

**Projektpraktikum Routing-Simulation
SS 2005**

**RIP-Konfiguration mittels
VNUML-Simulator**

Christian Perscheid

Vorwort

Dieses Tutorial wurde im Sommersemester 2005 im Rahmen des Projektpraktikums „Routing-Simulation“ erstellt. Übergeordnetes Ziel war es, ein möglichst umfangreiches Tutorial für den VNUML-Simulator zu schaffen.

Dieser Teil des Tutorials beschäftigt sich mit der Konfiguration von RIP und setzt auf den Teilbereichen „Installation“ und „ICMP“ auf. Es ist daher zu empfehlen sich vorher mit diesen Teilen des Tutorials beschäftigt zu haben.

Ich habe versucht mein Tutorial so aufzubauen, das wenn man es von Anfang bis Ende durcharbeitet, man in der Lage sein sollte, mit dem VNUML-Simulator selbstständig RIP-Netzwerke zu erstellen.

Nach einer kleinen Einführung in RIP folgen einige Grundlagen von VNUML und zum Abschluss einige Beispielszenarios, in die Aufgaben und Lösungen zum Selbsterarbeiten integriert worden sind.

Es empfiehlt sich diese Beispiele in der angegebenen Reihenfolge durchzuarbeiten, da man so schrittweise in die Kniffe der RIP-Konfiguration mittels VNUML eingeführt wird, und außerdem spätere Beispiele oft auf den Basisbeispielen aufsetzen.

Diese Beispiele finden sich vorprogrammiert auf der Ergebnis-DVD dieses Projektpraktikums. Der Name der zugehörigen Datei steht im Inhaltsverzeichnis in Klammern hinter dem jeweiligen Beispiel.

Um die Beispiele aus der Zip-Datei zu verwenden, bitte ich den Inhalt dieser Datei in den Ordner /usr/local/share/vnuml/examples/rip/ der VNUML-Installation zu entpacken.

Christian Perscheid

Kapitel 1: Einführung RIP	
1.1 Grundlagen Netzaufbau	4
1.2 Routing Information Protocol (RIP)	4
1.2.1 Erstellen der Routing Tabellen	4
1.2.2 Pakete weiterleiten	5
1.2.3 Routing Updates und Timer	5
1.2.4 Verbesserungen der Konvergenz	6
1.2.5 RIP Nachrichten Format	7
1.2.6 Classful Routing	8
1.3 RIPv2	9
Kapitel 2: RIP mit VNUML: Grundlagen	11
2.1 ZEBRA	11
2.1.1 Was ist ZEBRA ?	11
2.1.2 Die Daemons	11
2.1.3 einige generische Konfigurationsbefehle	12
2.1.4 Beispiel Konfigurationsdatei	12
2.1.5 Gemeinsame Aufrufoptionen	13
2.1.6 Überblick VTY	13
2.2 Der Routing-Daemon ripd	14
2.2.1 Starten und Stoppen von ripd	14
2.2.2 RIP netmask	14
2.2.3 RIP Konfiguration	15
2.2.4 RIP Informationen anzeigen	17
2.2.5 RIP Debug Befehle	17
Kapitel 3: RIP mit VNUML: Beispiel-Szenarios	18
3.1 Ein einfaches Beispielnetzwerk mit manueller RIP-Konfiguration	18
Aufgabe 1: Nachbau des Netzwerkes (ripbasic)	22
Aufgabe 2: Erweitern des Netzwerkes (ripextended)	22
Lösung	23
3.2 An- und Abschalten einzelner Interfaces	26
Aufgabe 3: Weitere Routen abschalten	26
3.3 Alternative Konfiguration der Daemons über Telnet	27
3.4 Booten mit automatischer RIP-Konfiguration (ripauto)	28
3.5 Filtern von Paketen	30
3.5.1 distribute lists	30
3.5.2 iptables	31
3.6 Erweiterungen der RIP-Konfiguration	34
3.6.1 Erweitern der ripd.conf (ripextconf)	34
Aufgabe 4: Notwendiges XML-Szenario erstellen	34
Lösung	35
Aufgabe 5: Konfigurationsdateien erstellen	36
Lösung	37
Aufgabe 6: Konfigurationsdateien für Andy, Goober, Opie	37
Lösung	38
3.6.2 RIPv1 Kompatibilität (ripv1)	39
Aufgabe 7: „version 1“ einfügen	40
Aufgabe 8: Warum fehlen plötzlich Einträge ?	40
Lösung	41
3.6.3 Debug-Funktionen und log-Datei (riplog)	41
Aufgabe 9: Log-Dateien aktivieren	41
3.6.4 Passive Interfaces	44
Anhang: Glossar aller unterstützter RIP-Konfigurationsbefehle	45

Kapitel 1

Einführung RIP

1. Netzaufbau

Um Pakete von einem Rechner zum anderen zu senden, könnte man alle Rechner mit jedem einzelnen verbinden. Dieser Aufwand wäre aber viel zu hoch. Ein *Netzwerk* fasst mehrere Rechner eines Bereiches zusammen. Diese Netzwerke kommunizieren über Router miteinander. Für die Kommunikation sind *Routing Protokolle* notwendig.

Des Weiteren fasst man mehrere Netze mit Routern zu einem *autonomen System* zusammen. Ein autonomes System steht meist unter einer administrativen Kontrolle.

Die Kommunikation der Router innerhalb eines autonomen Systems erfolgt über das *Interior Gateway Protocol (IGP)*. Dem gegenüber steht das *Exterior Gateway Protocol (EGP)*, das für die Weiterleitung zwischen den einzelnen autonomen Systemen verantwortlich ist.

Zu den IGP's zählen:

- Distance Vector Routing Protocol: *Routing Information Protocol (RIP)*
- Link State Routing Protocol: *Open Shortest Path First Protocol (OSPF)*

2. Routing Information Protocol (RIP)

RIP und OSPF zählen zu den *dynamischen Routing-Verfahren*. Dabei geht es darum, wie Router mit Hilfe von Protokollen automatisch ihre Routing Tabelle erstellen und aktualisieren. Der Weg eines Paketes ist beim dynamischen Routing nicht festgelegt. Er kann sich abhängig vom Netzzustand und den davon abhängigen Routerinformationen ändern.

Das RIP-Protokoll arbeitet mit einem *Distance Vector Protokoll Algorithmus*. Dabei kennt jeder Router seine direkten Nachbarn, den Zustand der Verbindungsleitungen zu diesen Nachbarn und die Kosten dieser Leitung.

Wenn der Router ein Datenpaket empfängt, muss er entscheiden über welches Interface er es weiterleitet. Er muss folglich Wissen über die Netzwerktopologie besitzen. Beim Routing werden die Topologie des Netzes und der optimale Pfad zu allen erreichbaren Knoten ermittelt. Diese Informationen speichert der Router in einer *Routing Tabelle*.

2.1 Erstellung der Routing Tabellen

Zu Beginn weiß jeder Router nur an welche Netzwerke seine Schnittstellen grenzen. Die Kosten dorthin betragen 0 (RIP 1). Diese Information speichert der Router in seiner Routing Tabelle. Diese Routing Tabelle schickt er an seine direkten Nachbarn, die unterdessen den gleichen Prozess vollzogen haben. Die Router aktualisieren ihre Tabellen mit den Informationen des Nachbarn usw. bis alle Router die Netzwerktopologie kennen. Die Zeit um diese Informationen zu versenden nennt man *Konvergenzzeit*. Die Tabellen befinden sich anschließend in einem stabilen konsistenten Zustand, d.h. es dürfen keine Schleifen entstehen. Ebenfalls muss sichergestellt werden, dass bei einem Verbindungsausfall alle Nachbarn informiert werden, diese wiederum ihre Nachbarn informieren usw. Wenn eine Aktualisierung mit einer besseren Metrik eintrifft, wird der Next Hop durch denjenigen Router ersetzt von dem die Information kam. Falls eine Route mit verschlechterter Metrik ankommt, wird diese nur in die Tabelle eingefügt, wenn es sich um die gleichen Next Hop Adressen handelt.

Der Router hat also nur eine eingeschränkte Sichtweise über das Netz. Es kann z.B. sein, dass er zwei Wege zu einer Zieladresse als gleichwertig erkennt, weil sie die gleiche Hop Anzahl haben, aber die Geschwindigkeit des einen Pfades geringer ist.

Es gibt zwei Nachrichten, die die Router untereinander austauschen:
Requests (Anfragen) und Response (Antworten)

2.2 Pakete weiterleiten

Wenn nun ein Paket eintrifft ermittelt der Router anhand der Routing Tabelle an welches Netz das Paket weitergeleitet werden muss und in der Weiterleitungstabelle erkennt er über welches Interface das Paket den Router verlassen muss.

Was ist aber, wenn eine Netzwerknummer über zwei verschiedene Wege erreichbar ist ? Welchen Weg soll der Router nun wählen ? Es muss ein Parameter für die Gewichtung der Pfade existieren, um den besten Weg zum Ziel in die Tabelle eintragen zu können: die *Metrik*. Die Metrik umfasst folgende Parameter: Hop Count, Bandweite, load, Verzögerung, Zuverlässigkeit und Kosten.

Bei RIP verwendet man nur eine Metrik: *Hop Count*. Das ist die Anzahl der Router die ein Paket passieren muss um das Ziel zu erreichen. Die Anzahl der Hops kann zwischen 1 und 15 liegen, während 16 als unendlich definiert ist. 1 bedeutet direkt verbundenes Netzwerk (meistens 0). Deshalb sollte RIP nur in Netzen verwendet werden, deren Wege nicht länger als 16 Hops sind.

2.3 Routing Updates und Timer

Broadcast Update

Wenn ein Router im Netzwerk ist, sendet er seine Routing Tabelle als Broadcast. Sprich zur IP 255.255.255.255. Alle Nachbarn hören diesen Broadcast und reagieren. CISCO verfährt so. Es gibt auch RIP Versionen bei denen die Router nur zu ihren direkten Nachbarn senden.

Periodische Updates

Die Router senden periodisch (Update Timer steht auf 30 Sekunden) ihre Routing Tabellen an ihre Nachbarn. Die Zeit hierfür sollte nicht zu hoch sein damit die Konvergenzzeit nicht zu stark ansteigt und nicht zu kurz, damit keine zu hohen Netzlasten entstehen.

Bei RIP enthält dieser Update Timer noch eine Random Variable um Synchronisation vorzubeugen.

Invalidation Timer – Lebenszeichen

Wenn ein Router ausfällt, schickt er keine periodischen Updates mehr. Da alle Router einen Timer (Invalidation Timer) für jeden Routing Tabellen Eintrag besitzen, läuft dieser irgendwann ab und der Eintrag zum ausgefallenen Router wird als unerreichbar markiert.

Somit wird sichergestellt, dass keine Pakete mehr an diesen geleitet werden.

Dieser Timer ist auf 6mal Update Zeit gestellt.

Garbage Collection Timer

Ist die Zeit, die benötigt wird, bis der Eintrag einer Route aus der Tabelle entfernt wird.

Triggered Update

Routing Aktualisierung wird gesendet, wenn sie in der Routing Tabelle etwas ändert. Bei RIP werden aber nur die Einträge versendet, die sich auch wirklich verändert haben. Ebenfalls im Unterschied zu den normalen periodischen Updates wird der Update Timer (Invalidation Timer) nicht auf 0 gesetzt (sonst Synchronisation möglich). Um eine Flut von triggered Updates zu vermeiden, setzt man einen nächsten Timer von 1,5 Sekunden, erst dann wird die Aktualisierung verschickt.

2.4 Verbesserungen der Konvergenz

Split Horizon

Was passiert, wenn ein Router A ein Paket erhält und unterschiedet dieses über B weiterzuleiten. In B angekommen wird aber entschieden über A weiterzuleiten ? Es ist eine Schleife entstanden. Der Ursprung dieses Problems liegt darin, dass ein Router die gelernte Information wieder an den Router sendet, von der er diese Information erhalten hat (*reverse route*). Split Horizon ist ein Verfahren, das die *reverse route* unterbindet. Man unterscheidet hierbei zwischen „*Split Horizon*“ und „*Split Horizon with poisoned reverse*“.

„Split Horizon“: Wenn ein Router Updates über seine Interfaces losschickt, lässt er das Interface aus, über dieses er die Information erhalten hat.

„Split Horizon with poisoned reverse“: Im Prinzip genauso, nur dass er die besagten Interfaces nicht weglässt, sondern als unerreichbar markiert.

➔ RIP unterstützt Split Horizon with poisoned reverse

Count to Infinity Problem

Split Horizon löst zwar das Problem von Schleifen zwischen Nachbarn aber nicht von Schleifen im Netzwerk.

Dieses Problem heisst „counting to infinity“ und wird gelöst durch das Definieren von infinity. Man definiert einen festen Wert an hops (16), der nicht überschritten werden darf. Wenn dies doch eintritt so wird dieser Eintrag als unerreichbar markiert. Aber je nach Update Zeit kann die Zeit bis die „Unerreichbarkeit“ eintritt sehr hoch sein.

Lösung: „*triggered update*“ und „*holddown time*“

Triggered Update (flash update):

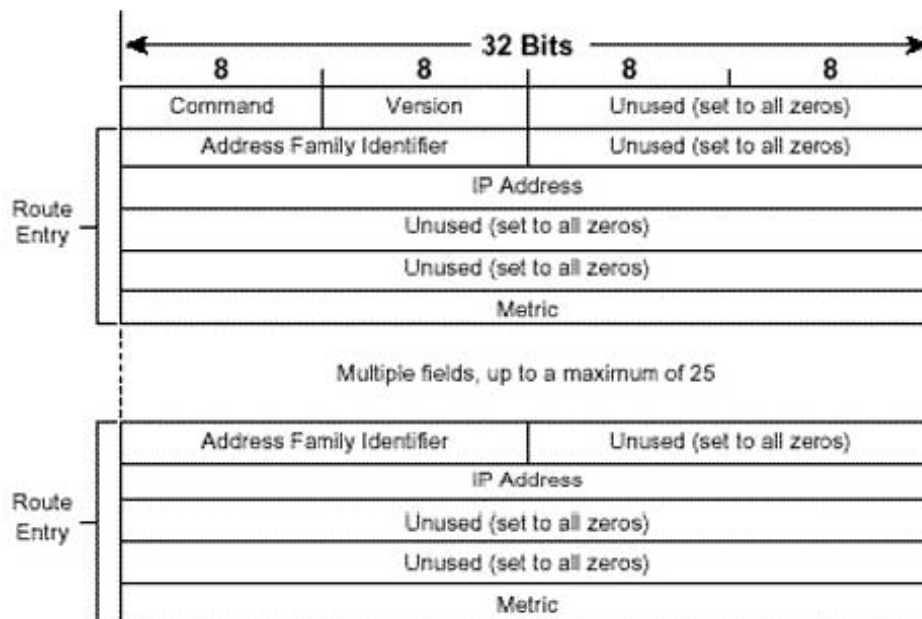
Nachdem sich die Metrik verbessert/verschlechtert hat erfolgt unmittelbar ein Update. Nicht erst nach der periodischen Update Zeit.

