UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4
Institut für Informatik
Arbeitsgruppe Rechnernetze

# Implementation und Evaluation automatischer Routen-Aggregation in einem schleifen-erkennenden Distanz-Vektor Verfahren

## Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Christian Henke

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Hannes Frey<br>(Institut für Informatik, AG Rechnernetze) |
| Zweitgutachter: | Dipl. Inform. Frank Bohdanowicz<br>(Institut für Informatik, AG Rechnernetze) |

Koblenz, im September 2013

# Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

| | Ja | Nein |
|---|---|---|
| Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden. | ☐ | ☐ |
| Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. | ☐ | ☐ |

.....
(Ort, Datum) (Unterschrift)

## Zusammenfassung

In dieser Arbeit wird das in letzter Zeit zunehmend wichtiger werdende Thema der Routenaggregation und deren Auswirkung auf die Verhinderung von Routingschleifen behandelt. Als Basis für die Implementation und Evaluation dient das an der Universität Koblenz entwickelte RMTI-Protokoll, bei dem es sich um eine Weiterentwicklung des in RFC2453 spezifizierten Routing Information Protocol Version 2 handelt. Dieses Protokoll kommt, in dieser Arbeit, innerhalb der virtuellen Netzwerkumgebung Virtual Network User-Mode-Linux (VNUML) zum Einsatz. Mit VNUML ist es möglich konkrete Netzwerkszenarien virtuell zu betreiben und zu untersuchen. Der RMTI ist bereits in der Lage topologische Schleifen zu erkennen und dadurch das Entstehen von Routingschleifen nachweislich zu verhindern. Im Rahmen der Arbeit wird die Funktionsweise des RMTI beschrieben und anschließend darauf eingegangen, unter welchen Umständen eine Routenaggregation vorgenommen werden darf ohne das die Aggregation Routinganomalien nach sich zieht. Um diese Änderungen vornehmen zu können ist ein tieferes Verständnis der Struktur von Routingtabellen notwendig, daher wird deren Aufbau anhand von Beispielen erläutert. Im Anschluss wird beschrieben an welchen Stellen Änderungen am RMTI vorgenommen werden müssen um trotz Aggregation eine Verhinderung von Routingschleifen bewirken zu können. Am Ende der Arbeit findet abschließend eine Evaluation der Reorganisationsfähigkeit des virtuellen Netzes bei vorgenommener Routenaggregation statt.



## Abstract

In this thesis we discuss the increasingly important routing aggregation and its consequences on avoiding routing loops. As basis for implementation and evaluation I will use the RMTI protocol, developed at the University of Koblenz, which is an evolution of the RFC2453 specified in the Routing Information Protocol version 2. The virtual network environment Virtual Network User Mode Linux (VNUML) is used within this thesis as environment. With VNUML it is possible to operate and evaluate real network scenarios in a virtual environment. The RMTI has already proven its ability to detect topological loops and thereby prevent the formation of these routing loops. In this thesis we will describe the function of the RMTI and then discuss under which circumstances we can use routing aggregation, without it resulting in routing anomalies. In order to implement these changes it is essential to have a deeper understanding of the structure of routing tables, so the construction will be explained using reference to examples. There follows a description of which points have to be changed, in the RMTI in order to avoid loops despite aggregation. In Summary we will evaluate the affects the routing aggregation has on the reorganization ability of the virtual network.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1. Einleitung

## 1.1. Motivation

Durch den rasanten Anstieg der Teilnehmeranzahl im Internet in den vergangenen Jahren, kam es zu einem starken Anwachsen der im weltweiten Internet veröffentlichten IP-Adressbereiche. Diese Entwicklung hält seit vielen Jahren an und ein Ende ist vorerst nicht abzusehen. Damit verbunden ist eine konsequente Vergrößerung der in Routern eingesetzten Routingtabellen, was zur Folge hat, dass der Prozess des Auffindens der passenden Route innerhalb eines Routers mehr Zeit beansprucht und der Austausch der Routinginformationen zwischen den Routern immer mehr Datenverkehr verursacht. Um dem entgegen zu wirken, wurde das Konzept der Routenaggregation entwickelt, hierbei wird versucht Teilnetze, wenn möglich, zu einem größeren zusammen zu fassen und dieses Aggregat, anstatt den kleineren Teilnetzen, aktiv im Netz zu veröffentlichen.

Diese Arbeit baut auf einem ersten Implementierungsversuch von automatischer Routenaggregation in den RMTI-Algorithmus (Routing with Metric based Topology Investigation) auf und hat das Ziel diese Implementierung zu verbessern, robuster zu gestalten und zu evaluieren. Es wird im Laufe der Arbeit anschaulich auf die Frage eingegangen, unter welchen Umständen Routen in Distanzvektorverfahren aggregiert werden dürfen und inwiefern sich dies auf die Routingschleifenerkennung des RMTI auswirkt. Zum Einsatz kommt die Netzwerkvirtualisierungssoftware VNUML (Virtual Network User-Mode-Linux), die es ermöglicht auf einem physikalischen Rechner, mehrere User-Mode-Linux (UML) Systeme zu emulieren und diesen zu einem Netzwerk zu verbinden. Komfortabel lassen sich diese virtuellen Netzwerke mit der VNUML-Sprache beschreiben, mit der es möglich ist ein Netzwerkszenario automatisiert zu starten. Jeder UML-Rechner innerhalb des Netzes ist gleichzeitig ein vollwertiger Linuxrechner, auf dem es zum Beispiel möglich ist beliebige Prozesse und Programme zu installieren, zu starten und zu konfigurieren. Innerhalb dieser UML-Rechner wird die Routingsuite Quagga benutzt, um IP-Routing zu ermöglichen und Anpassungen an den verwendeten Protokollen vorzunehmen. Das Protokoll auf welches sich die Arbeit bezieht ist der RMTI, eine Erweiterung des Routing Information Protocol (RIPv2) um die Erkennung von Routingschleifen, zur Verhinderung des Counting-to-Infinity Problems.

Ziel dieser Bachelorarbeit ist die Erweiterung des RMTI um die Möglichkeit der Routenaggregation, um diesen weiter an die an ein modernes Routingprotokoll gestellten Anforderungen anzupassen.

## 1.2. Aufbau

Nach einer kurzen Einführung im ersten Kapitel, wird in Kapitel zwei ein Überblick über die verwendeten Begriffe und Techniken gegeben. Insbesondere wird auf den RMTI-Algorithmus und seine Funktionsweise eingegangen und die Routenaggregation erklärt. Es werden einige möglicherweise auftretende Probleme bei der Routenaggregation genannt und eine häufig verwendete Datenstruktur für die Datenhaltung in Routingtabellen, die PATRICIA-Tries erläutert. Das dritte Kapitel verdeutlicht, wie Routingtabellen aufgebaut sind und bespricht, wann eine Aggregation durchgeführt werden darf. Im vierten Kapitel wird anhand von Quelltext erklärt, wie die Routenaggregation in den RMTI integriert wurde. In Kapitel fünf wird die Implementierung auf korrektes Verhalten und Reorganisationsfähigkeit überprüft um im Anschluss im sechsten Kapitel zu untersuchen ob die Schleifenerkennung des RMTI auch in aggregierbaren Netzen korrekt arbeiten kann. Das letzte Kapitel gibt einen kurzen Ausblick darauf, wie die Vorteile von Routenaggregation und Schleifenerkennung gleichzeitig genutzt werden können um eine möglichst hohe Effizienz beim erzeugten Routingtraffic und der Routingtabellenorganisation zu erreichen.

# 2. Grundlagen

Im folgenden Kapitel, werden Begriffe und Grundlagen erklärt um die bearbeitete Thematik besser greifbar zu machen.

## 2.1. Routing

### 2.1.1. Einführung

Der Begriff Routing bezeichnet den Vorgang der Pfadwahl zur Übertragung von Datenpaketen zwischen getrennten Netzwerken. Ein Paket wird, falls der Adressat sich nicht im selben Netz befindet, über mehrere dazwischenliegende Router(Hops) zum Ziel gesendet. Begrifflich wird zwischen Routing und Forwarding unterschieden. Während Routing die gesamte Pfadwahl eines Pakets vom Start- zum Zielpunkt bezeichnet, geht es beim Forwarding um den Entscheidungsprozess eines Routers für die Wahl des nächsten Hops, um das Datenpaket näher ans Ziel zu befördern. Beim Routing wird zwischen statischen und dynamischen Verfahren unterschieden. Statisches Routing bedeutet, die Routen werden manuell verwaltet und beim Ausfall einer gesetzten Route erfolgt keine automatische Reorganisation. Beim dynamischen Routing wiederrum, sind die Router in der Lage die Netztopologie selbständig zu erlernen und eigenständig auf Veränderungen im Netz zu reagieren. Die meisten dynamischen Routingprotokolle verfolgen einen von zwei unterschiedlichen Ansätzen. Zum einen gibt es hier die Link-State-Verfahren, die nach dem Prinzip "*Teile der Welt mit, wer deine Nachbarn sind*" arbeiten, zum Anderen werden sogenannte Distanzvektorprotokolle eingesetzt, die nach dem gegenteiligen Prinzip "*Teile deinen Nachbarn mit, wie für dich die Welt aussieht* " funktionieren. Distanzvektorprotokolle versenden somit ihre gesamte Routingtabelle an ihre direkten Nachbarn und haben kein Detailwissen über die eigentliche Netzwerktopologie während Linkstateprotokolle die Informationen über die Verbindungszustände zu ihren direkten Nachbarn im gesamten Netz verbreiten und so jeder Router in der Lage ist sich den Pfad zu jedem Ziel selbst zu berechnen. Der bekannteste Vertreter der Link-State-Protokolle ist Open Shortest Path First (OSPF), eines der heutzutage meistbenutzten Intradomain-Routingprotokolle. Bei den Distanzvektorprotokollen ist das Routing Information Protocol (RIP) und seine Erweiterungen ein häufig benutzter Vertreter dieser Protokollfamilie. Während es sich bei RIP und OSPF um Interior Gateway Protokolle handelt, also um Protokolle die innerhalb eines Autonomen Systems Verwendung finden, existiert eine weitere Protokollfamilie, die Exterior Gateway Protokolle. Diese werden zur Vermittlung zwischen Autonomen Systemen eingesetzt

### 2.1.2. Autonomes System

Ein Autonomes System ist ein größerer Verbund von Routern und Netzwerken die einheitlich administriert werden und durch Interior Gateway Protokolle verbunden sind. Autonome Systeme werden üblicherweise von Providern, Unternehmen und Organisationen betrieben und haben eine einzigartige Autonomous System Number zur Identifikation, die von einer Regional Internet Registry festgelegt wurden, welche wiederrum von der Internet Assigned Numbers Authority (IANA) kontrolliert werden. Um ein Routing zwischen einzelnen Autonomen Systemen zu ermöglichen kommen Exterior Gateway Protokolle zum Einsatz, wobei hier aktuell nur das BGP (Border Gateway Protokoll) Verwendung findet.
Die Gesamtheit aller Autonomen Systeme bildet das heutige Internet.

### 2.1.3. Konvergenz

Der Zustand der Konvergenz in einem Netzwerk, besagt, dass jeder Router die selbe konsistente Sicht auf den aktuellen Stand der Netzwerktopologie hat. Wenn das Netz konvergiert ist, finden somit keine grundsätzlichen Änderungen in den Routingtabellen der im Netz vertretenen Router mehr statt. Dies ändert sich erst, wenn ein Ereignis eintritt, wie zum Beispiel der Wegfall einer bestehenden Verbindung, welches eine Reorganisation des Netzwerks erfordert. Der Zeitraum der zum Erreichen eines konvergenten Zustands benötigt wird, hängt vom eingesetzten Routingprotokoll ab. Hierbei kann es je nach Netzaufbau und verwendetem Protokoll drastische Unterschiede geben. Die Minimierung der Konvergenzzeit ist daher eines der erklärten Ziele von Forschung und Entwicklung und wird auch in kommenden Entwicklungen weiter fokussiert werden.

### 2.1.4. Metrik

Die Metrik beschreibt die Verbindungsqualität in ein bestehendes Subnetz und dient als Selektionskriterium falls redundante Verbindungen zum Ziel existieren. Zur Berechnung der Metrik können unterschiedliche Faktoren, wie beispielsweise die zur Verfügung stehende Bandbreite einer entsprechenden Route, gemessene Latenzen, Paketverlustraten, Anzahl zu passierender Hops zum Ziel, sowie Nutzungskosten für entsprechende Teilverbindungen herangezogen werden. Beim RIP-Algorithmus beschränkt sich die Berechnung der Metrik lediglich auf den Hopcount, also die Anzahl zu passierender RIP-Router bis zum Ziel. Metriken werden für gewöhnlich als Ganzzahl vorgehalten und sind in den Routingtabellen einsehbar. Jeder Route wird eine entsprechende Metrik zugewiesen.

### 2.1.5. Topologische Schleifen

Bei topologischen Schleifen, handelt es sich um redundante Verbindungen in einem vermaschten Netz, die grundsätzlich als Alternativrouten erwünscht sind. So kann beim Ausfall einer Route, auf eine Alternative mit schlechterer Metrik zurückgegriffen werden, sodass die Erreichbarkeit des gesamten Netzes nach wie vor gewährleistet ist. Topologische Schleifen können allerdings sowohl in Distanz-Vektor-Protokollen, als auch in Link-State-Protokollen (transient Loops) negative Auswirkungen haben. So kann es in Distanz-Vektor-Protokollen dazu kommen, dass nach dem Ausfall einer Route fehlerhafte Informationen zu dieser ausgefallenen Routen in einer Schleife zirkulieren, bis die maximale Metrik erreicht ist, man spricht hier vom Counting-to-Infinity-Problem, welches in 2.2.2. erläutert wird. Während des Existierens einer Routingschleife, kann wie in [Bohdan08] gezeigt, der Traffic im an der Schleife beteiligten Netzsegment stark ansteigen, da die Nutzdaten, für das aktuell fehlgeroutete Netz bis zum Ablauf des TTL-Werts in der Schleife zirkulieren.

In Link-State-Verfahren führen topologische Schleifen bei dem Ausfall eines Routers nicht zum Counting to Infinity Problem, da veraltete Informationen hier zuverlässig an der Sequenznummer des Update-Pakets erkannt werden können.

## 2.2. Routing Information Protocol

### 2.2.1. Einführung

Beim Routing Information Protocol [RFC1058] handelt es sich um ein Interior-Gateway-Protokoll, welches in seiner ersten Version 1988 in RFC 1058 spezifiziert wurde. Während RIPv1 noch mit Netzklassen arbeitete, wurde die 1998 veröffentliche Version RIPv2 [RFC2453] durch Classless Inter-Domain Routing [RFC1519] ergänzt, womit Subnetzmasken variabler Länge nutzbar wurden.

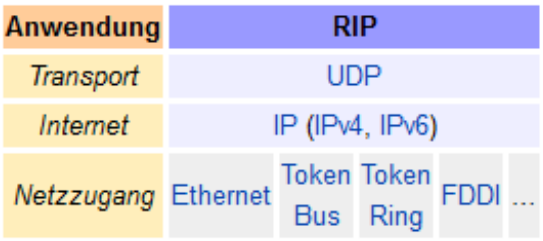



**Abb. 2.1 RIP im TCP/IP-Protokollstapel [wiki1]**

Bei Distanzvektorverfahren wie RIP, handelt es sich um die Realisierung des Bellman-Ford-Algorithmus zum Auffinden des kürzesten Weges in einem Graphen. Das Einsatzgebiet von RIP sind Netze kleinerer und mittlerer Größe, da das Protokoll nur einen maximalen Hop-Count von 15 unterstützt. Netze, die zum Beispiel durch eine Verbindungsunterbrechung, nicht mehr erreichbar sind werden in den Routing-Tabellen mit der Metrik 16 gekennzeichnet, was auch als RIP-Infinity bezeichnet und entsprechend propagiert wird. RIPs größter Vorteil ist die leicht verständliche Funktionsweise und die damit einhergehende Robustheit, sodass ein neuer RIP-Router schnell konfiguriert und in ein bestehendes Netz implementiert werden kann. Trotzdem wurde RIP mittlerweile im produktiven Bereich durch neuere Verfahren, wie OSPF, EIGRP und IS-IS verdrängt, obwohl es in seiner ursprünglichen Version sehr geringe Hardware-anforderungen stellt.

Die Routingupdates überträgt RIP im in Abbildung 2.2 dargestellten Format. Jede Nachricht hat einen Header in dem der Nachrichtentyp im Feld Command und die verwendete RIP-Version übergeben wird. Beim Command wird zwischen Response und Request unterschieden, die Version gibt an, ob es sich um eine RIPv1- oder RIPv2-Nachricht handelt.

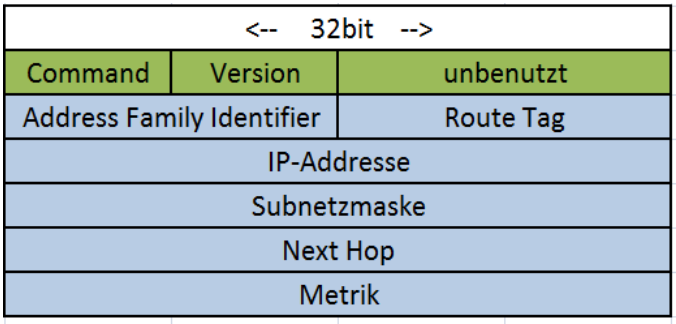



**Abb. 2.2 RIPv2-Nachrichtenaufbau**

Im Anschluss an den Header folgen bis zu 25 RIP-Einträge, die durch Netzadresse, Subnetzmaske, Next Hop und die Metrik beschrieben werden. Der Punkt Address Family Identifier gibt an, welches Adressierungsprotokoll verwendet wird, im Falle von IP wäre hier der Eintrag auf 2 gesetzt. Der Route Tag wird verwendet, wenn es sich um eine Route handelt, die von einem anderen Protokoll als RIP gelernt wurde.

