horizon with poisoned reverse.....	8
2.1.3	Triggered updates	9
2.2	Routing Information Protocol with Minimal Topology Information	9
2.2.1	X-Kombination	10
2.2.2	Y-Kombination	11
3.	Virtual Network User Mode Linux - VNUML.....	12
3.1	Installation von VNUML.....	12
3.2	Arbeiten mit VNUML	13
3.3	Syntax und Semantik	14
3.4	Erstellen einer neuen Simulation	15
3.4.1	Designphase	16
3.4.2	Implementierungsphase	16
3.4.3	Ausführungs-/ Testphase	17
3.4.4	Szenario herunterfahren.....	18
4.	Zebra & Quagga.....	19
4.1	Arbeiten mit Quagga.....	20
5.	Diagnose-Hilfsmittel.....	23
5.1	Ping	23
5.2	Traceroute	24
5.3	Show ip route	25
5.4	Tcpdump	26
5.5	Ethereal	27

6.	Szenario 1	27
6.1	Vergleich der Routing Protokolle	28
7.	Szenario 2 – CTI	36
7.1	Szenario Durchlauf unter Verwendung von RIP	38
7.2	Szenario Durchlauf unter Verwendung von RIP-MTI	42
7.3	Erweiterung des Szenario um eine Backupverbindung	48
8.	Ausblick	52
9.	Anhang	53
9.1	Szenario 1	53
9.2	Szenario 2	56
9.3	Szenario 2 – Neue Adressierung der Schnittstellen und Firewall	61
	Literaturverzeichnis	66

Abbildungsverzeichnis

Abb. 2.1.1: Kosten (Hops) unter RIP.....	7
Abb. 2.2.1: Routingschleife unter RIP.....	10
Abb. 2.2.1.1: X-Kombination einer Source Loop.....	11
Y-Kombination einer Source Loop.....	11
Abb. 3.4.4.1: Drei Phasen Modell – VNUML Arbeitszyklus.....	19
Abb. 4.1.1: Beispiel 1.....	20
Abb. 6.1: Szenario 1.....	28
Abb. 6.1.1: Ethereal Bildschirmausdruck.....	33
Abb. 7.1: Szenario 2.....	38
Abb. 7.1.1: Durchlauf des CTI Ereignisses (Metrik 4 bis 7).....	41
Abb. 7.1.2: Durchlauf des CTI Ereignisses (Metrik 7 bis 15).....	41
Abb. 7.1.3: Durchlauf des CTI Ereignisses (Metrik 15 bis 16).....	41
Abb. 7.2.1: Szenario 2 – RIP MTI	42
Abb. 7.2.2: Szenario 2 – RIP MIT mit Hostanschluss.....	45
Abb. 7.2.3: Ethereal Mitschnitt	47
Abb. 7.2.4: RIP-MIT erkennen und verwerfen der falschen Route.....	47
Abb. 7.3.1: Erweiterung des Szenario um eine Backupverbindung.....	48
Abb. 7.3.2: Subnetz 1 (Umfang 3).....	51
Abb. 7.3.3: Subnetz 2 (Umfang 7).....	51
Abb. 7.3.4: Subnetz 3 (Umfang 6).....	51

1. Einleitung

Thema dieser Arbeit ist ein Vergleich des Routing Information Protokoll (RIP) mit dem erweiterten Protokoll RIP-MTI¹. Das dynamische Routingprotokoll RIP, welches auf dem Distance Vektor Algorithmus basiert, wird in der Praxis vornehmlich in kleinen Netzwerken eingesetzt. Es basiert auf dem Prinzip, das jeder Netzknoten zu Beginn nur seine direkten Nachbarn kennt. Dies hat zur Folge das eine gewisse Zeit in Anspruch genommen wird, bis sich alle Netzknoten untereinander ausgetauscht haben. Die Zeit die benötigt wird, bis alle Routingtabellen im Netzwerk konvergieren, ist folglich im Vergleich zu anderen Routing Protokollen vergleichsweise hoch. Die Implementierung des Protokolls auf den einzelnen Netzwerkrechnern ist aber gerade auf Grund dieser simplen Struktur unkompliziert und schnell auch von Laien zu realisieren.

Der Einsatz des RIP Protokolls ohne Erweiterungen wie split horizon oder triggered update fordert in der Praxis eine Reihe von Nachteilen. So arbeitet Beispielsweise das Protokoll ineffektiv bei der Übertragung von Routing Informationen zwischen den einzelnen Netzwerkrechnern. Oder es treten Routingschleifen auf in denen das Übermittelte Paket feststeckt. Mit dem Ziel diese Nachteile auszuschalten wurde das RIP-MTI Protokoll entwickelt.

Der RIP-MTI Algorithmus setzt in der Entwicklung direkt auf den klassischen RIP Algorithmus auf². Das RIP-MTI Protokoll basiert auf der Idee, ein gegebenes Netzwerk in Hinblick auf genauere Topologieinformationen hin zu analysieren. So erkennt das Protokoll Schleifen die im Netzwerk enthalten sind und merkt sich diese und deren Umfang. Das Haupt Augenmerk liegt dabei auf der Verhinderung des so genannten Counting to Infinity Ereignisses.

Für eine Praxis bezogene Untersuchung der beiden Algorithmen werden mehrer virtuelle Netzwerke erstellt. Unter Linux wird zu diesem Zweck mit dem Tool VNUML³ gearbeitet.

¹ Routing Information Protocol with Minimal Topology Information (RIP-MTI)

² Andreas J. Schmid. RIP-MTI: Minimum-effort loop-free distance vector routing algorithm. Diplomarbeit, Universität Koblenz-Landau

³ Virtual Network User Mode Linux (VNUML)

VNUML bietet die Möglichkeit beliebige virtuelle Netzwerke zu simulieren. Auf diesen virtuellen Netzwerken können die Routing Protokolle realitätsnah ausgeführt und getestet werden.

2. Routingprotokolle

In diesem Kapitel wird das Routing Information Protokoll und die Erweiterung RIP with minimal topology information (RIP-MIT) erläutert. Es soll dem Leser einen kurzen Einstieg in den Themenbereich geben und zum grundsätzlichen Verständnis der Routing Protokolle beitragen.

2.1 RIP – Routing Information Protocol

Das Routing Information Protokoll (RIP) basiert auf dem Distanzvektor-Algorithmus. Es wird verwendet um dynamisch auf Netzwerkroutern die Routingtabellen zu erstellen. Dynamisch bedeutet, das der Weg eines Paketes nicht statisch festgelegt ist. Der Pfad eines Paketes kann sich Aufgrund von temporären Begebenheiten, wie zum Beispiel dem Ausfall einer Netzwerkverbindungen, verändern. Anzumerken ist an dieser Stelle, das der Begriff Route und Pfad, im folgenden Text gleich gesetzt werden kann. RIP wird vornehmlich in den Protokollen IP und IPX eingesetzt.

Ein realer Ablauf unter Verwendung von RIP läuft wie folgt ab. Wird ein Router hochgefahren, kennt er zu Beginn nur das direkt mit ihm Benachbarte Netzwerk. Die Netzwerkadressen zu diesen direkten Nachbarn stehen statisch in seiner lokalen Routingtabelle. Diese Routingtabelle tauscht er im nächsten Schritt mit seinen benachbarten Routern aus. Gleichzeitig fordert er von diesen benachbarten Routern deren Routingtabellen an. Aus diesen neuen Routingtabellen entnimmt sich der Router, die neuen Informationen und fügt sie seiner eigenen Routingtabelle hinzu. Er lernt folglich, welche Netze er über die benachbarten Router erreichen kann und zu welchen Kosten, Pfadlänge dies geschieht.

Um auf Änderungen im Netzwerk reagieren zu können, werden weiterhin in regelmäßigen Abständen die vollständigen Routingtabellen zwischen den einzelnen Routern ausgetauscht. Durch diesen regelmäßigen Austausch der Routing Informationen wird aber auch ein gewisser Traffic im Netzwerk erzeugt. Die Standardvorgabe bei RIP ist alle 30 Sekunden ein Austausch. Hierdurch wird sichergestellt, dass auch ein unbeabsichtigtes Ab- oder Einschalten eines Routers, im Netzwerk registriert wird. Wenn ein Router ausfällt schickt er keine periodischen Updates mehr. Alle Router besitzen einen so genannten Timer (invalidation timer) für jeden Eintrag in ihren Routing Tabellen. Dieser Timer läuft mit der Zeit ab. Ist der Timer abgelaufen wird der Eintrag zum ausgefallenen Router in diesem Fall als unerreichbar markiert. Das bedeutet dass seine Metrik auf 16 gesetzt wird. Kann innerhalb eines erneuten Timerdurchlaufes die Route nicht aktualisiert werden, wird sie aus der Routingtabelle entfernt.

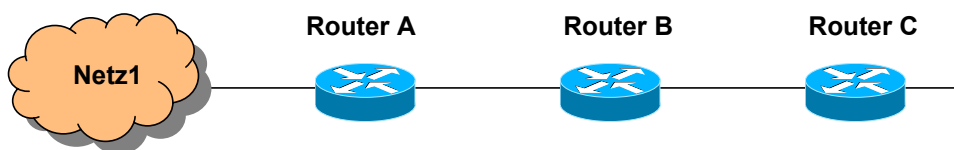


Abb. 2.1.1: Kosten (Hops) unter RIP.

Router A erreicht das Netz 1 mit dem Hop 1, Router B mit Hop 2 über Router A und Router C mit Hop 3 über Router B.

Das Routing Information Protokoll fügt weiterhin jedem Eintrag in einer Routingtabelle einen so genannten „Hop Count“ Vermerk an, die Metrik. Er steht für die Anzahl der Router die entlang eines Pfades zum Zielnetzwerk durchlaufen werden. Dieser Vermerk dient dazu einzelnen Pfaden im Netzwerk eine Gewichtung zu geben und sie vergleichbar zu machen. Wenn ein Netzwerk über zwei verschiedene Pfade zu erreichen ist, so kann über den „Hop Count“ Eintrag, der günstigere Pfad gewählt werden.

Im Gegensatz zu OSPF kennt ein Router der unter RIP läuft Anfangs immer nur seine direkten Nachbarn. Bei Änderungen im Netzwerk dauert es eine gewisse Zeit, bis alle Router wieder eine einheitliche Sicht auf das Netzwerk haben. Mit Erreichen dieses Zustandes spricht man von Konvergenz.

Aufgrund des Umstandes das das Routing Information Protokoll nur seine direkten Nachbarn kennt ist das Protokoll einfach in ein einem bestehendem Netzwerk zu realisieren. Allerdings entstehen hierdurch auch seine beiden größten Nachteile. Zum einen dauert es vergleichsweise lange bis sich alle Router untereinander ausgetauscht haben. Hohe Konvergenzzeiten sind die Folge. Zum anderen gibt es das Counting to Infinity (CTI) Ereignis. Während der Übertragung eines Paketes kann es vorkommen, das der Pfad zu seinem Ziel in einer Schleife endet. Theoretisch würde nun die Metrik unendlich hoch gezählt werden. Zur Verhinderung von Schleifen wurde aus diesem Grund die Anzahl der maximal möglichen Hops, also die Metrik auf eine Länge von 16 begrenzt. Wenn ein Pfad die Länge 16 erreicht, so wird der Eintrag als unerreichbar markiert. RIP erlaubt deshalb nur Netzwerke mit einem maximalen Durchmesser von 15 Routern.

2.1.1 Split Horizon

Ein add-on mit deren Hilfe die Stabilität beim Routing unter RIP erhöht werden kann ist Split-Horizont. Es gibt mehrere Versionen von Split Horizon. Die einfachste Ausführung läuft wie folgt ab. Ein Router sendet eine Routing-Aktualisierung an seine Nachbarn. Dabei schickt er nicht die ganze Routingtabelle mit, sondern nur die Einträge die nicht schon zuvor von dem jeweiligem Nachbarn gelernt worden sind. Diese Methode wird in der Literatur auch gerne als „Belehre nicht den Lehrer“ Prinzip bezeichnet. Wenn in der Praxis die übertragenen Pakete mit den jeweils enthaltenen Routen betrachtet werden, drängt sich allerdings vielmehr der Spruch auf „Belehre den Lernenden nicht mit unnötigen Routen“.

2.1.2 Split Horizon with poisoned reverse

Eine weitere Erweiterung für das Routing Information Protokoll ist Split Horizon with poisoned reverse. In dieser Erweiterung werden wieder alle, einem Router bekannten Routen, übertragen. Die Routen die von den jeweiligen Nachbarn gelernt worden sind, werden aber mit einer Metrik von unendlich markiert. Dieses Verfahren bietet eine höhere Sicherheit gegen Fehler. Ein möglicher Fall in dem es zu

Kollisionen kommen kann, tritt auf, wenn zwei Nachbarn Routen besitzen, die auf den jeweils anderen verweisen. Durch die Markierung mit einer Metrik von unendlich werden diese Routen sofort erkannt. Bei split horizon ohne poisoned reverse wäre in diesem Fall erst eine Zeitüberschreitung abzuwarten.⁴

2.1.3 Triggered updates

Zusätzlich gibt es noch die Option triggered updates im Routing Protokoll zu aktivieren. Im Normalfall aktualisieren sich die Netzwerkrouter in festgelegten Zeitabständen durch update Anfragen. Ist die Option triggered update aktiviert sendet ein Router diese Informationen erst, wenn er selbst ein Update bekommen hat. Das Verfahren bietet zwei Vorteile. Zum einem wird das Netzwerk nicht ständig mit Updateanfragen belastet, zum anderen werden Änderungen in der Netzwerktopologie schneller verbreitet.⁵

2.2 Routing Information Protocol with Minimal Topology Information

RIP-MTI wurde mit dem Ziel entwickelt, das Counting to Infinity Problem zu lösen. RIP-MTI verwendet weiterhin alle Funktionen von RIP, verfügt aber über zwei wesentliche Erweiterungen. Die eine Erweiterung besteht darin, ein grobes Abbild der Netzwerktopologie, aus den empfangenen Updatenachrichten zu erstellen. Die zweite Erweiterung besteht darin das grobe Abbild auf so genannte Source Loops zu untersuchen. Source Loops sind Routingschleifen in denen der Ausgangspunkt des Pfades wieder erreicht wird.

In dem Beispielnetzwerk aus Abbildung 1.2 führt folgende Konstellation zu einer Schleife. Router Berlin möchte ein Paket an Router München schicken. Inzwischen ist aber die Verbindung zwischen Bonn und München unterbrochen. Router Berlin hat als Next Hop Eintrag für die Route nach München den Router Bonn stehen. Wurde inzwischen dem Router Bonn mitgeteilt das der Pfad nach München über

⁴ Tobias Koch Diplomarbeit 2005 – Implementation und Simulation von RIP-MIT

⁵ http://de.wikipedia.org/wiki/Routing_Information_Protocol

Hamburg, Berlin führt entsteht eine Schleife. Das Paket kreist zwischen den drei Routern Hamburg, Berlin und Bonn. Hierbei handelt es sich sogar um eine Source Loop, da der Ausgangspunkt der Schleife immer wieder durchlaufen wird.

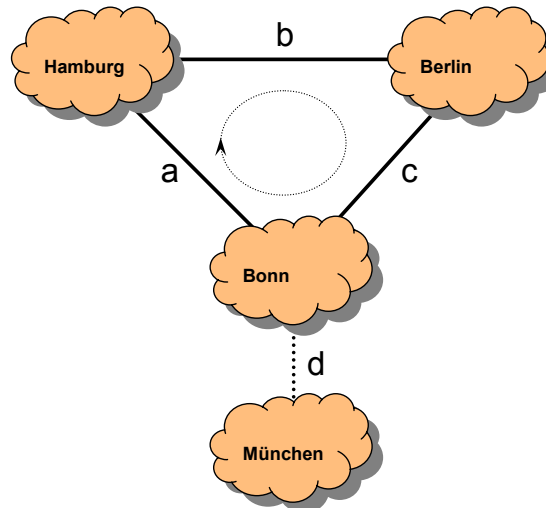


Abb. 2.2.1: Routingsschleife unter RIP

Nach dem Ausfall der Netzwerkverbindung „d“ können Pakete von „Berlin“, Ziele im Netzwerk „München“ nicht mehr erreichen. Es entsteht eine Routingsschleife. In diesem Fall sogar eine Source Loop, da der Ausgangspunkt (Berlin) innerhalb der Schleife liegt.

RIP-MTI kann Aufgrund der Fähigkeit Source Loops zu erkennen, den counting-to-infinity Zustand ausschließen. Ohne Routingsschleifen können bekannter Weise keine CTI entstehen.

Für das Verständnis von RIP-MTI müssen nachfolgend nur noch zwei Schleifenarten betrachtet werden. Durch die Vorgabe das split horizon aktiviert sein muss, können nur noch zwei mögliche Schleifentypen auftreten. Eine so genannte X Kombination aus 5 Routern oder eine Y Kombination aus 3 Routern.

2.2.1 X-Kombination

In einer X-Kombination kann die Netzwerktopologie in zwei Kreise unterteilt werden. Einen oberen und einen unteren Kreis. Aufgrund der Updatenachrichten kann MTI nicht nur einen solchen Kreis erkennen, sondern auch seinen Umfang bestimmen. Wird in der Praxis eine Route erreicht, deren Pfadlänge vom Startpunkt zum Endpunkt und zurück zum Router Bonn kleiner ist als die Summe der beiden Kreisförmigen Netzumfänge, so liegt keine X-Kombination vor.

Eine Formel zur Bestimmung von X Schleifen sähe wie folgt aus:

$$\text{Pfadlänge}_{\text{Startpunkt} \rightarrow \text{Endpunkt}} + \text{Pfadlänge}_{\text{Endpunkt} \rightarrow \text{Bonn}} > \text{Umfang}^{\text{Kreis1}} + \text{Umfang}^{\text{Kreis2}}$$

Natürlich Sprachliche Formulierung zur Bestimmung von Schleifen in X-Kombinationen.

Betrachten wir hierzu ein Beispiel. Router Hamburg möchte ein Paket an den Router München abschicken. Sind alle Router korrekt konfiguriert beträgt die Pfadlänge 2. Wird Router Hamburg aber mitgeteilt das die Pfadlänge zu Hamburg eine Metrik von 7 hat so kann der Router sofort erkennen das es sich dabei um eine Schleife handeln muss. Der Maximal längste Pfad zum Zielrouter kann nicht größer sein als die Summer der beiden Umfänge des oberen und unteren Kreises.

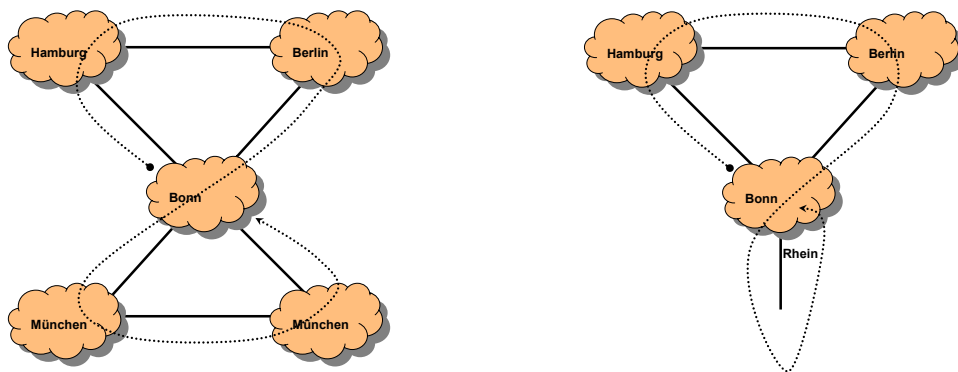


Abb. 2.2.1.1 Source Loops
 Links handelt es sich um eine X-Kombination einer Source Loop. Die aus zwei Schleifen besteht.
 Rechts ist eine Y-Kombination einer Source Loop dargestellt.