Problem: Wenn nun alle nahezu gleichzeitig updaten kommt es zu Verzögerungen.

Holddown Timers:

Wenn eine Hop-Distanz ansteigt wird die holddown Zeit gesetzt. Solange diese Zeit abläuft, wird der Router keine neuen Updates akzeptieren. CISCO definiert so einen Timer für RIP, nicht jedoch für RFC1058. CISCO setzt diesen Timer auf 6mal Update Zeit.

2.5 RIP Nachrichten Format



Jede Nachricht beinhaltet einen Befehl, die Versionsnummer und bis zu 25 Routen. Jeder Routen Eintrag hat: Eine Adress-Familien-Bezeichnung, die Ziel IP Adresse und die Metrik (die Hop Anzahl). Wenn mehr als 25 Einträge versendet werden müssen, müssen mehrerer RIP Nachrichten verwendet werden.

Command:

- 1: Request/Anfrage Nachricht Eine Anfrage an alle angeschlossenen Router, die komplette oder Teile ihrer Routing Tabelle an den anfragenden Router zu übermitteln.
- 2: Response/Antwort Nachricht Eine Antwort enthält die komplette oder Teile der Routing Tabelle des Senders. Diese Meldung wird als Reaktion auf einen RIP Request gesendet.

Version:

- 1: RIP v1
- 2: RIP v2

Address-Family Identifier:

- 2: TCP/IP Protokoll Verwendung

IP Address: Gibt die IP Adresse an, über die ein bestimmtes Netz zu erreichen ist. Entweder: Major Adresse, Subnetz Adresse, Host Adresse.

Metric: 1: direkt verbundenes Netzwerk
15: größt mögliche Entfernung
16: unerreichbar

Unused:

Alle Felder die nicht belegt sind enthalten eine 0.

Request Nachricht:

Adress-Familien Bezeichnung: 0 IP Adresse: 0 Metrik 16

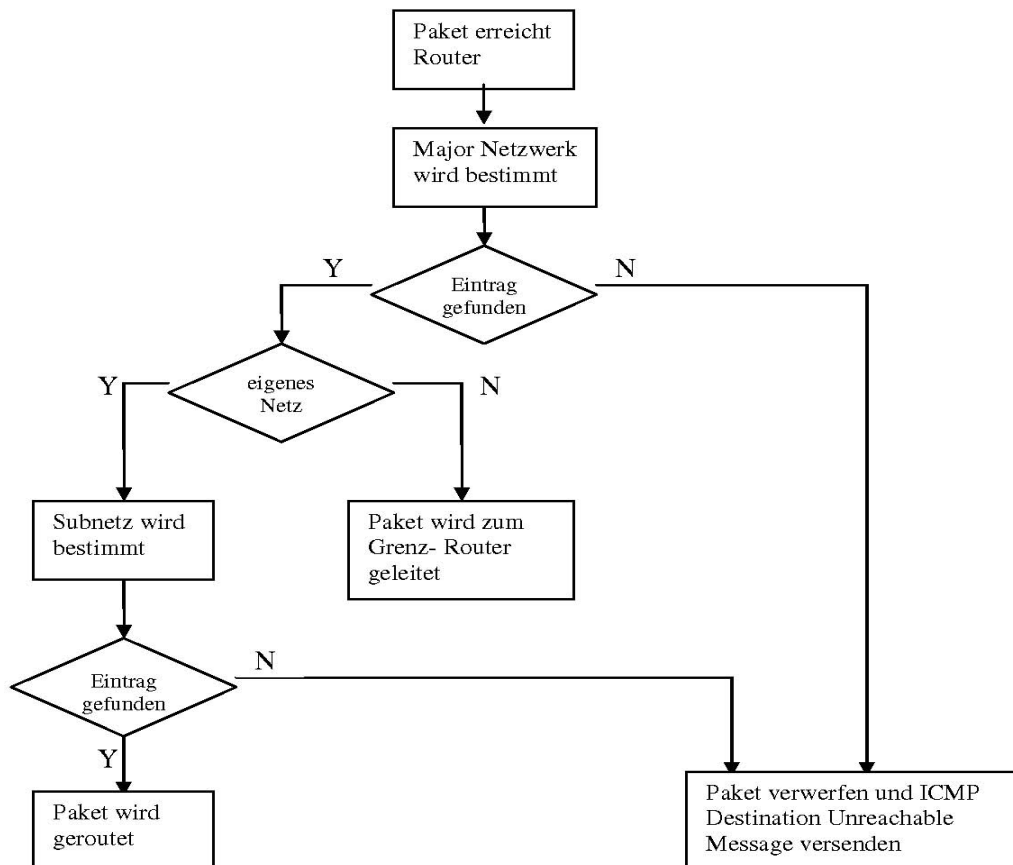
- ➔ man erhält von den Routern, die die Anfrage erhalten die komplette Tabelle geliefert.
- ➔ setzt man die IP Adresse auf eine spezielle Adresse so bekommt man Infos zur Route für diese Ziel Adresse.

2.6 Classful Routing

Router innerhalb eines Netzwerk-Adress-Bereichs besitzen innerhalb ihrer Routing Tabelle nur Subnetz Einträge für ihr Netzwerk-Adress-Bereich. Sie haben allerdings keine Subnetz Einträge für Subnetze anderer Netzwerke. Für die Übertragung von Pakete in andere Netzwerke haben diese Router in ihren Tabellen einen Eintrag zum Boundary Router. Boundary Router sind Router, die sich an Grenzen zwischen mehreren Netzwerken befinden. Sie führen route summarization/subnet hiding durch. Diese haben Subnetz Einträge für diese Netzwerke.

Das RIP Nachrichten Format hat keinen Eintrag für Subnetz Masken. Dadurch muss die Subnetz-Maske im gesamten Gebiet konstant sein, es ist nicht möglich eine variable Länge zu verwenden. Wenn ein Update geschickt wird mit einer anderen als der Subnetz-Nummer werden nur die Klasse A, B oder C Nummern veröffentlicht.

Routing Tabellen-Lookup bei der Paket-Weiterleitung

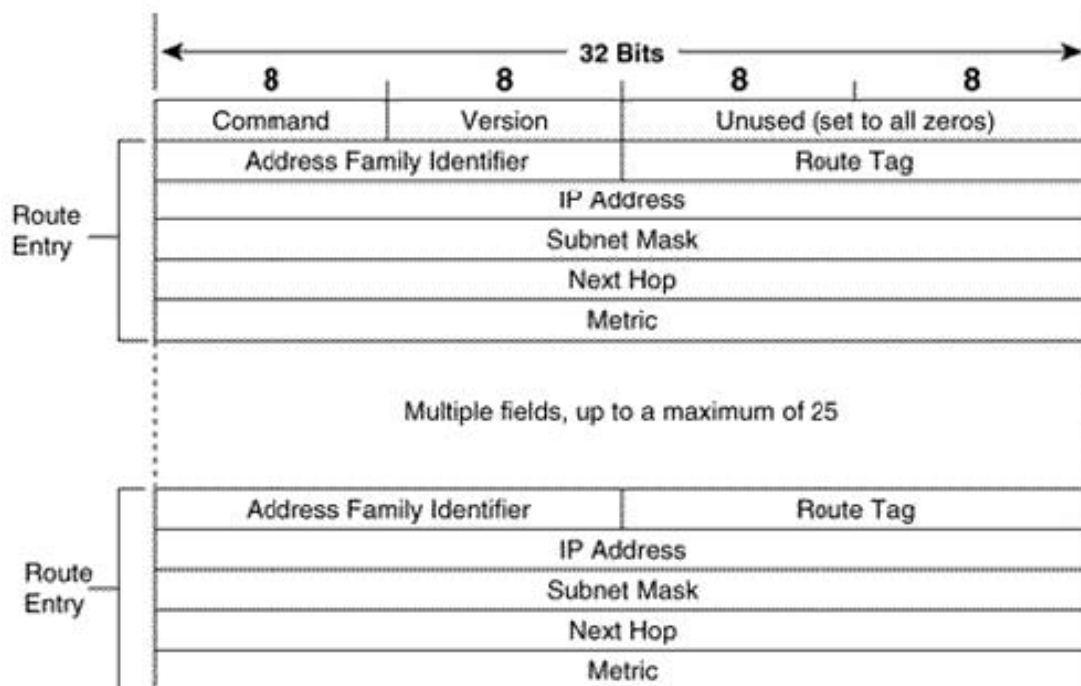


3. Routing Information Protocol Version 2 (RIP v2)

Die IP-basierten Netzwerke haben sich stark vergrößert und so wurde auch eine Verbesserung der RIP v1 erforderlich. RFC1723 beschreibt RIP v2 ab IOS Version 11.1 und später wird RIP v2 unterstützt. Sie löst nicht die ältere Version ab, sondern wird zusätzlich zur Verfügung gestellt. RIP v2 bietet die Möglichkeit mehr Informationen zu übertragen.

Multicast

Broadcast wurde durch Multicast ersetzt. Multicast ist viel effizienter. Multicast kopiert Datenpaketen bei deren Transport. Und zwar genau dort, wo eine Verzweigung vom Sender zu den Empfängern im Netz vorliegt. Broadcasting verläuft ungezielt. So ist die Ausstrahlung von Fernseh-oder Hörfunkprogrammen dem Broadcasting zuzuordnen, während ein Internet-Videostream mit auswählt versorgten Empfängern zum Multicasting gehört. Bei RIP v2 sieht ein Multicast so aus, dass ein Router seine Updates zu allen RIP v2 sprechenden Routern sendet, indem er das Updat zu der Klasse D Adresse 224.0.0.9 schickt. Der Vorteil ist, dass alle nicht RIP fähigen Router sich nicht mit unnötigen durch Broadcast empfangenden Nachrichten aufhalten müssen.



Command, Address Family Identifier, IP Address, Metric haben sich nicht verändert. Natürlich steht unter Version nun 2.

Subnet Mask:

Classless Inter Domain Routing wird bei Version 1 nicht unterstützt. In Version2 wird nun mit jeder Route die Netzmaske mit versendet. Eine 32 Bit Maske die Netzwerk und Subnetz Teil einer IP Adresse sichtbar macht.

Classless

RIPv2, OSPF kann Information über Subnetz-Masken in Updates tragen. Dadurch wird es möglich Subnetz-Masken mit variabler Länge zu verwenden (VSLM). Also kann man mit einem Protokoll das classless Routing unterstützt die CIDR (Classless Inter-Domain Routing) Technik verwenden. Ein Vorteil von Masken mit variabler Länge ist eine effiziente Adressraumverteilung. Mittels Subnetting ist es möglich eine Adresse in mehrere kleiner

Adressbereiche zu unterteilen (Subnetze). Wenn man aber nun Subnetting ohne VLSM verwendet, so kann man nur Masken der folgenden Form verwenden: 255.0.0.0/8 oder 255.255.0.0/16 oder 255.255.255.0/24. Das bedeutet, wenn man 256 Hostadressen benötigt, müsste Adresse mit einer 255.255.0.0 Maske verwendet werden. Damit sind aber 65534 Host Adressen möglich. Nicht gerade effizient. Nutzt man nun die Möglichkeit von VLSM so kann man eine 255.255.254.0/23 Maske verwenden, mit 510 Host Adressen. Wesentlich effizienter. Die Technik CIDR lässt Routing Tabellen nicht ins Unermessliche Wachsen, denn CIDR macht Supernetting möglich. Supernetting oder Aggregation fasst mehrere Netze zu einem größeren zusammen. Die Einträge 172.12.6.128 und 172.12.6.148 sind 27 Stellen lang identisch, sie können zu 172.12.6.128/27 zusammengefasst werden.

Authentication

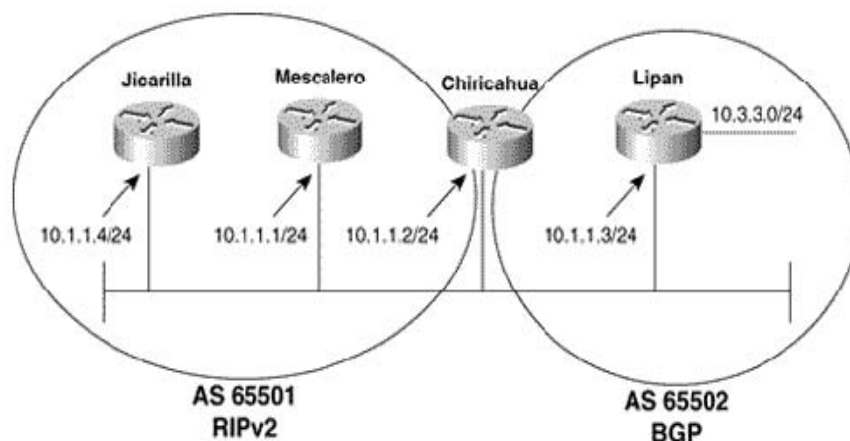
Aus Sicherheitsgründen ist es nun auch möglich Routingupdates mittels Passwort zu authentifizieren. Hiermit kann die Quelle des Updates bestimmt werden. Hierfür ist der erste Eintrag im Nachrichtenformat vorgesehen. Es gibt zwei Möglichkeiten für die Verschlüsselung md5 oder text.

Route Tag:

Markiert RIP Einträge, die externe Routen enthalten. Z.B. die Nummer des Autonomen Systems, die diese von einem externen Routing Protokoll kommen. Diesen Tag können auch andere Routing Protokolle lesen und somit können Infos über RIP Domains transportiert werden.

Next Hop:

Mit Version 2 ist es möglich Router, die nicht RIP sprechen, als Next Hop zu haben, der nächste Hop Adresse wird in die Routing Tabelle eingetragen. In der alten Version wurde immer der Router als nächster Hop genommen, von dem die Information kam. Wenn nächster Hop gleich zuzusender Hop ist, lautet der Eintrag 0.0.0.0



Hier erkennt man die Vorteile die Next Hop und das Route Tag mit sich bringen. Denn wenn Jicarilla etwas an

10.3.3.0 senden möchte, muss das Paket normalerweise den Gang über den Boundary Router Chiricahua machen. Nun sendet dieser Router aber an Mescalero und Jicarilla die Nachricht, dass das Netz 10.3.3.0 mit der Maske 255.255.255.0 in dem Autonomen System 65502 liegt und der nächste Hop nicht er sondern Lipan ist.

Kapitel 2

RIP mit VNUML

Die virtuellen Netzwerke, die man mit VNUML erstellt, verfügen im Allgemeinen nur über lokale Konnektivität. Um nun das dynamische Routingprotokoll RIP zu konfigurieren, benutzt VNUML eine open-source Implementation namens Zebra (www.zebra.org). Das Zebra Paket ist standardmäßig in den von VNUML unterstützten Filesystemen bereits installiert, deshalb sind keine zusätzlichen Installationen notwendig.

1. ZEBRA

1.1 Was ist Zebra ?

Zebra ist ein Routing Paket, das TCP/IP basierte Routing Services mit Routing Protokoll Support wie RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, BGP-4 and BGP-4+ bereitstellt. Zusätzlich zu den traditionellen IPv4 Routing Protokollen, unterstützt Zebra auch IPv6 Routing Protokolle.