Dadurch das jeder RIP-Router alle 30 Sekunden seine gesamte Routingtabelle an seine unmittelbaren Nachbarn weiter leitet, entsteht in einem Netz aus RIP-Routern verhältnismäßig viel Datenverkehr. Da auf der Transportschicht UDP zum Einsatz kommt ist eine gesicherte Übertragung der Routingupdates nicht gewährleistet.
In den Routingtabellen der RIP-Router herrscht keine Redundanz, was bedeutet, dass Alternativrouten mit einer schlechteren Metrik verworfen werden. Dadurch ist es der RIP-

Standard-Implementation nicht möglich topologische Schleifen zu erkennen, was beim Ausfall bestimmter Routen die sogenannte Counting to Infinity
Problematik auslösen kann, auf welche im Folgenden noch genauer eingegangen wird.

Der größte Nachteil bei der Verwendung von RIP sind die hohen Konvergenzzeiten bei Änderungen in der Netztopologie. Bei einem maximal möglichen Hopcount von 15, kann es im schlechtesten Fall passieren, dass nach einer Änderung ein konsistenter Zustand erst nach ungefähr sieben Minuten erreicht wird, da die Änderungsinformation nur alle 30 Sekunden von Knoten zu Knoten weiter gereicht wird.

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

     Network          Next Hop         Metric From            Tag Time
C(i) 10.0.1.0/24      0.0.0.0               1 self              0
R(n) 10.0.2.0/24      10.0.1.2              2 10.0.1.2          0 00:59
C(i) 10.0.3.0/24      0.0.0.0               1 self              0
R(n) 10.0.4.0/24      10.0.3.2              2 10.0.3.2          0 01:00
R(n) 10.0.5.0/24      10.0.3.2              3 10.0.3.2          0 01:00
Router>
```

**Abb. 2.3 Quagga-RIP-Routingtabelle**

Wenn nun ein neuer RIP-Router ins Netz integriert wurde passiert folgendes :
Beim Start kennt dieser nur seine direkt anliegenden Netze, weshalb er von seinen direkten Nachbarn per multicast deren Tabellen anfordert. Ab jetzt versendet der neue RIP-Router, ebenso wie die bereits im Netz befindlichen alle 30 Sekunden Updatepakete, die die jeweilige gesamte Routingtabelle beinhalten. Sollte nun eine in der Routingtabelle eingetragene Route 180 Sekunden lang nicht durch ein reguläres RIP-Update bestätigt werden, da beispielsweise der entsprechende Nachbarrouter ausgefallen ist, wird die Route mit Metrik 16 gekennzeichnet. Wenn nun in den nächsten 120 Sekunden bei keinem Interface des Routers ein RIP-Update mit einer besseren Metrik eintrifft wird die Route komplett aus der Routingtabelle entfernt.

In der nachfolgenden Abbildung ist der nach kurzer Zeit erreichte konvergente Zustand eines Netzes von RIP-Routern innerhalb einer Y-Topologie zu sehen.

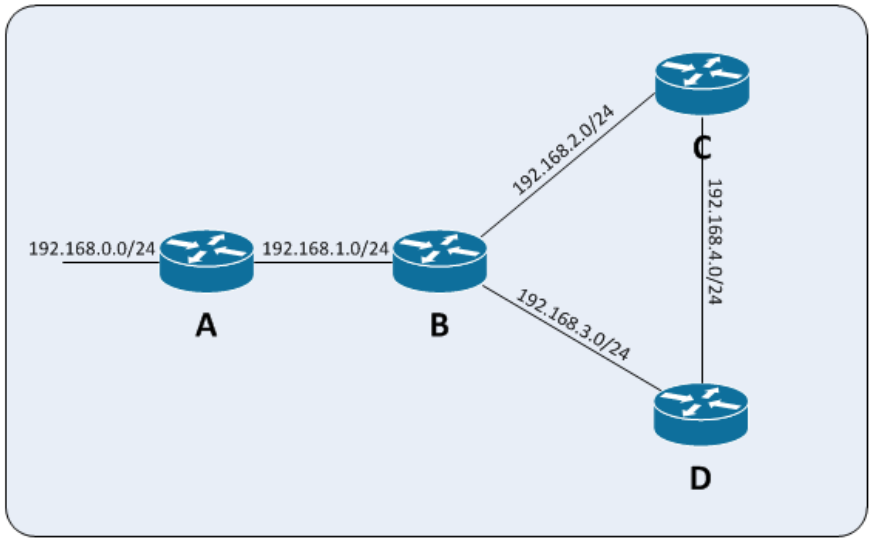



**Abb. 2.4 RIP-Router innerhalb Y-Topologie**

**Routingtabelle A**

| Ziel | Next Hop | Metrik |
|---|---|---|
| 192.168.0.0/24 | self | 1 |
| 192.168.1.0/24 | self | 1 |
| 192.168.2.0/24 | B | 2 |
| 192.168.3.0/24 | B | 2 |
| 192.168.4.0/24 | B | 3 |

**Routingtabelle B**

| Ziel | Next Hop | Metrik |
|---|---|---|
| 192.168.0.0/24 | A | 2 |
| 192.168.1.0/24 | self | 1 |
| 192.168.2.0/24 | self | 1 |
| 192.168.3.0/24 | self | 1 |
| 192.168.4.0/24 | C | 2 |

**Routingtabelle C**

| Ziel | Next Hop | Metrik |
|---|---|---|
| 192.168.0.0/24 | B | 3 |
| 192.168.1.0/24 | B | 2 |
| 192.168.2.0/24 | self | 1 |
| 192.168.3.0/24 | D | 2 |
| 192.168.4.0/24 | self | 1 |

**Routingtabelle D**

| Ziel | Next Hop | Metrik |
|---|---|---|
| 192.168.0.0/24 | B | 3 |
| 192.168.1.0/24 | B | 2 |
| 192.168.2.0/24 | B | 2 |
| 192.168.3.0/24 | self | 1 |
| 192.168.4.0/24 | self | 1 |

**Abb. 2.5 Routingtabellen zum Y-Szenario in Abb. 2.4.**

## 2.2.2. Split Horizon

Das Split-Horizon Verfahren, findet in vielen Distanz-Vektor-Protokollen Anwendung um die Entstehung einfacher Routing-Schleifen zu vermeiden. Der Kerngedanke ist, dass Informationen über die Erreichbarkeit eines gelernten Netzes nur über die Netzwerkinterfaces versendet werden, von denen die Netzinformation nicht gelernt wurde. An jedes Interface wird also die gesamte Routingtabelle abzüglich der Routen, die von eben diesem gelernt wurden gesendet.

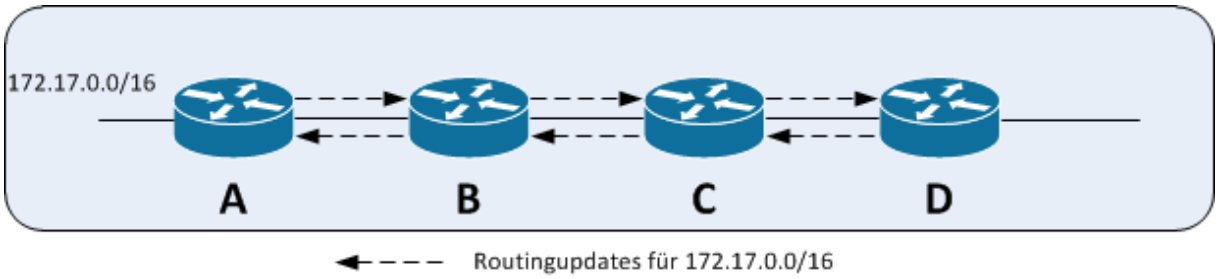



**Abb. 2.6 Verlauf der Routingupdates ohne Split Horizon**

Ohne Split-Horizon verteilen sich die Routingupdates für 172.17.0.0/16 wie in der Abbildung 2.6. Router A bekommt seine eigene Route von Router B erneut angeboten, allerdings mit einer um eins erhöhten Metrik und verwirft sie deshalb. Dies stellt in einem fehlerfreien Netzzustand  abgesehen vom minimal erhöhten Routingtraffic auch kein Problem dar.

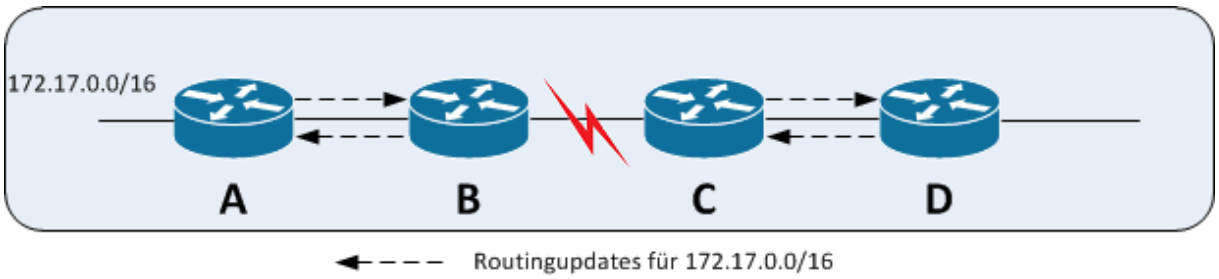



**Abb. 2.7 Verlauf der Routingupdates ohne Spilt Horizon und vorhandener Verbindungsunterbrechung**

Im Szenario in Abbildung 2.7 kam es nun zu einer Routenunterbrechung zwischen Router B und C, sodass das Netz 172.17.0.0/16 für die Router C und D nicht mehr erreichbar ist. Sobald es nun bei Router C zum Timeout des Routingeintrags gekommen ist, wird Router C ein Routingupdate von Router D empfangen und seinen eigenen Routeneintrag mit der Metrik 16 für 172.17.0.0/16  mit dem von D angebotenen, mit der Metrik 4 überschreiben. Router D erhöht die Metrik für das Netz beim nächsten empfangenen Routingupdate ebenfalls um 1. Diese Routingschleife löst sich erst auf, wenn die Metriken beider Router den Wert 16 erreicht haben.

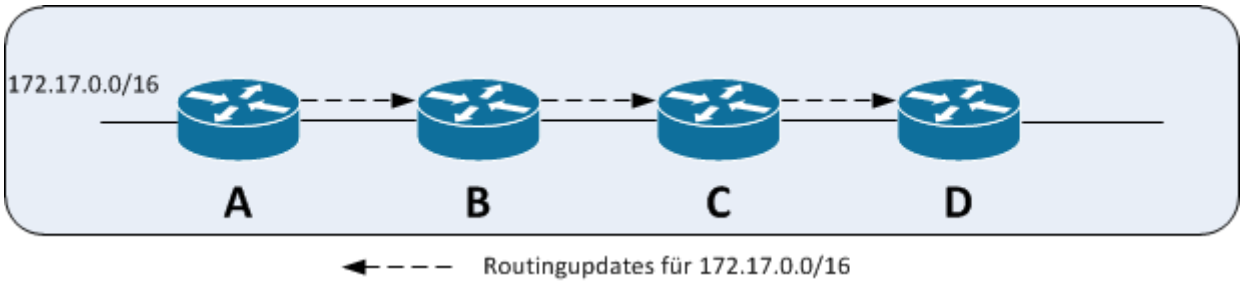



**Abb. 2.8 Verlauf der Routingupdate mit aktiviertem Split Horizon**

In der Abbildung 2.8 ist nun das selbe Szenario mit aktivertem Split Horizon zu sehen, die Routingupdates für 172.17.0.0/16 werden nur noch über das Interface geschickt, von dem das Prefix nicht gelernt wurde.

Triviale Routingschleifen können somit nicht mehr entstehen. Zum Split-Horizon existiert die Erweiterung "Split Horizon with Poison Reverse". Hier wird, bei aktiviertem Route Poisoning eine ausgefallene Route entgegen dem Split-Horizon-Verfahren auch auf dem Interface mit Metrik 16 veröffentlicht von dem sie ursprünglich gelernt wurde.

## 2.2.3. Counting to Infinity

Beim Counting to Infinity Problem handelt es sich um die Weitergabe von veralteten Routinginformationen in Netzwerkschleifen. Dadurch wird die Information über die nicht Erreichbarkeit eines Netzes immer wieder mit veralteten Informationen der Nachbarn überschrieben, welche mit einer besseren Metrik im Vergleich zu RIP-Infinity eine vermeintliche Alternative bieten.

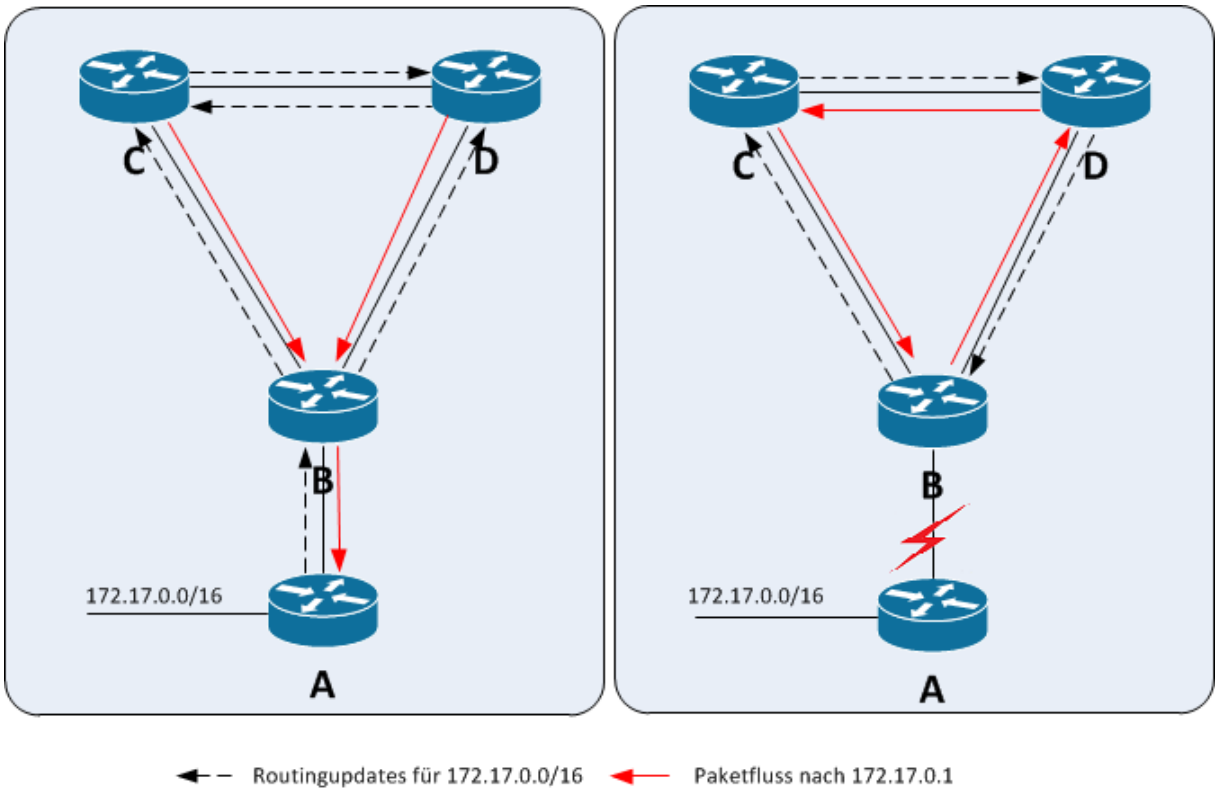



**Abb. 2.9 Counting to Infinity in Y-Topologie**

In der Abbildung 2.9 ist auf der linken Seite eine intakte Y-Topologie zu sehen und die dem Netz 172.17.0.0/16 zugehörigen Routingupdates bei eingeschaltetem Split Horizon, sowie Paketflüsse der Router B, C und D an einen Host im Netz 172.17.0.0/16. Nach kurzer Zeit stellt sich der in der rechten Tabelle abgebildete Zustand in den Routingtabellen für das relevante Netz ein.

| Router | Next-Hop | Metrik |
|---|---|---|
| A | self | 1 |
| B | A | 2 |
| C | B | 3 |
| D | B | 3 |

**Abb. 2.10 Blick der unterschiedlichen Router auf 172.17.0.0/16**

Im rechten Teil der Abbildung 2.9 kommt es nun zu einem Verbindungsausfall zwischen Router B und A, was zu einer Unerreichbarkeit des Netzes 172.17.0.0/16 für die Router B,C und D führt (t0 Abbildung 2.11). Je nachdem in welcher zeitlichen Abfolge nun die Routingupdates eintreffen, kann es innerhalb der Schleife zwischen B, C und D zum Hochzählen der Metrik bis 16 kommen.

| Router/Zeit | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t... | t13 | t14 | t15 | t16 | t17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 16 | 16 | 16 | 16 | 5 | 5 | ... | 11 | 14 | 14 | 14 | 16 |
| B | 2 | 16 | 16 | 16 | 5 | 5 | 5 | ... | 14 | 14 | 14 | 16 | 16 |
| C | 3 | 3 | 3 | 16 | 16 | 6 | 6 | ... | 12 | 15 | 15 | 15 | 16 |
| D | 3 | 3 | 16 | 4 | 4 | 4 | 7 | .. | 13 | 13 | 16 | 16 | 16 |

**Abb. 2.11 Zeitlinie eines ablaufenden CTI**

Die Tabelle in Abbildung 2.11 zeigt den jeweiligen Zustand der Routingtabellen mit Sicht auf das Netz 172.17.0.0/16 an. Das Hochzählen der Metrik ist farblich hervorgehoben. Nachdem der Timeout für die Route nach 172.17.0.0/16 beim Router B abgelaufen ist, setzt dieser die Metrik für dieses Subnetz auf 16 und benachrichtigt die Router C und D vom Routenausfall, die wiederrum ihre Metrik für das Netz auf 16 ändern (t2 Abbildung 2.11).
In unserem Szenario hat nun jedoch Router C ein periodisches Routingupdate an Router D gesendet, kurz bevor Router C selbst Kenntnis vom Ausfall des Netzes 172.17.0.0/16 erlangte. Dieses Routingupdate mit einer mittlerweile veralteten Metrikinformation trifft nun bei D ein und D ersetzt seine bisherige Route mit Metrik 16 mit der Fehlinformation von Router C und setzt C als Next-Hop und die Metrik auf 3. Das Counting to Infinity beginnt an dieser Stelle (t3 Abbildung 2.11). Router D propagiert diese weiter an Router B, welcher D als Next-Hop einträgt und die Metrik auf 4 erhöht. Das Routingupdate von Router B an C ersetzt den letzten RIP-Infinity Eintrag im Netz und nun haben alle drei an der Schleife beteiligten Router die Information das Netz 172.17.0.0/16 sei nach wie vor erreichbar (t5 Abbildung 2.11).

Router C sendet nun das nächste Update an D, dieses kommt jedoch mit der Metrik 5 bei D an, welcher diese übernimmt, da die ursprüngliche Route ebenfalls von C gelernt wurde. Das Hochzählen der Metrik innerhalb der Schleife endet erst, wenn alle drei Router erneut die Metrik 16 für das Netz 172.17.0.0/16 angenommen haben (t17 Abbildung 2.11). Nach dem Ablauf des Timeouts, der mit Metrik 16 einsetzt, wird erneut ein konvergenter Zustand erreicht.

Während des Bestehens der Routingschleife liefen alle Pakete mit dem Ziel 172.17.0.0/16 ebenfalls im Kreis, sodass der Traffic in diesem Netzsegment unter Umständen stark ansteigt und kein Datenpaket das Ziel 172.17.0.0/16 erreicht hat.

## 2.2.4. Triggered Updates

In der ursprünglichen RIP-Implementation versendet ein Router standardmäßig alle 30 Sekunden seine gesamte Routingtabelle an seine Nachbarn. Dies hat den Nachteil, dass sich neue Routinginformationen nur langsam im Netz ausbreiten. Um dies zu umgehen wurde bereits in RIPv1 Triggered Updates implementiert, die die Nachbarrouter sofort über Metrikänderungen in Kenntnis setzen sollen. Dies löst meist eine Änderungskaskade im entsprechenden Netz aus, da alle Router die ebenfalls von der Änderung betroffen sind ihrerseits ein Update versenden.

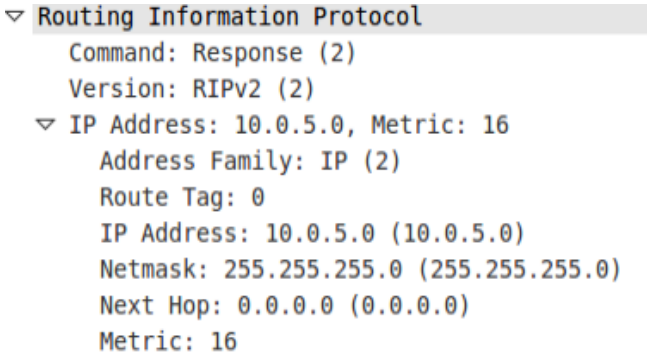



**Abb. 2.12 Wiresharkmitschnitt eines Triggered Updates**

Dadurch gibt es einen kurzzeitigen Anstieg des Routingtraffics, der positive Effekt, dass das Netz schneller konvergiert und die Entstehung von Routingschleifen dadurch häufig vermieden werden können überwiegt jedoch.

## 2.2.5. Erweiterung RMTI

Die Abkürzung RMTI, steht für *Routing with Metric based Topology Investigation* und wurde von Andreas Schmid im Rahmen seiner 1999 veröffentlichten Diplomarbeit an der Universität Koblenz als RIP-MTI vorgestellt [Schmid99]. In den letzten Jahren, war der RMTI Gegenstand vieler Untersuchungen bezüglich Effizienz und Korrektheit der Schleifenerkennung, sodass im Laufe der Zeit etliche Anpassungen der Ursprungsversion von Andreas Schmid erfolgten. Der RMTI ist eine abwärtskompatible Erweiterung der ursprünglichen RIPv2-Implementation und kann in Topologien eingesetzt werden, die aus Routern der normalen RIP-Version bestehen, da das RIP-Nachrichtenformat beibehalten wird. Ein RMTI-Router sammelt Informationen über die Netzwerkschleifen, an denen er selbst beteiligt ist. Dies geschieht über die Analyse der eintreffenden Routingupdates, die aufgrund einer schlechteren Metrik bei der konventionellen RIP-Implementation unbeachtet bleiben. Sollte es nun zu einem Routenausfall kommen, ist der RMTI in der Lage zu entscheiden, ob eine angebotene Alternativroute auf veralteten Informationen basiert, die zum CTI-Problem führen würden oder es sich um eine gültige Alternative handelt. Dadurch ist der RMTI in der Lage, das CTI-Problem zuverlässig zu verhindern. Um topologische Schleifen zu erkennen, analysiert der RMTI die eigentlich zu verwerfenden Routingupdates darauf, ob es ein Alternativangebot für ein bereits in der Routingtabelle vorhandenes Netz gibt, welches über ein anderes Netzwerkinterface erreicht werden kann.

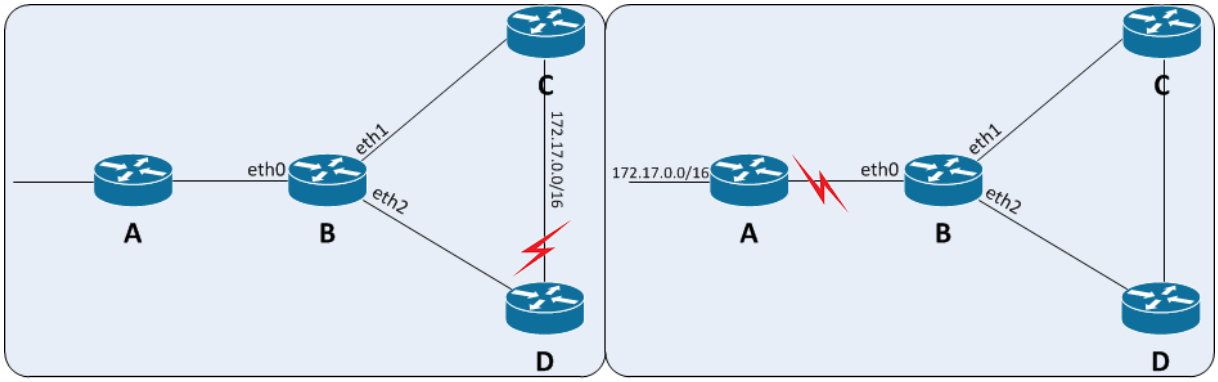



**Abb. 2.13 Verbindungsunterbrechung in Y-Topologie**

Auf der linken Seite von Abbildung 2.13 ist ein Netzwerkszenario beschrieben, in welchem das Netz 172.17.0.0/16 von Router B sowohl über das Interface eth1 als auch über eth2 erreicht werden kann. Kommt es nun zur eingezeichneten Verbindungsunterbrechung zwischen Router D und C, würde ein RIP-Router vom bisher benutzten Interface eth2 auf das Interface eth1 umschwenken, was in diesem Fall dem erwünschten Verhalten entspricht.
In der rechten Abbildung ist 172.17.0.0/16 für Router B nur über eth0 erreichbar, trotzdem stellt sich beim RIP-Protokoll der Sachverhalt genauso dar wie im Szenario links. Wenn es nun zum Verbindungsabbruch kommt, kann es beim zeitlich ungünstigen Erhalt von Routingupdates dazu kommen, dass Router B Alternativrouten über eth1 oder eth2 nach 172.17.0.0/16 bekommt, die aufgrund ihrer Metrik vermeintlich attraktiver erscheinen, als die RIP-Infinity Route über Interface eth0 (siehe Abbildung 2.11). In diesem Fall würde es bei RIP nun zum unerwünschten Counting to Infinity kommen, welches der RMTI, durch das vorherige erkennen der existierenden Netzwerkschleife verhindern könnte. Anhand der vom RMTI gesammelten Informationen über bestehende Netzwerkschleifen, dem Interface, über welches das Routingupdate eintrifft, sowie über die Metrik zum Zielnetz kann der RMTI

erkennen, ob es sich um eine gültige Alternativroute handelt oder die Übernahme der Information zum CTI-Problem führen würde.
In Abbildung 2.14 ist eine sogenannte Simple-Loop abgebildet, es handelt sich um eine Netzwerkschleife, die dadurch charakterisiert ist, dass es einen Pfad von einem Interface des Routers zu einem anderen Interface des Routers gibt, ohne diesen selbst zu passieren. Zur Identifikation von Simple Loops, achtet der RMTI auf Routenvorschläge für das selbe Subnetz, die an unterschiedlichen Interfaces ankommen. Somit existiert eine Simple Loop immer in Bezug auf ein Subnetz und hat eine maximale Metrik von 30(RMTI-Infinity = 31), welche sich aus 2 * RIP-Infinity - 2 zusammensetzt. Für die Schleifenerkennung zwischen zwei Interfaces benötigt der RMTI nur die Simple Loops mit der geringsten Metrik, die längeren Schleifen werden nicht benötigt. Jeder RMTI-Router speichert zwei Tabellen, die MSILM-Tabelle, sowie die MRPM-Tabelle, welche zur Bewertung von Alternativrouten heran gezogen werden.

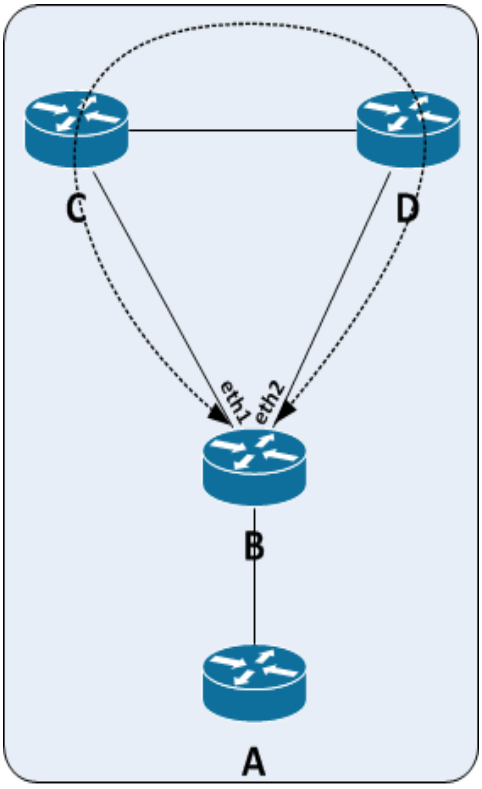



**Abb. 2.14 Simple Loop zwischen eth1 und eth2**

Die Minimal-Simple-Loop-Metric (MSILM) beschreibt die Metrik der kleinsten Simple Loop zwischen zwei spezifischen Interfaces eines Routers. Hat ein Router beispielweise drei Netzwerkinterfaces, so hat die MSILM-Tabelle sechs Einträge, jeweils einen von jedem Interface zu jedem anderen Interface. Initialisiert wird der MSILM-Wert mit 31, was bedeutet, dass kein Simple Loop zwischen den beiden Interfaces existiert. Außerdem wird für jedes Interface ein Wert namens Minimal-Return-Path-Metric (MRPM) vorgehalten, dieser beschreibt den kürzesten Simple Loop eines Interfaces zum Router zurück. Der MRPM-Wert für ein Interface ist also das Minimum aller MSILM-Werte des Interfaces.