2.2.2 Y-Kombination

In der Grafik 2.2 wird eine Beispielnetz für eine Y Kombination gezeigt. In dieser Y-Kombination tritt eine Schleife auf, wenn das an Bonn angehängte Netz „Rhein“ nicht mehr erreichbar ist. Im Normalfall ist das an Bonn angeschlossene Netz von Hamburg oder Berlin aus, mit einer Metrik von 2 zu erreichen. Fällt die Verbindung zwischen Bonn und Rhein weg, kann es zu folgender Konstellation kommen. Der Router Hamburg teilt dem Router Bonn mit das das Netz Rhein mit einer Metrik von 4 erreichbar ist. Bonn muss in diesem Fall herausfinden ob es sich bei der Route um eine Source Loop oder einem regulären Pfad handelt. Die Überprüfung wird ähnlich wie bei einer X Kombination ausgeführt. Router Bonn weiß das das oberen Netz einen Umfang von 3 hat. Ist die mitgeteilte Route zum Ziel größer als der Umfang

muss es sich dabei um eine Schleife handeln. Natürlich sprachlich formuliert würde die Berechnung wie folgt aussehen.

$$\text{Umfang}^{\text{Kreis}} < \text{Pfadlänge}^{\text{Startpunkt} \rightarrow \text{Endpunkt}}$$

Natürlich Sprachliche Formulierung zur Bestimmung von Schleifen in Y-Kombinationen.

3. Virtual Network User Mode Linux - VNUML

VNUML ist ein Instrument mit dessen Hilfe unter Linux Computernetzwerke simuliert werden können. Es basiert auf dem Tool UML⁶. UML gibt einem Linux Anwender die Möglichkeit, auf seinem physikalischen Rechner, mehrere virtuelle Linux Systeme parallel laufen zu lassen. Betrachtet man den Aufbau von VNUML, so fällt einem auf, das es aus zwei Modulen besteht. Es besteht zum einem aus einer Sprache und zum anderem aus einem dazugehörigem Parser. Die Sprache basiert auf XML Tags. Jede XML Datei beschreibt für sich ein Netzwerkszenario. Der Parser wird für den Aufbau und den Ablauf, bzw. die Simulation eines Szenarios benötigt. Der Parser an sich basiert auf einem Perl-Skript.

3.1 Installation von VNUML

Es bietet sich an, die jeweils aktuellste Version des Tools, von der VNUML-Homepage herunter zu laden⁷. Es ist zu beachten, das eine funktionsfähige Installation, mehrer Installations-Dateien benötigt. Generell muss während der Installation der User unter Linux immer als root angemeldet sein. Zu Beginn sollte das eigentliche VNUML Programm installiert werden⁸. Es beinhaltet den Parser und eine Routine, die die weiteren benötigten Module von den Webservern herunter lädt und selbstständig installiert. In der Praxis hat sich gezeigt, das diese Routine nicht immer zuverlässig funktioniert. Auf Grund dessen sollten eventuell auch alle Perl Module von der Internetseite herunter geladen und durch die beigefügten Perl Skripte installiert werden. In dieser Arbeit wurde die Parser-Version 1.5.0-1 vom 21.03.2005 verwendet. Weiterhin muss der Linux Kernel⁹ und das Root Dateisystem¹⁰ von der

⁶ User Mode Linux (UML)

⁷ <http://jungla.dit.upm.es/~vnuml/>

⁸ <http://prdownloads.sourceforge.net/vnuml/vnuml-1.5.0-1.tar.gz?download>

⁹ <http://prdownloads.sourceforge.net/vnuml/linux-2.6.10-1m.tar.bz2?download>

Internetseite herunter geladen werden. Sie müssen anschließend jeweils unter Linux entpackt und die extrahierten Dateien in die zugehörigen Verzeichnisse kopiert werden. Eventuell ist ein Umbenennen des Root Dateisystem und des Parsers nötig. Ansonsten kann es vorkommen das der Parser bei der Ausführung das Root Dateisystem oder den Linux Kernel nicht findet. Die Dateinamen müssen immer mit den in den XML Dateien verwendeten Namen übereinstimmen.

```
Root-Dateisystem: /usr/local/share/vnuml/filesystems/root_fs_tutorial
Kernel: /usr/local/share/vnuml/kernels/linux
```

Positionierung des Root Dateisystems und der Linux Kernels im Dateisystem unter Linux

Vor der Installation empfiehlt es sich zu überprüfen ob genügend freier Speicher bereit steht. VNUML benötigt durch das Anlegen von temporären Dateien während der Installation an die 900 Megabyte. Ist die Installation abgeschlossen werden noch gut 600 Megabyte Speicherplatz belegt. Soll unter VNUML gleichzeitig mit den Protokollen RIP und RIP-MTI gearbeitet werden, so ist durch die doppelte Installation des Root Dateisystem, sogar ein Platzbedarf von insgesamt 1.2 Gigabyte gegeben. Einen ausführlicheren Ablauf der Installation von VNUML kann in der Seminararbeit von Herrn Arndt¹¹ oder in der Diplomarbeit von Herrn Koch¹² nachgelesen werden.

3.2 Arbeiten mit VNUML

Für einen ersten Einstieg in die VNUML Welt bieten sich die der Installation beiliegenden Beispielszenarios an. Im Ordner „Examples“ befinden sich eine Reihe von einfachen Szenarien. Mit Hilfe dieser kann der User die Fähigkeiten und den generellen Umgang mit dem Tool erlernen. Anhand dieser Dateien sollte die Installation des Tools auch auf Vollständigkeit und Korrektheit überprüft werden. In besonderen Fällen treten Fehler in VNUML erst auf, wenn ein Szenario gestartet wird. Im Allgemeinen ist daraufhin der Ursprung eines Fehlers darin zu finden, das ein oder mehrer Module bei der Installation nicht vollständig installiert worden sind. Selbige Module müssen dann nachträglich von Hand installiert werden.

¹⁰ http://prdownloads.sourceforge.net/vnuml/root_fs_tutorial-0.2.3.bz2?download

¹¹ Richard Arndt Seminararbeit 2005 – VNUML - <http://www.uni-koblenz.de/~steigner/seminar-routingsim/arndt.pdf>

¹² Tobias Koch Diplomarbeit 2005 – Implementation und Simulation von RIP-MIT

3.3 Syntax und Semantik

Ein zu simulierendes Szenario wird in der VNUML Sprache in einer XML¹³ Datei formuliert und abgespeichert. Allgemein ist festzuhalten, dass XML Dateien aus Tags bestehen. Ein Tag beginnt mit einer spitzen Klammer auf "<" und endet mit einer spitzen Klammer zu ">". Es gibt Tags die alleine für sich stehen können und es gibt Tags zu denen ein StartTag "<TagName>" und ein EndTag "</TagName>" gehören. Hier heraus resultiert eine Baumstruktur die typisch für XML Dateien ist. Kommentare werden durch Spitzer Klammer auf und zwei horizontalen Strichen begonnen und am mit zwei horizontalen Strichen und Spitzer Klammer zu geschlossen. "*<-- Kommentar Text -->*".

Der Erste Tag-Eintrag einer VNUML Datei benennt die verwendete XML Version. Der darauffolgende Tag die Pfadangabe der vnuml.dtd Datei. Die dtd-Datei beinhaltet die VNUML Sprache und wird bei der Installation von VNUML angelegt. Die Ausformulierung der Simulation muss mit einem <vnuml> Tag beginnen und endet mit dem schließenden </vnuml> Tag. Zwischen diesem Tagpaar können die vier möglichen <global>, <net>, <vm> und <host> Tags stehen.

Der <global> Tag definiert die Rahmenbedingungen einer Simulation. Er folgt immer unmittelbar nach dem <vnuml> Tag. Der durch einen Start- und einen Endtag abgesteckte Ast beinhaltet eine Reihe von Konfigurationseinträgen. Als erstes wird die verwendete VNUML Version angegeben. Weiterhin wird der Name der Simulation gesetzt. Es können noch eine Reihe anderer Konfigurationen gesetzt werden. Diese sind aber für das Verständnis dieser Arbeit nicht von Nöten.

Der <net> Tag bezeichnet jeweils ein virtuelles Netzwerk. Er besitzt keinerlei weiteren Tags und dient lediglich dazu die einzelnen Netzwerknamen einer Simulation einzuführen. Sie dienen dazu ein Netzwerk in der späteren Simulation identifizieren zu können. Diese Namensvergabe ist somit zwingend notwendig.

¹³ Extensible Markup Language

Die Einzelnen virtuellen Rechner werden durch den `<vm>` Tag definiert. Der `<vm>` Tag besitzt eine ganze Reihe von weiteren Tags die auf seinem Ast Verwendung finden. Unter anderem definieren diese die Netzwerkverbindungen durch die die einzelnen virtuellen Rechner mit einander verknüpft sind. Die einzelnen virtuellen Rechner werden direkt am Anfang durch das Name Attribute zur Einfachen späteren Identifizierung mit einem Namen belegt. Im drauffolgenden `<filesystem>` Tag wird das Root Dateisystem angegeben. Das Dateisystem wird beim hochfahren des virtuellen Rechners auf diesem gemountet. In dieser Arbeit werden zwei Dateisysteme verwendet.

```
<filesystem  
type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>  
Standart VNUML Root-Dateisystem
```

```
<filesystem type="cow">/usr/local/share/vnuml/filesystems/MTI/root</filesystem>  
Root-Dateisystem mit RIP-MTI
```

Unterhalb des Tags, der das Dateisystem angibt, wird der Linux Kernel mit dem `<kernel>` Tag angegeben. Der angegebene Linux Kernel wird auf den virtuellen Rechnern verwendet.

Die eigentlichen Netzwerkschnittstellen werden mittels des Interface `<if>` Tags konfiguriert. Das Attribut `id` nummeriert verwendeten Interfaces. Anschließend wird das Interface mit der Bezeichnung `eth` und der vergebenen Nummer mit dem virtuellen Rechner verbunden. Parallel dazu wird im Host ein Interface mit der Bezeichnung `name-ethNummer` angelegt. Mit dem Attribut `net` wird das Netzwerk angegeben, mit dem die neue Schnittstelle verbunden werden soll. Die IP-Adresse des virtuellen Rechners wird durch den `<ipv4>` Tag angegeben.

Der Einsatz des `<host>` Tag sollte nur stattfinden, wenn ein Host unverzichtbarer Bestandteil einer Simulation ist. Dabei wird die Konfiguration des Hosts entsprechend zum `<vm>` Tag durchgeführt.

3.4 Erstellen einer neuen Simulation

Das Erstellen einer neuen Simulation unter VNUML läuft immer nach dem selben Schema ab. Die erste Phase ist die Designphase, die völlig ohne den Computer,

erstellt werden kann. Die zweite Phase ist die Implementierungsphase in der der VNUML Code erstellt wird. Die dritte Phase beinhaltet die Ausführung des Codes und in der letzten vierten Phase wird das Projekt wieder heruntergefahren.

3.4.1 Designphase

Phase Nummer Eins ist die Designphase. In dieser Phase wird die Simulation grundlegend geplant und entworfen. Es wird die Anzahl der benötigten virtuellen Rechner und deren jeweilige Verknüpfungen, Netzwerkverbindungen bestimmt. Zu diesem Zweck bietet wegen ihrer Einfachheit sich eine klassische Zeichnung mit Blatt und Papier an. Zugleich sollte in dieser Phase die virtuellen Rechner und die Verbindungen in einer Art benannt werden, die das spätere Arbeiten erleichtern. Es sollte bedacht werden, das Abhängig von der Anzahl der virtuellen Rechner Hardwarekapazitäten belegt werden. Das bedeutet je größer das virtuelle Netz wird, um so höher wird auch die Rechenanforderung an der realen PC der die Simulation ausführt. In der Praxis hat sich gezeigt das die CPU Geschwindigkeit weniger ausschlaggebend für die Performance ist, als ausreichender Hauptspeicher. Der Ablauf der Simulationen konnte erheblich beschleunigt werden in dem auf dem Testrechner der Hauptspeicher von 512MB auf 1024MB aufgerüstet wurde.

3.4.2 Implementierungsphase

Ist die Designphase abgeschlossen, muss das gewünschte Szenario in einer für die VNUML Sprache verständlichen Form erstellt werden. Hierfür wird mittels der VNUML Syntax eine XML Datei erstellt. Bei der Erstellung der XML Datei sollte immer darauf geachtet werden das sich die Syntax von VNUML in Version 1.4 zu 1.5 verändert hat. Testet man VNUML mit den Beispieldateien von der Homepage, so sind diese nicht alle mit der neuesten VNUML Version kompatibel. Ist dies der Fall, muss der User die jeweiligen XML Dateien mit der aktuellen VNUML-Syntax abgleichen.

```
Alt: <start type="verbatim">zebra -f zebra.conf.sample -d -P 2601</start> (VNUML 1.4)
Neu: <exec seq="start" type="verbatim">zebra -f zebra.conf.sample -d -P 2601</exec>
      (VNUML1.5)
```

Austausch des <start> Befehles unter VNUML 1.4 durch den <exec> Befehls in Version 1.4

Ein Beispiel das die Änderung des VNUML Syntax zeigt ist der <start> Tag. In der Version 1.4 findet er noch Verwendung. In der Version 1.5 wurde er durch den <exec> Tag ersetzt.

Wird eine XML Datei erstellt ist zu beachten das sowohl die Dateinamen des Root Dateisystems, als auch die des Linux Kernels korrekt angegeben werden müssen. Das bezieht sich sowohl auf den Dateinamen und die korrekte Angabe der Position der Dateien im System. Die Versionsangabe der verwendeten VNUML Version muss mit der Versionsangabe in der XML Datei übereinstimmen. Zu guter letzt sollte aus Gründen der Übersichtlichkeit der Dateiname immer mit dem verwendeten Namen des Szenarios übereinstimmen.

3.4.3 Ausführungs-/ Testphase

Als letzter Schritt folgt der praktische Test und die reale Ausführen des Szenarios. Diese letzte Phase kann noch mal in vier Einzelschritte Zergliedert werden.

In der ersten Aktion wird das Szenario hochgefahren. Hierbei geht der VNUML Parser wie folgt vor. Zuerst erstellt er die virtuellen Netzwerke und Verbinden hierdurch der virtuellen UML Rechner. Sind diese aufgebaut werden alle UML Rechner im Host gebootet und entsprechend konfiguriert. Der VNUML Parser bedient sich hier bei aus den Einträgen der XML Datei um die IP Adressen, die Routing Tabelle usw. zu konfigurieren.

Das hochfahren der virtuellen UML Rechner mit dem dazugehörigem mounten der Dateisystemen ist sehr rechenintensiv und kann aufgrund dessen bis zu zehn Minuten in Anspruch nehmen. Nach dem der Bootvorgang abgeschlossen ist, existieren die virtuellen Rechner im Host und das Szenario ist bereit für die eigentliche Simulation.

<code>vnumlparser.pl -t sample.xml -vB</code>	Der Anhang -vB veranlasst VNUML während die Netzwerktopologie aufgebaut wird, alle Schritte auf dem Bildschirm auszugeben. Er ist optional, kann also weggelassen werden. Zusätzlich wird die Konsole erst frei gegeben, sobald das Netzwerk vollständig hochgefahren ist.
---	--

Hochfahren eines Szenarios

Die zweite Aktion ist es auf dem hochgefahrenen Szenario die Simulation zu starten. Der Aufruf des VNUML Parsers mit dem Parameter `-x start@Dateiname.xml` bewirkt

das Ausführen der Execution Tags `<exec seq="start">` der XML Datei. Diese werden für jeden virtuellen Rechner in der Datei definiert. Die Tags werden vom Host, dem realen PC, aus über eine Remote-Shell auf den virtuellen Rechnern ausgeführt.

```
vnumlparser.pl -s sample.xml (VNUML 1.4)
vnumlparser.pl -x start@sample.xml (VNUML 1.5)

vnumlparser.pl -p sample.xml (VNUML 1.4)
vnumlparser.pl -x stop@sample.xml (VNUML 1.5)
```

Starten und Beenden einer Simulation.

Die dritte Aktion ist die gestartete Simulation wieder zu beenden. Dies geschieht durch den Aufruf `vnumlparser.pl -x stop@Dateiname.xml`. Aus der XML Datei werden alle `<exec seq="stopp">` Tags ausgeführt. Die Stop Kommandos sollten so gewählt sein, dass alle virtuellen Rechner wieder in einen reproduzierbaren Anfangszustand zurückkehren und abgeschaltet werden können.

3.4.4 Szenario herunterfahren

In der vierten Aktion wird das Szenario heruntergefahren. Es werden alle Ressourcen frei gegeben die zuvor von dem Szenario belegt wurden. Das Herunterfahren der UML Rechner ist wiederum sehr rechenintensiv und kann einige Zeit in Anspruch nehmen.

Diese vier Aktionen sollten immer beachtet werden. Wird ein Szenario nicht ordnungsgemäß heruntergefahren kann das gemountete Dateisystem Schaden nehmen. Eventuell ist sogar eine Neuinstallation des Root Dateisystems nötig. Hängt sich ein Szenario auf und kann nur noch gewaltsam abgebrochen werden müssen anschließend mehrere Dateien von Hand gelöscht werden. Im Verzeichnis `/var/vnuml/` muss die Datei „Lock“ und das Verzeichnis des soeben abgebrochenen Szenario entfernt werden. Anschließend sollte ein erneutes Hochfahren des Szenarios möglich sein.