Zebra stellt eine hochwertige Multi Server Routing Engine, bietet user interfaces für jedes Routing-Protokoll und unterstützt die gängigen Kommandozeilenbefehle.

1.2 Die Daemons

Es sind fünf Routing Daemons in Gebrauch und es gibt einen Manager Daemon. Diese Daemons können auf separaten Maschinen vom Manager Daemon lokalisiert werden. Jeder dieser Daemons wartet an einem eigenen Port auf ankommende VTY Verbindungen. (VTY steht für Virtual Teletype interface. Das bedeutet, dass man eine Telnet-Verbindung zu dem Daemon aufbauen kann)

Die Routing Daemons sind:

- ➔ `ripd`, `ripngd`, `ospfd`, `ospf6d`, `bgpd`
- ➔ `zebra`

In eine Config-Datei kann man die Debugging Optionen, ein vty Passwort, Routing Daemon Konfigurationen, einen log-file Namen usw. reinschreiben. Diese Informationen formen die Startbefehle die ausgeführt werden, wenn der Daemon gestartet wird.

Diese Konfigurationsdateien befinden sich generell hier:

```
/usr/local/etc/*.conf
```

Jeder Daemon hat sein eigene Konfigurationsdatei. Zum Beispiel ist die standardmäßig verwendete Konfigurationsdatei von Zebra:

```
/usr/local/etc/zebra.conf
```

Der Daemon Name plus .conf ist der standardmäßige Name der Konfigurationsdateien. Man kann auch eine eigene Konfigurationsdatei benutzen, indem man beim Starten des Daemons `-f` gefolgt von dem Pfad der Konfigurationsdatei anhängt. Zum Beispiel für den Zebra-Daemon:

```
zebra -f /usr/local/etc/zebra.conf
```

1.3 einige generische Konfigurationsbefehle

Befehl	Beschreibung
hostname hostname	Setzt den Hostnamen des Routers
password password	Setzt das Passwort für das vty-Interface. Wenn es keines gibt, akzeptiert ein vty keine Verbindungen
enable password password	Setzt das enable Passwort
log stdout / no log stdout	Setzt die Log-Ausgabe auf stdout
log file filename	Log-Ausgabe in Datei. z.B. log file /usr/local/etc/bgpd.log
log syslog / no log syslog	Setzt die Log-Ausgabe auf syslog
write terminal	Zeigt die aktuelle Konfiguration auf dem vty-Interface an
write file	Schreibt die aktuelle Konfiguration in die Config-Datei
configure terminal	Umschalten in den Konfigurationsmodus. Dieser Befehl ist der erste Befehl für die Konfiguration
terminal length <0-512>	Setzt die Terminalanzeigelänge. Bei der Länge 0 wird keine Anzeigekontrolle durchgeführt.
list	Zeigt alle Befehle an
service password-encryption	Encrypted das Passwort
show version	Zeigt die verwendete Zebra-Version und deren Build-Host Informationen an
line vty	In den vty-Konfigurationsmodus wechseln
exec-timeout minute	
exec-timeout minute second	Setzt eine Timeoutzeit für vty-Verbindungen
access-class access-list	Lehnt vty-Verbindungen mit einer Access-Liste ab

1.4 Beispiel Konfigurationsdatei

Unten ist eine Beispielkonfigurationsdatei für den Zebra-Daemon.

```
!  
! Zebra configuration file  
!  
hostname Router  
password zebra  
enable password zebra  
!  
log stdout  
!  
!
```

„!“ und „#“ sind Kommentierungszeichen. Wenn der erste Buchstabe eines Wortes eines dieser Symbole ist, wird der Rest der Zeile als Kommentar ignoriert.

```
password zebra!password
```

Wenn ein Kommentarsymbol nicht der erste Buchstabe des Wortes ist, ist es ein normaler Buchstabe. Also wird im obigen Beispiel „!“ nicht für ein Kommentarsymbol gehalten und „zebra!password“ wird das neue Passwort.

1.5 Gemeinsame Aufrufoptionen

Diese Optionen gelten für alle Zebra Daemons.

- d oder --daemon
Läuft in Daemon Modus.
- f *file* oder --config_file=*file*
Setzt den Namen der Konfigurationsdatei.
- h oder --help
Zeigt diese Hilfe an und geht raus.
- i *file* oder --pid_file=*file*
Beim Start wird der process identifier des Daemons in eine Datei geschrieben, typischerweise in /var/run. Diese Datei kann vom Init-System benutzt werden, um Befehle zu implementieren wie z.B.
.../init.d/zebra status, .../init.d/zebra restart
or .../init.d/zebra stop.
- P *port* oder --vty_port=*port*
Setzt die VTY Portnummer.
- v oder --version
Gibt die Programmversion aus.

1.6 Überblick VTY

VTY steht für Virtual Teletype Interface. Das bedeutet man kann via Telnet Protokoll eine Verbindung zu dem Daemon aufbauen. Um ein VTY Interface zu aktivieren, muss man ein VTY Passwort festlegen. Wenn kein VTY Passwort existiert, kann niemand sich zu dem Interface verbinden.

```
% telnet localhost 2601

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is zebra (version 0.94)
Copyright 1997-2000 Kunihiro Ishiguro

User Access Verification

Password: XXXXX
Router> ?
  enable          Turn on privileged commands
  exit            Exit current mode and down to previous mode
  help            Description of the interactive help system
  list            Print command list
  show            Show running system information
  who             Display who is on a vty
Router> enable
Password: XXXXX
Router# configure terminal
Router(config)# interface eth0
Router(config-if)# ip address 10.0.0.1/8
Router(config-if)# ^Z
Router#
'?' ist sehr nützlich um Befehle nachzusehen.
```

2. Der RIP-Daemon ripd

ripd unterstützt die RIP Version 1 wie beschrieben in RFC1058 und die RIP Version 2 wie beschrieben in RFC2453.

2.1 Starten und Stoppen von ripd

Die standardmäßige Konfigurationsdatei von ripd ist `ripd.conf`. Wenn ripd gestartet wird, sucht er im Verzeichnis `/usr/local/etc`. Wenn die `ripd.conf` sich dort nicht befinden sollte, sucht er als nächstes im aktuellen Verzeichnis danach.

RIP benutzt den UDP Port 521 zum Senden und Empfangen von RIP Paketen.

Das RIP Protokoll benötigt die Interface Informationen des ZEBRA Daemons. Deshalb ist ein laufender ZEBRA-Daemon zwingende Voraussetzung für ripd.

Daher muss vor dem Start von ripd immer zuerst zebra gestartet werden:

```
# zebra -d
# ripd -d
```

Um ripd zu stoppen, benutzt man den Befehl:

```
killall ripd (für zebra: killall zebra)
```

Verschiedene Signale haben spezielle Bedeutungen für ripd:

SIGHUP

Lädt erneut die Konfigurationsdatei `ripd.conf`. Alle Konfigurationen werden zurückgesetzt. Alle bisher gelernten Routen werden aus der Routing Tabelle gelöscht.

SIGUSR1

Rotiert das `ripd` logfile.

SIGINT

SIGTERM

ripd löscht alle installierten RIP Routen und terminiert korrekt.

2.2 RIP netmask

Die netmask Features von ripd unterstützen beide Version 1 und 2 von RIP. Version 1 von RIP beinhaltete ursprünglich keine netmask Informationen. In RIP Version 1 wurden die Netzwerkklassen ursprünglich dazu benutzt die Größe der Netzmaske zu bestimmen.

Klasse A Netzwerke benutzen 8 bits der Maske, Klasse B 16 bits und Klasse C Netzwerke 24 bits.

Version 2 von RIP unterstützt eine subnet mask von variabler Länge (VLSM). Wenn man die subnet mask erweitert kann die Maske unterteilt und wiederbenutzt werden. Jedes subnet kann für verschiedene Zwecke benutzt werden, so z.B. grosse bis mittelgrosse LAN und WLAN Verbindungen.

Zebra ripd unterstützt nicht die nicht-sequentiellen netmasks welche in RIP Version 2 beinhaltet sind.

2.3 RIP Konfiguration

router rip

Der `router rip` Befehl ist notwendig um RIP zu aktivieren. Um es zu deaktivieren benutzt man den Befehl `no router rip`. RIP muss aktiviert sein, bevor irgendwelche RIP Befehle ausgeführt werden können.

no router rip

RIP deaktivieren.

RIP kann so konfiguriert werden, dass es entweder Version 1 oder Version 2 Pakete verarbeiten kann, der Standard Modus ist Version 2. Wenn keine Version spezifiziert worden ist, wird der RIP Daemon Version 2 verwenden. Wenn RIP auf Version 1 gesetzt wurde, erscheint die Anzeige „Version 1“, aber wenn Version 2 benutzt wird erscheint dies nicht.

network network

no network network

Mit diesem Befehl gibt man die Interfaces an für die RIP aktiviert sein soll. Diejenigen Interfaces deren Adressen mit denen des Netzwerks übereinstimmen sind aktiviert.

Diese Gruppe von Befehlen aktivieren oder deaktivieren jeweils RIP interfaces zwischen bestimmten Nummern einer spezifizierten Netzwerkadresse. Wenn z.B. das Netzwerk für 10.0.0.0/24 RIP-aktiviert ist, würde sich daraus ergeben, dass alle Adressen von 10.0.0.0 bis 10.0.0.255 RIP-aktiviert sind.

network ifname

no network ifname

Bestimmt ein RIP-aktiviertes Interface *ifname*.. Sowohl das Senden als auch das Empfangen von RIP-Paketen wird aktiviert an dem im `network ifname` Befehl spezifizierten Port.

neighbor a.b.c.d

no neighbor a.b.c.d

Spezifiziert einen RIP Nachbarn. Wenn ein Nachbar kein Multicast versteht, ist dieser Befehl nützlich um Nachbarn zu spezifizieren. In manchen Fällen sind nicht alle Router fähig Multicast zu verstehen, wo die Pakete zu einem Netzwerk oder einer Gruppe von Adressen gesendet werden. In einer Situation wo ein Nachbar keine Multicast Pakete verarbeiten können, ist es notwendig eine direkte Verbindung zwischen den Routern herzustellen. Der `neighbor` Befehl erlaubt es dem Netzwerkadministrator einen Router als RIP Nachbarn zu spezifizieren. Der `no neighbor a.b.c.d` Befehl deaktiviert den RIP Nachbarn.

Unten ist eine sehr einfache RIP Konfiguration. Das Interface `eth0` und die Interfaces, deren Adressen zu 10.0.0.0/8 passen sind RIP-aktiviert.

```
!  
    router rip  
        network 10.0.0.0/8  
        network eth0  
!
```

Passive Interfaces:

```
passive-interface IFNAME  
no passive-interface IFNAME
```

Dieser Befehl versetzt das angegebene Interface in den Passivmodus. An einem passiven Interface werden alle ankommenden Pakete ganz normal verarbeitet und RIP sendet weder Multicast noch Unicast Pakete außer an die RIP Nachbarn, die mit dem `neighbor` Befehl spezifiziert worden sind.

Handling der RIP Versionen:

```
ripd(config-router)# version version
```

Setzt die verwendete RIP Version. `version` kann „1“ oder „2“ sein.

```
ripd(config-if)# ip rip send version version
```

`version` kann sein „1“, „2“, „1 2“. Dieser Konfigurationsbefehl überschreibt die gesetzte RIP Version des Routers. Durch diesen Befehl wird das angegebene Interface Pakete in RIP Version 1, Version 2 oder in beiden senden. In Fall „1 2“ werden Pakete sowohl per Multicast als auch per Broadcast gesendet.

```
ripd(config-if)# ip rip receive version version
```

Setzt die Version für ankommende RIP Pakete. Durch diesen Befehl wird das angegebene Interface Pakete in RIP Version 1, RIP Version 2 oder in beiden empfangen.

RIP split horizon:

```
ripd(config-if)# ip rip split-horizon
```

```
ripd(config-if)# no ip rip split-horizon
```

Damit wir split horizon auf dem Interface kontrolliert. Der Standard ist `ip split horizon`. Wenn man split horizon auf dem Interface nicht verwendet, benutzt man `no ip split-horizon`

RIP-Timer:

```
timers basic update timeout garbage
```

Das RIP Protokoll hat verschiedene Timer. Der Benutzer kann die Timer mit dem `timers basic` Befehl konfigurieren.

Die Standardwerte für die Timer sind:

1. Der Update Timer liegt bei 30 Sekunden. Bei jedem Update wird der RIP Prozess geweckt um eine Antwortnachricht zu allen benachbarten Routern zu senden, die die komplette Routing Tabelle enthält.
2. Der Timeout Timer liegt bei 180 Sekunden. Wenn der Timeout Timer abläuft, ist die nicht mehr länger aktiv. Trotzdem bleibt sie noch eine kurze Zeit in der Routing Tabelle, so dass die Nachbarn benachrichtigt werden dass die Route nicht mehr verfügbar ist.
3. Der Garbage Collect Timer liegt bei 120 Sekunden. Sobald der Timer abläuft, wird die Router endgültig aus der Routing Tabelle gelöscht.

no timers basic Der `no timers basic` Befehl setzt die Werte der Timer wieder auf die oben angegebenen Standardwerte.

2.4 RIP Informationen anzeigen

(Vorraussetzung hierfür ist eine Telnet-Verbindung zum RIP-Daemon)

show ip rip

Dieser Befehl zeigt alle RIP Routen an. Bei Routen die über RIP empfangen worden sind, wird die Zeit zu der das Paket gesendet wurde und die tag Informationen angezeigt. Außerdem wird diese Information für Routen die in RIP weiterverteilt worden sind angezeigt.

show ip rip status

Zeigt den aktuellen RIP Status an. Dieser enthält RIP Timer, Filter, Version, RIP-aktivierte Interfaces und RIP peer Informationen:

```
ripd> show ip rip status
Routing Protocol is "rip"
  Sending updates every 30 seconds with +/-50%, next due in 35
  seconds
    Timeout after 180 seconds, garbage collect after 120 seconds
    Outgoing update filter list for all interface is not set
    Incoming update filter list for all interface is not set
    Default redistribution metric is 1
    Redistributing: kernel connected
    Default version control: send version 2, receive version 2
      Interface          Send  Recv
Routing for Networks:
  eth0
  eth1
  1.1.1.1
  203.181.89.241
Routing Information Sources:
  Gateway              BadPackets BadRoutes  Distance Last Update
```

2.5 RIP Debug Befehle

```
debug rip events
```

debug rip zeigt die RIP events an. Gesendete und empfangene Pakete, Timer, und Änderungen in den Interfaces sind Events die mit ripd angezeigt werden.

```
debug rip packet
```

debug rip packet zeigt detaillierte Informationen über die RIP Pakete.

```
debug rip zebra
```

Dieser Befehl zeigt die Kommunikation zwischen ripd und zebra an.

```
show debugging rip
```

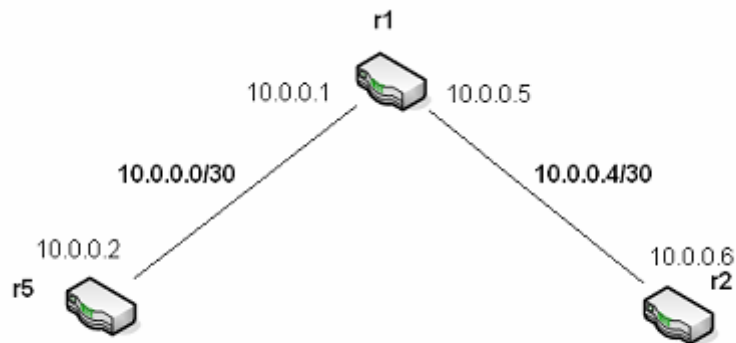
Zeigt ripd's debugging Optionen.

show debugging rip zeigt alle aktuell gesetzten Informationen von ripd debug an.