Bei einem Routenausfall zu einem Subnetz, kann eine gültige Alternativroute nur von einem Interface kommen, welches über eine Simple Loop mit dem Interface der alten Route verbunden ist, ansonsten handelt es sich um eine alte Routinginformation des Routers selbst, welche an ihn zurück gesendet wurde und zum CTI führen würde. In der ursprünglichen RMTI-Version, die im Rahmen der Diplomarbeit von Andreas Schmid entstand, musste jede vorgeschlagene Alternativroute folgende Ungleichung erfüllen, um als neue valide Route in die Routingtabelle aufgenommen zu werden :

$$m(N) + m(A) - 1 > msilm(N,A)$$

Bei m(N) handelt es sich um die angebotene Metrik ins Subnetz über das neue Interface. Analog dazu bezeichnet m(A) die Metrik für die Verbindung über das alte Interface zum Subnetz. Und bei msilm(N,A) handelt es sich um den in der MSILM-Tabelle hinterlegten minimalen Simple-Loop zwischen altem und neuem Interface. Es wird also überprüft, ob die Routenkombination aus alter und neuer Route größer ist als die kleinste Schleife zwischen beiden Interfaces, nur dann kann es sich um eine korrekte Alternative handeln, deren Annahme nicht zum Counting to Infinity führt.

Sollte es keine Schleife zwischen beiden Interfaces geben, handelt es sich bei msilm(N,A) um den Initialwert von 31, somit ist bei einer maximal gültigen Metrik von 15 keine Möglichkeit gegeben, die Ungleichung bei fehlender Schleife zu erfüllen. Das bis hierhin vorgestellte Verfahren, wird heute als *RMTI - Normal Mode* bezeichnet. Tiefergehende Untersuchungen in komplexen, schleifenbehafteten Netztopologien offenbarten jedoch, dass der *RMTI - Normal Mode* nicht immer in der Lage ist ein Counting to Infinity zu verhindern. Um dem CTI-Problem zu begegnen schlug Thomas Kleemann in seiner Diplomarbeit [Klee01] eine Erweiterung des RMTI um den sogenannten *Strict Mode* vor, der es zum Ziel hat, auch die damals noch auftretenden CTIs in verschachtelten Schleifen zu verhindern.

Source-Loops werden vom RMTI in zwei mögliche Kategorien eingeteilt. Dies sind die Y-Kombination und die X-Kombination. Beide Arten haben die Gemeinsamkeit, dass die Routinginformation vom Router bereits mindestens einmal verarbeitet wurde, was für den RIP-Algorithmus jedoch nicht mehr nachvollziehbar ist.
In der Abbildung X ist die Entstehung einer Y-Kombination zu sehen. Router B lernt die Route zum Netz 172.17.0.0/24 von Router A und gibt diese an C und D weiter. Anschließend kommt es zum Verbindungsausfall zwischen A und B und Router B kennzeichnet 172.17.0.0/24 mit Metrik 16 und gibt diese Information an C und D weiter, welche beide die RIP-Infinity-Metrik übernehmen. Zur gleichen Zeit war jedoch noch ein Routingupdate von C nach D unterwegs, welches mit einer veralteten Metrik bei Router D eintrift, welcher bereits den Wert 16 hinterlegt hat. Router D übernimmt nun den fehlerhaften Wert und sendet ihn an Router B weiter. Von Router B hängt es nun ab ob es zum Counting to Infinity kommt. Je nach dem, ob er den fehlerhaften, in einer Netzwerkschleife entstanden, Wert als solchen identifizieren kann oder nicht.
Router B wird durch die Position, die er in der Y-Kombintion einnimmt auch als Source-Router bezeichnet. Der Source-Router ist der Router in einer Netzwerkschleife, der dem ausgefallen Subnetz am nächsten liegt und im gegebenen Beispiel der einzige Router, auf dem der RMTI aktiv sein muss, um die sonst entstehende Routingschleife zu verhindern. Zur Verhinderung von Y-Kombinationen verwendet der RMTI den Y-Test.

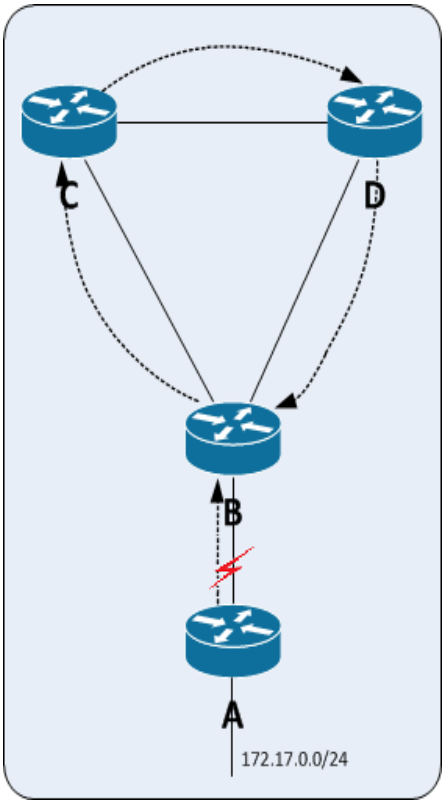


**Abb. 2.15 172.17.0.0/24 Counting to Infinity**

Zur Berechnung, ob eine angebotene Route in ein ausgefallenes Zielnetz eine Y-Kombination oder eine gültige Alternative ist, wird folgende Ungleichung heran gezogen :

$$(1) \quad mrpm(N) + m(A) > m(N)$$

wobei gilt :
- *mrpm(N)* ist die kleinste erkannte Simple-Loop für das Interface über welches die neue Routinginformation zum Zielnetz empfangen wurde
- *m(A)* ist die Metrik zum Zielnetz über das alte Interface, also die Metrik der ausgefallenen Route
- *m(N)* ist die Metrik zum Zielnetz über das neue Interface, also die Metrik des Routenvorschlags

Die Ungleichung ist so zu verstehen, dass geprüft wird, ob die neue Metrik geringer ist als die Metrik der kleinsten möglichen Y-Kombination für das neue Interface.

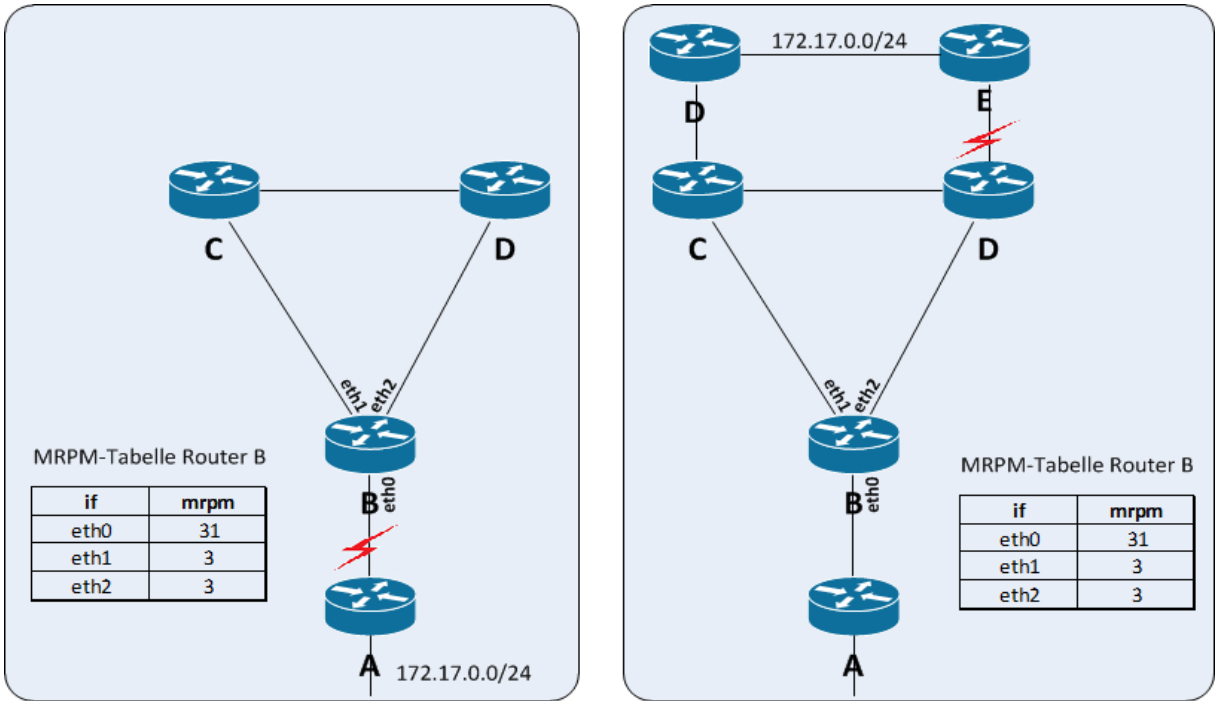



**Abb. 2.16 Test auf vorhandene Y-Kombination**

In Abbildung 2.16 ist auf der linken Seite eine Y-Topologie zu sehen. Das Subnetz 172.17.0.0/24 war bis zur Verbindungsunterbrechung für Router B mit einer Metrik von 2 über eth0 zu erreichen. Der kleinste Simple-Loop für die Interfaces eth1 und eth2 hat die Metrik 3. Sollten nach der Verbindungsunterbrechung nun veraltete Informationen im Simple-Loop zirkulieren, haben diese mindestens die Metrik 5, was bedeutet die Ungleichung ist nicht erfüllt.

$$\mathbf{3}(mrpm\_N) + \mathbf{2}(m\_A) > \mathbf{5}(m\_N) \quad \textbf{falsch}$$

Auf der rechten Seite von Abbildung 2.16 benutzte der Router B bis zur Verbindungs-unterbrechung zum Zielnetz 172.17.0.0/24 den Pfad B→D→E über eth2 mit der Metrik 3. Die Interfaces eth1 und eth2 befindet sich in einer Simple-Loop-Konstellation mit der Metrik 3. Nach dem Verbindungsbruch zwischen E und D bekommt der Router B nun eine Alternativroute mit dem Pfad B→C→D über eth1mit Metrik 3 angeboten und stellt folgende Berechnung an.

$$\mathbf{3}(mrpm\_N) + \mathbf{3}(m\_A) > \mathbf{3}(m\_N) \quad \textbf{wahr}$$

Die Angebotene Route wird somit als neue Alternative akzeptiert, da es sich nicht um eine Y-Kombination handeln kann.
Die zweite vom RMTI untersuchte Routenkombination ist die X-Kombination, die in Abbildung X dargestellt ist. Bei der X-Kombination handelt es sich um zwei über einen zentralen Router miteinander verbundene Simple-Loops.

Wie in der Abbildung 2.17 zu sehen, erhält der Source-Router B zum Zeitpunkt T1 ein Routingupdate mit der Erreichbarkeitsinformation für 172.17.0.0/24 von Router A. Dieses leitet er im folgenden Update an die Router C und D (T2). Nun kommt es zum Ausfall von 172.17.0.0/24 und Router B übernimmt die Metrik 16 und schickt ein Update über den Ausfall an Router C und D. Kurz vor dem Eintreffen des Routingupdates, schickt Router D jedoch noch ein Update basierend auf veralteten Informationen an Router C, welcher die Information in seine Routingtabelle einträgt, da dort für 172.17.0.0/24 mittlerweile Metrik 16 vorgehalten war (T3). In T4 empfängt der Source-Router B eine veraltete Routeninformation von Router E und trägt diese in seine Routingtabelle ein. Kurz darauf registriert allerdings Router E ebenfalls den Ausfall und sendet die Metrik 16 an B, welcher diese übernimmt. In T5 sendet Router C die mit veralteter Metrik ausgestattete Routinginformation nun an B, welche sogleich übernommen wird, woraufhin ein Couting to Infinity in der Schleife B→D→C→B einsetzt.

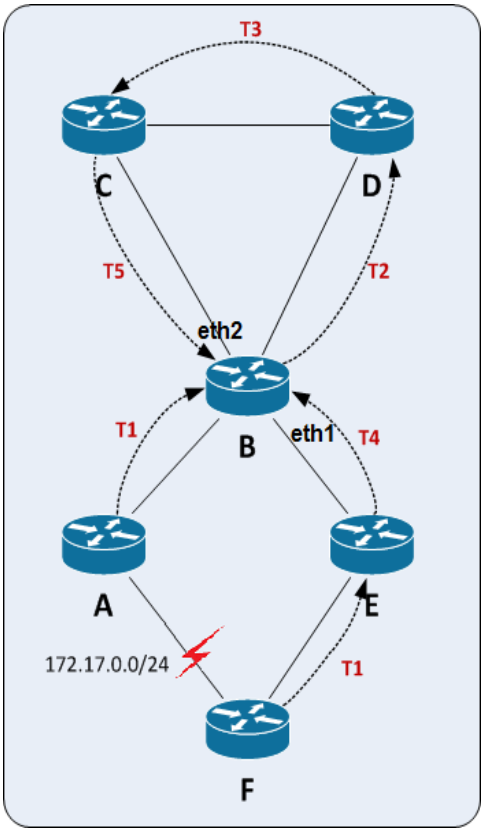


**Abb. 2.17 X-Kombination**

Die X-Kombination unterscheidet sich dadurch von der Y-Kombination, dass es sich hier nicht nur um die Übernahme von veralteten Informationen des Source-Routers handelt, sondern das kurzzeitig Fehlinformationen aus einer anderen Netzwerkschleife ebenfalls in den Routingprozess hineinspielen. Um dem entgegen zu wirken, verwendet der RMTI den X-Test.
Beim X-Test wird im Vergleich zum Y-Test ebenfalls die minimale Simple-Loop des Ursprungsinterfaces in die Ungleichung einbezogen.

$$(2) \quad mrpm(N) + mrpm(A) > m(N) + m(A) - 1$$

In Abbildung 2.17 beträgt der MRMP-Wert für das Interface eth1 4, die Simple-Loop kommt hier über den Pfad B→E→F→A→B zustande. Für das Interface eth2, an dem die neue Routinginformation ankommt, beträgt der Wert in der MRPM-Tabelle den Wert 3. Die fehlerhafte Erreichbarkeitsinformation für 172.17.0.0/24, die zum Zeitpunkt T4 bei eth2 eintraf hat eine Metrik von 3. Das zum Zeitpunkt T5 bei eth2 eintreffende Routingupdate, weißt den Weg nach 172.17.0.0/24 mit einer Metrik von 5 aus. Die Berechnung des X-Tests läuft nun wie folgt ab :

$$\mathbf{3}\,(mrpm\_N) + \mathbf{4}(mrpm\_A) > \mathbf{5}(m\_N) + \mathbf{3}(m\_A) - \mathbf{1} \quad \textbf{falsch}$$

Somit wird die X-Kombination erkannt und die angebotene Route verworfen.

Die bis hierhin vorgestellte Version des *RMTI - Strict Mode* ist in der Lage durch X-Test, Y-Test sowie das aktivierte Split Horizon Verfahren Source-Loops in allen bisher getesteten Netztopologien wirkungsvoll zu erkennen und somit das Counting to Infinity Problem zu verhindern. Wie von Frank Bohdanowicz in seiner Diplomarbeit [Bohdan08] gezeigt wurde, sind sowohl Y- als auch X-Test in bestimmten Netzwerkszenarien zu streng ausgelegt, was zur Folge hat, dass es dazu kommt, das gültige Routen wiederholt abgelehnt werden, bis die ausgefallene Route aus dem Speicher des Routers entfernt wurde und somit nicht mehr zur

Bewertung der Routingupdates heran gezogen werden kann. Dies führt in den entsprechenden Szenarien zu sehr schlechten Konvergenzzeiten, weshalb es eine erneute Überarbeitung des RMTI-Algorithmus gab. Nach [Bohdan08] kann ein verzögerter Y-Test den X-Test vollständig ersetzen, somit kommt es zu weniger abgelehnten validen Routen. Dies wird erreicht, indem der kleinste Metrikwert für ein ausgefallenes Netz für eine gewisse Dauer gespeichert wird und somit ein Überschreiben des Wertes durch einen anderen Router, wie es bei der X-Kombination passieren kann keine Auswirkung mehr auf den MTI-Test hat.

Der *RMTI - Careful Mode* stellt eine Weiterentwicklung des *Strict Mode* dar. Er soll die Effizienz bei der Schleifenerkennung des *Strict Modes* beibehalten, ohne dabei jedoch die schlechten Konvergenzeigenschaften dessen zu übernehmen. Die im Folgenden vorgestellten Versionen des *Careful Mode* basieren alle auf dem *Strict Mode.* Gemeinsam ist ihnen, dass sie nachdem ein alternativer Routenvorschlag zu einer ausgefallenen Route den X/Y-Test nicht besteht, die ausgefallene Route in der Routingtabelle markiert wird und ein Routingupdate mit RIP-Infinity für die ausgefallene Route an die Nachbarn ausgesendet wird. Ziel davon ist es, eine eventuell bestehende Source-Loop bei den Nachbarn aufzulösen. Ab hier unterscheiden sich die drei entwickelten Careful-Methoden.

**RMTI-Careful-DT (Deny Timer)**

Der Careful Mode mit Deny Timer, speichert eine durch X/Y-Test abgelehnte Route temporär und übernimmt diese in die Routingtabelle, wenn in einem festgelegten Zeitintervall kein Routingupdate mit Metrik RIP-Infinity von seinen direkten Nachbarn für diese Route eintrifft.

**RMTI-Careful-RT (Request Timer)**

Bei diesem Careful Mode handelt es sich um eine Weiterentwicklung des *DT-Mode*. Hier wird die Route die den X/Y-Test nicht besteht, nicht im Router zwischengehalten, sondern zuerst abgelehnt. Nach einem definierten Zeitintervall wird die Route von den Nachbarroutern angefordert und sofern sie nicht mit Metrik RIP-Infinity gekennzeichnet ist übernommen.

**RMTI-Careful-ESHC (External Split Horizon Check)**

Der letzte Careful Mode übernimmt eine neue Routinginformation für eine ausgefallene Route erst dann, wenn an jedem Interface, welches eine Schleife zum Eingangsinterface des Routingupdates hat, ebenfalls eine neue Routeninforation ins Subnetz erhalten wurde. Hintergedanke bei diesem Verfahren ist, dass bei Nicherhalt des Updates an einem mit Schleife verbundenen Interface der Split Horizon eines der Nachbarrouter das Zurücksenden der Routinginformation an den Router selbst unterdrückt haben muss, was die Existenz einer Source-Loop bedeutet.

In [Bohdan2008] wird die Verwendung des *RMTI-Careful-RT* Modus empfohlen, da es sich beim *Careful-ESHC* um ein Verfahren handelt, das speziell bei größeren Netzausfällen zu einer starken Speicherbelastung und stark erhöhter Rechenaktivität im Router führt. Denn für jedes ausgefallene Netz muss eine Interfacetabelle organisiert und vorgehalten werden. Vom Resourcenproblem ist, wenn auch in geringerem Umfang ebenfalls der *Careful-DT-Mode* betroffen, was den *RMTI-Careful-RT* als den attraktivsten Modus erscheinen lässt.

## 2.2.6. Routenaggregation

Als Routenaggregation wird die Zusammenlegung mehrerer nebeneinander liegender Netzbereiche zu einem einzigen Größeren bezeichnet. Häufig verwendete Begriffe für das selbe Verfahren sind auch supernetting oder route summarization. Der Vorteil beim Aggregieren von Subnetzen ist, das Einträge in den Routingtabellen eingespart werden können, was dazu führt, dass der Router weniger Speicherkapazität benötigt und ein Durchsuchen der Routingtabelle schneller durchgeführt werden kann. Durch das Zusammenlegen mehrerer Netze zu Einem, kann ebenfalls der Routingtraffic je nach Netztopologie drastisch reduziert werden, da in den Routingupdates nicht mehr alle Einzelrouten aufgeführt werden müssen.
Route Aggregation wurde zusammen mit dem Classless Inter Domain Routing eingeführt, welches seinerseits den Mangel an IPv4-Klasse-B-Netzen beheben sollte, was zeitgleich ein massives Anwachsen der Routingtabellen zur Folge hatte, welches nach wie vor anhält.

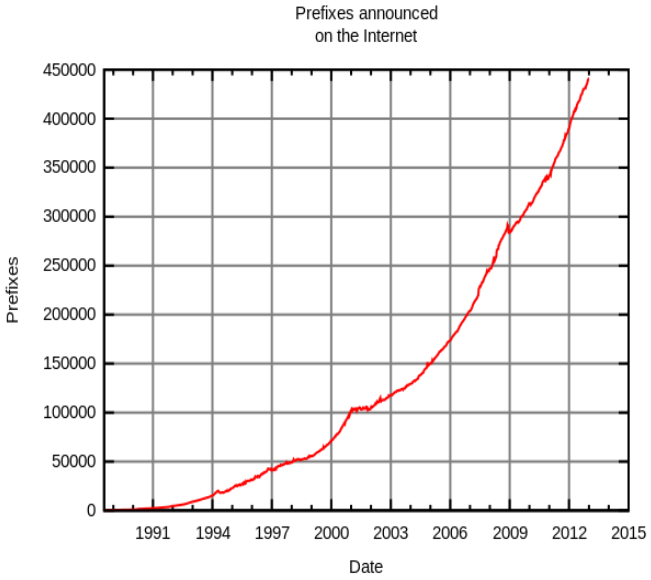
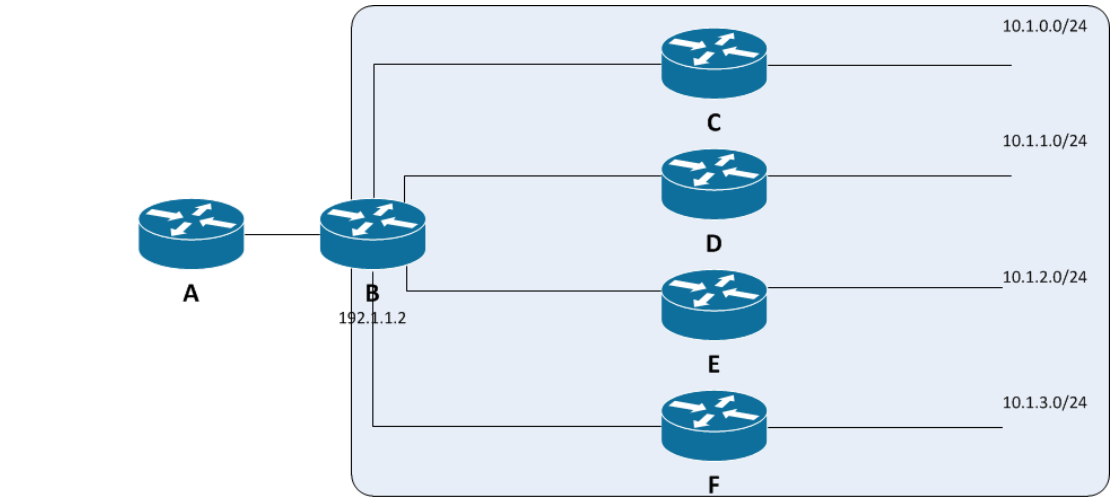



**Abb. 2.18 Anstieg der weltweit veröffentlichten Prefixe [wiki2]**

Diesem Problem sollte mit Route-Aggregation begegnet werden. Für Route-Aggregation existiert aktuell lediglich der [RFC1338], welcher nur das grundsätzliche Verhalten beim Aggregieren beschreibt, allerdings viele Detailfragen ungeklärt lässt. Um eine Routenaggregation vornehmen zu können, wird zwingend ein klassenloses Routingprotokoll, wie zum Beispiel OSPF oder RIPv2, benötigt, da bei der Zusammenlegung von Netzen der Netzanteil der Subnetzmaske verkleinert wird, um zwei oder mehr Netze in dem jetzt größeren Hostanteil unterbringen zu können.

Ein dynamisches Aggregationsverfahren ist beim Konzept der Link-State-Protokolle nicht realisierbar, da es keine Möglichkeit gibt die Link State Advertisements sinnvoll zu aggregieren, bei gleichzeitiger Aufrechterhaltung des Link-State-Konzepts. Eine Aggregation kann beispielweise bei OSPF somit nur an den Area Border Routern manuell durchgeführt werden. Bei Distanzvektorverfahren wie RIP, kann jedoch eine automatisierte Aggregation erfolgen, da die Router hier ihre kompletten Routingtabellen miteinander austauschen und nicht wie bei OSPF lediglich Nachbarschaftsinformationen. Im Folgenden werden einige Probleme die sich durch die konventionelle Art der Aggregation ergeben diskutiert.




| Ziel | Next-Hop |
|---|---|
| 10.1.0.0/24 | 192.1.1.2 |
| 10.1.1.0/24 | 192.1.1.2 |
| 10.1.2.0/24 | 192.1.1.2 |
| 10.1.3.0/24 | 192.1.1.2 |
| 192.1.1.0/24 | self |

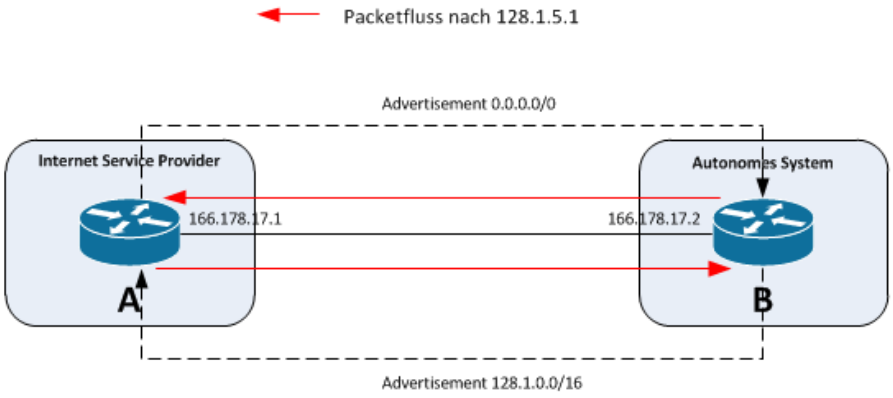
Routingtabelle von A mit aktivierter Routenaggregation bei Router B

| Ziel | Next-Hop |
|---|---|
| 10.1.0.0/22 | 192.1.1.2 |
| 192.1.1.0/24 | self |
| | |
| | |

Routingtabelle von A mit aktivierter Routenaggregation bei Router B

**Abb. 2.19 Beispieltopologie für Routenaggregation**

Sehr wichtig ist es, dass es sich bei den zu aggregierenden Netzen um nebeneinander liegende Netzbereiche handelt und diese das Aggregat vollständig ausfüllen. Ansonsten kommt es zu der Entstehung von Löchern in der Netztopologie. Diese Fehlinformationen verbreiten sich durch die Routingupdates daraufhin im gesamten Netz und führen zu Black Holes und Forwarding-Schleifen.