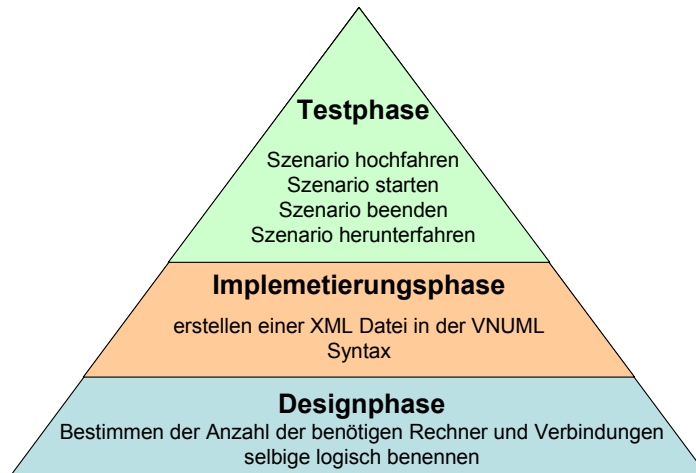


Abb. 3.4.4.1: Drei Phasenmodell – VNUML Arbeitszyklus

4. Zebra & Quagga

Zebra und Quagga sind frei erhältliche Routing Softwaren. Beide sind aus dem GNU Projekt heraus entstanden. Das GNU Projekt wurde im Jahr 1984 mit dem Ziel gegründet, ein auf Unix basierendes, freies Betriebssystem und zugehörige Software zu entwickeln. Das Kürzel GNU steht für "GNU's Not Unix", ausgesprochen "DJI-N-JUH"¹⁴. Zebra ist Modular aufgebaut. Es gibt einen zentralen Serverprozeß (zebra daemon) und einzelne untergelagerte Prozesse, die jeweiligen Routing Daemons. Der Zebra Daemon ist zuständig für die Erstellung der Kernel Routing Tabelle und der Regelung des Austausches von Routing Informationen zwischen den einzelnen Daemons. Die Routing Daemons übernehmen die eigentlichen Routing Aufgaben. Routing Daemons entsprechen den jeweiligen Routing Protokollen. So gibt es unter Zebra die folgenden Routing Daemons, ripd, bgpd und ospfd. Die Stärken von Zebra liegen vor allen Dingen auf TCP/IP basierendem Routing. Zebra ermöglicht es einen Computer so zu konfigurieren, dass er entweder als unkomplizierter Router, als Route Server oder als Route Reflektor einzusetzen ist. In den letzten Jahren ist Zebra nicht weiter entwickelt worden. Da aber nach wie vor ein Bedarf an einer aktuellen Version bestand, die die aktuellen Routing Protokolle unterstützt, wurde Quagga ins Leben gerufen. Quagga ist eine Weiterentwicklung die nahtlos an Zebra anknüpft. In der aktuellen Version 0.98 unterstützt Quagga die folgenden Protokolle: RIP, RIPv2 und RIPv3, OSPF und OSPFv2, IPv6, IGMP und IGMPv2, BGPv4 sowie SNMP. In

¹⁴ <http://www.gnu.org/home.de.html>

Zukunft soll zusätzlich auch noch eine Unterstützung von Multicast Routing Protokollen realisiert werden.¹⁵

4.1 Arbeiten mit Quagga

In dieser Studienarbeit wird mit Hilfe von Quagga das den Routing Daemon zur Verfügung stellt, auf virtuellen Rechnern das RIP Protokoll ausgeführt. VNUML ist nur für die Erstellung der Netzwerke zuständig. Die Routing Protokolle werden mittels Quagga auf den jeweiligen Routern hochgefahren. Dies kann entweder manuell oder auch automatisch durch anlegen von Konfigurationsdateien geschehen. Um ein Gefühl für den Umgang mit Quagga in Verbindung mit RIP zu bekommen, empfiehlt es sich Anfangs mit manuellen Eingaben zu experimentieren. Hierfür wird ein einfaches Netzwerkbeispiel verwendet.

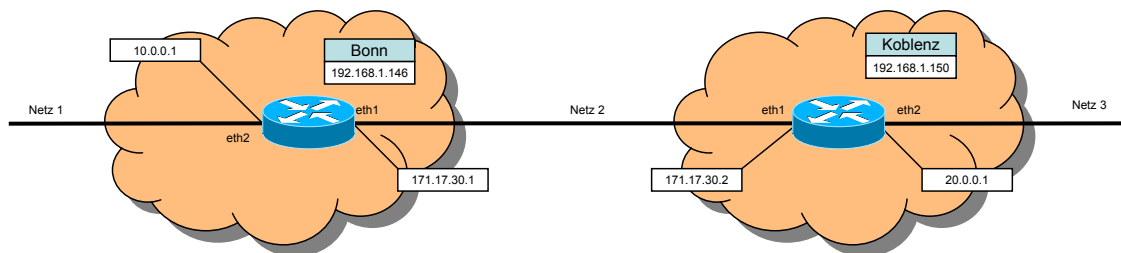


Abb. 4.1.1 – Szenario 1

Das in Abbildung 3.1 vorgestellte Netz wird durch Aufrufen der Datei RIP3.xml angefertigt. VNUML erzeugt die beiden Virtuellen Maschinen Bonn (192.168.1.146) und Koblenz (192.168.1.150). Dazu das Netz 1 mit dem Adressbereich 10.0.0.1/30. Das Netz 2 mit dem Adressbereich 171.17.30.17/30, das Bonn und Koblenz verbindet und das Netz 3 mit dem Bereich 20.0.0.1. Für den Zeitraum, der nötig ist die Netzwerktopologie hoch zu fahren, wird auf dem Bildschirm die folgende Anzeige ausgegeben:

```
Bonn sshd is not ready (socket style)
61 seconds elapsed...
Bonn sshd is not ready (socket style)
66 seconds elapsed...
Bonn sshd is ready (socket style): 192.168.1.146 (mng_if)
Koblenz sshd is ready (socket style): 192.168.1.150 (mng_if)
host> /bin/rm -f /var/vnuml/LOCK
Total time elapsed: 82 seconds
```

Bildschirmausdruck während des Hochfahrens eines Szenario.

¹⁵ Benjamin Zapilko – Seminar Routing WS 04/05 – GNU Zebra & Quagga

Anschließend muss die Simulation gestartet werden. Dies geschieht durch den Aufruf der XML-Datei mit folgendem Zusatz:

```
vnumlparser.pl -x start@RIP3.xml -vB Erst jetzt werden alle Ausführbaren Zeilen der
XML-Datei ausgeführt. <exec seq="start"...>
Das Szenario wird gestartet.
```

Befehlszeile für das starten einer Simulation auf dem Szenario RIP3

Zur Aktivierung von Quagga und des RIP Protokolls auf dem virtuellen Rechner muss die XML Datei folgende Zeile enthalten:

```
<exec seq="start" type="verbatim">ripd -f /usr/local/etc/ripd.conf.sample -d -P
2602</exec>
```

RIP Kommando Eintrag einer XML Datei für einen Virtuelle Rechner

Der Standardwert für den Port den Quagga im Verbund mit RIP verwendet ist 2602 und darf nicht verändert werden. Ist das Szenario hochgefahren kann mittels der Eingabe `telnet Bonn 2602` auf die virtuelle Maschine Bonn zugegriffen werden.

```
root@IBM:/home/Leif/VNUML/examples# telnet Bonn 2602
trying 192.168.1.146...
Connected to uml1.
Escape character is '^]'.

Hello, this is quagga (version 0.96.5).
Copyright 1996 - 2002 Kunihiro Ishiguro.

User Access Verification
Password:zebra
Bonn>
```

Login Screen von Quagga

Quagga begrüßt den User und fordert ihn zur Passwordeingabe auf. Das Standardpasswort für Quagga ist `zebra`. Gesetzt wird das Passwort in der Datei `„usr/local/etc/ripd.conf.sample“`. Das Passwort kann vom User beliebig durch einen Eintrag in der Konfigurationsdatei geändert werden. Nachdem sich der User autorisiert hat befindet er sich auf dem Virtuellen Rechner Bonn. Zunächst nur im unprivilegierten Modus (normal Mode). In normal Mode kann der User lediglich auf ein paar Grundfunktionen des Systems zugreifen. Er kann sich den Systemstatus, die Routingtabelle sowie einfache Systeminformationen anzeigen lassen.

```
Bonn> 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
```

Ausdruck des Befehls `show ip rip`

Direkt nach dem Aufbau der Netzwerktopologie ist noch kein Routing-Daemon aktiv. Auf Grund dessen sind zu diesem Zeitpunkt noch keinerlei Einträge in der Routingtabelle. Der obige Bildschirmausdruck der Routingtabelle bleibt leer. Dem normal Mode übergeordnet ist der privilegierte Modus. Der privilegierte Modus bietet dem User die Möglichkeit zur Konfiguration der Router-Betriebsparameter und der Ausführung von Prüfschritten. In den privilegierten Modus (enable Mode) gelangt der User durch Eingabe des Befehles `enable`.

```
Bonn>
Bonn>enable
Bonn#                                     (privilegierte Modus erkennbar an der Route)
```

Wechsel vom unprivilegierten in den privilegierten Modus unter Quagga

Möchte der User den RIP-Routing-Daemon konfigurieren, muss er zuerst in den privilegierten Modus und dann in den globalen Konfigurationsmodus wechseln. Im Konfigurationsmodus wird ein IP-Routing-Protokoll konfiguriert. Der Wechsel vom privilegierten in den Konfigurationsmodus geschieht über folgende Eingabe.

```
Bonn>enable
Bonn#configure terminal
Bonn(config)#
```

Wechsel vom privilegierten in den globalen Konfigurationsmodus

Der eigentliche RIP-Daemon wird durch die Eingabe `router rip` hochgefahren. Der Router wird hierdurch RIP fähig und wechselt automatisch in den Routerkonfigurationsmodus.

```
Bonn(config)#router rip
Bonn(config-router)#
```

Durch Eingabe von `router rip` Wechsel in der Routerkonfigurationsmodus

Jetzt können dem Router Bonn seine direkten Nachbarn bekannt gegeben werden. Hierfür wird mittels des Befehles `network` die Netzwerknummer jedes direkt an den Router Bonn angeschlossenen Netzwerkes in die Routingtabelle eingetragen. Es werden nur so genannte Major Klassen ABC Netzwerk Adressen angegeben, da RIP nur selbige akzeptiert.

```
Bonn(config-router)#
Bonn(config-router)# network 171.17.30.0/24
Bonn(config-router)# network 10.0.0.0/24
```

Konfiguration der Routen des Routers Bonn

Bonn lernt durch die obige Eingabe seine benachbarten Netzwerke kennen und tauscht sofort seine Routingtabelle mit den Tabellen der anderen Routern aus. Vom

Router Koblenz lernt er zum Beispiel das Netz 20.0.0.0/24. Entsprechend zu den Einträgen von Bonn muss der Router von Koblenz konfiguriert werden. Vorher kann keine Kommunikation stattfinden.

```
Koblenz(config-router)#
Koblenz(config-router)# network 171.17.30.0/24
Koblenz(config-router)# network 20.0.0.0/24
```

Konfiguration der Routen des Routers Koblenz

Zu diesem Zeitpunkt ist das Beispielsnetz und der Routing-Daemon komplett hochgefahren und konfiguriert. Durch anschließendes Ausdrucken der Routing Tabelle mittels `show ip rip` auf den Bildschirm, werden die statischen Routen und die hinzu gelernten Routen angezeigt. Wenn alle Konfigurationen korrekt vorgenommen worden sind, sollte ein Ausdruck in der folgenden Art am Bildschirm erscheinen.

```
root@IBM:/home/Leif/VNUML/examples# telnet Bonn 2602
Bonn> 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
C(i) 10.0.0.0/24      0.0.0.0           1 self               0
R(n) 20.0.0.0/24      171.17.30.2       2 171.17.30.2         0 02:36
C(i) 171.17.30.0/24  0.0.0.0           1 self               0
```

Ausdruck der Routingtabelle des Routers Bonn durch Eingabe des Befehles `telnet Bonn 2602`

Unter Quagga ist `split horizon per default` aktiv. Ist es erforderlich eine Simulation ohne `split horizon` durchzuführen, müssen folgende Einträge für jedes gewünschte Interface vorgenommen werden:

```
Bonn>enable
Bonn#configure terminal
Bonn(config-router)# interface eth1
Bonn(config-if)# no ip rip split-horizon
```

Deaktivieren von `split-horizon` auf der Schnittstelle `eth1` des Routers Bonn

5. Diagnose-Hilfsmittel

5.1 Ping

Für eine erste Diagnose hat sich das Kommando `ping` bewährt. Das Kommando sendet ein sendet ein ICMP-Echo-Request-Paket an die Zieladresse des zu überprüfenden Rechners (Host). Unter einem Host versteht man in der Informatik

einen Computer, der an ein Computernetzwerk angeschlossen ist und auf dem eins oder mehrere Serverprogramme laufen. Wenn dieser Zielrechner angeschaltet und korrekt an das Netzwerk angebunden ist, sendet er das Paket an den Ausgangsrechner zurück. Der User weiß folglich das der „angepinkte“ Rechner korrekt funktioniert. Zusätzlich liefert ping eine kurze Statistik über die Anzahl der gesendeten, empfangenen und unbeantworteten Pakete. Sowie kürzeste, längste und mittlere Antwortzeit.

```
root@IBM:/home/Leif/VNUML/examples# ping Bonn
PING Bonn (192.168.1.146): 56 data bytes
64 bytes from 192.168.1.146: icmp_seq=0 ttl=64 time=6.4 ms
64 bytes from 192.168.1.146: icmp_seq=1 ttl=64 time=0.3 ms
64 bytes from 192.168.1.146: icmp_seq=2 ttl=64 time=0.3 ms
64 bytes from 192.168.1.146: icmp_seq=3 ttl=64 time=0.3 ms

--- Bonn ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss round-trip min/avg/max =
0.3/1.8/6.4 ms
```

Überprüfung des Routers Bonn durch "anpingen"

5.2 Traceroute

Mit `traceroute` wird der Weg, den ein Datenpaket vom Rechner A zum Rechner B nimmt, ausgegeben. Die erste Spalte gibt die Anzahl der Knoten an, die zwischen dem Anfangs und jeweiligen Rechner liegen. Also der wievielte Knoten in der Verbindung der bezeichnete Rechner gerade ist. Der nächste Eintrag ist der Rechnername und dessen IP-Adresse. Die folgenden drei Einträge zeigen die Antwortzeiten für drei aufeinander folgende Datagramme an dem jeweiligen Rechner.

```
Elbe:~# traceroute 192.168.32.2
traceroute to 192.168.32.2 (192.168.32.2), 30 hops max, 38 byte packets
 1 192.168.30.1 (192.168.30.1)  2.361 ms  2.215 ms  1.074 ms
 2 172.17.30.2 (172.17.30.2)  1.657 ms  1.493 ms  8.251 ms
 3 192.168.32.2 (192.168.32.2) 14.723 ms  5.589 ms  4.447 ms Elbe:~#
```

Anzeige der Route eines Paketes mittels traceroute

Ein !N hinter den Antwortzeiten gibt an, das ein Netzwerk nicht erreichbar ist. Es ist ein Anzeichen dafür, dass der Router nicht wusste, wohin er das Datagramm weiterleiten soll. Der Router quittiert diesen Fehler mit einem network unreachable Datagramm. In der Praxis ist das ein Anzeichen dafür, dass eine Netzverbindung zusammengebrochen ist. Ein !H ist ein Hinweis darauf, dass ein Rechner nicht erreicht werden konnte. Dies bedeutet, dass es möglich war bis in das lokale

Netzwerk des Zielrechners zu gelangen, dieser sich aber nicht gemeldet hat, weil er zum Beispiel abgeschaltet war.¹⁶ Die beiden Beispiele zu traceroute beziehen sich auf die Netzwerktopologie RIP1.xml.

```
Elbe:~# traceroute 192.168.32.2
traceroute to 192.168.32.2 (192.168.32.2), 30 hops max, 38 byte packets
 1 192.168.30.1 (192.168.30.1)  2.824 ms  1.188 ms  1.061 ms
 2 * * 192.168.30.1 (192.168.30.1)  3010.347 ms !H
 3 192.168.30.1 (192.168.30.1)  3753.907 ms !H  3009.161 ms !H  3010.111 ms !H
Elbe:~#
```

Fehlermeldung !H bei tracerout, Unerreichbarkeit eines Rechners

5.3 Show ip route

Eine weitere Möglichkeit zur Ausgabe der Routingtabelle ist der Befehl show ip route. Er zeigt zusätzlich die Schnittstellen, über die eine Netzwerkverbindung besteht, an. Anders als „show ip rip“, wird „show ip route“ aber nicht über Quagga im Verbund mit dem RIP Port 2602 aufgerufen. Es ist erforderlich sich mittels Telnet über den Port 2601 und Zebra auf dem gewünschten Rechner einzuloggen.

```
root@IBM:/home/Leif/VNUML/examples# telnet Bonn 2601
Router> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 10.0.0.0/24 is directly connected, eth2
R>* 20.0.0.0/24 [120/2] via 171.17.30.2, eth1, 00:33:53
C>* 127.0.0.0/8 is directly connected, lo
C>* 171.17.30.0/24 is directly connected, eth1
C>* 192.168.1.144/30 is directly connected, eth0
```

Ausgabe aller bekannten Routen mittels show ip route

Die Funktionen beider Befehle kombiniert das Kommando route -n. Das Kommando kann nur direkt auf einem der virtuellen Rechner ausgeführt werden. Der User muss sich folglich mittels ssh auf einem der Rechner einloggen. Das Standardpasswort für die virtuellen Rechner ist xxxx.

```
root@IBM:/home/Leif/VNUML/examples# ssh Bonn
Password: xxxx
Bonn:~# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
192.168.1.144    0.0.0.0         255.255.255.252 U        0    0      0 eth0
10.0.0.0         0.0.0.0         255.255.255.0  U        0    0      0 eth2
20.0.0.0         171.17.30.2    255.255.255.0  UG       2    0      0 eth1
171.17.30.0     0.0.0.0         255.255.255.0  U        0    0      0 eth1
```

Ausgabe der Routingtabelle mittels route -n

¹⁶ <http://www.64-bit.de/dokumentationen/howto/de/html/DE-NET2-HOWTO-14.html>

Die bisher vorgestellten Tools ermöglichen dem User eine Kontrolle, ob die Netzwerktopologie korrekt hochgefahren und alle Router vollständig konfiguriert sind. Eine genau Analyse des Datenverkehrs ermöglichen sie nicht. Hierfür sollten die Programme tcpdump oder Ethereal verwendet werden.

5.4 Tcpcdump¹⁷

Tcpcdump ist ein Netzwerkanalyse-Tool. Es befindet sich standardmäßig auf den virtuellen Rechner von VNUML und kann aus der Konsole heraus gestartet werden. Eine Installation ist nicht erforderlich. Tcpcdump gibt dem User die Möglichkeit, den gesamten Netzwerkverkehr der über ein Interface abgewickelt wird, mitzuverfolgen und zu analysieren. Tcpcdump ist ein klassischer Netzwerk-Sniffer. Tcpcdump versetzt dazu das entsprechende Netzwerkinterface in den so genannten Promiscuous Mode. Promiscuous Mode bedeutet, das der Sniffer den gesamten Datenverkehr sammelt, der an die in diesen Modus geschaltete Netzwerkinterface geschickt wird. Es werden also nicht nur die an ihn adressierten Frames empfangen, sondern auch die nicht an ihn adressierten. Der Adressat eines Datenpaketes wird in einer Netzwerktopologie anhand seiner MAC-Adresse festgelegt.¹⁸ Das Interface liest nun nicht mehr nur die für sie bestimmten Datenpakete, sondern schneidet alle Datenpakete mit und gibt sie auf dem Bildschirm oder in eine Datei aus. Tcpcdump protokolliert den gesamten Datenverkehr. Das bedeutet, das auch Passwörter mitgeschnitten werden. Wenn selbige unverschlüsselt sind kann sie der User direkt auslesen, was ein Sicherheitsrisiko darstellt. Für die Arbeit mit tcpcdump sind Aufgrund dessen zwingend root-Rechte erforderlich.¹⁹

Wird tcpcdump ohne Argumente aufgerufen, bindet es sich an das erste Interface in der Systemliste, das up ist. Ein Interface lässt sich gezielt mittels `-i` zuweisen. Wird zum Beispiel `tcpcdump -i eth1` aufgerufen, wird der gesamte Verkehr über eth1 mitgeschnitten und am Bildschirm ausgegeben. Sind tiefer gehende Informationen nötig, kann das Argument `-v` angehängt werden. Es bewirkt, das tcpcdump in den

¹⁷ <http://www.tcpdump.org/>

¹⁸ <http://de.wikipedia.org/wiki/Sniffer>

¹⁹ <http://www-e.uni-magdeburg.de/steschum/tcpdump.pdf>

Verbose-Modus geht. Zusätzlich werden dann ACK-Pakete und Headerinformationen wie RX ID, Rufnummer, Sequenznummer, Seriennummer und RC Paketflags ausgegeben.