Kapitel 3

Beispiel-Szenarios

1. Ein einfaches RIP-Netzwerk mit manueller RIP-Konfiguration



Dieses Netzwerk besteht aus 3 Routern (r1,r2,r5). Hierbei kennt zu Beginn der Router r1 als einziger die beiden anderen. r5 und r2 müssen sich erst über das RIP-Protokoll gegenseitig kennen lernen bzw. eine Route dorthin finden und diese in die Routing-Tabelle eintragen.

Stufe 1: Erstellen und Booten des XML-Szenarios

Zunächst erstellt man die benötigte XML-Datei:

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

<global>
<version>1.5</version>
<simulation_name>ripbasic</simulation_name>
<ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
<automac/>
<host_mapping/>

</global>
<net name="Net0"/>
<net name="Net1"/>
<vm name="r1">

<filesystem
type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
</filesystem>
<if id="1" net="Net0">
<ipv4 mask="255.255.255.252">10.0.0.1</ipv4>
</if>
<if id="2" net="Net1">
<ipv4 mask="255.255.255.252">10.0.0.5</ipv4>
</if>
<forwarding/>
```

```

</vm>

<vm name="r2">
<filesystem
type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
</filesystem>
<if id="1" net="Net1">
<ipv4 mask="255.255.255.252">10.0.0.6</ipv4>
</if>
<forwarding/>
</vm>

<vm name="r5">
<filesystem
type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
</filesystem>
<if id="2" net="Net0">
<ipv4 mask="255.255.255.252">10.0.0.2</ipv4>
</if>
<forwarding/>
</vm>

</vnuml>

```

Man definiert die zwei benötigten Netzwerke mit dem <net> tag. Den virtuellen Maschinen werden nach folgendem Schema IP-Adressen vergeben:

Network	Link	IP Address Assignment (Lower-numbered router first)	Broadcast Address
10.0.0.0/30	r1 – r5	10.0.0.1, 10.0.0.2	10.0.0.3
10.0.0.4/30	r1 – r2	10.0.0.5, 10.0.0.6	10.0.0.7

Die Netzwerkmaske für alle Netzwerke wird auf 255.255.255.252 gesetzt. Das <forwarding/> tag wird in jeder virtuellen Maschine verwendet, um sie wie einen Router funktionieren zu lassen. In der Annahme, dass das obige Szenario in einer Datei mit Namen „ripbasic.xml“ gespeichert worden ist, wird dieses Szenario gestartet mit:

```
host# vnumlparser.pl -t ripbasic.xml -v -o /tmp/umlboot
```

Durch das -v werden beim Startvorgang diverse Meldungen ausgegeben, die beim Debuggen helfen können und die -o Option sorgt dafür, dass die VM boot Nachrichten in verschiedene Dateien mit Namen umlboot.rN (N->Routernummer) im Verzeichnis /tmp. Diese Option kann auch weggelassen werden. Das Booten der virtuellen Rechner dauert 1-2 Minuten, je nach Rechenleistung des verwendeten Computers und läuft im Hintergrund ab. Nachdem das vnumlparser Skript ausgeführt worden ist, muss man noch diese 1-2 Minuten warten, bevor man die UML's zum ersten Mal anpingen kann. Wenn eine UML nicht erreichbar ist, kann man einen Blick in die Bootmessages in /tmp/umlboot.rN werfen um dem Problem näher zu kommen. Die Simulation kann gestoppt werden mit:

```
vnumlparser.pl -d ripbasic.xml -v
```

Wenn alles gut gegangen ist, sollte man jetzt in der Lage sein, sich per ssh in die einzelnen UML's einzuloggen („ssh rN“ bzw „ssh IP-Adresse“). Im Normalfall werden hier die

virtuellen Maschinen über ihre Namen r1, r2 und r5 und über die IP-Adressen 192.168.0.2, 192.168.0.6 und 192.168.0.10 erreichbar sein.

VNUML konfiguriert automatisch die Interface-Adressen nach der im Szenario angegebenen Spezifikation. Man kann nun die IP Verbindungen der direkt verbundenen Router mit dem ping-Befehl überprüfen, und die Routing-Tabellen mit dem Befehl `netstat -rn` einsehen.

Stufe 2: RIP konfigurieren

Dazu muss man sich in jeden Router einloggen und dort die Konfigurationsdateien des RIP-Deamons editieren. Die Konfigurationsdatei, die wir hier verwenden wollen, befindet sich im Verzeichnis `/etc/zebra/ripd.conf`. Sie lässt sich unter Linux beispielsweise mit dem kommandozeilenbasierten Editor „vi“ bearbeiten. (Ein Tutorial für vi gibt es z.B. hier: <http://www.eng.hawaii.edu/Tutor/vi.html>). Die Konfigurationdateien sehen für alle 3 Router gleich aus:

```
r1:~# head -15 /etc/zebra/ripd.conf
! *- rip *-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
! password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 10.0.0.0/8
```

Während man den überwiegenden Teil der Datei beibehalten kann, da er aus Kommentaren besteht, sind die letzten beiden Zeilen besonders wichtig. Mit `router rip` wird RIP für diesen Router aktiviert und mit `network 10.0.0.0/8` wird bestimmt, dass der Router die RIP Nachrichten an alle Adressen in dem angegebenen Bereich schickt. Bitte beachten Sie, dass diese nicht hinter „!“ stehen dürfen.

Nachdem man auf allen 3 Routern die Konfigurationsdateien entsprechend bearbeitet hat, startet man auf jedem Router die Zebra und RIP Daemons in dieser Reihenfolge mit:

```
# zebra -f /etc/zebra/zebra.conf -u root -d
# ripd -f /etc/zebra/ripd.conf -u root -d
```

Das Log-Datei Problem:

Falls beim Starten eines Daemons eine Meldung erscheint, dass eine log-Datei nicht geschrieben werden konnte, liegt das daran, dass in den Standard-Konfigurationsdateien in der letzten Zeile ein Eintrag zum Anlegen einer log-Datei vorhanden ist. Der Pfad in der diese log-Datei angelegt werden soll, ist aber standardmäßig nicht vorhanden. Es gibt mehrere Möglichkeiten dieses Problem zu beheben:

- man löscht entweder die Zeile aus der conf-Datei oder
- man legt das gewünschte Verzeichnis vorher mit dem `mkdir`-Befehl an oder
- man ändert den Pfad entsprechend ab und wählt ein existierendes Verzeichnis

Stufe 3: RIP stabilisieren lassen und überprüfen

Man muss RIP ein wenig Zeit lassen zum stabilisieren lassen. Nach ein paar Sekunden kann man nachsehen, ob die Routing Tabellen (von Router r5 und r2) aktualisiert worden sind. Unten angegeben sind die Routing-Tabellen der Router r5 und r2, mit den RIP-aktualisierten Einträgen in Fettdruck.

```
r2:~# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask          Flags            Iface
10.0.0.4         0.0.0.0         255.255.255.252 U                eth1
10.0.0.0         10.0.0.5         255.255.255.252 UG               eth1
192.168.0.4     0.0.0.0         255.255.255.252 U                eth0
```

```
r5:~# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask          Flags            Iface
10.0.0.4         10.0.0.1         255.255.255.252 UG               eth2
10.0.0.0         0.0.0.0         255.255.255.252 U                eth2
192.168.0.8     0.0.0.0         255.255.255.252 U                eth0
```

Zu beachten ist, dass jede Routing Tabelle einen Eintrag für jedes der beiden Netzwerke in dem virtuellen Netzwerk hat. Nun sollte jeder Router in der Lage sein, jeden anderen Router im Netzwerk zu erreichen. Zum Beispiel kann man versuchen, r5 von r2 aus anzupingen:

```
r2:~# ping 10.0.0.2 # router r5
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=63 time=42.8 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=63 time=2.69 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=63 time=2.02 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=63 time=3.59 ms
```

Außerdem kann man einen traceroute von r5 nach r2 ausführen:

```
r5:~# traceroute 10.0.0.6 # router r2
traceroute to 10.0.0.6 (10.0.0.6), 30 hops max, 38 byte packets
 1 10.0.0.1 (10.0.0.1) 38.196 ms 3.592 ms 3.956 ms
 2 10.0.0.6 (10.0.0.6) 11.640 ms 3.924 ms 3.908 ms
```

(Hinweis zur Fehlersuche:

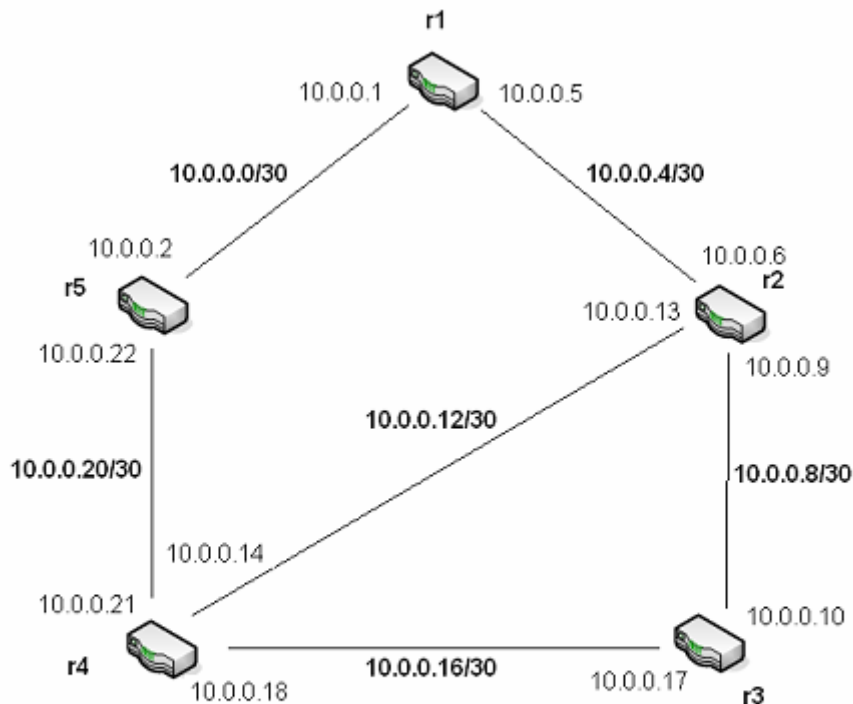
Falls nachdem alle Daemons gestartet worden sind, ein Router nicht erreichbar sein sollte, d.h. er weder von anderen Routern angepingt werden kann, noch irgendeine RIP-Route über ihn verläuft und man sicher ist, dass sowohl die XML-Datei und die RIP-Konfigurationsdateien korrekt sind, empfehle ich das Szenario noch einmal komplett herunterzufahren und es dann wieder hochzufahren und dann die Daemons noch einmal zu starten.)

Aufgabe 1:

Erstellen Sie obiges Szenario Schritt für Schritt nach der oben angegebenen Anleitung.

Aufgabe 2:

Erweitern Sie obiges Netzwerk, so dass es folgendes Aussehen hat:



- Erstellen Sie dazu ein neues XML-Szenario und speichern Sie es unter dem Namen „ripextended.xml“
- Booten Sie das Szenario und konfigurieren Sie die Konfigurationsdateien des RIP Daemons in gleicher Weise im Beispielnetz beschrieben.
- Starten Sie anschließend die Daemons und überprüfen Sie zuletzt mit `netstat -rn`, `ping` und `traceroute` die korrekte Funktionsweise des Netzwerks.

Den virtuellen Maschinen werden nach folgendem Schema IP-Adressen vergeben:

Network	Link	IP Address Assignment (Lower-numbered router first)	Broadcast Address
10.0.0.0/30	r1 – r5	10.0.0.1, 10.0.0.2	10.0.0.3
10.0.0.4/30	r1 – r2	10.0.0.5, 10.0.0.6	10.0.0.7
10.0.0.8/30	r2 – r3	10.0.0.9, 10.0.0.10	10.0.0.11
10.0.0.12/30	r2 – r4	10.0.0.13, 10.0.0.14	10.0.0.15
10.0.0.16/30	r3 – r4	10.0.0.17, 10.0.0.18	10.0.0.19
10.0.0.20/30	r4 – r5	10.0.0.21, 10.0.0.22	10.0.0.23

Lösung zu Aufgabe 2:

1. Das XML-Szenario besteht nun aus 6 Netzen und 5 Routern:

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

  <global>
    <version>1.5</version>
    <simulation_name>ripextended</simulation_name>
    <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
    <automac/>
    <host_mapping/>
  </global>
  <net name="Net0"/>
  <net name="Net1"/>
  <net name="Net2"/>
  <net name="Net3"/>
  <net name="Net4"/>
  <net name="Net5"/>
  <vm name="r1">

    <filesystem
      type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
    <if id="1" net="Net0">

      <ipv4 mask="255.255.255.252">10.0.0.1</ipv4>
    </if>
    <if id="2" net="Net1">

      <ipv4 mask="255.255.255.252">10.0.0.5</ipv4>
    </if>
    <forwarding/>
  </vm>
  <vm name="r2">
    <filesystem
      type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
    <if id="1" net="Net1">

      <ipv4 mask="255.255.255.252">10.0.0.6</ipv4>
    </if>
    <if id="2" net="Net2">

      <ipv4 mask="255.255.255.252">10.0.0.9</ipv4>
    </if>
    <if id="3" net="Net3">

      <ipv4 mask="255.255.255.252">10.0.0.13</ipv4>
    </if>
    <forwarding/>
  </vm>
  <vm name="r3">
    <filesystem
      type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
```

```

<if id="1" net="Net2">
  <ipv4 mask="255.255.255.252">10.0.0.10</ipv4>
</if>
<if id="2" net="Net4">
  <ipv4 mask="255.255.255.252">10.0.0.17</ipv4>
</if>
<forwarding/>
</vm>

<vm name="r4">
  <filesystem
  type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
  </filesystem>
  <if id="1" net="Net4">
    <ipv4 mask="255.255.255.252">10.0.0.18</ipv4>
  </if>
  <if id="2" net="Net5">
    <ipv4 mask="255.255.255.252">10.0.0.21</ipv4>
  </if>
  <if id="3" net="Net3">
    <ipv4 mask="255.255.255.252">10.0.0.14</ipv4>
  </if>
  <forwarding/>
</vm>

<vm name="r5">
  <filesystem
  type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
  </filesystem>
  <if id="1" net="Net5">
    <ipv4 mask="255.255.255.252">10.0.0.22</ipv4>
  </if>
  <if id="2" net="Net0">
    <ipv4 mask="255.255.255.252">10.0.0.2</ipv4>
  </if>
  <forwarding/>
</vm>
</vnuml>