| Ziel | Next-Hop |
|---|---|
| 128.1.0.0/16 | 166.178.17.2 |
| ... | ... |
| ... | ... |
| ... | ... |

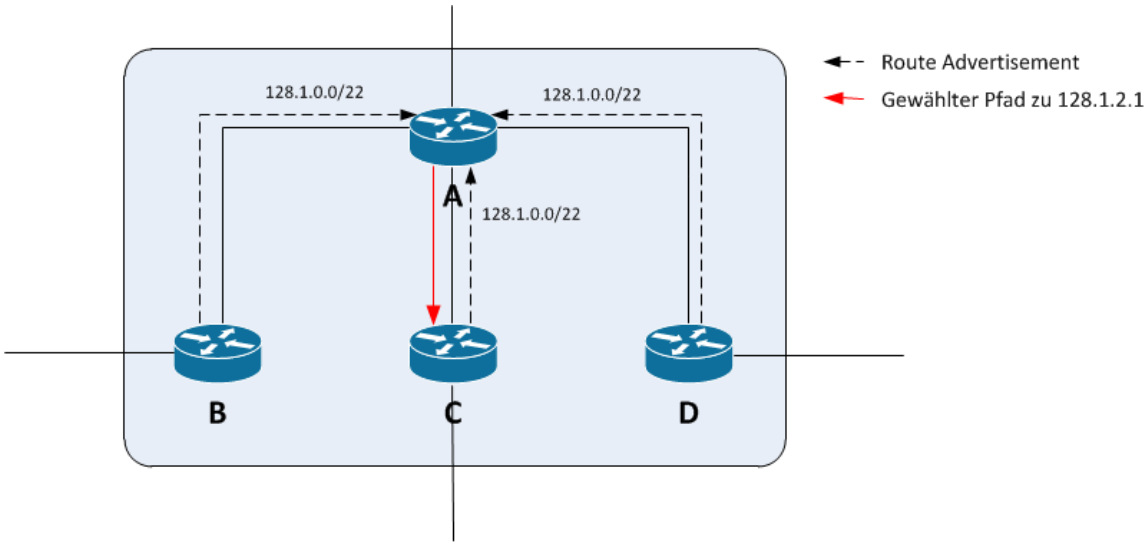
Routingtabelle A

| Ziel | Next-Hop |
|---|---|
| 128.1.1.0/24 | 10.0.1.1 |
| 128.1.2.0/24 | 10.0.1.1 |
| 128.1.4.0/24 | 10.0.1.2 |
| 0.0.0.0/0 | 166.178.17.1 |

Routingtabelle B

**Abb 2.20 Forwardingloop aufgrund zu großem Aggregat und Default-Gateway-Konfiguration**

In der Abbildung 2.20 ist ein mögliches Szenario für das Entstehen einer Forwarding-Loop durch Routenaggregation zu sehen. Router B fasst in seinem Routingupdate die drei Subnetze 128.1.1.0/24, 128.1.2.0/24 und 128.1.4.0/24 zu 128.1.0.0/16 zusammen. Der Router A des Internet Service Providers wiederrum propagiert sich als Default-Gateway für B. Im folgenden Szenario muss Router A nun ein an 128.1.5.1 adressiertes Paket zustellen. Bei der Auswertung der Routingtabelle wird erkannt, dass der Präfix der auf Router B als Next-Hop zeigt dem Gesuchten am nächsten kommt und das Paket wird deswegen an 166.178.17.2 weiter geleitet. Router B jedoch findet keinen passenden Routeneintrag in der eigenen Routingtabelle und sendet das Paket daher an seinen Default Gateway. Somit bekommt Router A erneut das Paket und sendet es wieder zu Router B. Diese Schleife läuft nun solange weiter, bis der Timeout für das Paket abgelaufen ist und es aufgrund dessen verworfen wird.
Das obige Szenario würde sich mit dem Eintragen einer Sink-Route in die Routingtabelle von Router B lösen lassen. Dies bedeutet, es wird die Route aus dem Routingupdate ebenfalls in die Routingtabelle eingetragen, allerdings zeigt der Next-Hop dabei auf ein Null-Device.
Dadurch werden die wirklich existierenden Netze nach wie vor erreicht, da der Longest-Prefix-Match Algorithmus dafür sorgt, dass aus der Routingtabelle immer der speziellste Eintrag als Next-Hop gewählt wird. In unserem Fall wäre dies also die Wahl eines /24- gegenüber eines /16-Netzes. Sollte die Sink Route die größte Übereinstimmung aufweisen, wird das Paket an ein Null-Device übergeben und damit verworfen.




| Ziel | Next-Hop |
|---|---|
| 128.1.0.0/24 | self |
| 128.1.1.0/24 | self |
| 128.1.2.0/24 | self |
| 128.1.3.0/24 | self |

Routingtabelle B

| Ziel | Next-Hop |
|---|---|
| 128.1.0.0/24 | self |
| 128.1.1.0/24 | self |
| 128.1.3.0/24 | self |
| | |

Routingtabelle C

| Ziel | Next-Hop |
|---|---|
| 128.1.1.0/24 | self |
| 128.1.3.0/24 | self |
| | |
| | |

Routingtabelle D

**Abb. 2.21 Entstehung eines Black Hole**

In der Abbildung 2.21 ist ein Szenario aufgezeigt, in dem es zu der Entstehung eines Black Holes kommt. Die Router B, C und D fassen ihre Routingtabelle jeweils als 128.1.0.0/22 zusammen. Allerdings füllt nur der Router B dieses Netz vollständig aus. Wenn Router A nun versucht ein Paket nach 128.1.2.1 zu übermitteln hat er drei scheinbar valide Möglichkeiten. Im Beispiel wird Router C gewählt und bekommt das Paket und stellt nach dem Durchsuchen der Routingtabelle fest, dass für 128.1.2.1 gar kein weiterführender Eintrag existiert und verwirft das Paket daher. Da Router A von dem Vorgang in Router C keinerlei Kenntnis hat, wird er auch zukünftige Pakete an Router C weiter leiten und somit wird das Netz 128.1.2.0/24 über Router A dauerhaft unerreichbar.

In der Praxis werden Router häufig mit mehreren gleichzeitig arbeitenden Routingprotokollen betrieben. Um zu entscheiden, welcher Eintrag letztlich in die Routingtabelle übernommen wird, bemühen handelsübliche Router den durch den Hersteller Cisco eingeführten, mit administrativer Distanz bezeichneten Wert. In Abbildung 2.22 sind die von Cisco definierten Standardwerte aufgeführt. Je kleiner die administrative Distanz umso glaubwürdiger wird die Routinginformation eingeschätzt und umso wahrscheinlicher in die Routingtabelle übernommen. Den direkt verbundenen und statisch gesetzten Routen wird gegenüber den durch dynamische Verfahren ermittelten Werten der Vorzug gewährt. Die verschiedenen Routingprotokolle wiederrum werden erneut in ihrer Glaubwürdigkeit unterteilt.

| Protokoll | administrative Distanz |
|---|---|
| Directly connected | 0 |
| Static route | 1 |
| EIGRP summary route | 5 |
| External BGP | 20 |
| Internal EIGRP | 90 |
| IGRP | 100 |
| OSPF | 110 |
| IS-IS | 115 |
| RIP | 120 |
| EGP | 140 |
| ODR | 160 |
| External EIGRP | 170 |
| Internal BGP | 200 |
| Unknown | 255 |

**Abb. 2.22 administrative Distanz [wiki3]**

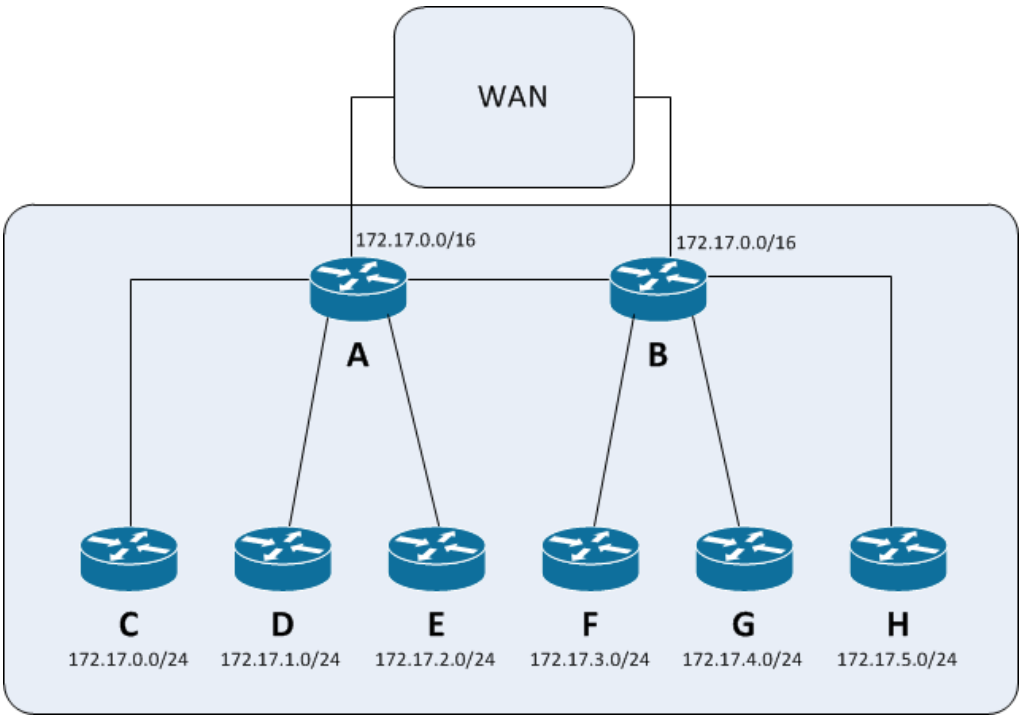



**Abb. 2.23 Topologie die Routenflimmern begünstigt**

In der Abbildung 2.23 ist ein Szenario zu sehen, welches in der Praxis zum sogenannten Routenflimmern führen kann.

Die Router A und B propagieren beide das Netz 172.17.0.0/16 und tragen gleichzeitig eine Sink-Route in eben dieses Netz in ihre Routingtabellen ein um zu vermeiden, dass ein Prefix aus 172.17.0.0/16, wie beispielsweise 172.17.10.0/24 zu einer Forwarding-Schleife führt. Je nachdem mit welcher administrativen Distanz die Sink-Route eingetragen wurde, kann es sein das Router A ein Routingupdate von B für 172.17.0.0/16 bekommt, welches die Sink-Route von A überschreibt, da die Administrative Distanz, basierend auf dem verwendeten Routingprotokoll kleiner ist als die der Sink-Route. Dieser Vorgang geschieht nun zeitgleich genauso im Router B, womit die Router A und B für 172.17.0.0/16 beide auf den jeweils anderen zeigen. Je nach Routerhersteller kommt es nun entweder zu einer dauerhaft bestehenden Forwarding-Schleife zwischen A und B, für alle nicht erreichbaren Prefixe aus 172.17.0.0/16 oder zu einem Counting to Infinity, wobei sich die Metriken für 172.17.0.0/16 zwischen A und B bei jedem Routingupdate um eins erhöhen, bis die maximale Metrik erreicht wird. Dann wird die nicht mehr verfügbare Route aus der Routingtabelle genommen und erneut die Sink-Route eingetragen, woraufhin sich das Counting to Infinity erneut wiederholt.

Während sich die erste vorgestellte Forwarding- Schleife noch auf nicht erreichbare Subnetze beschränkte, wurde durch [FrGeHu10] auch eine Forwarding-Schleife, welche die Erreichbarkeit existierender Subnetze verhindert entdeckt.

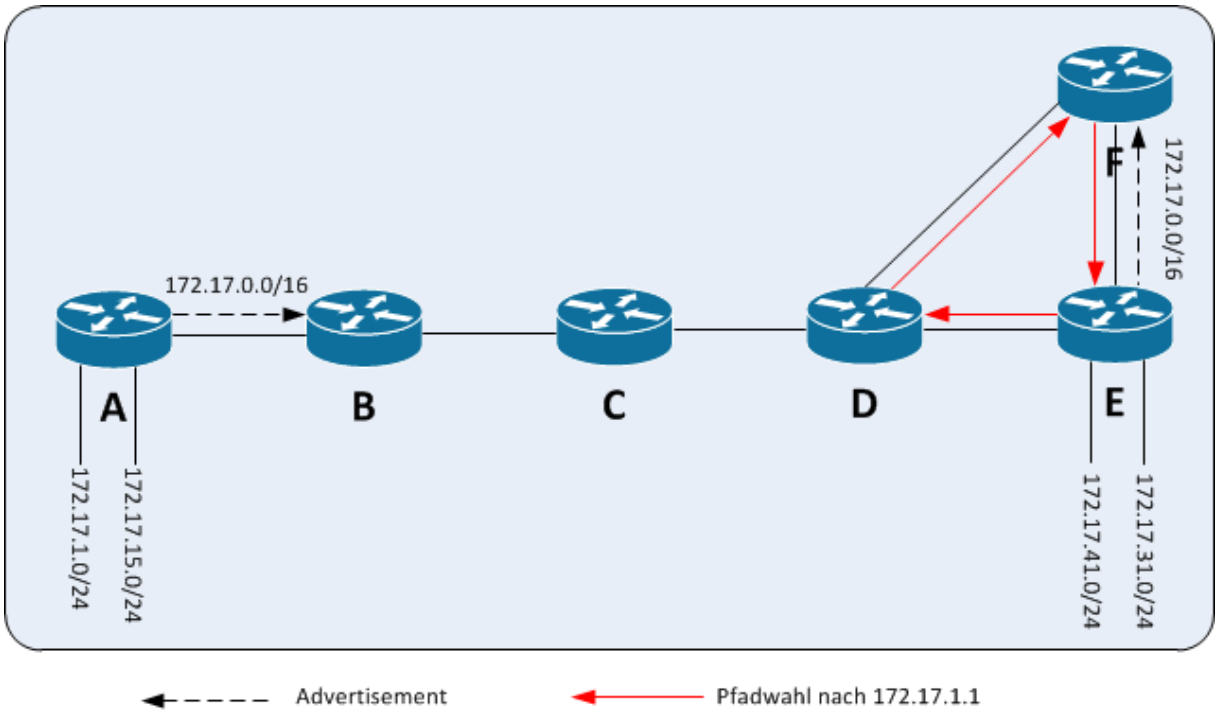



**Abb. 2.24 unerreichbare Prefixe aufgrund eines zu großen Aggregats**

Im in Abbildung 2.24 dargestellten Szenario fassen die Router A und E in ihren Routingupdates beide ihre vorhandenen Netze jeweils als 172.17.0.0/16 zusammen. Router A propagiert seine Routingtabelle nun an B und Router E seine zeitgleich an F. Router F schickt die Information wiederrum weiter an Router D, der das Aggregat in seine Routingtabelle übernimmt und an Router E weiter sendet. Da Router E selbst keinen Weg zu 172.17.0.0/16 kennt,wird 172.17.0.0/16 mit Next-Hop D in die Routingtabelle aufgenommen.

Wenn nun das Routingupdate von Router A bei Router D eintrifft, wird das Netz 172.17.0.0/16 von Router A verworfen, weil D bereits einen Eintrag dafür von Router F mit der Metrik zwei in der Routingtabelle hat und die Metrik des Routingupdates von A höher ausfällt. Das Netz befindet sich nun in einem konvergenten Zustand und die durch A angebundenen Prefixe sind für die Router D,E und F unerreichbar. Wenn D nun beispielsweise ein Paket an einen Host im Netz 172.17.15.0/24 sendet wird dieses zwischen D,E und F zirkulieren, bis der TimeToLive-Wert erreicht ist.
Wie dargestellt, sind die meisten entstehenden Routinganomalien darauf zurück zu führen, dass Aggregatrouten propagiert werden, die vom Router der das Routingupdate versendet nicht vollständig erreicht werden können. Trotz der beschriebenen Problematik wird dies in der Praxis so gehandhabt und einigen, der so entstehenden Anomalien versucht mit Sink-Routes zu begegnen.
Ob die naheliegendste Lösung, nämlich Aggregate nur zu propagieren, wenn sie vollständig vom jeweiligen Router erreichbar sind in der praktischen Umsetzung noch einen Vorteil bringt müssen nähere Untersuchungen an praxisnahen Netztopologien zeigen.

Um Routenaggregation außerhalb des Intradomain-Routings effektiv betreiben zu können, kommt der Zuweisung der IP-Adressblöcke an Provider und Organisationen, wie in [RFC4632] beschrieben, besondere Bedeutung zu. Die Serviceprovider bekommen einen größeren Adressbereich zugewiesen, den sie unterteilen und dann wiederum an ihre Kunden weiter geben. Der Serviceprovider muss lediglich seinen gesamten Adressblock im Netz propagieren, ohne auf die seinerseits vergeben Teilnetze einzugehen. Diese aktuelle, der Aggregation zuträgliche Handhabe kann allerdings zum Beispiel durch den Wechsel eines Kunden zu einem anderen Serviceprovider bei gleichzeitiger Mitnahme des benutzten Adressblocks unterlaufen werden. In so einem Fall wird der Provider weiterhin sein Aggregat propagieren und der neue Provider des Kunden ein Teilnetz dieses Aggregats, somit erreichen die Pakete aufgrund des Longest Prefix Match ihr Ziel, jedoch beinhalten die Routingtabellen dadurch wieder ein Prefix mehr. Um diesem Anwachsen entgegen zu wirken sollte eine Mitnahme von Adressräumen bei Providerwechsel unterbunden werden und der wechselnde Kunde muss innerhalb des Adressblocks des neuen Providers eingruppiert werden.

## 2.2.7. Patricia Trie

Bei Tries und im Speziellen Patricia Tries, handelt es sich um eine Datenstruktur, die häufig zum Speichern von Zeichenketten benutzt wird, da sich hier normalerweise mehrere Schlüssel einen gemeinsamen Prefix teilen. Deshalb findet diese Form der Datenhaltung in der Praxis besonders bei Autovervollständigungen und Rechtschreibprüfungen Einsatz.
In einem Trie befinden sich somit alle Vertreter mit dem selben Prefix innerhalb desselben Teilbaums, was Tries auch zur Datenhaltung in Routingtabellen, gerade im Hinblick auf Routenaggregation interessant macht. Im Gegensatz zu Suchbäumen speichern Tries keine Werte in den Knoten, sondern der Wert ergibt sich durch die bisher passierten Kanten. Wobei jede Kante mit einem Buchstaben des Alphabets beschriftet ist. Somit hat ein Knoten eines Tries für das Alphabet von A bis Z maximal 27 mögliche Kindknoten, 26 für die Buchstaben und ein mögliches Abschlusssymbol. Das Auffinden eines Wertes innerhalb eines Tries funktioniert ähnlich dem Prinzip longest prefix match, was ebenfalls in Routern zum Einsatz kommt um das Netz mit der speziellsten Netzmaske zu identifizieren.

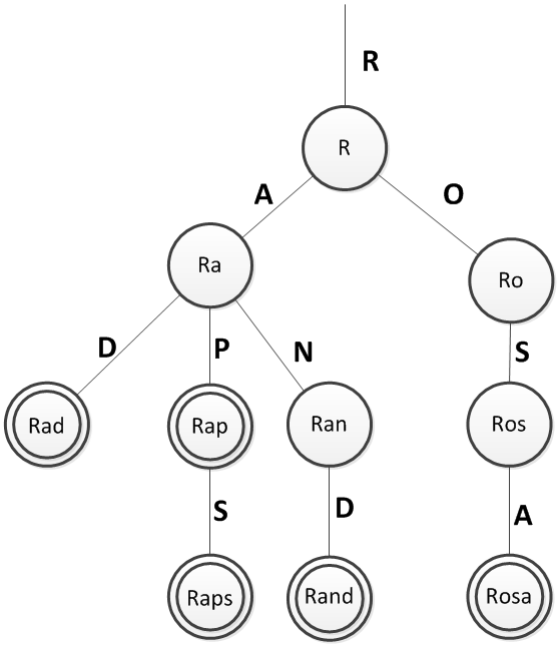



**Abb. 2.25 ein Trie mit enthaltenen Daten**

In Abbildungen 2.25 ist ein Trie dargestellt, in den die Worte *Rad*, *Rap*, *Raps*, *Rand* und *Rosa* aufgenommen wurden, die Bezeichnungen in den Knoten dienen nur dem Verständnis, existieren aber nicht in der praktischen Umsetzung. Wie zu sehen ist, sind Werte wie *Rosa* nicht speichereffizient hinterlegt, wodurch ebenfalls unnötige Vergleiche zum Auffinden des Schlüssels notwendig wären. Aus diesem Grund kam es zur Weiterentwicklung *Practical Algorithm to Retrieve Information Coded in Alphanumeric* kurz PATRICIA-Trie, bei der sich mehrere hintereinander liegende Knoten mit lediglich einem Kind zu einer einzelnen Kanteninformation zusammen fügen.

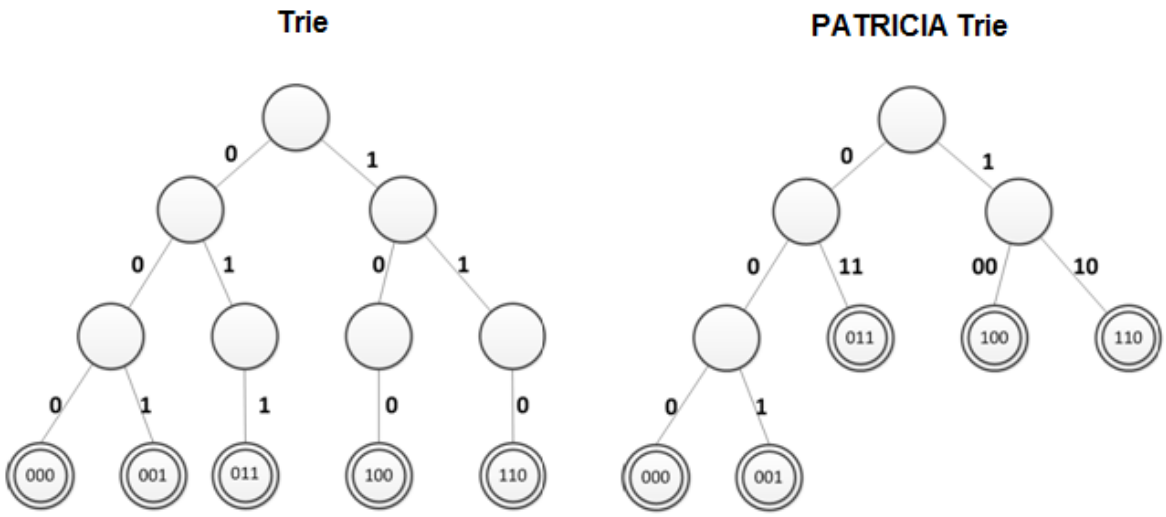



**Abb. 2.26 selber Datensatz als Trie / PATRICIA Trie**

Wie in der Abbildung 2.26 erkennbar ist, benötigt das Auffinden eines Schlüssels der Länge k innerhalb eines PATRCIA-Tries im schlechtesten Fall k Vergleiche, dies trifft ebenfalls auf das Einfügen und Löschen zu, die jeweils in *O*(k) liegen.
Somit sind PATRICIA-Tries eine ernsthafte Alternative zu balancierten binären Suchbäumen, deren Effizienz in Abhängigkeit zu der bereits im Baum vorhanden Schlüsselmenge n zu betrachten ist. In diesen Bäumen liegt der Aufwand zum Auffinden eines Schlüssels bei *O*(log n), was im ersten Moment als Vorteil erscheint, jedoch findet hier bei jedem Vergleich ein Abgleich mit dem gesamten Schlüssel der Länge k statt, bis eine Abweichung gefunden wird. Je tiefer die Suche im Baum fortgeschritten ist, umso länger wird das gleiche Prefix, welches immer wieder erneut abgeglichen wird. Dadurch entsteht den Tries ein Vorteil, da hier immer nur kurze Zeichenketten verglichen werden. Im Vergleich zu Hashtabellen, die im Idealfall eine Komplexität von *O*(1) aufweisen benötigen PATRICIA-Tries weniger Reorganisationsaufwand, da es hier nicht zu Kollisionen wie bei den Hashtabellen kommt. Somit bieten sich PATRICIA-Tries für die Datenhaltung in Routingtabellen, als schnelle und speichereffiziente Datenstruktur an.

## 2.3. Werkzeuge

Im folgenden wird auf die in der Arbeit verwendete Software eingegangen.

### 2.3.1. VNUML

VNUML steht für Virtual-Network-User-Mode-Linux und ist eine unter der GNU Public Licence stehende Netzwerksimulationssoftware, die von 2002-2009 am Telematics Engineering Department an der Technischen Universität Madrid entwickelt wurde. Mit ihr ist es möglich, rein virtuell Netzwerkszenarien nachzustellen und das Zusammenspiel und Verhalten von unterschiedlichsten Netzwerkprotokollen zu beobachten. VNUML benutzt wiederrum User Mode Linux (UML), womit es möglich ist einen kompletten Linux-Kernel innerhalb eines Linux-Systems als Prozess zu starten. VNUML verbindet dann die virtuellen UML-Rechner zu einer vollständigen Netztopologie. Auf den einzelnen UML-Rechnern ist es möglich zur Laufzeit Programme zu installieren und zu starten sowie Änderungen an der Systemkonfiguration vorzunehmen. In dieser Arbeit fanden die Kernel-Version *linux2.6.18.1-bb2-xt-4m* sowie das Root-File-System *root_fs_tutorial-0.5.2* und VNMUL in der Version 1.8.9 Verwendung. Zur Beschreibung der Szenarien wurde die sogenannte VNUML-Sprache entwickelt, sie basiert auf XML und liegt in unterschiedlichen Versionen vor, sodass jedes Szenario über eine Angabe, der vom VNMULparser zu benutzenden Sprachversion verfügt. In der Szenario-Datei sind sowohl globale Konfigurationsinformationen, wie zu verwendendes Dateisystem, Kernel, zugesicherter Speicher hinterlegt, sowie auch Einstellungen für die virtuellen Maschinen, wie zum Beispiel Konfiguration der vorhandenen Netzwerkschnittstellen. Außerdem können Skripte eingefügt werden, die beim Start, bzw. dem Beenden des Szenarios zur Ausführung kommen.
Im Folgenden ist ein kleines Beispielszenario in der VNUML-Sprache beschrieben. Es existiert lediglich aus einem Netz und zwei darin befindlichen Hostsystemen. Zum besseren Verständnis werden einige der wichtigsten Parameter kurz genauer beschrieben.

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

<vnuml>
  <?vde enabled="false"?>
  <global>
    <version>1.8</version>
    <simulation_name>New_Scenario.xml</simulation_name>
    <ssh_version>2</ssh_version>
    <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
    <automac offset="0" />
    <vm_mgmt type="private" network="192.168.0.0" mask="24" offset="100">
      <host_mapping />
    </vm_mgmt>
    <vm_defaults exec_mode="mconsole">
      <filesystem type="cow">/usr/share/vnuml/filesystems/root_fs_tutorial</filesystem>
      <mem>50M</mem>
      <kernel>/usr/share/vnuml/kernels/linux</kernel>
      <forwarding type="ipv4" />
    </vm_defaults>
  </global>
  <net name="a" mode="virtual_bridge" type="lan">
  </net>
  <vm name="R1">
    <if id="1" net="a">
      <ipv4 mask="/24">10.0.0.1</ipv4>
    </if>
  </vm>
  <vm name="R2">
    <if id="1" net="a">
```

```
      <ipv4 mask="/24">10.0.0.2</ipv4>
    </if>
  </vm>
</vnuml>
```