Die Einsatzgebiete von tcpdump sind:

- Diagnose von Netzwerkproblemen
- Eindringungsversuche entdecken
- Netzwerktraffic-Analyse und Filterung nach verdächtigem Inhalt
- Datenspionage

5.5 Ethereal²⁰

Ethereal ist eine grafische Erweiterung. Es gestattet eine komfortablere Analyse von Netzwerkprotokollen als tcpdump. Im Unterschied zu tcpdump ist Ethereal nicht Kommandozeilen orientiert, sondern verfügt über eine grafische Benutzeroberfläche. Es gibt sowohl eine Windows, also auch eine Linux Version. Ethereal zeichnet den gesamten Verkehr in einem Netzwerk auf und zeigt anschließend die einzelnen Pakete an. Die Pakete können entsprechend dem jeweiligen Bedarf nach Zeit, Source-Adresse, Destination-Adresse und dem jeweils zugrunde liegendem Protokoll sortiert werden.²¹ Ethereal benötigt ebenfalls wie tcpdump unter Linux Root-Rechte. Das Programm bietet die Möglichkeit mehrere oder alle angeschlossenen Interfaces zu einer Gruppe zusammen zu fassen und aufzuzeichnen.

6. Szenario 1

Für einen ersten Vergleich zwischen den Routingprotokollen bietet sich das aus Kapitel 3.1 bekannte Szenario an. Die beiden Router Bonn und Koblenz sind durch das Netz 171.17.30.0/30 mit einander verbunden. An jedem der beiden Router ist ein weiteres Netz angeschlossen. Router Bonn ist direkt mit dem Netz 10.0.0.0/30 und

²⁰ <http://www.ethereal.com/>

²¹ <http://www.ethereal.com/docs/user-guide/>

Koblenz mit dem Netz 20.0.0.0/30 verbunden. Für die Untersuchung werden Mitschnitte des Netzverkehrs mit tcpdump und ethereal verwendet. Es gibt zwar auch die Möglichkeit mittels Quagga Log Dateien zu erstellen, allerdings beinhalten diese kaum Informationen anhand derer eine Untersuchung durchzuführen wäre.

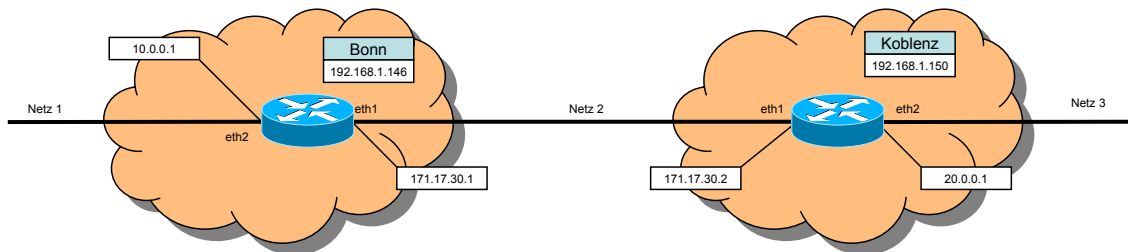


Abb. 6.1: Szenario 1A

6.1 Vergleich der Routing Protokolle

Das Szenario wird wie gewohnt hochgefahren. Anschließend wird direkt mittels SSH auf Bonn eingelockt und tcpdump gestartet. Es ist wichtig tcpdump vor dem RIP Daemon auf einem der Router zu starten. Nur so kann ein vollständiger Mitschnitt der gesendeten Pakete erstellt werden. Zusätzlich wird sichergestellt das die Ausgangssituation bei jeder Aufzeichnung die Selbe ist. Das ist unbedingt erforderlich um eine Vergleichbarkeit zu gewährleisten. Tcpdump wird so konfiguriert das es den Traffic auf dem Router Bonn über die Schnittstelle eth1 aufzeichnet.

Die anschließende Konfiguration der beiden Router erfolgt über vorgefertigte Konfigurationsdateien die von VNUML geladen werden.

```
<filetree when="start"
  root="/usr/local/etc"/>/home/Leif/VNUML/examples/RIP3/Bonn</filetree>
<exec seq="start" type="verbatim">ripd -f /usr/local/etc/Bonn.conf -d -P 2602</exec>
```

Dabei muss beachtet werden, dass der RIP Daemon unter Quagga, standardmäßig mit Split Horizon betrieben wird. Da in diesem Beispiel die einfachste Fall von RIP gezeigt werden soll, muss Split Horizon explizit deaktiviert werden. Die Deaktivierung muss einzeln für jede Interface Schnittstelle des Routers vorgenommen werden. Während Quagga nun den RIP Daemon hochfährt und die Konfigurationen vornimmt protokolliert tcpdump zeitgleich auf Bonn die übertragenen Pakete auf der Schnittstelle eth1. Für das Verständnis im folgenden

Text, die IP Adresse 171.17.30.1 entspricht der Schnittstelle eth1 auf dem Router Bonn, also Bonn-eth1. Die IP Adresse 171.17.30.2 der Schnittstelle eth1 auf dem Router Koblenz, quasi Koblenz-eth1. Die übertragenen Pakete haben dabei folgende Struktur:

Das erste übertragene Paket ist immer eine Request Anfrage. Sie ist bei allen drei untersuchten Protokollen, RIP ohne split horizon, RIP mit split horizon und RIP-MTI immer gleich. Ausgehend vom Router Bonn geht die Request Anfrage an den Multicast Channel 224.0.0.9. Der Multicast Channel 224.0.0.9 wird unter RIP2 immer dann verwendet wenn eine Kommunikation innerhalb von Routern statt findet. Für OSPF wird zum Beispiel der Channel 224.0.0.5 und 224.0.0.6 verwendet.

Der Aufbau eines Paketes ist immer gleich. Im Anfang der ersten Zeile steht der Timestamp. Dieser Zeiteintrag entspricht dem Zeitpunkt an dem das Paket die Schnittstelle eth1 erreicht und von tcpdump aufgezeichnet wurde. Gezählt wird die Zeit, die der Hostrechner hochgefahren ist. In der nächsten Zeile stehen die source Adresse gefolgt von der destination Adresse, abgetrennt durch ein „>“. Darauf folgt welche Funktion das Paket hat. Dabei wird unterschieden zwischen request und response, Anfrage und Antwort. Handelt es sich um ein response Paket sind die zu übermittelnden Netzadressen mit Metric und next-hop Einträgen angehängt.

```
14:28:55.974630 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], length: 52)
  Bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Request, length: 24
```

Paket 1 – Szenario 1 - RIP ohne Split Horizon

Das darauf folgende Paket ist eine Response Nachricht vom Router Bonn an die Multicast Schnittstelle. In ihr wird die dem Router Bonn bekannte Verbindung zum Netz 10.0.0.0/30 übermittelt. Dieses Netz ist in direkter Nachbarschaft von eth1 zu Bonn angeschossen. Hat Aufgrund dessen eine Metric von 1 und der next-hop Eintrag ist self. Diese Response Nachricht ist ebenfalls bei allen drei Routingprotokollen gleich.

```
14:28:56.361788 IP (tos 0x0, ttl 1, id 1, offset 0, flags [DF], length: 52)
  Bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
  AFI: IPv4: 10.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
```

Paket 2 – Szenario 1 - RIP ohne Split Horizon

Das dritte Paket ist eine IGMP²² Nachricht. Von einer Multicast-Anfrage ausgehend meldet sich der Router Bonn an der Multicast Schnittstelle an. Im Gegensatz zu einer direkten RIP Paketen wird jetzt die Multicast Adresse 224.0.0.22 benutzt. Das Ziel dieses Paketes ist es, einer Multicast Gruppe beizutreten. Dazu schickt Bonn dem Router einen so genannten IGMP Report, der die Multicast Adresse enthält. Der Router nimmt die IP-Adresse von Bonn in seine interne Tabelle auf und leitet die Pakete ab sofort an ihn weiter. Das Verlassen einer Gruppe ist je nach Version von IGMP unterschiedlich. Bei Version 1 schickt der Router periodisch alle 60 bis 120 Sekunden eine Anfrage an alle ihm bekannte Rechner im Subnetz. Jedes Mitglied einer Multicast Gruppe meldet seine Zugehörigkeit an den Router zurück. Meldet sich ein vorher eingetragener Client innerhalb einer gewissen Zeit nicht zurück, so wird er aus der Tabelle ausgetragen. Das IGMP Paket ist ebenfalls in allen drei untersuchten Routingprotokollen gleich.

```
14:28:55.997986 IP (tos 0xc0, ttl 1, id 0, offset 0, flags [DF], length: 40,
    optlength: 4 ( RA ))
Bonn > 224.0.0.22: igmp v3 report, 1 group record(s) [gaddr 224.0.0.9 to_ex, 0
source(s)]
```

Paket 3 - Szenario 1 - RIP ohne Split Horizon

Das vierte Paket unterscheidet sich je nach verwendetem Protokoll, deutlich voneinander. Unter RIP, ohne split horizon, werden in einer Response Message von Bonn.route an die Multicast Schnittstelle zwei Routen übertragen. Zum einem die Route zu 10.0.0.0/30 mit Metrik 1 und next-hop self, als auch die route 171.17.30.0/30 mit Metrik 1 und next-hop self. Wird hingegen RIP mit split horizon oder RIP-MIT verwendet wird nur eine Route übertragen. Die Route 10.0.0.0/30 mit Metrik 1 und next-hop self.

Eine Mitteilung über die Route 171.17.30.0/30 kann eingespart werden. Tcpcdump protokolliert momentan auf eth1 mit der IP-Adresse 171.17.30.1 und gehört folglich selbst zu jener Gruppe. Hier spielt zum ersten mal RIP unter Verwendung von split horizon seine Stärken aus. Anders als das einfache RIP erkennt es das die Route zu 171.17.30.0/30 unnötig ist und lässt sie weg. Gleiches geschieht unter Verwendung von RIP-MIT, da split horizon eine Voraussetzung von RIP-MIT ist.

²² Internet Group Message Protokoll

```

14:28:56.778663 IP (tos 0x0, ttl 1, id 3, offset 0, flags [DF], length: 72)
  Bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 44, routes: 2
  AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
  AFI: IPv4:      171.17.30.0/30, tag 0x0000, metric: 1, next-hop: self

```

Paket 4 – Szenario 1 – RIP ohne Split Horizon

```

09:14:43.111620 IP (tos 0x0, ttl 1, id 3, offset 0, flags [DF], length: 52)
  Bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
  AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 1, next-hop: self

```

Paket 4 – Szenario 1 – RIP mit Split Horizon

Im Paket Nummer Fünf geht eine Request Anfrage von 171.17.30.2 an den Multicast Channel heraus. Dies ist das erste Paket die den bis dahin für Bonn-eth1 unbekanntem Eintrag 171.17.30.2 enthält. Dieses Paket ist ebenfalls bei allen drei Routingprotokollen gleich und an fünfter Position.

```

14:28:56.872647 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], length: 52)
  171.17.30.2.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Request, length: 24

```

Paket 5 – Szenario 1 – RIP ohne Split Horizon

Die nachfolgenden Pakete unterscheiden sich in ihrer Art und Reihenfolge stark in Abhängigkeit vom verwendeten Routing Protokoll. So wiederholt sich unter RIP ohne split horizon das Paket 4 an der Position 6. Verändert hat sich lediglich die Source und die Destination Adresse. Im Paket Nummer 4 war noch Bonn.route die Source und 224.0.0.9 die Destination Adresse. Dies verändert sich im Paket Nummer 6. Bonn.route ist immer noch die Source aber die Destination wurde zu 171.17.30.2. Die Schnittstelle eth1 von Bonn übermittelt folglich mittels eines Response Paketes seine bekannten Routen 10.0.0.0/30 und 171.17.30.0/30 an seinen Nachbarn 171.17.30.2.

```

14:28:56.878587 IP (tos 0x0, ttl 64, id 5, offset 0, flags [DF], length: 72)
  Bonn.route > 171.17.30.2.route: [udp sum ok]
  RIPv2, Response, length: 44, routes: 2
  AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
  AFI: IPv4:      171.17.30.0/30, tag 0x0000, metric: 1, next-hop: self

```

Paket 6 – Szenario 1 – RIP ohne Split Horizon

Paket Nummer 7 ist eine Response Nachricht von 171.17.30.2 an die Multicast Adresse. Es übermittelt die bis zu diesem Zeitpunkt dem Router Bonn unbekanntem Adresse 20.0.0.0/30. Aufgrund dessen das dieses Paket von der Schnittstelle 171.17.30.2, also von Koblenz-eth1 kommt, beträgt die Metric 1 und der next-hop Eintrag ebenfalls 1.

```
14:28:56.889467 IP (tos 0x0, ttl 1, id 1, offset 0, flags [DF], length: 52)
  171.17.30.2.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
  AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
```

Paket 7 – Szenario 1 – RIP ohne Split Horizon

Paket Nummer 8 ist die direkte Antwort auf das Paket Nummer 7. Ausgehend vom Router Bonn. Dieser hat das vorherige Paket angenommen und die Route 20.0.0.0/30 in seine Routing Tabelle aufgenommen. Dieses Meldet er an die Multicast Adresse weiter. Ausgehend von der Neuen Positionierung verändern sich die Metric auf den Wert 2 und der next-hop Eintrag wird auf 171.17.30.2 gesetzt.

```
14:28:56.925553 IP (tos 0x0, ttl 1, id 6, offset 0, flags [DF], length: 52)
  Bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
  AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 2, next-hop: 171.17.30.2
```

Paket 8 – Szenario 1 – RIP ohne Split Horizon

Im darauf folgenden neunten Paket meldet sich Koblenz-eth1, also 171.17.30.2 per IGMP Paket bei 224.0.0.22 an. Zu diesem Zeitpunkt befindet sich also sowohl 171.17.30.1, als auch 171.17.30.2 in der Group Tabelle. Das gleiche Paket wird auch unter den anderen ROuting Protokollen angewendet. Unter Verwendung des RIP Protokolls mit split horizon wird dieser Eintrag schon im achten Paket vorgenommen. Unter RIP-MIT allerdings ebenfalls erst an neunter Stelle. Dafür hat RIP-MIT aber andere Pakete, die bei einfachen RIP später kommen, schon vorher abgearbeitet.

```
14:28:56.995836 IP (tos 0xc0, ttl 1, id 0, offset 0, flags [DF], length: 40,
optlength: 4 ( RA ))
  171.17.30.2 > 224.0.0.22: igmp v3 report, 1 group record(s) [gaddr 224.0.0.9 to_ex, 0
source(s)]
```

Paket 9 – Szenario 1 – RIP ohne Split Horizon

Paket 10 ist die Mitteilung von 171.17.30.2 an den Multicast Channel über die ihm bekannten Routen. Zum einem kennt Koblenz-eth1 inzwischen sowohl die Verbindung zu 10.0.0.0/30, als auch die Verbindung zu 20.0.0.0/30. Entsprechen sind die Metric und dir next-hop Einträge. Ausgehend von Koblenz-eth1 ist die Metric zu 10.0.0.0/30 gleich 2 und der next-hop Bonn. Zu 20.0.0.0/30, das sein direkter Nachbar ist, ist die Metric 1 und der next-hop self.

Auffällig ist die Angabe das drei Routen übertragen wurden. Dies lässt sich erklären das zusätzlich noch der Eintrag 171.17.30.0/30 mitgeteilt wurde. Dieser aber fallen

gelassen wurde, da die Quelle des Paketes selbst zu diesem Netz gehört. Aus dem Protokoll von tcpdump ist dies nicht direkt zu schließen. Mittels einer Überprüfung durch ethereal, das die dritte Route nicht ausblendet, konnte 171.17.30.0/30 als dritte Route identifiziert werden.

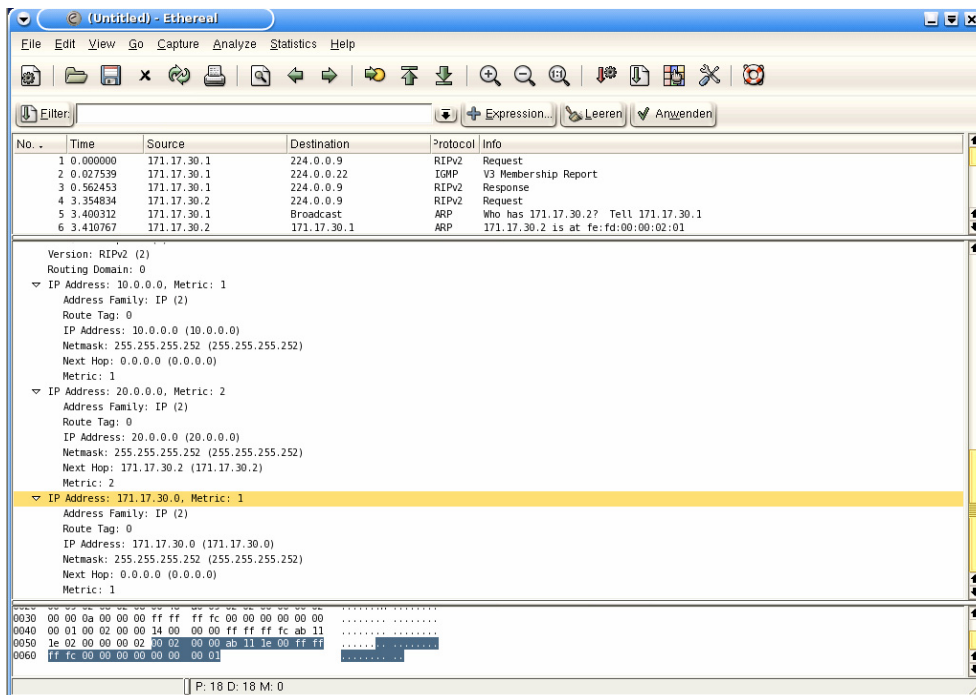


Abb. 6.1.1 Ethereal Screen Shot
Darstellung eines Paketinhaltes mit 3 Pfadangaben

```
14:28:57.738907 IP (tos 0x0, ttl 1, id 3, offset 0, flags [DF], length: 92)
  171.17.30.2.route > 224.0.0.9.route:
  RIPv2, Response, length: 64, routes: 3
  AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 2, next-hop: Bonn
  AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 1, next-hop: self[rip]
```

Paket 10 – Szenario 1 – RIP ohne Split Horizon

Das gleiche Paket wird auch unter Verwendung der anderen Routingprotokollen übermittelt. Es lässt sich dabei sehr gut die unterschiedliche Arbeitsweise der Protokolle erkennen. Während das einfache RIP ohne split horizon sämtliche Routen die es kennt überträgt, findet unter RIP mit split-horizon eine Vorselektierung statt. Es werden nur zwei Routen übertragen. Routen die der vom anderen Knoten gelernt wurden oder doppelt vorhanden sind werden nicht übertragen.

```
09:14:44.011393 IP (tos 0x0, ttl 1, id 3, offset 0, flags [DF], length: 72)
  171.17.30.2.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 44, routes: 2
  AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 2, next-hop: Bonn
  AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
```

Paket 12 – Szenario 1 – RIP mit Split Horizon

Unter RIP-MTI funktioniert die Vorselektierung noch besser. Hier wird nur eine Route übermittelt.

```
10:41:26.481948 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], length: 52)
  171.17.30.2.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
  AFI: IPv4: 20.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
```

Paket 10 – Szenario 1 – RIP MTI

Die Pakete 11 und 12 sind jeweils ARP²³ Anfrage und ARP Antwort auf den für Bonn-eth1 neu gelernten Nachbarn 171.17.30.2. Das ARP Protokoll findet Verwendung bei der Adressierung von Hardware. Dies geschieht indem aus Hardwareadressen IP-Adressen generiert werden. Dies geschieht folgender weiße. Als erstes wird eine ARP Anforderung, ein Request-Broadcast mit der IP-Adresse des gesuchten Computers gesendet. Ein Host, der die gesuchte IP-Adresse kennt, antwortet indem er die zugehörige MAC²⁴-Adresse über ein ARP-Reply zurücksendet. Als MAC-Ziel wird die Quelladresse der Anforderung verwendet. Der antwortende Host muss nicht unbedingt der gesuchte Host sein, da jeder teilnehmende Host über einen Cache von MAC- und IP-Adressen verfügt.²⁵

Paket 11 beinhaltet die Anfrage von Bonn nach der Adresse von 171.17.30.2. Diese wird an alle Hosts geschickt die mit Bonn verbunden sind. Paket 12 ist die ARP-Antwort auf die Anfrage. Sie beinhaltet mit fe:fd:00:00:02:01 die MAC Adresse des Rechners zur zugehörigen IP-Adresse.

```
14:29:01.865555 arp who-has 171.17.30.2 tell Bonn
```

Paket 11 – Szenario 1 – RIP ohne Split Horizon

```
14:29:01.866436 arp reply 171.17.30.2 is-at fe:fd:00:00:02:01
```

Paket 12 – Szenario 1 – RIP ohne Split Horizon

Die ARP-Anfrage und die dazugehörige ARP-Antwort treten bei RIP mit split horizon in den Paketen 9 und 10 auf. Wird RIP-MTI eingesetzt wird die ARP Anfrage und Antwort schon nach den Paketen 6 und 7 abgewickelt. Hier lässt sich deutlich die verbesserte Effektivität der beiden Routingprotokolle erkennen. Sie kommen zu einem deutlich früherem Zeitpunkt zu der ARP Kommunikation.