```

2. Die Konfigurationsdateien der Daemons können aus der Beispielaufgabe übernommen werden.

3. Unten angegeben sind die Routing-Tabellen der Router r1 bzw r4, so wie sie nach dem Start der Daemons aussehen sollten (RIP-Einträge in Fettdruck):

```

r1:~# netstat -rn
Kernel IP routing table
Destinat      Gateway      Genmask      Flags      ... Iface
10.0.0.4      0.0.0.0      255.255.255.252  U      ... eth2
10.0.0.0      0.0.0.0      255.255.255.252  U      ... eth1
10.0.0.12     10.0.0.6     255.255.255.252  UG     ... eth2
10.0.0.8      10.0.0.6     255.255.255.252  UG     ... eth2
10.0.0.20     10.0.0.2     255.255.255.252  UG     ... eth1
10.0.0.16     10.0.0.6     255.255.255.252  UG     ... eth2
192.168.0.0  0.0.0.0      255.255.255.252  U      ... eth0

```

```

r4:~# netstat -rn
Kernel IP routing table
Destinat      Gateway      Genmask      Flags      ... Iface
10.0.0.4      10.0.0.13    255.255.255.252  UG     ... eth3
10.0.0.0      10.0.0.22    255.255.255.252  UG     ... eth2
10.0.0.12     0.0.0.0      255.255.255.252  U      ... eth3
10.0.0.8      10.0.0.17    255.255.255.252  UG     ... eth1
10.0.0.20     0.0.0.0      255.255.255.252  U      ... eth2
10.0.0.16     0.0.0.0      255.255.255.252  U      ... eth1
192.168.0.12 0.0.0.0      255.255.255.252  U      ... eth0

```

Ein ping von r1 nach r3 liefert:

```

r1:~# ping 10.0.0.10 # router r3
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=63 time=7.24 ms

```

Ein traceroute von r4 nach r1 liefert:

```

r4:~# traceroute 10.0.0.1 # router r1
traceroute to 10.0.0.1 (10.0.0.1), 64 hops max, 40 byte packets
1 10.0.0.22 (10.0.0.22) 3 ms 2 ms 2 ms
2 10.0.0.1 (10.0.0.1) 54 ms 2 ms 3 ms

```

Mittels tcpdump kann man den Austausch der RIP-Nachrichten beobachten. Zum Beispiel kann man in r5 das Interface 2 beobachten mit:

```

r5: ~# tcpdump -i eth2

```

2. An und Abschalten einzelner Interfaces

Um beispielsweise die Interfaces an der r1 – r5 Verbindung abzuschalten benutzt man die Befehle:

```
r1:~# ifconfig eth1 down
r5:~# ifconfig eth2 down
```

RIP wird die fehlende Verbindung erkennen und die betroffenen Routen neu justieren. Zum Beispiel ist dies die Routing-Tabelle des Routers r1 nach Abschaltung der obigen Route:

```
r1:~# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags ... Iface
10.0.0.4 0.0.0.0 255.255.255.252 U ... eth2
10.0.0.12 10.0.0.6 255.255.255.252 UG ... eth2
10.0.0.8 10.0.0.6 255.255.255.252 UG ... eth2
10.0.0.20 10.0.0.6 255.255.255.252 UG ... eth2
10.0.0.16 10.0.0.6 255.255.255.252 UG ... eth2
192.168.0.0 0.0.0.0 255.255.255.252 U ... eth0
```

Man erkennt, dass die direkte Route zu 10.0.0.0, mit dem Interface eth1 abgeschaltet worden ist. Ein traceroute von r1 nach r5 benötigt jetzt 3 Hops:

```
r1:~# traceroute 10.0.0.22

traceroute to 10.0.0.22 (10.0.0.22), 30 hops max, 38 byte packets
 1 10.0.0.6 (10.0.0.6) 12.951 ms 3.427 ms 4.400 ms
 2 10.0.0.14 (10.0.0.14) 11.680 ms 7.751 ms 3.342 ms
 3 10.0.0.22 (10.0.0.22) 12.103 ms 8.213 ms 7.559 ms
```

Um die Interfaces wieder anzuschalten benutzt man:

```
r1:~# ifconfig eth1 up
r5:~# ifconfig eth2 up
```

Aufgabe 3:

Fahren Sie fort und deaktivieren Sie noch einige weitere Interfaces. RIP wird, soweit möglich, dynamisch die Routen anpassen.

3. Alternative Konfiguration der Daemons über Telnet

In dem einführenden Beispiel hab ich beschrieben, wie man die Konfigurationsdateien der Daemons mit einem Editor bearbeiten kann. Darüber hinaus kann man die Daemons über eine Telnet-Verbindung konfigurieren.

Voraussetzung dafür ist, dass man in den Konfigurationsdateien ein VTY-Passwort festlegt. Unsere Konfigurationsdatei für den RIP Daemon aus dem Anfangsbeispiel würde dann so aussehen:

```
r1:~# head -15 /etc/zebra/ripd.conf
! *- rip *-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 10.0.0.0/8
```

Man sieht der einzige Unterschied besteht darin, dass wir nun ein VTY-Passwort festgelegt haben, welches "zebra" lautet.

Hat man die Konfigurationsdateien entsprechend editiert und die Daemons in derselben Weise wie zuvor gestartet (ebenfalls zwingende Voraussetzung), kann man sich nun folgendermaßen via Telnet in die Daemons einloggen:

- Man loggt sich per ssh in den entsprechenden Router ein. Wir nehmen als Beispiel r5.

```
ssh r5
```

- Man stellt eine Telnet-Verbindung zu dem gewünschten Daemon her. In unserem Fall ist dies der RIP-Daemon. Dieser hat den Port 2602. (Zebra hätte 2601, OSPF 2604):

```
r5:# telnet r5 2602
```

Es erscheint folgende Ausgabe:

```
Trying 10.0.0.2...
Connected to r5.
Escape character is '^]`.

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

User Access Verification

Password:
```

- Hier gibt man nun das zuvor gesetzte Passwort ein. In unserem Fall: zebra

Nun hat man sich in den Daemon eingeloggt und es erscheint:

```
ripd>
```

- Um den Daemon nun zu konfigurieren, gibt man folgendes ein:

```
enable (hat man in der Konfigurationsdatei ein enable-Passwort gesetzt muss man dies an dieser Stelle eingeben.)
```

```
configure terminal
```

- Es erscheint:

```
ripd(config)#
```

- An dieser Stelle kann man nun die gleichen Konfigurationsbefehle verwenden, wie sie in den einzelnen Konfigurationsdateien vorkommen. Für unser Beispiel würde man eingeben:

```
router rip
```

- Es erscheint:

```
ripd(config-router)#
```

- Dann folgt die Eingabe der networks:

```
z.B.: network 10.0.0.0/8
```

- Um zusätzlich noch einzelne Interfaces zu konfigurieren gibt man zuerst das Interface an:

```
z.B. interface Ethernet0 für Interface 0.
```

- Es erscheint:

```
ripd(config-if)#
```

- Um die Verbindung zu beenden benutzt man (ggf. mehrmals) das exit-Kommando

(Eine komplette Liste aller möglichen Befehle befindet sich im Anhang)

4. Booten eines Szenarios mit automatischer RIP-Konfiguration

Bisher haben wir nach dem Start eines Szenario immer zuerst in jeden Router einloggen, die Daemons konfigurieren und dann Starten müssen, um für ein Netzwerk RIP zu aktivieren. Dies ist je nach Größe eines Netzwerks ein sehr mühsames und zeitraubendes Unterfangen. VNUML bietet jedoch die Möglichkeit diesen Vorgang zu erleichtern, indem man die Konfigurationsdateien der Daemons extern anlegen und die benötigten Startbefehle der Daemons direkt in die XML-Datei hineinschreiben kann.

Dazu legt man am besten im selben Verzeichnis in dem sich auch die .xml-Datei befindet für jeden Router ein separates Verzeichnis an. Für unser Basisbeispiel würde man also 3 Ordner mit dem Namen „r1“, „r2“ und „r5“ anlegen. In diesen Ordner legt man nun die Konfigurationsdateien des RIP-Deamons (ripd.conf) für jeden der drei Router an und konfiguriert sie genauso wie bisher.

Damit der vnumlparser nun beim die Daemons mit diesen Konfigurationsdateien direkt mitstartet, fügt man nun bei jedem Router eine Start-Sequenz in die xml-Datei ein.

Ausgehend von unserem Basisbeispiel ändern wie die XML-Datei wie folgt ab:

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

  <global>
    <version>1.5</version>
    <simulation_name>ripauto</simulation_name>
    <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
    <automac/>
    <host_mapping/>

  </global>
  <net name="Net0"/>
  <net name="Net1"/>
  <vm name="r1">
```

```

<filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
</filesystem>
<if id="1" net="Net0">

  <ipv4 mask="255.255.255.252">10.0.0.1</ipv4>
</if>
<if id="2" net="Net1">
<ipv4 mask="255.255.255.252">10.0.0.5</ipv4>
</if>
<forwarding/>
<filetree when="start"
root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripauto/r1</filetree>
<exec seq="start" type="verbatim">hostname</exec>
<exec seq="start" type="verbatim">/usr/local/sbin/zebra -f /etc/zebra/zebra.conf
-u root -d</exec>
<exec seq="start" type="verbatim">ripd -f /usr/local/etc/ripd.conf -u root -
d</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="r2">
<filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
</filesystem>
<if id="1" net="Net1">

  <ipv4 mask="255.255.255.252">10.0.0.6</ipv4>
</if>
<forwarding/>
<filetree when="start"
root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripauto/r2</filetree>
<exec seq="start" type="verbatim">hostname</exec>
<exec seq="start" type="verbatim">/usr/local/sbin/zebra -f /etc/zebra/zebra.conf
-u root -d</exec>
<exec seq="start" type="verbatim">ripd -f /usr/local/etc/ripd.conf -u root -
d</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="r5">
<filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
</filesystem>
<if id="2" net="Net0">

  <ipv4 mask="255.255.255.252">10.0.0.2</ipv4>
</if>
<forwarding/>
<filetree when="start"
root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripauto/r5</filetree>
<exec seq="start" type="verbatim">hostname</exec>
<exec seq="start" type="verbatim">/usr/local/sbin/zebra -f /etc/zebra/zebra.conf
-u root -d</exec>
<exec seq="start" type="verbatim">ripd -f /usr/local/etc/ripd.conf -u root -
d</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>

```

Man erkennt, dass bei jedem Router eine Start- und eine Stop-Sequenz hinzugefügt worden ist. Unsere selbstgeschriebenen externen ripd.conf liegen in diesem Fall in den Verzeichnissen: /usr/local/share/vnuml/examples/rip/ripauto/rN (N: 1,2,5).

Dieser Pfad muss entsprechend abgeändert werden, je nachdem wo die Dateien liegen.

Damit die virtuellen Maschinen diese Dateien verwenden können, müssen sie mit Hilfe des filetree-Befehls "emuliert" werden. Das bedeutet in diesem konkreten Fall, dass die angegebene ripd.conf für die virtuelle Maschine im Verzeichnis /usr/local/etc/ liegen wird. Es folgen die Start- und Stop-Sequenz Befehle, die in diesem Fall überwiegend aus dem Starten und Stoppen der Daemons bestehen.

Zu beachten ist, dass wir den Zebra-Daemon mit der ursprünglichen Konfigurationsdatei starten und den RIP-Daemon mit der emulierten Datei, was die unterschiedlichen Verzeichnisse erklärt.

Analog kann man so natürlich auch die Zebra-Konfigurationsdateien extern anlegen.

Man speichert das Szenario nun unter dem Namen „ripauto.xml“.

Anschließend bootet man die Maschinen wieder mit dem bekannten Befehl:

```
vnumlparser.pl -t ripauto.xml -v
```

Sind alle Router hochgefahren, startet man nun die Start-Sequenz mit folgendem Befehl:

```
vnumlparser.pl -x start@ripauto.xml -v
```

Damit werden die Daemons anhand der festgelegten Konfigurationsdateien nun automatisch gestartet und das Netzwerk befindet sich im Stufe3-Status des Beispielszenarios (3.1)

Am Ende führt man die Stop-Sequenz aus mit dem Befehl:

```
vnumlparser.pl -x stop@ripauto.xml -v
```

und fährt das Szenario wieder wie üblich herunter mit:

```
vnumlparser.pl -d ripauto.xml -v
```

5. Filtern von Paketen

Es gibt mehrere Möglichkeiten Pakete innerhalb des Netzwerks zu filtern.

5.1 Filtern mit distribute-list

RIP Routen können mit einer sogenannten distribute-list gefiltert werden:

Befehl: **distribute-list** access_list direct ifname

Mit diesem Befehl kann man access-lists für ein Interface festlegen. access_list ist der Name der access list. direct legt die Richtung fest. Sie kann sein „in“ oder „out“. Wenn direct auf in gesetzt wird bezieht sich die zugehörige access list auf einlaufende Pakete.

Der distribute-list Befehl kann benutzt werden um den RIP path zu filtern.

distribute-list kann access-lists für ein gewähltes Interface festlegen. Als erstes sollte man die access-list spezifizieren. Als erstes wird der Name der access-list und das betreffende Interface in dem distribute-list Befehl angegeben. Zum Beispiel wird in der Folgenden Konfiguration das Interface eth0 nur die Pfade akzeptieren, die mit der Route 10.0.0.0/8 übereinstimmen.