Unter dem Punkt *global* werden die Rahmenbedingungen für den generellen Ablauf des Szenarios definiert. Zum Beispiel die verwendete Sprachversion, in unserem Fall 1.8, der Name des Szenarios, sowie die verwendete SSH-Version und der Ablageort des Schlüssels. Der optionale *automac*-Tag, gibt an ob eine automatische Zuweisung von Mac-Adressen an die Interfaces der virtuellen Maschinen stattfinden soll oder diese gesondert angegeben werden. Der Punkt *vm_mgmt* spezifiziert die Reichweite des von VNUML zur Steuerung der einzelnen virtuellen Maschinen benutzten Managment Netzwerkes. IP-Adressen aus diesem Bereich dürfen nicht für die weiter unten definierten Netzwerkschnittstellen der virtuellen Maschinen benutzt werden. Durch *host_mapping* werden die Hostnamen der unten aufgeführten virtuellen Maschinen in die Datei \etc\host geschrieben und mit ihrer IP-Adresse verknüpft. Dadurch ist beispielweise ein SSH-Login mit *ssh R1* statt *ssh 10.0.0.1* möglich.
Der Punkt *vm_defaults* beschreibt die Standards für alle virtuellen Maschinen, welcher in der Definition der speziellen virtuellen Maschine jedoch überschrieben werden kann. Hier wird auch angegeben, mit wie viel RAM, welchem Linux und welchem Linuxsystem die virtuellen Maschinen betrieben werden sollen. Im obigen Beispiel werden die virtuellen Maschinen im Copy-On-Write Modus betrieben, sodass ein Master-Filesystem benutzt wird und Änderungen an diesem für jede einzelne virtuelle Maschine in eine eigene Datei geschrieben werden, wodurch viel Speicherplatz eingespart wird. Der Punkt *forwarding* gibt an, ob eine virtuelle Maschine, wenn sie ein Paket auf einem Interface bekommt, in der Routingtabelle nachschauen soll, ob es für das Ziel des Pakets einen entsprechenden Eintrag gibt und dieses entsprechend weiterleiten soll. In unserem Fall werden nur IPv4 Pakete weitergeleitet.
Im *net*-Tag werden die Netze definiert. Für jede virtuelle Maschine, wird ein eigenes *vm*-Tag geöffnet, welche mehrere Interfaces besitzen kann, die mit dem *if* -Tag definiert werden. Jedem Interface wird eine eindeutige *id* und ein Netz zugewiesen in dem es sich befindet. Außerdem wird die IP-Adresse und die dazugehörige Netzmaske angegeben.