Paket 13 ist Inhaltlich eine Wiederholung des Paketes 10. Paket 14 ist ein Response Paket. Ausgehend von Bonn wird an den Multicast Channel alle Bonn bekannten

²³ Address Resolution Protokoll

²⁴ Media Access Control

²⁵ http://de.wikipedia.org/wiki/Address_Resolution_Protocol

Routen weitergegeben. Bekannt sind Bonn jetzt drei Routen. Eine Verbindung zum Netz 10.0.0.0/30 mit einer Metric von 1 und next-hop self. Eine mögliche Verbindungen zu 20.0.0.0/30 und die ausgeblendete Route zum Netz 171.17.30.0/30. Ausgehend von Bonn-eth1 ist die Route zu 20.0.0.0/30 mit einer Metric von 2 und einem next-hop 171.17.30.2 festgelegt.

```
14:29:23.798759 IP (tos 0x0, ttl 1, id 8, offset 0, flags [DF], length: 92)
  Bonn.route > 224.0.0.9.route:
  RIPv2, Response, length: 64, routes: 3
  AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
  AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 2, next-hop:
171.17.30.2[rip]
```

Paket 14 – Szenario 1 – RIP ohne Split Horizon

Nach dem vierzehnten Paket wiederholen sich die Pakete 10 und 14. Es treten keine neuen Pakete auf. Die Routingtabellen der Router sind folglich komplett konfiguriert. Festzuhalten ist, dass ein kompletter Durchlauf unter RIP ohne split horizon 14 Pakete benötigt. Die Anfänglichen fünf Pakete waren unter den verschiedenen RIP Lösungen gleich oder sehr ähnlich. Unterschiede treten nur in der Hinsicht auf das unter dem einfachen RIP ohne split horizon eine Reihe von Paketen doppelt auftreten und eigentlich unnütze Routen übertragen werden.

Wird RIP mit split horizon betrieben ist die Konvergenz nach 13 Paketen erreicht. Unter RIP MIT wird die Konvergenz sogar schon nach 10 Paketen erreicht.

	RIP ohne Split Horizon	RIP mit Split Horizon	RIP-MTI
Benötigte Pakete	14	13	10
Zeit (1/100 sec)	67,824126	69,844211	1,889362

Da das Szenario nur simuliert wird sind die Zeitangaben die benötigt werden bis das Netzwerk konvergiert mit Vorsicht zu betrachten. Teilweise sind unter dem selben Szenario mit der selben Konfiguration und der selben Ausgangssituation deutliche Zeitunterschiede messbar. So erklärt sich auch der extreme Ausreißer RIP-MTI in der obigen Tabelle. Die Zeitunterschiede sind dermaßen außergewöhnlich dass sie nur durch die Verwendung des VNUML Simulators erklärt werden können.

Deutliche Unterschiede bei den Routing Protokollen sind ab dem Zeitpunkt der Konvergenz wahrzunehmen. Ist die Konvergenz erreicht wiederholt sich zwar in allen drei Routing Protokollen immer das gleichen Paketpaar, aber unterscheiden sich diese in der Anzahl der übertragen Routen. Das eine Paket ist immer ausgehend

von 171.17.30.2 (Koblenz.route) an den Multicast Channel, mit allen ihm bekannten Routen. Das andere Paket geht aus von 171.17.30.1 (Bonn.route) mit den ihm bekannten Routen. Unter RIP mit split horizon sehen dieses Paket wie folgt aus.

```

09:15:12.121120 IP (tos 0x0, ttl 1, id 8, offset 0, flags [DF], length: 72)
  Bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 44, routes: 2
    AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 1, next-hop: self
    AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 2, next-hop: 171.17.30.2

09:15:13.023111 IP (tos 0x0, ttl 1, id 5, offset 0, flags [DF], length: 72)
  171.17.30.2.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 44, routes: 2
    AFI: IPv4:      10.0.0.0/30, tag 0x0000, metric: 2, next-hop: Bonn
    AFI: IPv4:      20.0.0.0/30, tag 0x0000, metric: 1, next-hop: self

```

Pakete 13 und 14 – Szenario 1 – RIP mit Split Horizon

Hierbei sticht insbesondere der Eintrag über die Anzahl der übermittelten Routen hervor. Unter RIP ohne split horizon steht an selbiger Stelle eine Drei. Es wird zusätzlich zu den beiden Routen 10.0.0.0/30 und 20.0.0.0/30 die Route zu dem Netz 171.17.30.0/30 übermittelt. Unter RIP mit split horizon hat der gleiche Eintrag als Wert nur eine Zwei. Die Route 171.17.30.0/30 wird nicht übermittelt. Das erklärt sich aus der Begebenheit, dass sowohl Bonn.route als auch Koblenz.route an das 171.17.30.0/30 Netz angeschlossen sind. Die Mitteilung über das 171.17.30.0/30 Netz ist für beide Router folglich nicht relevant. RIP mit split horizon unterdrückt diese für die Router unnütze Route.

Eine weitere Eigenschaft die unter RIP ohne Split horizon auftritt, ist dass in Next Hop Einträgen teilweise vermerkt wird von wo die jeweilige Route gelernt wurde. So steht bei der Route 20.0.0.0/30 im Paket Nummer 14 im next-hop Eintrag zusätzlich zum 171.17.30.2/30 noch als weiterer Eintrag [[rip].

Dieses Verhalten ändert sich, wenn das erweiterte RIP-MTI Routing Protokoll eingesetzt wird.

7. Szenario 2 – CTI

Im vorhergehenden Kapitel wurde ein einfach gehaltenes Szenario, bestehend aus zwei Routern und drei Teilnetzen betrachtet. Mit Hilfe des Szenarios konnte ein erster Qualitätsunterschied zwischen den einzelnen Routing Protokollen gezeigt werden. Dieses Kapitel betrachtet ein weiteres Szenario. Szenario 2 behandelt den

Counting To Infinity (CTI) Zustand. Hierzu wird auf eine Netzwerktopologie zurückgegriffen die aus 3 Routern (Hamburg, Berlin und Bonn) und 3 Netzwerkrechnern (Elbe, Spree und Rhein) besteht. An jedem Router ist jeweils ein Netzwerkrechner angeschlossen.

Ziel ist es in der simulierten Netzwerktopologie einen Counting To Infinity Zustand zu erzeugen. Hierfür wird die Simulation in zwei Versionen durchgeführt. Eine Version mit dem einfachen Routing Information Protokoll und eine weitere Version in der der erweiterten RIP-MTI Algorithmus Anwendung findet. Das Netzwerk entspricht einer Y-Kombination. RIP-MTI müsste folglich das Auftreten einer Schleife im Routingpfad erkennen.

In der Theorie könnte eine Routingschleife entstehen wenn folgende Konstellation auftritt. Die Verbindung zwischen Bonn und Rhein bricht zusammen. Hamburg wird nicht unmittelbar mitgeteilt das der Netzwerkrechner Rhein nicht mehr erreichbar ist. Beim nächsten Update, das von Hamburg ausgeht, werden die Router Berlin und Bonn in die Irre geführt. Es entsteht eine Schleife zwischen Hamburg, Berlin und Bonn. In der Praxis ist das CTI Ereignis ein deutliche Rarität. Aufgrund dessen muss bei der Simulation des Ereignisses nachgeholfen werden. Hierzu kann auf ein in der Diplomarbeit von Krechel Martin erstelltes Szenario zurückgegriffen werden. Ein Counting To Infinity Ereignis wird in diesem Beispiel durch temporärem Einsatz von Firewalls herbeigeführt.

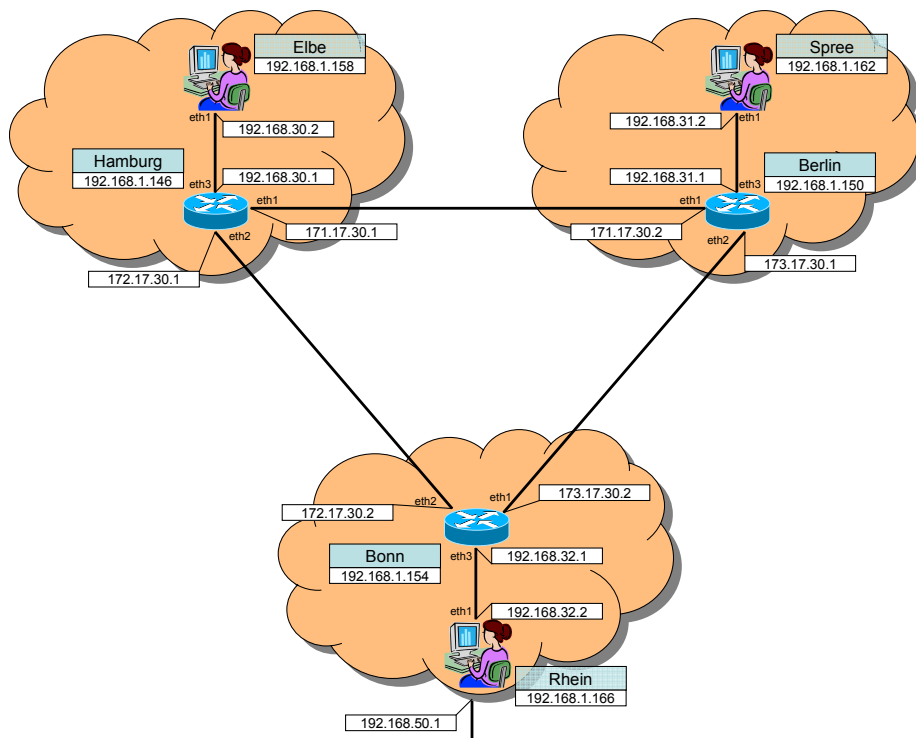


Abb. 7.1: Szenario 2 - Y-Kombination

7.1 Szenario Durchlauf unter Verwendung von RIP

Die Durchführung des Szenarios läuft wie folgt ab. Ist das Szenario hochgefahren und alle Router haben ihre Routingtabellen konfiguriert, wird die Netzwerkverbindung zwischen Bonn und Rhein gekappt. Dies geschieht durch folgenden Befehl.

```
vnumlparser.pl -x cut@RIP6.xml -vb
```

Trennen der Verbindung zwischen den Routern Bonn und Rhein

Der Aufruf des Befehles führt die in der RIP6.XML Datei enthaltenen `<exec seq="cut">` Zeilen aus. Diese Befehlszeilen bewirken, dass auf Router Bonn das Interface eth2 und auf Router Rhein das Interface eth1 geblockt werden. Zur Kontrolle kann anschließend auf Router Bonn die Routingtabelle betrachtet werden. Dazu muss sich mittels „telnet Bonn 2602“ auf dem Router Bonn eingelockt werden. Durch Eingabe des Befehles „show ip rip“ wird die Routingtabelle am Bildschirm ausgegeben. Ausschlaggebend ist der Timer zur Route nach 192.168.32.0/30, der Route zum Router Rhein. Der Timer, dessen Wert Anfangs noch drei Minuten beträgt, läuft aus. Hat der Timer einen Wert kleiner 30 Sekunden erreicht, wird Hamburg vom Netzwerk getrennt. Dies wird durch folgenden Befehl erreicht.

```
vnumlparser.pl -x cutHamburg@RIP6.xml -vb
```

Trennen des Router Hamburg vom Netz, dadurch bekommt Hamburg die Nachricht, dass Router Rhein nicht mehr erreichbar ist nicht mit.

Durch die Trennung des Router Hamburg vom übrigen Netzwerk bekommt dieser die Nachricht, dass der Router Rhein nicht mehr erreichbar ist, nicht mitgeteilt. Der Router Hamburg blockt allerdings nur Pakete die vom übrigen Netzwerk an ihn gehen. Ausgehende Pakete vom Router Hamburg sind weiterhin möglich. Durch diese Begebenheit werden beim nächsten Update das von Hamburg ausgeht den Routern Berlin und Bonn falsche Routen übermittelt.

Ist die Verbindung zwischen Hamburg und dem übrigen Netzwerk gekappt, wird das Tool Ethereal gestartet. Ethereal wird so konfiguriert, das es den Traffic aller an den Host angeschlossenen Netzwerke aufzeichnet. Daraufhin kann die Verbindung zwischen Hamburg und dem restlichen Netzwerk wieder hergestellt werden.

```
vnumlparser.pl -x repairHamburg@RIP6.xml -vb
```

Deaktivierung der Firewall auf dem Router Hamburg

Sobald die Verbindung zum Router Hamburg wieder hergestellt ist kommt es zur Schleife. Ein CTI Ereignis findet statt. Das CTI Ereignis lässt sich gut an zwei Stellen nachvollziehen. Zum einem wird die Routingtabelle auf dem Router Bonn verändert. Anfangs ist die Metrik von Bonn zum Router Rhein auf einem Wert von 16. Die Verbindung wird von der Firewall geblockt wird. Router Rhein ist quasi unerreichbar.

Dies ändert sich sobald vom Router Hamburg die Update Nachricht kommt. Sie enthält den Hinweis das Router Rhein noch erreichbar ist. Folglich wird die Metrik verändert. Wird ab diesem Zeitpunkt die Routingtabelle auf dem Router Bonn wiederholt ausgegeben kann mitverfolgt werden wie die Metrik der Route zum Router Rhein langsam anwächst. Sie wird innerhalb von 18 Sekunden von einer dreier auf sechzehner Metrik erhöht.

```
10:34:36.666154 IP (tos 0x0, ttl 1, id 38, offset 0, flags [DF], length:52)
bonn.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
    AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 5, next-hop:self

10:34:45.405720 IP (tos 0x0, ttl 1, id 28, offset 0, flags [DF], length:52)
172.17.30.1.route > 224.0.0.9.route: [udp sum ok]
  RIPv2, Response, length: 24, routes: 1
```

```

AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 6, next-hop:bonn
10:34:45.435301 IP (tos 0x0, ttl 1, id 44, offset 0, flags [DF], length:52)
bonn.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 8, next-hop:self
10:34:48.475582 IP (tos 0x0, ttl 1, id 30, offset 0, flags [DF], length:52)
172.17.30.1.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 9, next-hop:bonn
10:34:48.497115 IP (tos 0x0, ttl 1, id 47, offset 0, flags [DF], length:52)
bonn.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 11, next-hop:self
10:34:49.535831 IP (tos 0x0, ttl 1, id 32, offset 0, flags [DF], length:52)
172.17.30.1.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 12, next-hop:bonn
10:34:50.522393 IP (tos 0x0, ttl 1, id 50, offset 0, flags [DF], length:52)
bonn.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 14, next-hop:self
10:34:54.583840 IP (tos 0x0, ttl 1, id 34, offset 0, flags [DF], length:52)
172.17.30.1.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 15, next-hop:bonn
10:34:54.589119 IP (tos 0x0, ttl 1, id 53, offset 0, flags [DF], length:52)
bonn.route > 224.0.0.9.route: [udp sum ok]
RIPv2, Response, length: 24, routes: 1
AFI: IPv4: 192.168.50.0/24, tag 0x0000, metric: 16, next-hop:self

```

Tcpdump Ausgabe auf Router Bonn. Schnittstelle eth1. Durchlauf des CTI Ereignisses

Die Zeitangabe verdeutlicht, dass das CTI Ereignis nicht von einer Sekunde zur Anderen abläuft, sondern eine gewisse Zeit benötigt um sich aufzubauen. Die 30 Sekunden sind sicher aufgrund der Simulation mittels VNUML nicht repräsentativ, verdeutlichen aber gut den zeitlichen Ablauf des CTI Ereignisses.

Eine zweite Möglichkeit das CTI Ereignis zu verfolgen ist die Analyse des Ethereal Mitschnittes. Ethereal sammelt mit der gewählten Konfiguration alle Datenpakete auf allen Schnittstellen. Diese Datenmenge wird schnell sehr groß und folglich Unübersichtlich. Zur Steigerung der Übersichtlichkeit, sollten nachdem des CTI Ereignisses abgeschlossen ist, unter Ethereal die gesammelten Pakete gefiltert werden. Es werden nur die RIP Version 2 Pakete benötigt. Wertet man diese Pakete aus, kann sehr gut der Kreislauf nachvollzogen werden. Das Paket wandert im Uhrzeigersinn durch das Netzwerk von einem Router zum nächsten. Dabei wird die Metrik zum Erreichen des Router Rhein bei jedem Schritt hoch gezählt.

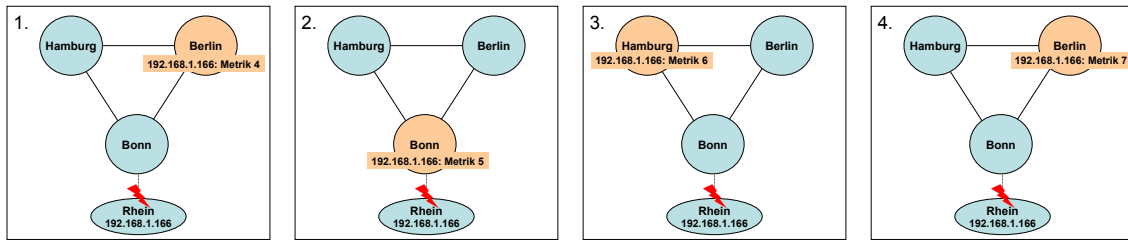


Abb. 7.1.1 Durchlauf des CTI Ereignisses, Router Rhein ist unerreichbar (Metrik 4 bis 7)

Die ersten beiden Response Pakete gehen vom Router Berlin aus. Sie beinhalten die Route zum Ziel Rhein mit einer Metrik von 4 Hops. Die beiden darauf folgenden Pakete haben als Source Adresse den Router Bonn. Inzwischen hat der Pfad zum Router Rhein eine Metrik von 5 Hops. In den beiden anschließenden Pakten, die vom Router Hamburg ausgehen, ist die Metrik auf 6 Hops angestiegen.