```
!
router rip
 distribute-list private in eth0
!
access-list private permit 10 10.0.0.0/8
access-list private deny any
```

Die distribute-list kann sowohl für ein als auch für ausgehende Dateien verwendet werden:

Befehl: `distribute-list prefix prefix_list (in|out) ifname`

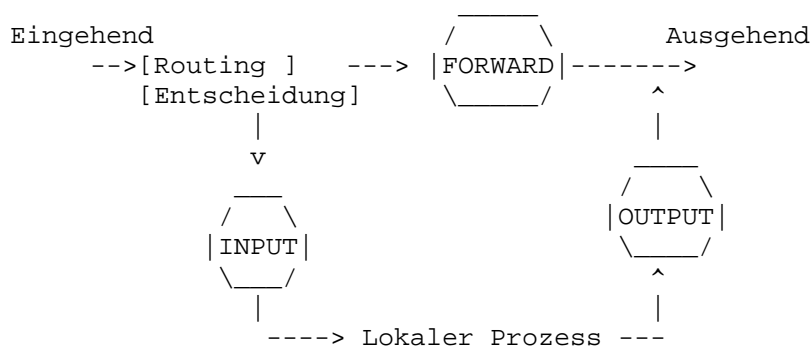
Man kann auch prefix lists für ein Interface with einem distribute-list Befehl festlegen. `prefix_list` ist der Name der prefix list. Als nächstes folgt die Richtung (in oder out). Wenn `direct` auf in gesetzt wird, bezieht sich die zugehörige access-list auf einlaufende Pakete.

5.2 Filtern mit iptables

iptables sind eine weitere Möglichkeit IP-Pakete zu filtern. Es sollte standardmäßig in dem verwendeten Kernel integriert sein.

Der Kernel beginnt mit drei Listen von Regeln in der Filtertabelle; diese Listen werden **Firewall-Ketten** oder nur **Ketten** (von englisch "chain" = Kette) genannt. Diese drei Ketten heissen **INPUT**, **OUTPUT** und **FORWARD**.

Diese Ketten sind so arrangiert:



Die drei Kreise repräsentieren die drei oben erwähnten Ketten. Wenn ein Paket einen Kreis im Diagramm erreicht, wird diese Kette untersucht, um über das Schicksal des Pakets zu entscheiden. Wenn die Kette besagt, dass das Paket zu DROPPEN ist, wird das Paket hier getötet. Wenn die Kette jedoch sagt, dass das Paket zu akzeptieren (ACCEPT) ist, kann es weiter durch das Diagramm reisen.

Eine Kette ist eine Checkliste von **Regeln**. Jede Regel sagt 'Wenn der Paket-header so-und-so aussieht, ist das-und-das mit dem Paket zu tun'. Wenn die Regel nicht auf das Paket zutrifft, wird die nächste Regel befragt. Wenn es endlich keine Regeln zum Befragen mehr gibt, sieht sich der Kernel die **Policy** der Kette an und entscheidet, was zu tun ist. In einem sicherheitsbewussten System sagt diese Policy dem Kernel normalerweise, dass er das Paket DROPPEN soll.

1. Wenn ein Paket eingeht (z.B. durch die Netzwerkkarte), sieht sich der Kernel zunächst die Zieladresse des Pakets an: Das wird 'Routing' genannt.
2. Wenn das Paket für diesen Rechner bestimmt ist, wandert es im Diagramm an die INPUT-Kette. Wenn es diese passiert, wird es der auf dieses Paket wartende Prozess erhalten.
3. Andernfalls, wenn der Kernel Forwarding nicht aktiviert hat, oder er nicht weiss, wie er das Paket weiterleiten soll, wird das Paket verworfen. Wenn Forwarding aktiviert

ist und das Paket für eine andere Netzwerkschnittstelle (wenn Du eine hast) bestimmt ist, geht das Paket in unserm Diagramm direkt zur FORWARD-Kette. Wenn es dort akzeptiert (ACCEPT) wird, wird es weitergeleitet.

4. Schliesslich kann ein Programm, das auf dem Rechner läuft, auch Netzwerkpakete verschicken. Diese Pakete gehen direkt zur OUTPUT-Kette. Wenn diese das Paket akzeptiert, wandert es weiter zu der Schnittstelle, für die es bestimmt ist.

iptables hat eine recht detaillierte Man-page (`man iptables`), wenn man mehr Informationen über Besonderheiten braucht.

Es gibt verschiedene Dinge, die man mit `iptables` machen kann. Man fängt an mit den drei eingebauten Ketten `INPUT`, `OUTPUT` und `FORWARD`, die man nicht löschen kann. Lass uns einen Blick auf die Operationen werfen, mit denen man auf ganzen Ketten arbeiten kann:

1. Eine neue Kette erstellen (-N).
2. Eine leere Kette loeschen (-X).
3. Die Policy fuer eine eingebaute Kette aendern (-P).
4. Die Regeln einer Kette auflisten (-L).
5. Die Regeln aus einer Kette ausspielen (flush) (-F).
6. Paket- und Bytezaehler aller Regeln einer Kette auf Null stellen (-Z).

Es gibt verschiedene Wege, die Regeln in einer Kette zu manipulieren:

1. Eine neue Regel an eine Kette anhaengen (-A).
2. Eine neue Regel an eine bestimmte Position in der Kette einfuegen (-I).
3. Eine Regel an bestimmter Position in der Kette ersetzen (-R).
4. Eine Regel an einer bestimmten Position in der Kette loeschen (-D).
5. Die erste passende Regel in einer Kette loeschen (-D).

Bevor irgendwelche iptables-Befehle ausgefuehrt werden, wird es keine Regeln in einer der eingebauten Ketten (`INPUT`, `OUTPUT`, `FORWARD`) geben, und die Policy aller Ketten wird auf 'ACCEPT' stehen. Man kann die Standard-Policy der FORWARD-Kette ändern, indem man die Option `forward=0` im `iptables_filter` module mitgibt.

Dies ist das A-und-O des Paketfilters: Regeln manipulieren. Meistens wirst man vermutlich den Befehl zum Anhängen (-A) oder Löschen (-D) einer Regel verwenden. Die anderen (-I zum Einfügen und -R zum Ersetzen) sind einfache Erweiterungen dieser Konzepte.

Jede Regel bestimmt eine Reihe von Bedingungen, die ein eintreffendes Paket durchlaufen muss und was weiterhin mit dem Paket geschehen soll ('target'). Zum Beispiel möchte man vielleicht alle ICMP-Pakete, die von der IP-Adresse 127.0.0.1 kommen, verwerfen. Unsere Bedingungen sind in diesem Fall also, dass das Protokoll ICMP und die Quelladresse 127.0.0.1 sein müssen. Das Ziel ist Verwerfen (DROP).

127.0.0.1 ist die Loopback-Schnittstelle, welche man auch dann hat, wenn man keine wirkliche Netzwerkverbindung hast. Man kann das 'ping' Programm verwenden, um solche Pakete zu generieren (es sendet einfach ein ICMP Typ 8 (echo request), auf das alle kooperierenden Hosts verbindlich mit einem ICMP Typ 0 Paket (echo reply) antworten sollten). Das macht es nützlich fuer einen Test.

```
# ping -c 1 127.0.0.1
```

```

PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
# iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
#

```

Man kann hier sehen, dass der erste Ping ankommt (das `-c 1` sagt dem Programm, dass es nur ein einziges Paket schicken soll).

Dann erweitern wir die INPUT-Kette mit einer Regel ('-A'), die besagt, dass Pakete, die von 127.0.0.1 ('-s 127.0.0.1') kommen und das Protokoll ICMP ('-p icmp') verwenden, zu verwerfen sind ('-j DROP').

Dann testen wir unsere Regel mit einem zweiten Ping. Es wird eine Pause geben, bevor das Programm aufgibt, auf ein Paket zu warten, das niemals ankommen wird.

Wir können die Regel mit einer von zwei Möglichkeiten löschen. Erstens, da wir wissen, dass es die einzige Regel in der INPUT-Kette ist, können wir sie nach der Nummerierung löschen, so wie:

```

# iptables -D INPUT 1
#

```

Dies löscht Regel Nummer 1 in der INPUT-Kette.

Der zweite Weg ist wie das `-A`-Kommando, man muss nur das `-A` durch ein `-D` ersetzen. Dies ist nützlich, wenn Du eine komplexe Kette von Regeln hast und nicht alle erst durchzählen möchtest, um herauszufinden, dass es Regel 37 ist, die Du loswerden willst. In diesem Fall würden wir folgendes verwenden:

```

# iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
#

```

Die Syntax von `-A` muss genau dieselben Optionen haben wie die Syntax des `-A` (oder `-D`) Befehls. Wenn es mehrere identische Regeln in derselben Kette gibt, wird nur die erste gelöscht.

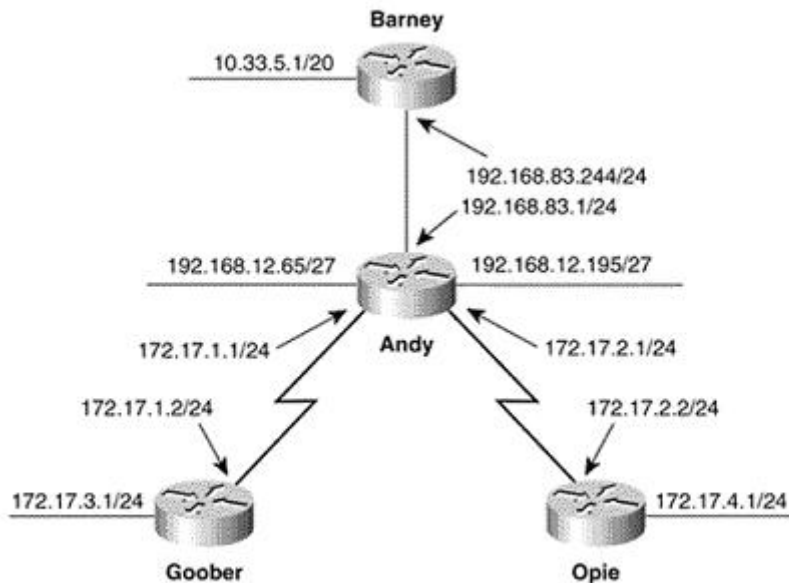
(Dies soll nur ein Beispiel für die Möglichkeiten sein, die iptables bietet. Für umfangreichere Informationen verweise ich an dieser Stelle auf die Man-Page von iptables (<http://www.linuxguruz.com/iptables/howto/maniptables.html>) und auf www.netfilter.org.)

6. Erweiterungen der RIP-Konfiguration

6.1 Erweitern der ripd.conf

Bisher wurde in den Beispiel immer seine sehr einfache Konfiguration des ripd benutzt, die auch bei allen Router gleich war. Es wurde jeweils immer nur RIP für das network 10.0.0.0/8 aktiviert, so dass die Router ihre RIP-Nachrichten an alle Netzwerke, die die Form 10.*.*.* haben gesendet worden sind.

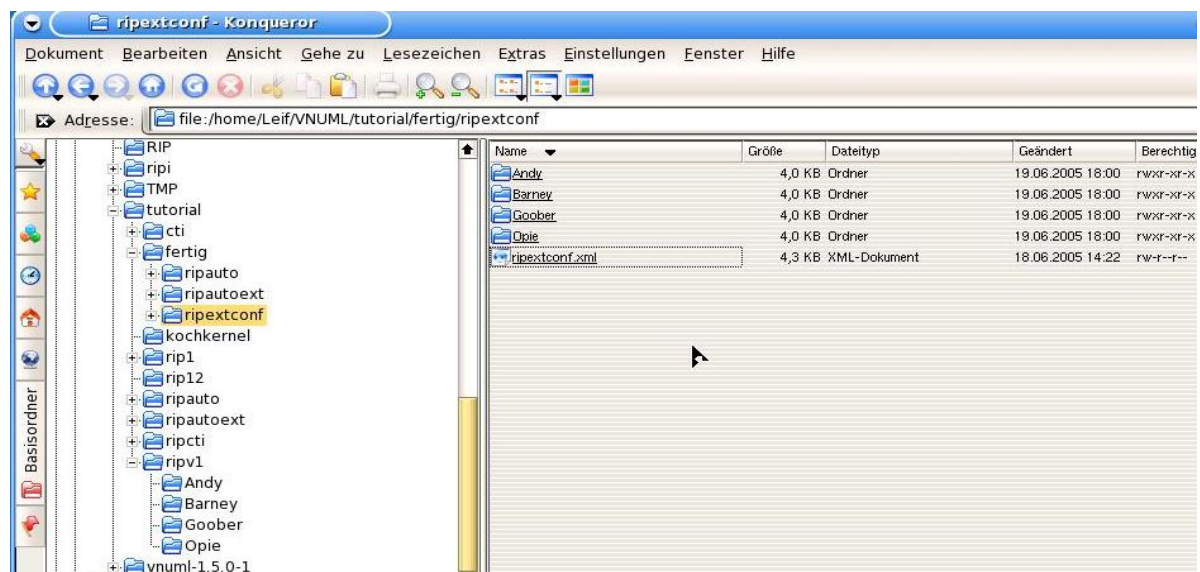
Im folgenden Beispiel sollen nun die Router spezifisch konfiguriert werden.



Aufgabe 4:

Erstellen Sie dazu einen neuen Ordner mit Namen „ripextconf“.

Erstellen Sie darin das hierfür notwendige XML-Szenario. Beachten Sie dabei die unterschiedlichen Subnetmasken. Außerdem sollen sowohl ripd als auch zebra mit externen Konfigurationsdateien automatisch zu starten sein. Speichern Sie anschließend die Dabei unter dem Namen "ripextconf.xml" ab. Legen sie dann die Unterordner für die Konfigurationsdateien der einzelnen Router an.



Lösung zu Aufgabe 4:

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

  <global>
    <version>1.5</version>
    <simulation_name>ripextconf</simulation_name>
    <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
    <automac/>
    <host_mapping/>
  </global>

  <net name="Net0"/>
  <net name="Net1"/>
  <net name="Net2"/>
  <net name="Net3"/>
  <net name="Net4"/>
  <net name="Net5"/>
  <net name="Net6"/>
  <net name="Net7"/>

  <vm name="Barney">
    <filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
    <if id="1" net="Net0">
      <ipv4 mask="255.255.255.240">10.33.5.1</ipv4>
    </if>
    <if id="2" net="Net1">
      <ipv4 mask="255.255.255.0">192.168.83.244</ipv4>
    </if>
    <forwarding/>
    <filetree when="start"
      root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripextconf/Barney</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/local/sbin/zebra -f
    /usr/local/etc/zebra.conf -u root -d</exec>
    <exec seq="start" type="verbatim">/usr/local/sbin/ripd -f
    /usr/local/etc/ripd.conf -u root - d</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="Andy">
    <filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
    </filesystem>
    <if id="1" net="Net1">
      <ipv4 mask="255.255.255.0">192.168.83.1</ipv4>
    </if>
    <if id="2" net="Net2">
      <ipv4 mask="255.255.255.224">192.168.12.65</ipv4>
    </if>
    <if id="3" net="Net3">
      <ipv4 mask="255.255.255.224">192.168.12.195</ipv4>
    </if>
    <if id="4" net="Net4">
      <ipv4 mask="255.255.255.0">172.17.1.1</ipv4>
    </if>
    <if id="5" net="Net5">
      <ipv4 mask="255.255.255.0">172.17.2.1</ipv4>
    </if>

    <forwarding/>
    <filetree when="start"
      root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripextconf/Andy</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/local/sbin/zebra -f
    /usr/local/etc/zebra.conf -u root -d</exec>
    <exec seq="start" type="verbatim">/usr/local/sbin/ripd -f
    /usr/local/etc/ripd.conf -u root - d</exec>
  </vm>
</vnuml>
```

```

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

<vm name="Goober">
  <filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
  </filesystem>
  <if id="1" net="Net4">
    <ipv4 mask="255.255.255.0">172.17.1.2</ipv4>
  </if>
  <if id="2" net="Net6">
    <ipv4 mask="255.255.255.0">172.17.3.1</ipv4>
  </if>
  <forwarding/>
  <filetree when="start"
  root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripextconf/Goober</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">/usr/local/sbin/zebra -f
  /usr/local/etc/zebra.conf -u root -d</exec>
  <exec seq="start" type="verbatim">/usr/local/sbin(ripd -f
  /usr/local/etc/ripd.conf -u root - d</exec>
  <exec seg="stop" type="verbatim"hostname"</exec>
  <exec seg="stop" type="verbatim"killall zebra"</exec>
  <exec seg="stop" type="verbatim"killall ripd"</exec>
</vm>

<vm name="Opie">
  <filesystem type="cow">/usr/local/share/vnuml/filesystems/root_fs_tutorial
  </filesystem>
  <if id="1" net="Net5">
    <ipv4 mask="255.255.255.0">172.17.2.2</ipv4>
  </if>
  <if id="2" net="Net7">
    <ipv4 mask="255.255.255.0">172.17.4.1</ipv4>
  </if>
  <forwarding/>
  <filetree when="start"
  root="/usr/local/etc/">/usr/local/share/vnuml/examples/rip/ripextconf/Opie</filetree>
  <exec seq="start" type="verbatim">hostname</exec>
  <exec seq="start" type="verbatim">/usr/local/sbin/zebra -f
  /usr/local/etc/zebra.conf -u root -d</exec>
  <exec seq="start" type="verbatim">/usr/local/sbin(ripd -f
  /usr/local/etc/ripd.conf -u root - d</exec>
  <exec seg="stop" type="verbatim"hostname"</exec>
  <exec seg="stop" type="verbatim"killall zebra"</exec>
  <exec seg="stop" type="verbatim"killall ripd"</exec>
</vm>
</vnuml>

```

Aufgabe 5:

Erstellen Sie nun in den Unterverzeichnissen die Konfigurationsdateien ripd.conf und zebra.conf.