## 2.3.2. Quagga

Quagga ist eine Routing Software Suite, die Implementierungen der häufig benutzten Routingprotokolle OSPF, RIP und BGP für IPv4 sowie IPv6 für UNIX-basierende Systeme bietet. Bei Quagga handelt es sich um eine Abspaltung der Routingsoftware GNU Zebra, die von Kunihiro Ishiguro entwickelt wurde. Das Ziel der Entwicklung von Quagga ist es eine weniger auf eine Einzelperson zentrierte Entwicklung wie es bei Zebra der Fall ist zu verfolgen und stattdessen eine gemeinschaftliche Entwicklung anzustreben. Daher ist Quagga Open-Source.
Quagga besteht aus dem Core-Deamon Zebra, der die Schnittstelle zur vom Linux-Kernel verwalteten Forwardingtabelle bildet und aus den unterschiedlichen Routingdeamons, welche mit dem Zebra-Deamon kommunizieren. Jeder Routingdeamon läuft als eigenständiger Prozess und kann durch einen eigenen TCP-Port über Telnet erreicht werden. Die für diese Arbeit interessanten Ports sind 2601, für den Zebra-Deamon und 2602 für den RIP-Deamon.
Die Konfiguration der Deamons ist über telnet durch ein Command Line Interface möglich, welches an das Internetwork Operating System von Cisco angelehnt ist. Um nicht jeden

Deamon einzeln über telnet konfigurieren zu müssen ist es auch möglich mit dem Befehl vtysh gleichzeitig Zugriff auf alle Routingdeamons zu erhalten.