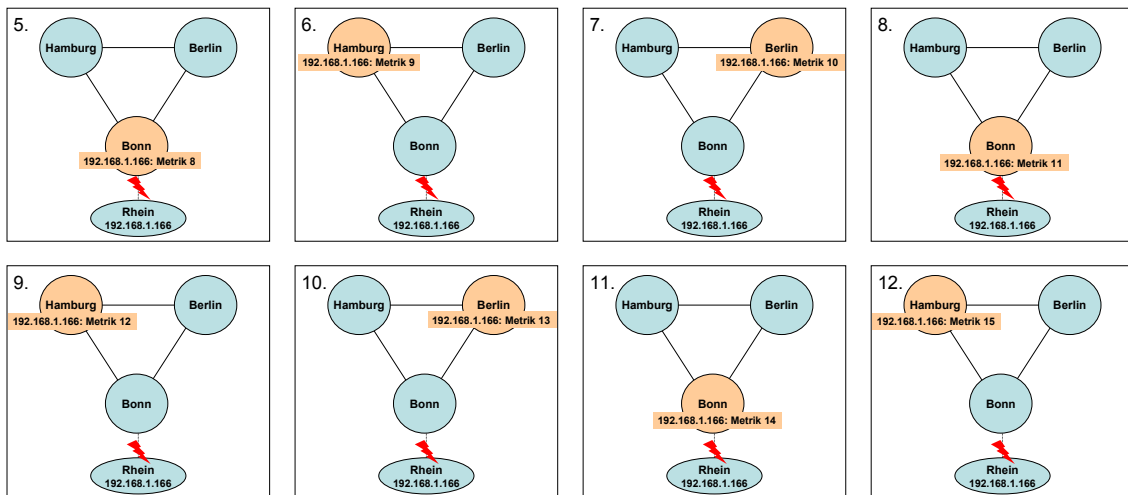


Abb. 7.1.2 Durchlauf des CTI Ereignisses, Router Rhein ist unerreichbar (Metrik 7 bis 15)

Bei jedem Schritt, der einen Wechsel von einem Router zum nächsten beinhaltet, wird die Metrik um den Wert Eins erhöht. Dieser Kreislauf zwischen den Routern Berlin, Bonn und Hamburg wird so lange fortgeführt bis die Metrik einen Maximalwert von 16 erreicht hat.

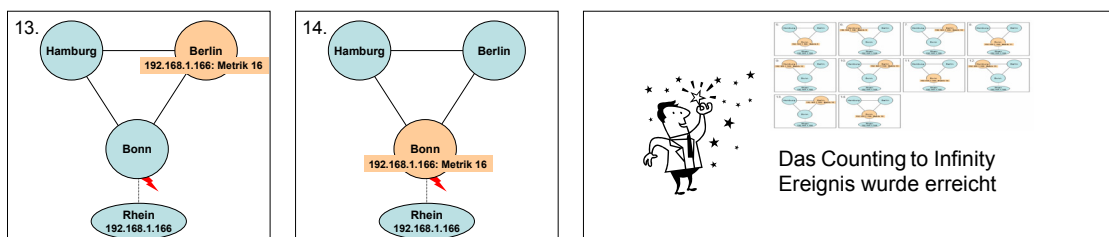
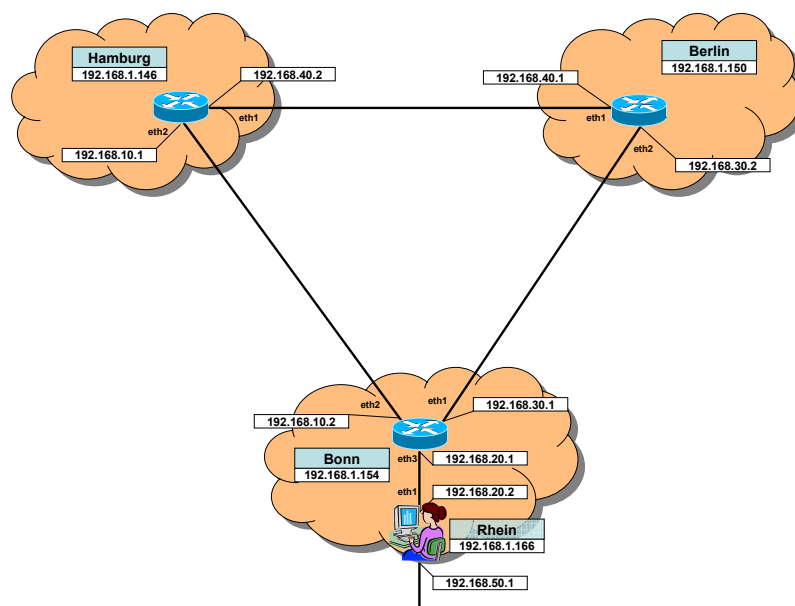


Abb. 7.1.3 Durchlauf des CTI Ereignisses, Router Rhein ist unerreichbar (Metrik 16)

Mit dem Ziel ein Aussagenkräftiges Ergebnis zu erhalten, wurde das Szenario mehrfach wiederholt. In jedem Durchlauf konnte erneut ein Counting to Infinity Zustand erzeugt werden. Dieses reproduzierbare Ergebnis ist Voraussetzung für die nächste Untersuchung

7.2 Szenario Durchlauf unter Verwendung von RIP-MTI

Kapitel 7.1 zeigte das unter Verwendung des einfachen Routing Protokolls RIP beständig ein CTI-Ereignis erzeugt werden konnte. In diesem Kapitel wird das gleiche Szenario unter Verwendung des erweiterten RIP-MTI Protokolls durchgeführt. Arbeitet des RIP-MTI Protokoll korrekt, so wird die Schleife erkannt und die nicht erreichbare Route aus der Routingliste entfernt.



Grafik 7.2.1: Szenario 2 – RIPMTI

Zur Vorbereitung muss das in der XML Datei abgespeicherte VNUML Szenario bearbeitet werden. Ziel ist es, das auf den virtuellen Maschinen laufende RIP Protokoll, durch das RIP-MTI Protokoll zu ersetzen. Dazu wird das Root

Dateisystem von dem Standart- auf das MTI-Dateisystem umgestellt. Ansonsten sind keinerlei Veränderungen an der VNUML Datei notwendig.

Das Szenario wird wie zuvor im Kapitel 7.1 gestartet. Haben sich alle Router untereinander ausgetauscht und die Routingtabellen wurde vollständig erstellt, kann das CTI Ereignis vorbereitet werden. Die Netzwerkverbindung zwischen Bonn und Rhein wird durch den Einsatz einer Firewall auf beiden Routern unterbrochen.

```

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
C(i) 192.168.10.0/24   0.0.0.0           1 self               0
C(i) 192.168.20.0/24   0.0.0.0           1 self               0
C(i) 192.168.30.0/24   0.0.0.0           1 self               0
R(n) 192.168.40.0/24   192.168.10.1      2 192.168.10.1        0 02:58
R(n) 192.168.50.0/24   192.168.20.2      2 192.168.20.2        0 00:04

```

Show ip rip ausgeführt auf Router Bonn. Die Verbindung zwischen Bonn und Rhein ist unterbrochen. Der Timer läuft ab. In 4 Sekunden wird die Verbindung auf 16 gesetzt werden.

Von diesem Zeitpunkt an wird auf dem Router Bonn die Routingtabelle betrachtet. Läuft in der Routing Tabelle der Timer zum Eintrag Router Rhein ab, wird die Firewall auf Router Hamburg aktiviert. Wichtig ist hierbei das dies ca. 30 Sekunden vor dem Ablauf des Timers geschieht. Die Firewall blockiert wie im vorher gehendem Szenario nur die Eingehende Verbindung des Routers Hamburg. Folglich ist Hamburg nicht mehr in der Lage neue Routen zu lernen, bzw. bestehende zu aktualisieren. Ist der Timer abgelaufen wird die Firewall auf Hamburg wieder deaktiviert. Router Hamburg teilt jetzt seinen benachbarten Routern Bonn und Berlin die ihm noch bekannte Route zum Router Rhein mit. Durch diese Fehlinformation wird die Basis für einen Counting To Infinity Ereignis gelegt.

Ab so fort kann das CTI Ereignis eintreten. Zur Kontrolle wird auf Router Bonn das Tool tcpdump ausgeführt. Tcpdump wird so konfiguriert, das es alle Pakte auf der Schnittstelle zwischen Bonn und Hamburg aufzeichnet. Diese Aufzeichnung dient zur direkten Kontrolle der Route nach Router Rhein. Wird der Eintrag zur Route nach Rhein hoch gezählt ist das CTI Ereignis eingetreten. Bleibt die Route hingegen auf nicht erreichbar, das heißt Metrik 16, wird sie nach Durchlauf des Timers gelöscht.

```

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
C(i) 192.168.10.0/24  0.0.0.0           1 self               0
C(i) 192.168.20.0/24  0.0.0.0           1 self               0
C(i) 192.168.30.0/24  0.0.0.0           1 self               0
R(n) 192.168.40.0/24  192.168.10.1      2 192.168.10.1       0 02:51
R(n) 192.168.50.0/24  192.168.20.2      16 192.168.20.2       0 01:57

```

Die Verbindung zwischen Bonn und Rhein wurde unterbrochen. Der Timer ist abgelaufen. Die Verbindung wurde auf 16 gesetzt. Es läuft ein neuer Timer ab. Ist dieser Abgelaufen und 192.168.50.0 ist weiterhin unerreichbar wird der Eintrag gelöscht. Ausgeführt auf Router Bonn.

```

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
C(i) 192.168.10.0/24  0.0.0.0           1 self               0
C(i) 192.168.20.0/24  0.0.0.0           1 self               0
C(i) 192.168.30.0/24  0.0.0.0           1 self               0
R(n) 192.168.40.0/24  192.168.10.1      2 192.168.10.1       0 02:28
R(n) 192.168.50.0/24  192.168.20.2      16 192.168.20.2       0 00:01

```

```

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
C(i) 192.168.10.0/24  0.0.0.0           1 self               0
C(i) 192.168.20.0/24  0.0.0.0           1 self               0
C(i) 192.168.30.0/24  0.0.0.0           1 self               0
R(n) 192.168.40.0/24  192.168.10.1      2 192.168.10.1       0 02:27

```

Der Timer läuft vom oberen Mitschnitt zum unteren Mitschnitt ab. Die Route wird aus der Tabelle entfernt.

Ist die Route entfernt wurde das CTI Ereignis, die Schleife korrekt vom Protokoll erkannt und verhindert. Zur weitergehenden Analyse wird Ethereal eingesetzt. Ethereal ermöglicht es alle Pakete die zwischen den Routern übermittelt werden abzufangen. Zu diesem Zwecke wurde in das Szenario eine Verbindung zum Hostrechner eingebaut. Durch den Einsatz des Host Rechner wird erreicht, das der direkte Datenverkehr auf den jeweiligen Netzwerkverbindungen aufgezeichnet werden kann. Der Hostrechner wird zu diesem Zwecke mit jeder Netzwerk-Verbindungen zwischen den Routern verbunden.

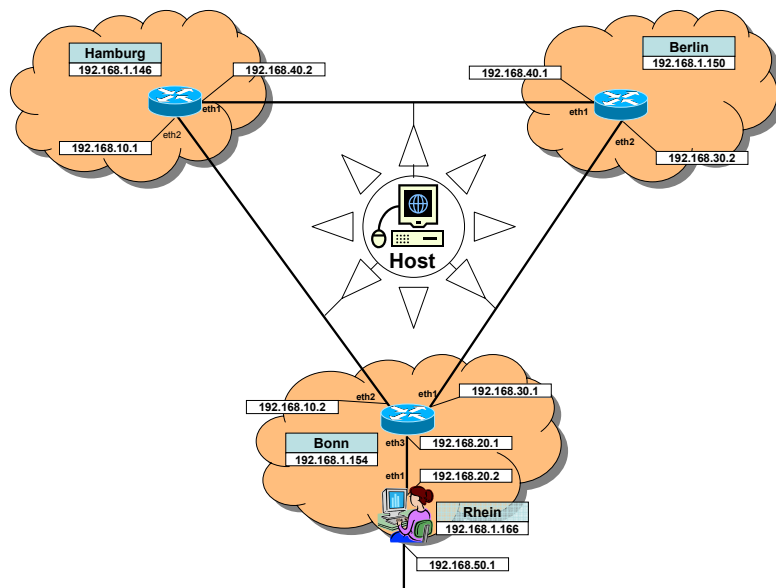


Abb. 7.2.2: Szenario 2 mit Einbindung an den Host.

Die Analyse des Mitschnittes von Ethereal zeigt den Ablauf des Ereignisses, bzw. das nicht Auftreten des CTI Ereignisses. Der Mitschnitt wird Sekundengenau protokolliert. Hierdurch lässt sich gut der Verlauf und die Updates der Routinginformationen verfolgen. In Sekunde 47, gezählt ab dem Beginn der Protokollierung, kennt und erreicht Schnittstelle 192.168.40.1 (eth1 von Berlin) noch alle Schnittstellen. In einem Paket an den Multicast Channel teilt Schnittstelle Berlin-eth1 diesem vier Routen mit.

```

47.140036      Source: 192.168.40.1  Destination: 224.0.0.9      RIPv2  Response
Routing Information Protokoll
  Command: Response (2)
  Version: RIPv2
  Routing Domain: 0
IP Address: 192.168.10.0, Metric: 1
IP Address: 192.168.20.0, Metric: 2
IP Address: 192.168.30.0, Metric: 2
IP Address: 192.168.50.0, Metric: 3
    
```

Ausschnitt des Ethereal Protokolls, Szenario 2 RIP-MTI

Alle vier Routen besitzen die korrekte Metrik. Das nächste Paket wird in Sekunde 54 protokolliert. Die Schnittstelle eth2 von Bonn teilt der Multicast Schnittstelle mit, das der Router Rhein mit einer Metrik von 16 versehen worden ist. Router Rhein ist folglich nicht mehr erreichbar. Die Firewall ist auf den benachbarten Rechnern Bonn und Rhein aktiviert worden.

```

54.073614      Source: 192.168.10.2  Destination: 224.0.0.9      RIPv2  Response
IP Address: 192.168.50.0, Metric: 16
    
```

Mitteilung von Bonn-eth2, das Router Rhein nicht mehr erreichbar ist.

Zeitlich in sehr kurzem Abstand folgt das nächste Paket. Es geht aus von der zweiten Schnittstelle des Router Bonn, eth1. Bonn ist der direkte Nachbar zum Router Rhein. Folglich ist es logisch nachzuvollziehen, dass die Mitteilungen des Router Rhein nicht mehr erreichbar ist, von ihm ausgehen.

54.076624	Source: 192.168.30.1	Destination: 224.0.0.9	RIPv2	Response
IP Address: 192.168.50.0, Metric: 16				

Mitteilung von Bonn-eth1, das Router Rhein nicht mehr erreichbar ist.

In den nachfolgenden Paketen verbreitet sich nach und nach die Information dass die Schnittstelle 192.168.50.0 nicht mehr erreichbar ist. Der Pfad zum Interface 192.168.50.0 wird in allen Routingtabellen mit einer Metrik von 16 versehen.

Anschließend wird die Verbindung zum Router Hamburg wieder hergestellt. Der Router kann dem übrigen Netzwerk seine falschen Routing-Informationen mitteilen. Ausgehend von der Schnittstelle 192.168.10.1 (Hamburg eth2) wird der Multicastschnittstelle die Route 192.168.40.0 mit einer Metrik von 1 mitgeteilt. Diese Routinginformation ist noch aktuell, hat folglich keinen Einfluss auf das übrige Netzwerk. Wichtiger ist die Mitteilung, die die Schnittstelle eth1 von Hamburg (192.168.40.2) verbreitet. In diesem Paket teilt Hamburg mit dass 192.168.50.0 mit einer Metrik von 3 erreichbar ist.

Diese Information ist falsch, da die Netzwerkverbindung zwischen Bonn und Rhein, immer noch unterbrochen ist. Die Frage ist nun wie die anderen Router auf diese Fehlinformation reagieren. Unter RIP würde durch diese Fehlinformation das CTI Ereignis ausgelöst werden. Das nächste Paket das Ethereal abfängt stammt von der Schnittstelle 192.168.30.2 (Berlin-eth1). In diesem Paket steht die Information dass 192.168.50.0 mit einer Metrik von 4 zu erreichen wäre.

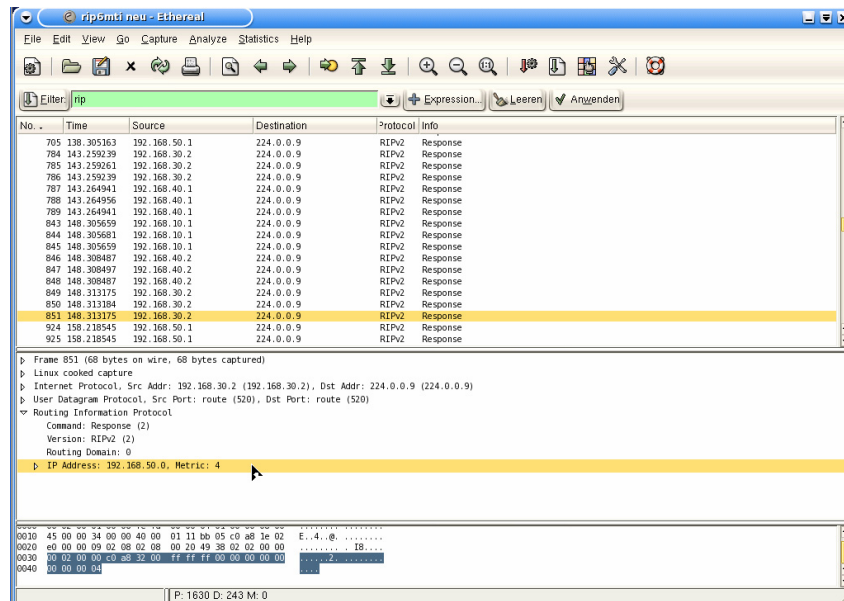


Abb. 7.2.3 Ethereal Mitschnitt: Übermittlung der Falschen Route zum Ziel 192.168.50.0

Die Fehlinformation von Hamburg ist folglich zum Router Berlin vorgedrungen. An dieser Stelle erkennt das Routing Protokoll RIP-MIT, das die Routinginformation nicht stimmt und lässt sie verfallen. In den anschließenden Paketen wird wieder die richtige Information verbreitet. Das 192.168.50.0 nicht zu erreichen ist und folglich mit einer Metrik von 16 zu versehen ist.