Lösung zu Aufgabe 5:

Das Aussehen der zebra.conf ist für alle Router gleich. Es handelt sich hierbei um die Standard-Konfigurationsdatei, in der nur die letzte Zeile, die für das Anlegen der log-Datei verantwortlich ist, entfernt worden ist. Dies soll verhindern, dass der Start der Daemons abbricht, weil der Pfad in der die log-Datei angelegt wird, standardmäßig nicht vorhanden ist.

```
! *- zebra *-
!
! zebra sample configuration file
!
hostname Router
! password zebra
! enable password zebra
!

!
! Static default route sample
!
!ip route 0.0.0.0/0 203.181.89.241
!

!log file zebra
```

Barney hängt an zwei Netzwerken 10.0.0.0 und 192.168.83.0. Daher müssen in der ripd.conf beide Netzwerke spezifiziert werden:

```
! *- rip *-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 10.0.0.0/8
network 192.168.83.0/24
```

Aufgabe 6:

Schreiben Sie Konfigurationsdateien für Andy, Goober und Opie.

Lösung zu Aufgabe 6:

Andy:

```
! *- rip *-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 172.17.0.0/16
network 192.168.12.0/24
network 192.168.83.0/24
```

Goober:

```
! *- rip *-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 172.17.0.0/16
```

Opie:

```
! *- rip *-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 172.17.0.0/16
```

Booten Sie das Szenario mit:

```
vnumlparser.pl -t ripextconf.xml -v -B
```

und starten Sie die Daemons mit:

```
vnumlparser.pl -x start@ripextconf.xml -v
```

Die Routing-Tabellen von Barney und Andy sehen beispielsweise dann so aus:

Barney:~# route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Iface
192.168.0.0	*	255.255.255.252	U	0	0	eth0
192.168.12.64	192.168.83.1	255.255.255.224	UG	2	0	eth2
192.168.12.192	192.168.83.1	255.255.255.224	UG	2	0	eth2
192.168.83.0	*	255.255.255.0	U	0	0	eth2
172.17.4.0	192.168.83.1	255.255.255.0	UG	3	0	eth2
172.17.2.0	192.168.83.1	255.255.255.0	UG	2	0	eth2
172.17.3.0	192.168.83.1	255.255.255.0	UG	3	0	eth2
172.17.1.0	192.168.83.1	255.255.255.0	UG	2	0	eth2
10.33.0.0	*	255.255.240.0	U	0	0	eth1

Andy:~# route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Iface
192.168.0.4	*	255.255.255.252	U	0	0	eth0
192.168.12.64	*	255.255.255.224	U	0	0	eth2
192.168.12.192	*	255.255.255.224	U	0	0	eth3
192.168.83.0	*	255.255.255.0	U	0	0	eth1
172.17.4.0	172.17.2.2	255.255.255.0	UG	2	0	eth5
172.17.2.0	*	255.255.255.0	U	0	0	eth5
172.17.3.0	172.17.1.2	255.255.255.0	UG	2	0	eth4
172.17.1.0	*	255.255.255.0	U	0	0	eth4
10.33.0.0	192.168.83.244	255.255.240.0	U	2	0	eth1

Stoppen Sie nun die Daemons mit:

```
vnumlparser.pl -x stop@ripextconf.xml -v
```

6.2 RIPv1 Kompatibilität

Bisher wurde beim Senden und Empfangen der Pakete RIPv2 verwendet. Dies ist die Standardeinstellung von VNUML, d.h. wenn nicht extra angegeben, wird immer RIPv2 benutzt. Um nun die im obigen Szenario verwendeten Router auf RIPv1 zu setzen, muss man jeweils die ripd-Konfigurationsdateien entsprechend anpassen, indem man die Zeile `version 1` einfügt.

z.B für Opie:

```
! *- rip *-
hostname ripd
password zebra
!
router rip
version 1
network 172.17.0.0/16
```

Aufgabe 7:

Stoppen Sie die Daemons und fügen Sie in alle ripd-Konfigurationsdateien `version 1` ein. Starten Sie anschließend wieder die Daemons.

Die Routing-Tabellen sehen nun so aus:

Barney:~# route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Iface
192.168.0.0	*	255.255.255.252	U	0	0	eth0
192.168.83.0	*	255.255.255.0	U	0	0	eth2
10.33.0.0	*	255.255.240.0	U	0	0	eth1

Andy:~# route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Iface
192.168.0.4	*	255.255.255.252	U	0	0	eth0
192.168.12.64	*	255.255.255.224	U	0	0	eth2
192.168.12.192	*	255.255.255.224	U	0	0	eth3
192.168.83.0	*	255.255.255.0	U	0	0	eth1
172.17.4.0	172.17.2.2	255.255.255.0	UG	2	0	eth5
172.17.2.0	*	255.255.255.0	U	0	0	eth5
172.17.3.0	172.17.1.2	255.255.255.0	UG	2	0	eth4
172.17.1.0	*	255.255.255.0	U	0	0	eth4

Goober:~# route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Iface
192.168.0.8	*	255.255.255.252	U	0	0	eth0
192.168.83.0	172.17.1.1	255.255.255.0	UG	2	0	eth1
172.17.4.0	172.17.1.1	255.255.255.0	UG	3	0	eth1
172.17.2.0	172.17.1.1	255.255.255.0	UG	2	0	eth1
172.17.3.0	*	255.255.255.0	U	0	0	eth2
172.17.1.0	*	255.255.255.0	U	0	0	eth1

Opie:~# route

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Iface
192.168.0.12	*	255.255.255.252	U	0	0	eth0
172.17.4.0	*	255.255.255.0	U	0	0	eth2
172.17.2.0	*	255.255.255.0	U	0	0	eth1

Aufgabe 8:

Warum fehlen in den Tabellen plötzlich Einträge ?

Lösung zu Aufgabe 8:

RIPv1 unterstützt keine Übermittlung von Subnet-Masken, was außerhalb des jeweiligen Subnetzes ein gravierendes Informationsdefizit darstellen kann, wenn die Subnetze untereinander nicht streng hierarchisch aufgebaut sind.

Man stelle sich als Beispiel ein Netzwerk mit IP 10.0.0.0 und Mask 255.0.0.0 vor, das Verbindung zur Außenwelt hat, und je eine Verbindung zu zwei Subnetzen mit IP 10.1.0.0 und 10.2.0.0, jeweils mit Maske 255.255.0.0, die untereinander keine Verbindung haben. Muss nun jedoch ein Paket von außerhalb an eines der Subnetze weitergereicht werden, so besteht mittels RIPv1 keinerlei Möglichkeit, das richtige Gateway zu ermitteln, jedoch kann bei RIPv2 die Maske für das Subnetz überliefert werden, womit das Problem gelöst ist.

6.3 Debug-Funktionen und log-Datei

Das Quagga-Paket stellt für RIP drei Debug-Optionen für Verfügung (siehe Kapitel 2: 2.5). Damit man Sie nutzen kann, muss in der ripd.conf ein logfile angelegt worden sein.

Zu beachten ist dabei das Problem, dass das logfile in einem existierenden Verzeichnis angelegt werden muss (siehe Kapitel 3: 1). Die Debug-Nachrichten werden dann in dieser log-Datei ausgegeben.

Die Aktivierung der Debug-Funktionen kann wieder per Telnet oder direktem Eintrag in der ripd.conf geschehen:

Die ripd.conf für das Basiszenario (Kapitel 3: 1) sieht dann beispielsweise so aus:

```
! -*- rip -*-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample,v 1.11 1999/02/19 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
debug rip events
debug rip packet
debug rip zebra
!
router rip
network 10.0.0.0/8
log file /usr/local/etc/ripd.log
```

In diesem Fall wurden alle drei möglichen Debug-Funktionen aktiviert. Deren Ausgaben befinden sich dann in der entsprechenden log-Datei des Routers, die in diesem Fall im Verzeichnis /usr/local/etc angelegt worden ist.

Aufgabe 9:

Aktivieren Sie beispielsweise für das Szenario aus Kapitel 3: 4 die Debug-Funktionen. Fügen Sie dazu in die ripd.conf der einzelnen Router die notwendigen Zeilen für die Debug-Funktionen und das logfile ein.

Beobachten Sie dann die Nachrichten in der ripd.log. Spielen Sie etwas herum, indem Sie beispielsweise Interfaces abschalten, Filterregeln setzen und beobachten Sie, wie sich die Nachrichten verändern.

Ausgaben aus der log-Datei von r1 im Szenario aus Kapitel 3: 4

Start der Daemons:

Zuerst werden die neuen Adressen hinzugefügt:

```
2005/06/27 09:12:00 RIP: interface add dummy0 index 5 flags 130 metric 1 mtu 1500
2005/06/27 09:12:00 RIP: interface add eth0 index 3 flags 4163 metric 1 mtu 1500
2005/06/27 09:12:00 RIP: connected address 192.168.0.2/30 is added
2005/06/27 09:12:00 RIP: interface add eth1 index 1 flags 4419 metric 1 mtu 1500
2005/06/27 09:12:00 RIP: connected address 10.0.0.1/30 is added
2005/06/27 09:12:00 RIP: turn on eth1
2005/06/27 09:12:00 RIP: Redistribute new prefix 10.0.0.0/30 on the interface eth1
2005/06/27 09:12:00 RIP: triggered update!
2005/06/27 09:12:00 RIP: interface add eth2 index 2 flags 4419 metric 1 mtu 1500
2005/06/27 09:12:00 RIP: connected address 10.0.0.5/30 is added
2005/06/27 09:12:00 RIP: turn on eth2
2005/06/27 09:12:00 RIP: Redistribute new prefix 10.0.0.4/30 on the interface eth2
2005/06/27 09:12:00 RIP: interface add lo index 4 flags 73 metric 1 mtu 16436
2005/06/27 09:12:00 RIP: connected address 127.0.0.1/8IP: Terminating on signal
2005/06/27 09:12:00 RIP: interface add dummy0 index 5 flags 130 metric 1 mtu 1500
2005/06/27 09:12:00 RIP: interface add eth0 index 3 flags 4163 metric 1 mtu 1500
2005/06/27 09:12:00 RIP: connected address 192.168.0.2/30 is added
2005/06/27 09:12:00 RIP: interface add sit0 index 6 flags 128 metric 1 mtu 1480
```

Per Multicast erfolgen nun die Requests an die jeweiligen Interfaces:

```
2005/06/27 09:12:01 RIP: multicast join at eth1
2005/06/27 09:12:01 RIP: multicast request on eth1
2005/06/27 09:12:01 RIP: Can't bind socket: Invalid argument
2005/06/27 09:12:01 RIP: SEND to 224.0.0.9.520
2005/06/27 09:12:01 RIP: multicast join at eth2
2005/06/27 09:12:01 RIP: multicast request on eth2
2005/06/27 09:12:01 RIP: Can't bind socket: Invalid argument
2005/06/27 09:12:01 RIP: SEND to 224.0.0.9.520
```

Pakete die von sich selbst stammen, werden ignoriert:

```
2005/06/27 09:12:01 RIP: ignore packet comes from myself
2005/06/27 09:12:01 RIP: ignore packet comes from myself
```

Ein erstes Update erfolgt:

```
2005/06/27 09:12:02 RIP: update timer fire!
2005/06/27 09:12:02 RIP: SEND UPDATE to eth1 ifindex 1
2005/06/27 09:12:02 RIP: Can't bind socket: Invalid argument
2005/06/27 09:12:02 RIP: multicast announce on eth1
2005/06/27 09:12:02 RIP: update routes on interface eth1 ifindex 1
2005/06/27 09:12:02 RIP: SEND to 224.0.0.9.520
2005/06/27 09:12:02 RIP: SEND RESPONSE version 2 packet size 24
2005/06/27 09:12:02 RIP: 10.0.0.4/30 -> 0.0.0.0 family 2 tag 0 metric 1
2005/06/27 09:12:02 RIP: SEND UPDATE to eth2 ifindex 2
2005/06/27 09:12:02 RIP: Can't bind socket: Invalid argument
2005/06/27 09:12:02 RIP: multicast announce on eth2
2005/06/27 09:12:02 RIP: update routes on interface eth2 ifindex 2
2005/06/27 09:12:02 RIP: SEND to 224.0.0.9.520
2005/06/27 09:12:02 RIP: SEND RESPONSE version 2 packet size 24
2005/06/27 09:12:02 RIP: 10.0.0.0/30 -> 0.0.0.0 family 2 tag 0 metric 1
2005/06/27 09:12:02 RIP: ignore packet comes from myself
```

```

2005/06/27 09:12:02 RIP: ignore packet comes from myself
2005/06/27 09:12:02 RIP: RECV packet from 10.0.0.6 port 520 on eth2
2005/06/27 09:12:02 RIP: RECV REQUEST version 2 packet size 24
2005/06/27 09:12:02 RIP: 0.0.0.0/0 -> 0.0.0.0 family 0 tag 0 metric 16
2005/06/27 09:12:02 RIP: update routes to neighbor 10.0.0.6
2005/06/27 09:12:02 RIP: SEND to 10.0.0.6.520
2005/06/27 09:12:02 RIP: SEND RESPONSE version 2 packet size 24
2005/06/27 09:12:02 RIP: 10.0.0.0/30 -> 0.0.0.0 family 2 tag 0 metric 1
2005/06/27 09:12:03 RIP: RECV packet from 10.0.0.6 port 520 on eth2
2005/06/27 09:12:03 RIP: RECV RESPONSE version 2 packet size 24
2005/06/27 09:12:03 RIP: 10.0.0.0/30 -> 10.0.0.5 family 2 tag 0 metric 2
2005/06/27 09:12:03 RIP: Nexthop address 10.0.0.5 is myself
2005/06/27 09:12:03 RIP: RECV packet from 10.0.0.2 port 520 on eth1
2005/06/27 09:12:03 RIP: RECV REQUEST version 2 packet size 24
2005/06/27 09:12:03 RIP: 0.0.0.0/0 -> 0.0.0.0 family 0 tag 0 metric 16
2005/06/27 09:12:03 RIP: update routes to neighbor 10.0.0.2
2005/06/27 09:12:03 RIP: SEND to 10.0.0.2.520
2005/06/27 09:12:03 RIP: SEND RESPONSE version 2 packet size 24
2005/06/27 09:12:03 RIP: 10.0.0.4/30 -> 0.0.0.0 family 2 tag 0 metric 1
2005/06/27 09:12:03 RIP: RECV packet from 10.0.0.2 port 520 on eth1
2005/06/27 09:12:03 RIP: RECV RESPONSE version 2 packet size 24
2005/06/27 09:12:03 RIP: 10.0.0.4/30 -> 10.0.0.1 family 2 tag 0 metric 2
2005/06/27 09:12:03 RIP: Nexthop address 10.0.0.1 is myself
2005/06/27 09:12:04 RIP: RECV packet from 10.0.0.2 port 520 on eth1
2005/06/27 09:12:04 RIP: RECV RESPONSE version 2 packet size 24
2005/06/27 09:12:04 RIP: 10.0.0.4/30 -> 10.0.0.1 family 2 tag 0 metric 2
2005/06/27 09:12:04 RIP: Nexthop address 10.0.0.1 is myself