### 2.3.3. XTPeer

Beim XT-Peer handelt es sich um es sich um ein Programm zur Steuerung und Visualisierung von Quagga RIP-Deamons. Ursprünglich entstanden ist der XTPeer im Rahmen der Diplomarbeit von Daniel Pähler um eine Möglichkeit zu haben RMTI-Szenarien zentral steuern zu können. Der XTPeer ist in der Lage sich mit laufenden RIP- bzw. RMTI-Deamons zu verbinden und ihr Updateverhalten zu protokollieren und visuell aufzubereiten. So ist es möglich, eine VNUML-Szenariodatei in den XTPeer zu laden, woraufhin der Netzaufbau graphisch dargestellt wird. Die einzelnen RIP-Router sind über den XTPeer konfigurierbar, so kann beispielweise zur Laufzeit zwischen verschiedenen RMTI-Modi gewechselt werden.
Auch ist es möglich, Netzwerkinterfaces zu deaktivieren und Routingupdates manuell auszulösen. Der XTPeer zeichnet außerdem alle Routingupdates eines Interfaces auf und gibt detaillierte Informationen zu diesen. Zum Beispiel ob X/Y-Test bestanden wurden und wie der aktuelle Zustand der MSILM und MRPM-Tabellen des Routers ist.
Dadurch erleichtert der XTPeer die Analyse des Verhaltens von RMTI-Szenarien maßgeblich.

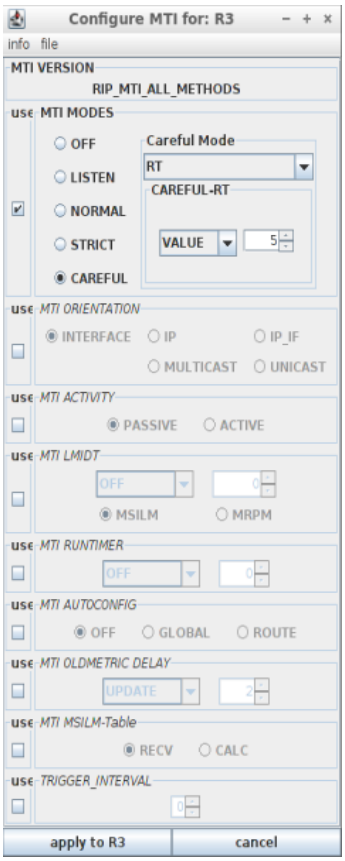



**Abb. 2.27 Konfiguration eines RMTI-Routers**

# 3. Visualisierung von Routingtabellenstrukturen

In diesem Kapitel gehen wir auf den Aufbau der RMTI-Routingtabellen und welche Konsequenzen sich daraus im Hinblick auf die Routenaggregation ergeben ein.

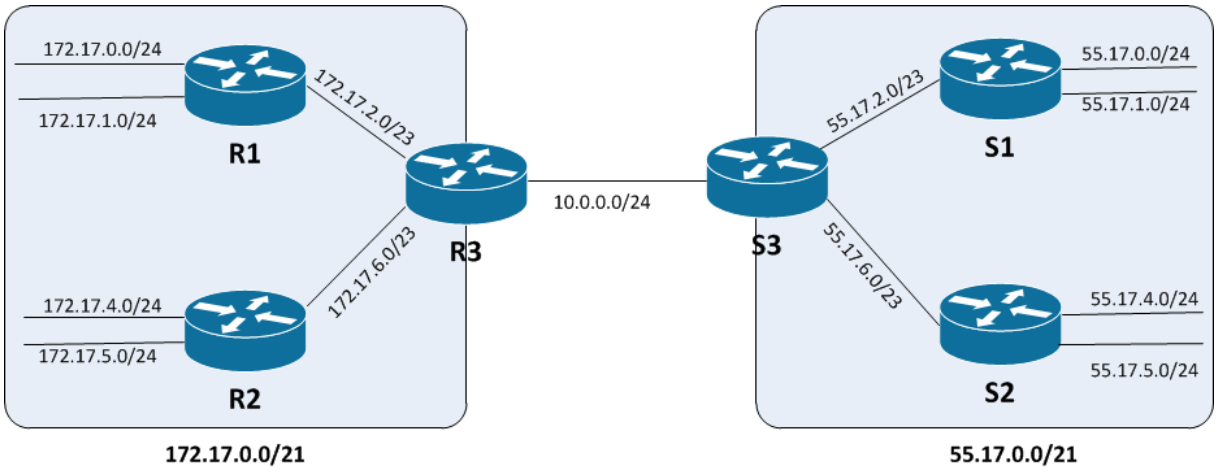



**Abb. 3.1 Szenario mit Aggregationsmöglichkeit**

Um die Funktionsweise von Patricia Tries zur Datenhaltung in Routingtabellen zu verdeutlichen werden wir im Folgenden auf das in Abbildung 3.1 dargestellte Netzwerkszenario zurück greifen. Beim Szenario handelt es sich um zwei Router-Verbunde, von denen sowohl der S- als auch der R-Verbund seine kleineren anliegenden Netze zu einem Aggregat (172.17.0.0/21 bzw. 55.17.0.0/21) zusammenfügen kann. Verbunden sind die Router R3 und S3 mit einem nicht zum Aggregat gehörigen Verbindungnetz. Bei funktionierender Aggregation tauschen R3 und S3 somit nur noch die beiden Super-Netze unter sich aus, R1, R2 und R3 erhalten somit keine Informationen über die eigentliche Zusammensetzung des Netzes 55.17.0.0/21. Das Selbe gilt für den S-Verbund und 172.17.0.0/21. Später wird das Szenario in Abbildung 3.1 noch um Netzwerkschleifen ergänzt um zu zeigen, dass die Implementierung in der Lage ist auch in redundanten Topologien zu aggregieren und dabei alle Teilnetze uneingeschränkt erreichbar bleiben.

```
struct route_node {
    struct prefix p;
    struct route_table *table;
    struct route_node *parent;
    struct route_node *link[2];
#define l_left   link[0]
#define l_right  link[1]
    unsigned int lock;
    void *info;
    int aggregator;
};
```

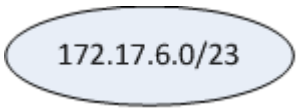
```
struct rip_info {
    int type;
    int sub_type;
    struct in_addr nexthop;
    unsigned int ifindex;
    u_int32_t metric;
    time_t mti_oldtimer;
    u_int32_t mti_oldmetric;
    u_char distance;
    ...
};
```

Die beiden C-Structs in den Quellcode-Boxen beschreiben zwei zum Verständnis der Routingtabellen wichtige Datenstrukturen. Beim route_node handelt es sich um einen Knoten innerhalb des Baumes, der die Routingtabelle repräsentiert. Abgesehen von der Wurzel hat jeder Knoten einen Elternknoten(parent) und wenn es sich nicht um einen Blattknoten handelt

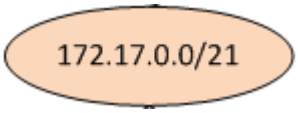
auch einen oder zwei Kindknoten(l_left,l_right). Die Variable lock ermittelt ob ein Knoten noch benötigt wird oder aus dem Baum entfernt werden kann. Je nach Verwendung des Knotens wird lock(*route_node) oder unlock((*route_node) aufgerufen und die Variable in- bzw. dekrementiert. Sollte sie kleiner als eins sein, wird der Knoten automatisch aus dem Baum entfernt. Aggregator gibt an, ob es sich beim aktuellen Knoten um ein Aggregat aus seinen Kindknoten handelt. Info gibt an, ob der Knoten eine Routinginformation besitzt und wenn vorhanden deren zugehörige Speicherstelle. Nur Knoten mit vorhandener Info werden vom Router propagiert, sind also existierende Routen.

Beim struct rip_info handelt es sich um die eben angesprochene Info, hier werden Informationen wie der NextHop in ein Zielnetz und die Metrik zum Ziel hinterlegt. Type gibt an, ob es sich bei der Route um eine direkt mit einem Interface verbundene (ZEBRA_ROUTE_CONNECT) oder um eine von einem anderen Router gelernte (ZEBRA_ROUTE_RIP) handelt. Ifindex gibt an, von welchem Interface eine Routeninformation empfangen wurde und distance beschreibt die administrative Distanz, also die Zuverlässigkeit der Information. Diese unterscheidet sich je nach dem Informationsursprung (RIP = 120, statisch gesetzte Route = 1) und wird bei der Auswahl des Zebra-Deamons für die bestmögliche Forwarding-Option benutzt. Zusätzlich sind im rip_info Struct noch einige für die Schleifenerkennung des RMTI benötigte Informationen, die aber mit Blick auf die Routenaggregation nicht relevant sind.

Im RIP bzw. RMTI Algorithmus ohne Aggregation sind im Baum, der die Routingtabelle repräsentiert zwei unterschiedliche Typen von Knoten im Einsatz.




Die hellblau gefüllten Knoten beschreiben eine von einem anderen Router gelernte beziehungsweise eine direkt per Interface verbundene Route in ein Zielnetz. Diese Knoten dürfen aktiv weiter im Netz propagiert werden. Sie sind immer mit einer RIP-Info, wie oben im C-Struct beschrieben, ausgestattet und das Aggregator-Flag ist auf 0 gesetzt.




Diese hellorange gefüllten Knoten sind virtuelle Knoten, die zur Organisation der Routingtabelle genutzt werden. Da ein Knoten immer nur zwei Kindknoten haben kann, kann es passieren, dass neu aufzunehmende Netze sich nicht in die bisherige Struktur einpassen, beispielweise wenn der Knoten unter dem der neue Knoten eingruppiert werden soll bereits zwei Kindknoten besitzt. In diesem Fall wird dann ein solcher virtueller Knoten erstellt, dessen Prefix und Prefix-Länge so gewählt wird, daß er zwei Knoten aufnehmen kann. Diese Knoten dienen nur zur Organisation der Routingtabelle, sie werden nicht vom Router im Netz propagiert und haben als route_node keine RIP-Info und ein auf 0 gesetztes Aggregator-Flag.

Zum besseren Verständnis betrachten wir nun den schrittweisen Aufbau der Routing-Tabelle, der in Abbildung 3.1 abgebildeten Topologie. Wir gehen davon aus, dass alle Router zeitgleich gestartet werden und auf allen die Routen-Aggregation deaktiviert ist, wie es beispielweise in RIPv2 der Fall ist. Unsere Betrachtung richtet sich auf den zentral liegenden Router R3.

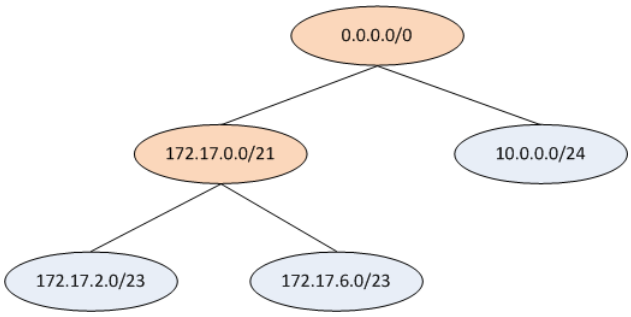



**Abb. 3.2 Tabellenstruktur nach Einfügen der angeschlossenen Netze**

Als erstes gruppiert R3 seine direkt über ein Interface anliegenden Netze in die Routingtabelle ein. Zuerst ist dies 172.17.2.0/23, welches kurzzeitig die Wurzel bildet. Als nächstes wird 172.17.6.0/23 aufgenommen und es wird ein virtueller Knoten gebildet, 172.17.0.0/21, der die beiden existierenden Netze zusammenfasst und als neue Wurzel fungiert. Als letztes wird 10.0.0.0/24 einsortiert und da es nicht in den bisherigen Baum passt wird ein neues virtuelles Netz 0.0.0.0/0 angelegt. Das Netz muss die maximale Größe beinhalten, da 10.0.0.0/24 und 172.17.0.0/21 sich bereits im ersten Bit des Prefixes unterscheiden. Nach der Einordnung der direkt per Interface angebundenen Routen hat der Baum der Routingtabelle den Zustand wie in Abbildung 3.2 dargestellt.

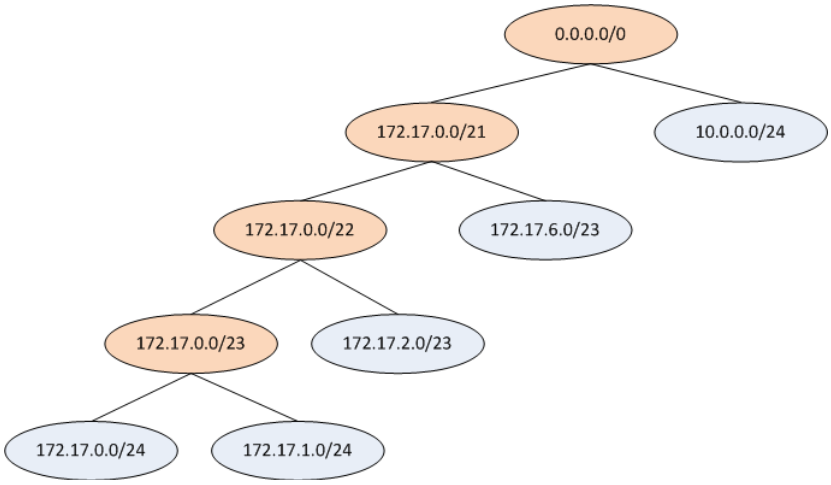



**Abb. 3.3 Tabellenstruktur nach Empfang eines Updates von R1**

Nun empfängt R3 ein RIP-Update von R1, welches die Netze 172.17.0.0/24 und 172.17.1.0/24 beinhaltet. Da das virtuelle Netz 172.17.0.0/21 bereits zwei Kindknoten hat, wird ein neuer virtueller Knoten erstellt. Unter diesem (172.17.0.0/22) wird das bereits in der Tabelle befindliche Netz 172.17.2.0/23 eingeordnet und ein weiterer virtueller Knoten 172.17.0.0/23 erstellt, der die beiden durch das Update neu erfassten Netze enthält. Am Ende hat die Routingtabelle von R3 den in Abbildung 3.3 dargestellten Zustand.

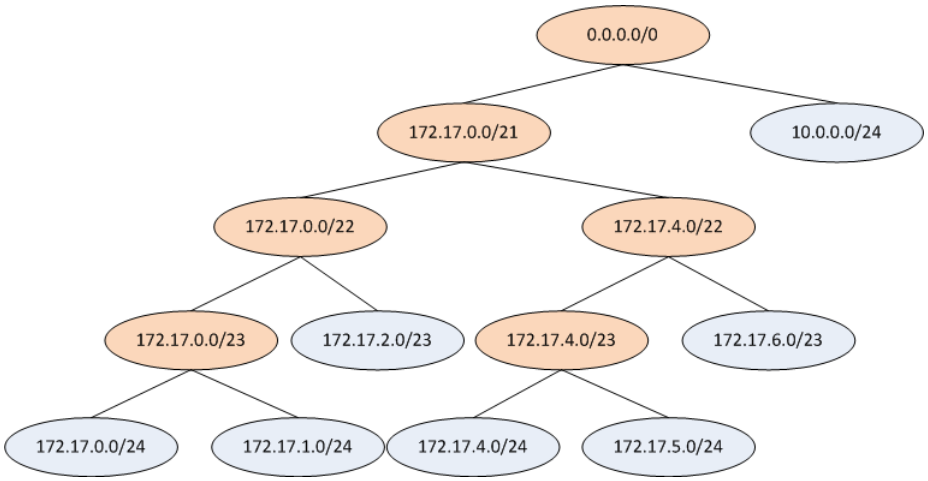



**Abb. 3.4 Tabellenstruktur nach Empfang eines Updates von R2**

Als nächstes sendet R2 seine beiden R3 noch nicht bekannten Netze 172.17.4.0/24 und 172.17.5.0/24. Es wird der neue virtuelle Knoten 172.17.4.0/22 erstellt und das bereits bekannte Netz 172.17.6.0/23 und ein weiterer neuer virtueller Knoten unter diesem eingeordnet. Der zweite neue virtuelle Knoten ist der Elternknoten der beiden von R2 soeben erlernten Netze.