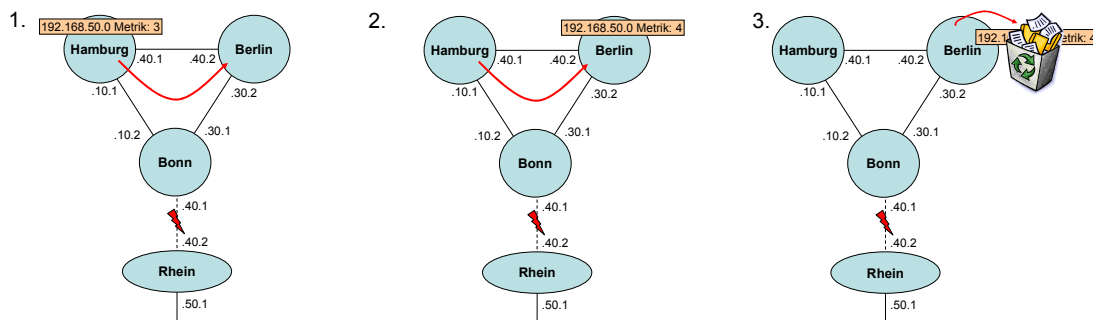


Abb. 7.2.4 RIP-MTI: Router Hamburg übermittelt die Falsche Route an Berlin. Router Berlin lernt diese zuerst, lässt sich aber nicht von der Falschen Angabe beirren und verwirft die falsche Route.

Die Schwierigkeit bei der Untersuchung des Szenarios 7.2 ist, das das nicht eintreten eines Ereignisses gezeigt werden soll. Es kann schließlich nur bedingt etwas protokolliert werden, das nicht statt findet.

Durch die Analyse der Pakete wird aber gezeigt das Routing Protokoll sich nicht, wie im vorher gehenden Kapitel, durch eine falsche Update Message in die Irre führen lassen. Das Protokoll erkennt richtig das die Schnittstelle zur Adresse 192.168.50.0 nicht mehr erreichbar ist. Daraufhin werden auf allen Routern die Routingtabellen aktualisiert. Der Pfad zur Adresse 191.168.50.0 wird mit einer Metrik von 16 versetzt und ein neuer Timer gestartet. Ist dieser Timer abgelaufen und die Schnittstelle ist weiterhin nicht erreichbar wird sie aus allen Routingtabellen entfernt.

7.3 Erweiterung des Szenario um eine Backupverbindung

Im vorhergehenden Kapitel konnte gezeigt werden das der RIP-MTI Algorithmus ein CTI Ereignis korrekt erkennt und verhindert. Es stellt sich jetzt die Frage ob der Algorithmus auch eine alternative Route zum Ziel zulässt, wenn diese dem Anschein nach wie eine weitere Schleife aussieht. Zu diesem Zweck wird das Szenario um drei weitere Router ergänzt. Diese bilden einen alternativen Weg zum Ziel 192.168.50.0.

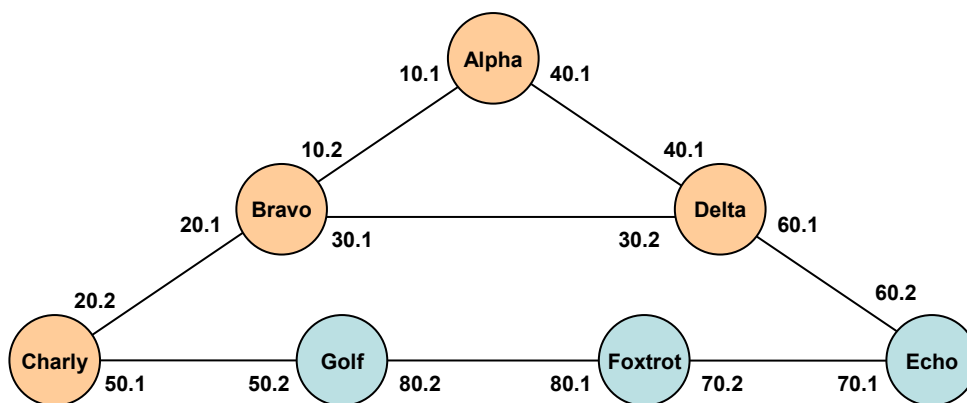


Abb. 7.3.1 Erweitertes Szenario 2: alternativ Weg zum Netz 192.168.50.0 über die drei neuen Router Echo, Foxtrot und Golf.

Betrachten wir noch einmal den Ablauf des vorher gehenden Szenario. Die Verbindung zwischen Bravo und Charlie wurde unterbrochen. Alpha wird kurz vor dem Ablauf des Timers vom Netz getrennt, damit der Eintrag zur Route 192.168.50.0 erhalten bleibt. Wir sind an der Stelle an der Alpha wieder an das Netz angeknüpft wird. Alpha teilt Delta die falsche Information mit, das 192.168.50.0 mit einer Metrik von 3 zu erreichen ist. Delta lernte diesen Eintrag und versieht ihn mit

einer Metrik von 4. Der RIP-MTI Algorithmus erkannte aber das der Pfad nicht korrekt ist und verwirft selbigen.

Durch Anfügen der drei weiteren Router an Delta wird versucht den Algorithmus mit einer vermeintlichen Schleife die ebenfalls eine Pfadlänge von 4 hat, zu täuschen. Die Frage ist, ob der Algorithmus, die vermeintliche Schleife über Echo, Foxtrot und Golf, als korrekten Pfad erkennt. Anderenfalls wird der Pfad fälschlich für eine Schleife gehalten und ebenfalls gelöscht werden.

Ein Durchlauf des Szenario zeigt ob der RIP-MTI Algorithmus korrekt arbeitet. Zu Beginn führt der Pfad von Delta nach Charlie über 192.168.30.1 (Bravo) mit einer Metrik von 3.

```
delta> 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)	192.168.10.0/24	192.168.30.1	2	192.168.30.1	0	02:39
R(n)	192.168.20.0/24	192.168.30.1	2	192.168.30.1	0	02:39
C(i)	192.168.30.0/24	0.0.0.0	1	self	0	0
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	0
R(n)	192.168.50.0/24	192.168.30.1	3	192.168.30.1	0	02:20
C(i)	192.168.60.0/24	0.0.0.0	1	self	0	0
R(n)	192.168.70.0/24	192.168.60.2	2	192.168.60.2	0	02:20
R(n)	192.168.80.0/24	192.168.60.2	3	192.168.60.2	0	02:20

Ausdruck der Routingtabelle auf Router Delta, vor der Abtrennung der Verbindung Bravo, Charlie.

Nachdem die Verbindung zwischen Bravo und Charlie gekappt wurde, wird auf Delta die Routingtabelle erneut abgefragt. Zu diesem Zeitpunkt wird der Pfad zum Ziel 192.168.50.0 mit einer Metrik von 4 und Next Hop 192.168.60.2 (Echo) angegeben.

```
delta> 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)	192.168.10.0/24	192.168.30.1	2	192.168.30.1	0	02:38
R(n)	192.168.20.0/24	192.168.30.1	2	192.168.30.1	0	02:38
C(i)	192.168.30.0/24	0.0.0.0	1	self	0	0
C(i)	192.168.40.0/24	0.0.0.0	1	self	0	0
R(n)	192.168.50.0/24	192.168.60.2	4	192.168.60.2	0	03:00
C(i)	192.168.60.0/24	0.0.0.0	1	self	0	0
R(n)	192.168.70.0/24	192.168.60.2	2	192.168.60.2	0	03:00
R(n)	192.168.80.0/24	192.168.60.2	3	192.168.60.2	0	03:00

Ausdruck der Routingtabelle auf Router Delta, nach der Abtrennung der Verbindung Bravo, Charlie.

Werden jetzt die Schnittstellen an Alpha wieder frei gegeben, kann sich die fehlerhafte Information, das 192.168.50.0 über den Pfad Alpha, Bravo zu erreichen ist verbreiten. Delta bekommt folglich mitgeteilt das es zu 192.168.50.0 einen weiteren Pfad gibt und versieht diesen mit einer Metrik von 4.

Zu diesem Zeitpunkt gibt es zwei theoretische Pfade zum Ziel 192.168.50.0. Der eine Pfad ist die CTI Schleife über Alpha, Bravo und Delta. Der andere Pfad geht über Echo, Foxtrot und Golf. Beide haben die Länge 4. Betrachtet man den RIP-MTI Algorithmus, so darf die maximal Pfadlänge in dieser Y-Topologie nur 3 betragen.

$$\text{Umfang}^{\text{Kreis}} < \text{Pfadlänge}^{\text{Startpunkt} \rightarrow \text{Endpunkt}}$$

Kreis 1 besteht aus Alpha, Bravo, Delta, also einen Umfang von 3. Die Pfadlänge beträgt für beide Routen nach 192.168.50.0 aber 4. Ginge es allein nach diesem Kriterium müssten folglich die beiden Einträge gelöscht werden. RIP-MTI lässt aber nur die Route über Alpha, Bravo und Delta verfallen. Nach der Mitteilung von Delta an den Multicastchannel taucht die Route in keinem Eintrag mehr auf. Sie wurde folglich korrekt als Schleife erkannt und gelöscht. Die Verbindung über Echo, Foxtrot und Golf bleibt bestehen, was sich anhand der Routingtabelle belegen lässt.

```

delta> 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) 192.168.10.0/24   192.168.30.1     2 192.168.30.1      0 01:38
R(n) 192.168.20.0/24   192.168.30.1     2 192.168.30.1      0 01:38
C(i) 192.168.30.0/24   0.0.0.0          1 self              0
C(i) 192.168.40.0/24   0.0.0.0          1 self              0
R(n) 192.168.50.0/24   192.168.60.2     4 192.168.60.2      0 02:00
C(i) 192.168.60.0/24   0.0.0.0          1 self              0
R(n) 192.168.70.0/24   192.168.60.2     2 192.168.60.2      0 02:00
R(n) 192.168.80.0/24   192.168.60.2     3 192.168.60.2      0 02:00
delta>

```

Das Verhalten lässt sich folgendermaßen erklären. Das erweiterte Szenario lässt sich in drei Subnetze unterteilen, die jeweils für einen Zyklus stehen. Subnetz Nummer Eins ist äquivalent zu dem Zyklus aus dem vorhergehendem Kapitel. Es besteht aus Alpha, Bravo und Delta. Der Zyklus hat folglich einen Umfang von drei. RIP-MTI erkennt diesen Umfang und verhindert das in diesem Kreislauf Routen entstehen die eine Länge von drei überschreiten. Der zweite Subnetz wird aus den Routern Alpha,

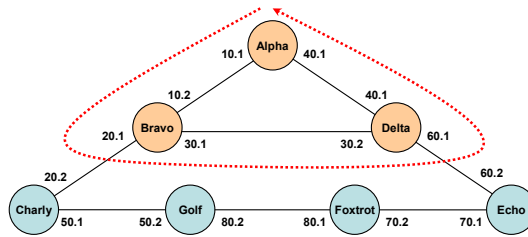


Abb. 7.3.2 Subnetz 1 (Umfang 3)

Delta, Echo, Foxtrott, Golf, Charly und Bravo gebildet. Innerhalb dieses Kreises sind Routen mit einer maximalen Länge von 7 zugelassen. Das dritte Subnetz stellt sich zusammen aus, Bravo, Delta, Echo, Foxtrott, Golf und Charly. Innerhalb dieses Zyklus werden Routen mit bis zu einer Länge von 6 vom RIP-MTI Algorithmus zugelassen.

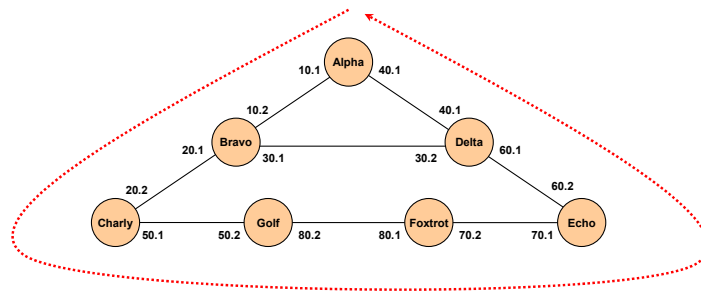


Abb. 7.3.3 Subnetz 2 (Umfang 7)

Betrachten wir jetzt wieder den Ausgangspunkt, das Delta zwei theoretische Pfade zum Zielnetz 192.168.50.0 kennt. Beide Pfade haben die Länge 4. Der eine Pfad betrifft das Subnetz 1. In diesem Subnetz dürfen Routen eine Länge von 3 nicht überschreiten. Folglich wird die Route nicht weitergegeben und verfällt. Die zweite Route mit der Länge 4 betrifft das Subnetz 2. Hier sind Routen mit einer Länge von bis zu 7 Routern erlaubt. Folglich wird die Route nicht gelöscht und findet Anwendung bis die Verbindung zwischen Bravo und Delta wieder aufgebaut wird. Das ganze funktioniert aber nur wenn dem RIP-MTI Protokoll genügend Zeit gegeben wird diese Zyklen der einzelnen Subnetze zu erlernen.

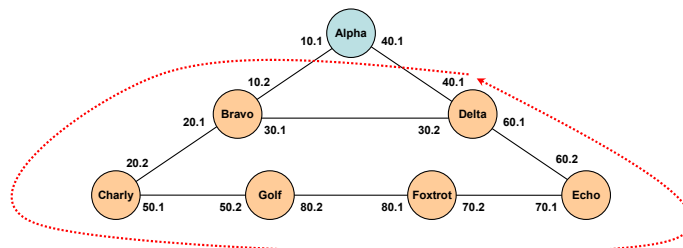


Abb. 7.3.4 Subnetz 3 (Umfang 6)

8. Ausblick

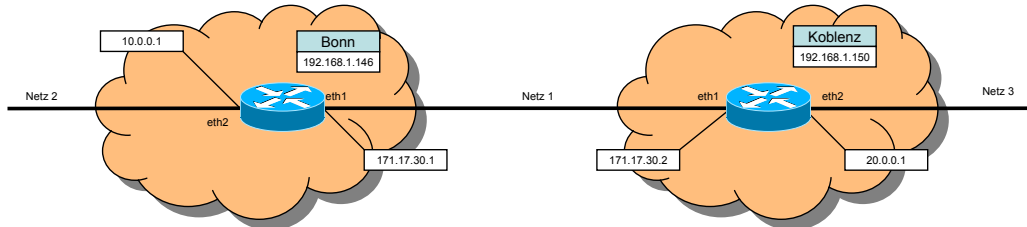
Das Simulationstool VNUML erlaubt auf einfachem Wege komplexe Netzwerke zu simulieren. VNUML erfordert zwar eine gewisse Zeit für die Installation und generell Einiges an Linux Kenntnissen. Sobald aber das Tool vollständig installiert ist, ist das Erstellen neuer Netzwerkszenarios sehr schnell zu erlernen. Für eine genaue Analyse der Szenarien stehen dem Anwender unter Linux sowohl eine Reihe von Systembefehlen zur Verfügung, als auch weitere mächtige Werkzeuge wie Ethereal und Tcpcap.

Der RIP-MTI Algorithmus zeigt eine wesentliche Verbesserung gegenüber dem klassischen RIP Algorithmus. Diese Vorteile fallen um so mehr ins Auge wenn unter RIP die Erweiterungen Split horizon und triggered updates nicht aktiviert werden. Hier kann der RIP-MTI Algorithmus klar seine Stärken ausspielen. Werden nur die übertragenen Aktualisierungspakete betrachtet, so arbeitet das erweiterte Protokoll erheblich effizienter. Es werden keine doppelten oder allgemein unnütze Routen übertragen.

Klar im Vorteil ist der MTI Algorithmus durch seine Fähigkeit zur Erkennung von Routing Schleifen. Durch die im Vorfeld stattfindende Analyse der Routinginformation erkennt der Algorithmus Zyklen im Netzwerk. Er bestimmt den Umfang der Zyklen und kann hierdurch Routen erkennen die über die maximal mögliche Länge herausgehen. Diese Routen markiert der Algorithmus folglich als Routing-schleifen und entfernt diese aus den Routingtabellen. Durch diese im Vorfeld stattfindende Analyse werden auch vermeintliche Schleife bei einer genaueren Betrachtung als korrekte Alternativrouten erkannt und nicht verworfen.

9. Anhang

9.1 Szenario 1



VNUML Datei - Szenario 1 (RIP):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
<vnuml>

  <global>
    <version>1.5</version>
    <simulation_name>RIP3</simulation_name>
    <ssh_key version="1">/root/.ssh/identity.pub</ssh_key>
    <automac/>
    <ip_offset>100</ip_offset>
    <host_mapping/>
    <shell>/bin/sh</shell>
  </global>

  <!--Networks-->

  <net name="Netz1"/>
  <net name="Netz2"/>
  <net name="Netz3"/>

  <!--Nodes-->

  <vm name="Bonna">
    <filesystem type="cow">
      /usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
    <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

    <if id="1" net="Netz1">
      <ipv4 mask="255.255.255.252">171.17.30.1</ipv4>
    </if>

    <if id="2" net="Netz2">
      <ipv4 mask="255.255.255.252">10.0.0.1</ipv4>
    </if>

    <forwarding type="ip" />

    <filetree when="start" root="/usr/local/etc">
      /home/Leif/VNUML/examples/RIP3/Bonn
    </filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">zebra -f
      /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
    <exec seq="start" type="verbatim">
      ripd -f /usr/local/etc/Bonn.conf -d -P 2602</exec>
    <exec seq="stop" type="verbatim">hostname</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
  </vm>
</vnuml>
```

```

</vm>

<vm name="Koblenz">
  <filesystem type="cow">
    /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netz1">
    <ipv4 mask="255.255.255.252">171.17.30.2</ipv4>
  </if>

  <if id="2" net="Netz3">
    <ipv4 mask="255.255.255.252">20.0.0.1</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/RIP3/Koblenz</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/Koblenz.conf -d -P 2602</exec>
  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>

</vm>

</vnuml>

```

VNUML Datei - Szenario 1 (RIP MTI):

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
<vnuml>

  <global>
    <version>1.5</version>
    <simulation_name>RIP3MTI</simulation_name>
    <ssh_key version="1">/root/.ssh/identity.pub</ssh_key>
    <automac/>
    <ip_offset>100</ip_offset>
    <host_mapping/>
    <shell>/bin/sh</shell>
  </global>

  <!--Networks-->

  <net name="Netz1"/>
  <net name="Netz2"/>
  <net name="Netz3"/>

  <!--Nodes-->

  <vm name="Bonna">
    <filesystem type="cow">
      /usr/local/share/vnuml/filesystems/MTI/root</filesystem>
    <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

    <if id="1" net="Netz1">
      <ipv4 mask="255.255.255.252">171.17.30.1</ipv4>
    </if>

    <if id="2" net="Netz2">
      <ipv4 mask="255.255.255.252">10.0.0.1</ipv4>
    </if>
  </vm>

```

```

<forwarding type="ip" />

<filetree when="start" root="/usr/local/etc">
/home/Leif/VNUML/examples/RIP3/BonnMTI</filetree>
<exec seq="start" type="verbatim">hostname</exec>
<exec seq="start" type="verbatim">zebra -f
/usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
<exec seq="start" type="verbatim">
ripd -f /usr/local/etc/Bonn.conf -d -P 2602</exec>
<exec seq="stop" type="verbatim">hostname</exec>
<exec seq="stop" type="verbatim">killall zebra</exec>
<exec seq="stop" type="verbatim">killall ripd</exec>

</vm>

<vm name="Koblenz">
<filesystem type="cow">
/usr/local/share/vnuml/filesystems/MTI/root</filesystem>
<kernel>/usr/local/share/vnuml/kernels/linux</kernel>

<if id="1" net="Netz1">
<ipv4 mask="255.255.255.252">171.17.30.2</ipv4>
</if>

<if id="2" net="Netz3">
<ipv4 mask="255.255.255.252">20.0.0.1</ipv4>
</if>

<forwarding type="ip" />

<filetree when="start" root="/usr/local/etc">
/home/Leif/VNUML/examples/RIP3/KoblenzMTI</filetree>
<exec seq="start" type="verbatim">hostname</exec>
<exec seq="start" type="verbatim">zebra -f
/usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
<exec seq="start" type="verbatim">
ripd -f /usr/local/etc/Koblenz.conf -d -P 2602</exec>
<exec seq="stop" type="verbatim">hostname</exec>
<exec seq="stop" type="verbatim">killall zebra</exec>
<exec seq="stop" type="verbatim">killall ripd</exec>

</vm>

</vnuml>

```

Quagga – RIP Daemon Konfiguration:

Bonn.conf

```

log file /tmp/Bonn.rip.log
!
hostname Bonn
password zebra
!
interface eth1
no ip rip split-horizon
!
interface eth2
no ip rip split-horizon
!
router rip
network 171.17.30.0/24
network 10.0.0.0/24
!