```

Ausschalten von eth1 an r1:

```

Poison 10.0.0.0/30 on the interface eth1 with an infinity metric [delete]
2005/06/27 09:54:08 RIP: turn off eth1
2005/06/27 09:54:08 RIP: interface eth1 index 1 flags 4354 metric 1 mtu 1500 is down
2005/06/27 09:54:08 RIP: triggered update!
2005/06/27 09:54:08 RIP: SEND UPDATE to eth2 ifindex 2
2005/06/27 09:54:08 RIP: Can't bind socket: Invalid argument
2005/06/27 09:54:08 RIP: multicast announce on eth2
2005/06/27 09:54:08 RIP: update routes on interface eth2 ifindex 2
2005/06/27 09:54:08 RIP: SEND to 224.0.0.9.520

2005/06/27 09:53:53 RIP: update routes on interface eth2 ifindex 2
2005/06/27 09:53:53 RIP: SEND to 224.0.0.9.520
2005/06/27 09:53:53 RIP: SEND RESPONSE version 2 packet size 24
2005/06/27 09:53:53 RIP: 10.0.0.0/30 -> 0.0.0.0 family 2 tag 0 metric 1
2005/06/27 09:53:53 RIP: ignore packet comes from myself
2005/06/27 09:53:53 RIP: ignore packet comes from myself

```

Reaktivierung von eth 1:

```

2005/06/27 09:59:00 RIP: interface eth1 index 1 flags 4419 metric 1 mtu 1500 is up
2005/06/27 09:59:00 RIP: turn on eth1
2005/06/27 09:59:00 RIP: Redistribute new prefix 10.0.0.0/30 on the interface eth1
2005/06/27 09:59:00 RIP: interface eth1: passive = 0
2005/06/27 09:59:00 RIP: triggered update!
2005/06/27 09:59:00 RIP: SEND UPDATE to eth2 ifindex 2
2005/06/27 09:59:00 RIP: Can't bind socket: Invalid argument
2005/06/27 09:59:00 RIP: multicast announce on eth2
2005/06/27 09:59:00 RIP: update routes on interface eth2 ifindex 2
2005/06/27 09:59:00 RIP: SEND to 224.0.0.9.520
2005/06/27 09:59:00 RIP: SEND RESPONSE version 2 packet size 24

```

Stoppen der Daemons:

```
2005/06/27 10:01:25 RIP: Terminating on signal
2005/06/27 10:01:25 RIP: interface dummy0: passive = 0
2005/06/27 10:01:25 RIP: interface eth0: passive = 0
2005/06/27 10:01:25 RIP: interface eth1: passive = 0
2005/06/27 10:01:25 RIP: interface eth2: passive = 0
2005/06/27 10:01:25 RIP: interface lo: passive = 0
2005/06/27 10:01:25 RIP: interface sit0: passive = 0
```

6.4 Passive Interfaces

Im Folgenden soll im Beispiel 6.1 der Router Andy über das Interface 1, welches für die Verbindung um Netz zwischen Andy und Barney zuständig ist, keine RIP Nachrichten mehr versenden, selber aber jedoch noch Updates empfangen und verarbeiten können. Dazu fügt man in seiner Konfigurationsdatei den passive-Interface-Befehl für das entsprechende Interface hinzu:

```
router rip
passive-interface Ethernet1
network 172.17.0.0/16
network 192.168.12.0/24
network 192.168.83.0/24
```

Zum Vergleich die tcpdump-Ausgaben von beiden Seiten:

```
Andy:~# tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
12:28:46.029393 IP Andy.route > 224.0.0.9.route: RIPv2, Response, length: 144
12:28:50.478563 IP 192.168.83.244.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:29:11.070693 IP Andy.route > 224.0.0.9.route: RIPv2, Response, length: 144
12:29:19.533462 IP 192.168.83.244.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:29:38.110403 IP Andy.route > 224.0.0.9.route: RIPv2, Response, length: 144
12:29:44.549093 IP 192.168.83.244.route > 224.0.0.9.route: RIPv2, Response, length:
144
```

```
Barney:~# tcpdump -i eth2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 96 bytes
12:27:15.439564 IP 192.168.83.244.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:27:21.954537 IP 192.168.83.1.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:27:42.990278 IP 192.168.83.1.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:27:45.446589 IP 192.168.83.244.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:28:09.458172 IP 192.168.83.244.route > 224.0.0.9.route: RIPv2, Response, length:
144
12:28:11.030434 IP 192.168.83.1.route > 224.0.0.9.route: RIPv2, Response, length:
144
```

Man erkennt, Barney sendet die Nachrichten sowohl an 192.168.83.1 und 192.168.83.244 während Andy nur dann die Nachrichten von Barney 192.168.83.244 erhält.

passive-interface ist kein RIP-spezifischer Befehl. Er könnte für jedes IP Routing Protokoll konfiguriert werden.

Anhang

Glossar ripd-Befehle

Diese Befehle kann man sich in einer Telnet-Verbindung mit der Tastenkombination „Shift+ß“ bzw „?“ anzeigen lassen.

Benutzer-Modus:

ripd>

ripd>	enable exit help list quit show history show ip prefix-list WORD show ip prefix-list detail show ip prefix-list detail WORD show ip prefix-list summary show ip prefix-list summary WORD show ip rip show ip rip status show ipv6 prefix-list WORD show ipv6 prefix-list detail show ipv6 prefix-list detail WORD show ipv6 prefix-list summary show ipv6 prefix-list summary WORD show memory show thread cpu show version terminal length <0-512> terminal no length who	Turn on privileged mode command Exit current mode and down to previous mode Description of the interactive help system Print command list Exit current mode and down to previous mode Display the session command history Show prefix-list (WORD=name of prefix-list) Details of prefix-lists Details of a prefix-list (WORD=name) Summary of prefix-lists Summary of a prefix-list (WORD=name) Show RIP routes Process parameters and statistics Show prefix-list (WORD=name of prefix-list) Details of prefix-lists Details of a prefix-list (WORD=name) Summary of prefix-lists Summary of a prefix-list (WORD=name) Memory statistics Thread CPU usage Displays zebra version Set number of lines on a screen Negate a command or set its defaults Display who is on vty
-------	---	---

privilegierter Modus: (enable)

ripd#

ripd#	clear ip prefix-list clear ip prefix-list WORD clear ipv6 prefix-list clear ipv6 prefix-list WORD configure terminal	Clear prefix-lists Clear a prefix-list (WORD=name) Clear prefix-lists Clear a prefix-list (WORD=name) Configuration terminal
ripd(config)#	access-list <1-99> remark LINE access-list <1-99> deny A.B.C.D access-list <1-99> deny any access-list <1-99> deny host A.B.C.D access-list <1-99> permit A.B.C.D access-list <1-99> permit any access-list <1-99> permit host A.B.C.D access-list <100-199> ... access-list <1300-1999> ... access-list <2000-2699> ... access-list WORD ...	Access-list entry comment Specity packets to reject (A.B.C.D=address to match) Reject any source host Reject a single host address Spectify packets to forward (A.B.C.D=address to match) Forward any source host Forward a single host address IP extended access list IP standard access list (expanded range) IP extended access list (expanded range) IP zebra access-list

	<p>banner motd default debug rip events debug rip packet debug rip packet rcv debug rip packet send debug rip zebra enable password 8 enable password LINE end exit help hostname NAME interface IFNAME</p>	<p>Set banner string Debug RIP events Debug RIP packet Debug RIP receive packet Debug RIP send packet Debug RIP and ZEBRA communication Specifies a HIDDEN password will follow The UNENCRYPTED (cleartext) 'enable' password End current mode and change to enable mode Exit current mode and down to previous mode Description of the interactive help system Set system's network name Select an interface of configure (IFNAME=ethX)</p>
ripd(config-if)#	<p>description end exit help ip rip authentication key-chain LINE ip rip authentication mode md5 ip rip authentication mode text ip rip authentication string LINE ip rip receive version 1 ip rip receive version 2 ip rip send version 1 ip rip send version 2 ip rip split-horizon ip rip split-horizon poisoned-reverse list no description no ip rip ... quit show running-config write write file write memory write terminal</p>	<p>Interface specific description End current mode and change to enable mode Exit current mode and down to previous mode Description of the interactive help system Authentication key-chain (LINE=name of key-chain) Authentication mode Keyed message digest Authentication mode Clear text authentication Authentication string (LINE=authentication string) Advertisement reception RIP version 1 Advertisement reception RIP version 1 Advertisement transmission RIP version 1 Advertisement transmission RIP version 2 Perform split-horizon Perform split-horizon with poisoned-reverse Print command list No Interface specific description Negate ip rip commands Exit current mode and down to previous mode Show running system information Write running configuration Write to configuration file Write configuration to the file (same as write file) Write to terminal</p>
ripd(config)#	<p>ip prefix-list WORD ip prefix-list sequence-number ipv6 access-list WORD ipv6 prefix-list WORD ipv6 prefix-list sequence-number key chain WORD line vty list log file FILENAME log record-priority log stdout log syslog log syslog facility auth log syslog facility cron log syslog facility daemon log syslog facility kern log syslog facility localX (X=0-7) log syslog facility lpr log syslog facility mail log syslog facility news log syslog facility syslog log syslog facility user log syslog facility uucp log trap... no ...</p>	<p>Build a prefix list (WORD=name of a prefix-list) Include/exclude sequence numbers in NVGEN Add an access list entry (WORD=IPv6 zebra access list) Build a prefix list (WORD=name of a prefix-list) Include/exclude sequence numbers in NVGEN Key-chain management (WORD=Key-chain name) Configure a virtual terminal line Print command list Logging to file (FILENAME=Logging filename) Log the priority of the message within the message Logging goes to stdout Logging goes to syslog Authorisation system Cron/at facility System daemons Kernel Local use Line printer system Mail system USENET news Syslog itself User process Unix-to-Unix copy system Limit logging to specified level Negates a command or set its defaults</p>

	password 8 WORD password LINE quit route-map WORD router zebra router rip	Specifies a HIDDEN password (WORD=hidden password) The UNENCRYPTED (cleartext) line password Exit current mode and down to previous mode Create route-map oder enter route-map command mode Make connection to zebra daemon Routing Information Protocol (RIP)
ripd(config-router)#	default-information originate default-metric <1-16> distance <1-255> distribute-list WORD distribute-list prefix WORD end exit help list neighbour A.B.C.D network A.B.C.D/M network WORD no ... offset-list WORD in <0-16> offset-list WORD in <0-16> passive-interface IFNAME passive-interface default quit redistribute bgp redistribute bgp metric <0-16> redistribute bgp route-map WORD redistribute connected ... redistribute kernel ... redistribute ospf ... redistribute static ... route A.B.C.D/M route-map RMAP in IFNAME route-map RMAP out IFNAME show running-config timers basic U T G version <1-2> write write file write memory write terminal	Distribute a default route Set a metric of redistribute routes Administrative distance Filter networks in routing updates (WORD=access-list) Filter prefixes in routing updates (WORD=prefix-list) End current mode and change to enable mode End current mode and down to previous mode Description of the interactive help system Print command list Specify a neighbour router Enable routing on an IP network (M=length z.b 10.0.0.0/8) Enable routing on an interface (WORD=interface) Negate a command or set its defaults Modify RIP metric for incoming updates Modify RIP metric for outgoing updates Suppress routing updates on an interface Default for all interfaces Exit current mode and down to previous mode Redistribute information from bgp protocol Metric Route map reference (WORD=Pointer to entries) Redistribute information from connected protocols Redistribute kernel routes Redistribute ospf protocol Redistribute static routes RIP static route configuration Route map set for input filtering (RMAP=Route map) Route map set for output filtering (RMAP=Route map) Show running system information U=Update T=Timeout G=Garbage Collector Timer Set routing protocol version Write running configuration Write to configuration file Write configuration to the file (same as write file) Write to terminal
ripd(config)#	service advanced-vty service password-encryption service terminal-length <0-512> show history show running-config write write file write memory write terminal	Enable advanced mode vty interface Enable encrypted passwords System wide terminal length configuration Display the session command history Show running configuration Write running configuration Write to configuration file Write configuration to the file (same as write file) Write to terminal
ripd#	copy running-config startup-config debug rip events debug rip packet debug rip packet recv debug rip packet send debug rip zebra disable end exit help list	Copy running-configuration to startup-configuration Debug RIP events Debug RIP packet Debug RIP receive packet Debug RIP sent packet Debug RIP and ZEBRA communication Turn off privileged mode command End current mode and change to enable mode End current mode and down to previous mode Description of the interactive help system Print command list

no ... quit show debugging rip show history show ip access-list show ip access-list <1-99> show ip access-list <100-199> show ip access-list <1300-1999> show ip access-list <2000-2699> show ip access-list WORD show ip prefix-list WORD A.B.C.D/M show ip prefix-list WORD seq show ip prefix-list detail show ip prefix-list detail WORD show ip prefix-list summary show ip prefix-list summary WORD show ip rip show ip rip status show ipv6 ... show memory show memory all show memory bgp show memory lib show memory ospf show memory ospf6 show memory rip show memory ripng show running-config show startup-config show thread cpu [FILTER] show version terminal length <0-512> terminal monitor terminal no ... who write write file write memory write terminal	Negate a command or set its defaults Exit current mode and down to previous mode Show debugging functions Display session command history List IP access lists IP standard access list IP extended access list IP standard access list (expanded range) IP extended access list (expanded range) IP zebra access list Build a prefix list (WORD=name of a prefix list) Sequence number of an entry Detail of prefix lists Detail of a prefix list Summary of prefix lists Summary of a prefix list Show RIP routes IP routing protocol process parameters and statistics IPv6 information Memory statistics All memory statistics BGP memory Library memory OSPF memory OSPF6 memory RIP memory RIPng memory Running configuration Contentes of the startup configuration Thread information Displays zebra version Set number of lines on a screen Copy debug output to the current terminal line Negate a command or set its defaults Display who is on vty Write running configuration Write to configuration file Write configuration to the file (same as write file) Write to terminal
---	--