```

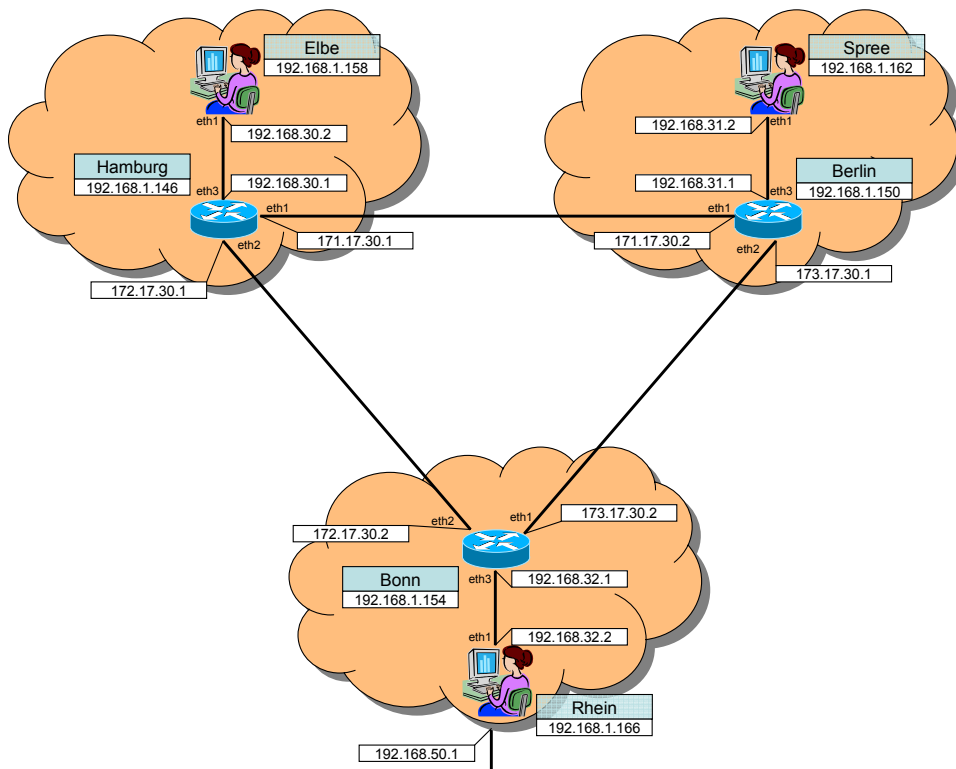
```

Koblenz.conf

log file /tmp/Koblenz.rip.log
!
hostname Koblenz
password zebra
!
interface eth1
no ip rip split-horizon
!
interface eth2
no ip rip split-horizon
!
router rip
network 171.17.30.0/24
network 20.0.0.0/24
!

```

9.2 Szenario 2



VNUML Datei - Szenario 2 (RIP):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
<vnuml>

  <global>
    <version>1.5</version>
    <simulation_name>RIP1</simulation_name>
    <ssh_key version="1">/root/.ssh/identity.pub</ssh_key>
    <automac/>
    <ip_offset>100</ip_offset>
    <host_mapping/>
    <shell>/bin/sh</shell>
  </global>

  <!--Networks-->

  <net name="Netz1"/>
  <net name="Netz2"/>
  <net name="Netz3"/>
  <net name="Netz11"/>
  <net name="Netz12"/>
  <net name="Netz13"/>

  <!--Nodes-->

  <vm name="Hamburg">
    <filesystem type="cow">
      /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
    <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

    <if id="1" net="Netz1">
      <ipv4 mask="255.255.255.0">171.17.30.1</ipv4>
    </if>

    <if id="2" net="Netz2">
      <ipv4 mask="255.255.255.0">172.17.30.1</ipv4>
    </if>

    <if id="3" net="Netz11">
      <ipv4 mask="255.255.255.0">192.168.30.1</ipv4>
    </if>

    <forwarding type="ip" />

    <filetree when="start" root="/usr/local/etc">
      /home/Leif/VNUML/examples/RIP1/Hamburg</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">
      zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
    <exec seq="start" type="verbatim">
      ripd -f /usr/local/etc/Hamburg.conf -d -P 2602</exec>
    <exec seq="stop" type="verbatim">hostname</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>

  </vm>

  <vm name="Berlin">
    <filesystem type="cow">
      /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
    <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

    <if id="1" net="Netz1">
      <ipv4 mask="255.255.255.0">171.17.30.2</ipv4>
    </if>

    <if id="2" net="Netz3">
      <ipv4 mask="255.255.255.0">173.17.30.1</ipv4>
    </if>

    <if id="3" net="Netz12">
```

```

    <ipv4 mask="255.255.255.0">192.168.31.1</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/RIP1/Berlin</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/Berlin.conf -d -P 2602</exec>
  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>

</vm>

<vm name="Bonn">
  <filesystem type="cow">
    /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netz3">
    <ipv4 mask="255.255.255.0">173.17.30.2</ipv4>
  </if>

  <if id="2" net="Netz2">
    <ipv4 mask="255.255.255.0">172.17.30.2</ipv4>
  </if>

  <if id="3" net="Netzl3">
    <ipv4 mask="255.255.255.0">192.168.32.1</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/RIP1/Bonn</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/Bonn.conf -d -P 2602</exec>
  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>

</vm>

<vm name="Elbe">
  <filesystem type="cow">
    /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netzl1">
    <ipv4 mask="255.255.255.0">192.168.30.2</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/RIP1/Elbe</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/Elbe.conf -d -P 2602</exec>
  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>

</vm>

```

```

<vm name="Spree">
  <filesystem type="cow">
    /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netzl2">
    <ipv4 mask="255.255.255.0">192.168.31.2</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/RIP1/Spree</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/Spree.conf -d -P 2602</exec>
  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>

<vm name="Rhein">
  <filesystem type="cow">
    /usr/local/share/vnuml/filesystems/root_fs_tutorial</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netzl3">
    <ipv4 mask="255.255.255.0">192.168.32.2</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/RIP1/Rhein</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/quagga/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/Rhein.conf -d -P 2602</exec>
  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>
</vnuml>

```

Quagga – RIP Daemon Konfiguration:

Berlin.conf

```

log file /tmp/Berlin.rip.log
!
hostname Berlin
password zebra
!
router rip
!
network 192.168.31.0/24
network 171.17.30.0/24
network 173.17.30.0/24
!

```

Bonn.conf

```
log file /tmp/Bonn.rip.log
!  
hostname Bonn  
password zebra  
!  
router rip  
!  
network 192.168.32.0/24  
network 172.17.30.0/24  
network 173.17.30.0/24  
!
```

Elbe.conf

```
log file /tmp/Elbe.rip.log  
!  
hostname Elbe  
password zebra  
!  
router rip  
!  
network 192.168.30.0/24  
!
```

Hamburg.conf

```
Hamburg.conf  
log file /tmp/Hamburg.rip.log  
!  
hostname Hamburg  
password zebra  
!  
router rip  
!  
network 192.168.30.0/24  
network 171.17.30.0/24  
network 172.17.30.0/24  
!
```

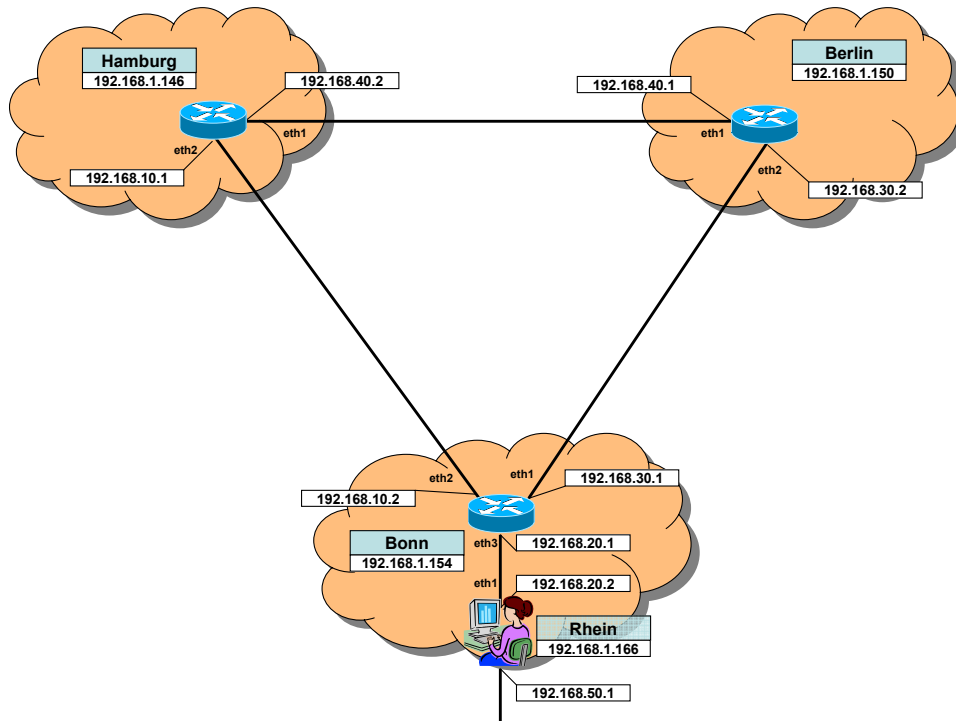
Rhein.conf

```
log file /tmp/Rhein.rip.log  
!  
hostname Rhein  
password zebra  
!  
router rip  
!  
network 192.168.32.0/24  
!
```

Spree.conf

```
log file /tmp/Spree.rip.log  
!  
hostname Spree  
password zebra  
!  
router rip  
!  
network 192.168.31.0/24  
!
```

9.3 Szenario 2 – Neue Adressierung der Schnittstellen und Firewall



VNUML Datei - Szenario 2 (RIP-MTI) mit Firewall:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">

<vnuml>

  <global>
    <version>1.5.0</version>
    <simulation_name>RIP6MTI</simulation_name>
    <ssh_key version="1">/root/.ssh/identity.pub</ssh_key>
    <automac/>
    <ip_offset>100</ip_offset>
    <host_mapping/>
    <shell>/bin/sh</shell>
  </global>

  <net name="Netz1"/>
  <net name="Netz2"/>
  <net name="Netz3"/>
  <net name="Netz4"/>
  <net name="Netz5"/>

  <vm name="Hamburg">
    <filesystem type="cow">/usr/local/share/vnuml/filesystems/MTI/root</filesystem>
    <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

    <if id="1" net="Netz1">
      <ipv4 mask="255.255.255.0">192.168.10.1</ipv4>
    </if>

    <if id="3" net="Netz4">
```

```

    <ipv4 mask="255.255.255.0">192.168.40.2</ipv4>
</if>

<forwarding type="ip" />

<filetree when="start" root="/usr/local/etc">
  /home/Leif/VNUML/examples/cti/rip6/alpha</filetree>
<filetree when="cutalpha" root="/usr/local/bin">
  /home/Leif/VNUML/examples/cti/rip6/binalpha</filetree>

<exec seq="start" type="verbatim">hostname</exec>
<exec seq="start" type="verbatim">
  zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601</exec>
<exec seq="start" type="verbatim">
  ripd -f /usr/local/etc/alpha.conf -d -P 2602</exec>

<exec seq="stop" type="verbatim">hostname</exec>
<exec seq="stop" type="verbatim">killall zebra</exec>
<exec seq="stop" type="verbatim">killall ripd</exec>

<exec seq="cutalpha" type="verbatim">hostname</exec>
<exec seq="cutalpha" type="verbatim">cutalpha.sh</exec>

<exec seq="repairalpha" type="verbatim">hostname</exec>
<exec seq="repairalpha" type="verbatim">repairalpha.sh</exec>

</vm>

<vm name="Bonn">
  <filesystem type="cow">/usr/local/share/vnuml/filesystems/MTI/root</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netz1">
    <ipv4 mask="255.255.255.0">192.168.10.2</ipv4>
  </if>

  <if id="2" net="Netz2">
    <ipv4 mask="255.255.255.0">192.168.20.1</ipv4>
  </if>

  <if id="3" net="Netz3">
    <ipv4 mask="255.255.255.0">192.168.30.1</ipv4>
  </if>

  <forwarding type="ip" />

  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/cti/rip6/bravo</filetree>
  <filetree when="cut" root="/usr/local/bin">
    /home/Leif/VNUML/examples/cti/rip6/bin</filetree>

  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/bravo.conf -d -P 2602</exec>

  <exec seq="cut" type="verbatim">hostname</exec>
  <exec seq="cut" type="verbatim">cutline.sh eth2</exec>

  <exec seq="repair" type="verbatim">hostname</exec>
  <exec seq="repair" type="verbatim">repairl.sh eth2</exec>

  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>

<vm name="Rhein">
  <filesystem type="cow">/usr/local/share/vnuml/filesystems/MTI/root</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netz2">

```

```

    <ipv4 mask="255.255.255.0">192.168.20.2</ipv4>
  </if>

  <if id="2" net="Netz5">
    <ipv4 mask="255.255.255.0">192.168.50.1</ipv4>
  </if>

  <forwarding type="ip" />
  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/cti/rip6/charly</filetree>
  <filetree when="cut" root="/usr/local/bin">
    /home/Leif/VNUML/examples/cti/rip6/bin</filetree>

  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/charly.conf -d -P 2602</exec>

  <exec seq="cut" type="verbatim">hostname</exec>
  <exec seq="cut" type="verbatim">cutline.sh eth1</exec>

  <exec seq="repair" type="verbatim">hostname</exec>
  <exec seq="repair" type="verbatim">repairl.sh eth1</exec>

  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>

<vm name="Berlin">
  <filesystem type="cow">/usr/local/share/vnuml/filesystems/MTI/root</filesystem>
  <kernel>/usr/local/share/vnuml/kernels/linux</kernel>

  <if id="1" net="Netz3">
    <ipv4 mask="255.255.255.0">192.168.30.2</ipv4>
  </if>

  <if id="2" net="Netz4">
    <ipv4 mask="255.255.255.0">192.168.40.1</ipv4>
  </if>

  <forwarding type="ip" />
  <filetree when="start" root="/usr/local/etc">
    /home/Leif/VNUML/examples/cti/rip6/delta</filetree>

  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">
    zebra -f /usr/local/etc/zebra.conf.sample -d -P 2601</exec>
  <exec seq="start" type="verbatim">
    ripd -f /usr/local/etc/delta.conf -d -P 2602</exec>

  <exec seq="stop" type="verbatim">hostname</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>

<host>
  <hostif net="Netz1">
    <ipv4 mask="255.255.255.0">192.168.10.5</ipv4>
  </hostif>
  <hostif net="Netz3">
    <ipv4 mask="255.255.255.0">192.168.30.5</ipv4>
  </hostif>
  <hostif net="Netz4">
    <ipv4 mask="255.255.255.0">192.168.40.5</ipv4>
  </hostif>
</host>
</vnuml>

```

Quagga – RIP Daemon Konfiguration:

Hamburg.conf

```
!  
hostname ripd  
password zebra  
!  
router rip  
!  
network 192.168.10.0/24  
network 192.168.40.0/24  
!
```

Bonn.conf

```
!  
hostname ripd  
password zebra  
!  
router rip  
!  
network 192.168.10.0/24  
network 192.168.20.0/24  
network 192.168.30.0/24  
!
```

Rhein.conf

```
!  
hostname ripd  
password zebra  
!  
router rip  
!  
network 192.168.20.0/24  
network 192.168.50.0/24  
!
```

Berlin.conf

```
!  
hostname ripd  
password zebra  
!  
router rip  
!  
network 192.168.40.1/24  
network 192.168.30.2/24  
!
```

Konfiguration der Firewall

Cut Kommando (*vnumlparser.pl cut@rip6.xml -vB*)

Trennen der Verbindung zwischen Bonn und Rhein

Cutline.sh

```
#!/bin/bash  
iptables -I FORWARD -i $1 -j DROP  
iptables -I FORWARD -o $1 -j DROP  
iptables -I INPUT -i $1 -j DROP  
iptables -I OUTPUT -o $1 -j DROP  
iptables -L -n -v
```

Repair Kommando (*vnumlparser.pl repair@rip6.xml -vB*)
Wiederherstellung der Verbindung zwischen Bonn und Rhein
repairl.sh

```
#!/bin/bash
iptables -D INPUT -i $1 -j DROP
iptables -D OUTPUT -o $1 -j DROP
iptables -D FORWARD -i $1 -j DROP
iptables -D FORWARD -o $1 -j DROP
iptables -L -n -v
```

Cutalpha Kommando (*vnumlparser.pl cutalpha@rip6.xml -vB*)
Trennen der Verbindung zwischen Hamburg und restlichem Netzwerk. Nur
eingehende Pakete werden geblockt.
cutalpha.sh

```
#!/bin/bash
iptables -I INPUT -s 192.168.10.2 -j DROP
iptables -I INPUT -s 192.168.40.1 -j DROP
```

Repairalpha Kommando (*vnumlparser.pl repairalpha@rip6.xml -vB*)
Wiederherstellung der Verbindung zwischen Hamburg und restlichem Netzwerk.
repairalpha.sh

```
#!/bin/bash
iptables -D INPUT -s 192.168.40.1 -j DROP
iptables -D INPUT -s 192.168.10.2 -j DROP
```

Literaturverzeichnis

Andreas J. Schmid, RIP-MTI: Minimum-effort loop-free distance vector routing algorithm.

Diplomarbeit, Universität Koblenz-Landau, 1999.

Benjamin Zapilko, GNU Zebra & Quagga.

Seminararbeit, Universität Koblenz-Landau 2005.

Larry L. Peterson and Bruce S. Davie. Computernetze.

Dpunkt Verlag, 3rd Edition 2004.

Richard Arndt, Virtual Network User Mode Linux.

Seminararbeit, Universität Koblenz-Landau 2005.

Thomas Kleeman, RIPvval – Evaluierung und Weiterentwicklung des RIP-MTI Algorithmus.

Diplomarbeit, Universität Koblenz-Landau, 2001.

Tobias Koch, Implementation und Simulation von RIP-MTI,

Diplomarbeit AG Rechnernetze und Rechnerarchitektur, Universität Koblenz-Landau, April 2005

Quagga Software Routing Suite. <http://quagga.net/>.

Verena Kinder, RIP Routing Configuration.

Seminararbeit, Universität Koblenz-Landau 2005.

VNUML Project home page, <http://jungla.dit.upm.es/~vnuml/>.

Wikipedia, http://de.wikipedia.org/wiki/Routing_Information_Protocol