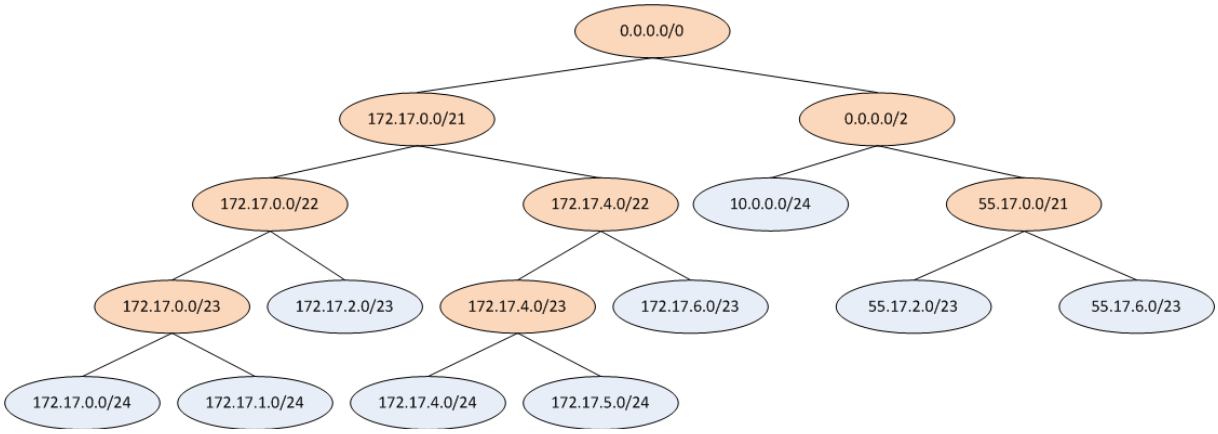



**Abb. 3.5 Tabellenstruktur nach Empfang eines Updates von S3**

Auch S3 hat ein RIP-Update an R3 gesendet, welches seine aktuell bekannten Netze beinhaltet. Dies sind die direkt an S3 anliegenden 55.17.2.0/23 und 55.17.6.0/23. Da diese nicht in die bisherige Struktur eingefügt werden können wird der virtuelle Knoten 0.0.0.0/2 gebildet, unter dem 10.0.0.0/24 und der virtuelle Knoten 55.17.0.0/21 eingefügt werden, unter dem dann wiederrum die beiden neu gelernten Netze eingefügt werden können. Am Ende der Einfügeoperation hat der Baum den in Abbildung 3.5 dargestellten Zustand.

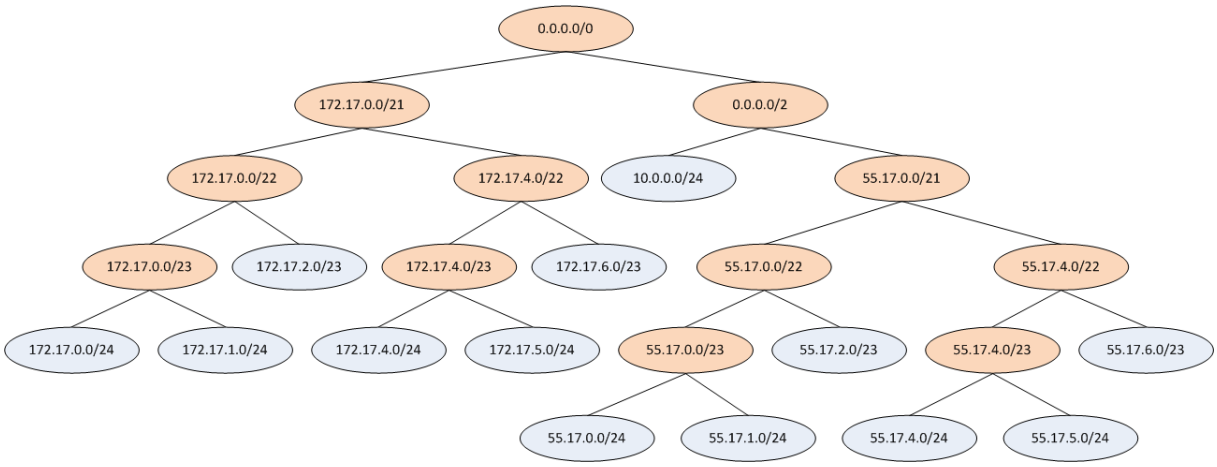



**Abb. 3.6 Tabellenstruktur nach erneutem Empfang eines Updates von S3**

Nach kurzer Zeit erhält R3 erneut ein Update von S3, da S3 mittlerweile ebenfalls die Netze seiner Nachbarn S1 und S2 erlernt hat und diese nun weiter reicht. In dem bei R3 eintreffenden Update sind die vier Netze 55.17.0.0/24, 55.17.1.0/24, 55.17.4.0/24 und 55.17.5.0/24, enthalten. Um diese eingruppieren zu können werden wiederum die vier neuen virtuellen Knoten 55.17.0.0/22, 585.17.4.0/22, 55.17.0.0/23, 55.17.4.0/23 erstellt. Der in Abbildung 3.6 dargestellte Zustand ist der konvergente Zustand in der Routingtabelle von R3, solange es keine Topologieänderungen gibt.

Würde man nun einen weiteren Router ins Netz aufnehmen und diesen über ein einziges Netz nur mit R3 verbinden, würde R3 alle 13 in den Blattknoten gespeicherten Netze an diesen Neuen Router melden, obwohl sich aufgrund des Netzaufbaus die vorhandenen Netze effizient in lediglich drei zusammenfassen lassen würden. Dies wären 172.17.0.0/21, das Verbindungsnetz 10.0.0.0/24 sowie 55.17.0.0/21.

Dies wirft die Frage auf, wann es möglich ist zwei Teilnetze zu einem Aggregat zusammen zu fassen und dieses zu kommunizieren. Damit keine Lücken entstehen und alle Netze die als erreichbar gemeldet werden auch erreichbar sind, müssen die zwei zu aggregierenden Netze direkt aneinander angrenzen und gleich groß sein.

Betrachten wir die Netze 172.17.0.0/24 und 172.17.1.0/24.

172.17.0.0/24

| | |
|---|---|
| Netz: | 172.17.0.0 |
| HostMin: | 172.17.0.1 |
| HostMax: | 172.17.0.254 |
| Broadcast: | 172.17.0.255 |

172.17.1.0/24

| | |
|---|---|
| Netz: | 172.17.1.0 |
| HostMin: | 172.17.1.1 |
| HostMax: | 172.17.1.254 |
| Broadcast: | 172.17.1.255 |

Beide Netze liegen direkt nebeneinander, es existiert keine Lücke zwischen ihnen und sie haben eine identische Netzmaske. Somit ist eine Aggregation zu 172.17.0.0/23 hier möglich.

| | |
|---|---|
| 172.17.0.0/24 : | 10101100.00010001.0000000**0** .00000000 |
| 172.17.0.1/24 : | 10101100.00010001.0000000**1** .00000000 |

In der Binärdarstellung der beiden Prefixe wird deutlich, dass sich bei einer lückenlosen Aggregation zweier Prefixe, der Netzanteil dieser nur im letzten Bit unterscheiden darf.

172.17.0.0/23

| | |
|---|---|
| Netz: | 172.17.0.0 |
| HostMin: | 172.17.0.1 |
| HostMax: | 172.17.1.254 |
| Broadcast: | 172.17.1.255 |

Das bloße nebeneinander liegen von zwei Teilnetzen mit gleichgroßer Netzmaske genügt allerdings noch nicht als Bedingung für eine Aggregation. Betrachten wir beispielweise die ebenfalls nebeneinander liegenden Netze 172.17.1.0/24 und 172.17.2.0/24.

172.17.1.0/24

| | |
|---|---|
| Netz: | 172.17.1.0 |
| HostMin: | 172.17.1.1 |
| HostMax: | 172.17.1.254 |
| Broadcast: | 172.17.1.255 |

172.17.2.0/24

| | |
|---|---|
| Netz: | 172.17.2.0 |
| HostMin: | 172.17.2.1 |
| HostMax: | 172.17.2.254 |
| Broadcast: | 172.17.2.255 |

| | |
|---|---|
| 172.17.0.1/24 : | 10101100.00010001.0000000**1** .00000000 |
| 172.17.0.2/24 : | 10101100.00010001.0000001**0** .00000000 |

Wie die Binärdarstellung zeigt, unterscheidet sich der Netzanteil der beiden Prefixe bereits im vorletzten Bit, sodass das kleinste Netz in dem sich beide Teilnetze befinden 172.17.0.0/22 ist.

172.17.0.0/22

| | |
|---|---|
| Netz: | 172.17.0.0 |
| HostMin: | 172.17.0.1 |
| HostMax: | 172.17.3.254 |
| Broadcast: | 172.17.3.255 |

Die Teilnetze füllen 172.17.0.0/22 jedoch nicht vollständig aus, es bleiben die Netzbereiche 172.17.0.0/24 und 172.17.3.0/24, die nicht durch Teilnetze abgedeckt wären, weshalb hier nicht aggregiert werden darf. Als nächste Bedingung gilt also, das ein potentielles Aggregat für zwei Teilnetze eine lediglich um eins größere Netzmaske aufweisen darf. Hier kommt uns die Struktur der Patricia Tries, in denen die Routen vorgehalten werden zu gute, da wir somit nur prüfen müssen, ob der Elternknoten der beiden zu aggregierenden Knoten einen um ein Bit vergrößerten Hostanteil der Subnetzmaske gegenüber den Kindknoten aufweist.

Außerdem darf es keine Aggregation von Knoten geben, bei dem einer, oder sogar beide, mit einer RIP-Infinity-Metrik versehen ist. Sollte einer der Kindknoten per Routingupdate auf RIP-Infinity gesetzt werden, müssen alle Aggregate die diesen Teilknoten beinhalten aufgelöst werden. Desweiteren darf eine Aggregation nur ausgeführt werden, wenn der Elternknoten der beiden zu aggregierenden Netze weder eine RIP-Info enthält, noch bereits ein Aggregatknoten ist, also nur, wenn es sich bei ihm um einen virtuellen Knoten handelt.


Die hellgrün gefüllten Knoten symbolisieren ab jetzt die Aggregatknoten. Ein Aggregatknoten hat immer eine RIP-Info, die aus den RIP-Infos seiner beiden Kindknoten errechnet wurde sowie ein gesetztes Aggregator-Flag. Die Kindknoten eines Aggregatknotens können sowohl von einem anderen Router gelernte Routen sein, direkt per Interface angebundene, oder auch erneut Aggregatknoten. Dies ist bei einer Mehfachaggregation der Fall, wenn durch die Bildung eines Aggregates ebenfalls der Elternknoten des neuen Aggregats alle Vorrausetzungen zur Bildung eines weiteren Aggregats erfüllt. Für die Zuweisung der Metrik gibt es mehrere Möglichkeiten, beispielweise kann das Aggregat die schlechteste Metrik seiner Kindknoten, oder den Durchschnitt beider annehmen. Im Rahmen dieser Arbeit wird die schlechtere der Kindknoten verwendet, der Grund dafür wird in Kapitel 6 genauer erläutert.

In der beschriebenen Implementierung wird der Aggregatknoten in seiner RIP-Info als ZEBRA_ROUTE_CONNECT typisiert, mit dem Subtyp RIP_ROUTE_STATIC. Diese Routen werden zwar propagiert, jedoch nicht in die Forwardingtabelle des Kernels übernommen, da aufgrund des longest prefix match Algorithmus für das Forwarding immer die Einträge der Teilnetze benutzt werden.

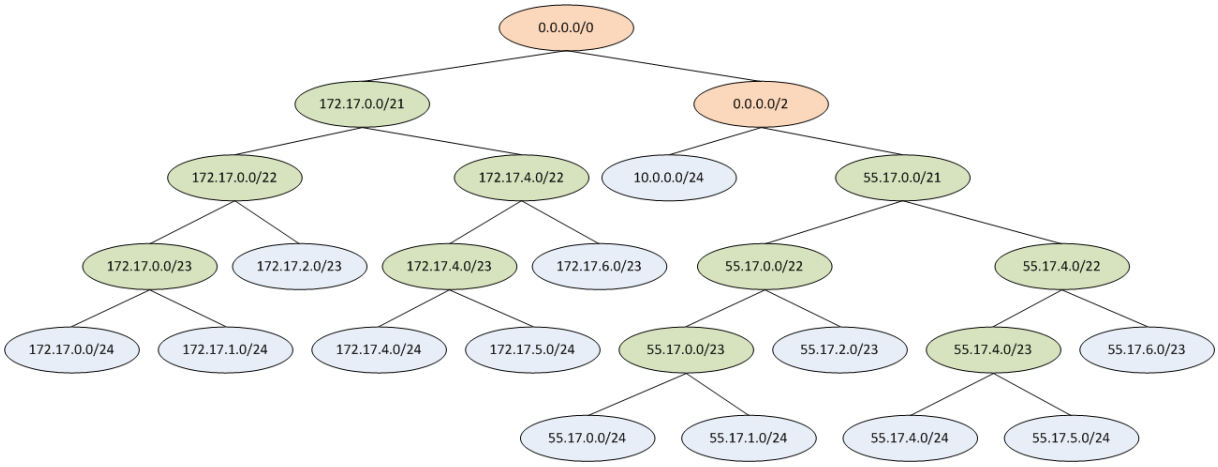



**Abb. 3.7 Tabellenstruktur bei aktivierter Aggregation in R3**

Wenn nun wie beschrieben aggregiert wird, baut sich die RIP-Routingtabelle von R3 wie die in Abbildung 3.7 dargestellte Baumstruktur auf. Dieser Zustand würde sich ergeben, wenn die Routenaggregation lediglich auf R3 aktiviert ist und die restlichen Router des Autonomen Systems keine Aggregation ihrer eigenen Routen vornehmen. Sollte man nun einen wie vorhin bereits angesprochen, einen neuen Router in die in Abbildung 3.1 dargestellte Topologie integrieren und direkt mit R3 verbinden, würde R3 diesem lediglich 3 Netze

mitteilen. Dies wären die beiden Aggregate 172.17.0.0/21 und 55.17.0.0/21, sowie das Netz 10.0.0.0/24. Somit teilt R3 mit aktivierter Aggregation nur noch drei Netze mit, gegenüber den 13 ohne Aggregation. Dies senkt den Netzwerkverkehr, der fürs Routing benötigt wird massiv.

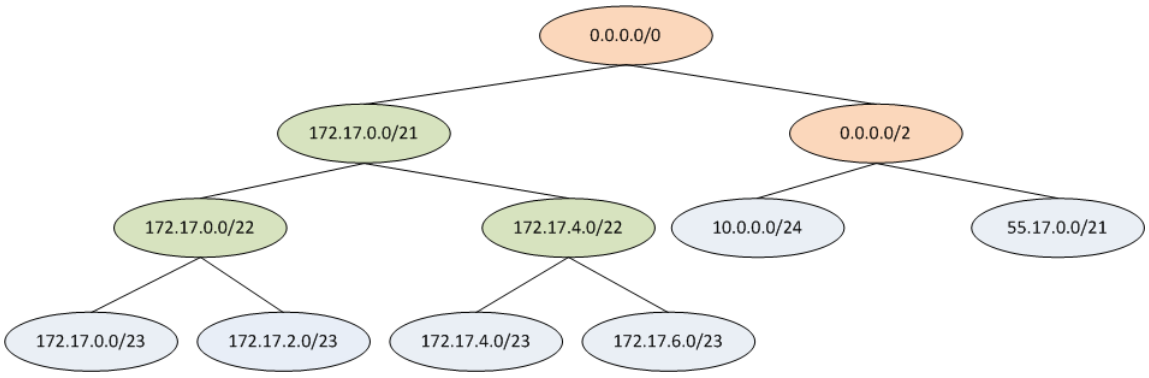



**Abb. 3.8 Tabellenstruktur bei aktivierter Aggregation im gesamten Netz**

Wenn wir nun die Aggregation auf allen Routern im Netz aktivieren, stellt sich die Situation für R3 wie in Abbildung 3.8 beschrieben dar. Das Supernetz 55.17.0.0/21 wird nun bereits von S3 zusammen gefasst und als Ganzes an R3 übergeben. Dadurch hat R3 keinerlei Kenntnis darüber, dass es sich bei diesem Netz eigentlich um ein Aggregat handelt.

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

     Network          Next Hop        Metric From           Tag Time       Aggregator
C(i) 10.0.0.0/24      0.0.0.0              1 self             0  0
R(n) 55.17.0.0/21     10.0.0.2             2 10.0.0.2         0 00:30  0
C(s) 172.17.0.0/21    0.0.0.0              1 self             0  1
C(s) 172.17.0.0/22    0.0.0.0              1 self             0  1
R(n) 172.17.0.0/23    172.17.2.1           2 172.17.2.1       0 00:28  0
C(i) 172.17.2.0/23    0.0.0.0              1 self             0  0
C(s) 172.17.4.0/22    0.0.0.0              1 self             0  1
R(n) 172.17.4.0/23    172.17.6.1           2 172.17.6.1       0 00:29  0
C(i) 172.17.6.0/23    0.0.0.0              1 self             0  0
Router>
```

**Abb. 3.9 RIP-Tabelle des Routers R3 bei Aggregation im gesamten Netz**

In Abbildung 3.9 ist die RIP-Routingtabelle des Routers R3 aus der Topologie in Abbildung 3.1 dargestellt. Die mit C(i) gekennzeichneten Routen sind direkt über ein eigenes Interface des Routers erreichbar *connected(interface)*. R(n) bedeutet, dass die Route über das RIP-Protokoll von einem anderen Router gelernt wurde *RIP(normal)*. Die Routen die der Router zu einem Aggregat zusammen gefasst hat sind als C(s) eingetragen *connected(static)*, hier ist ebenfalls das Aggregator-Flag gesetzt und als NextHop self angegeben. Die Kind-Routen auf denen das Aggregat basiert, werden nicht weiter im Netz propagiert, was bedeutet der gesamte Teilbaum unterhalb eines Aggregats bleibt nur dem aggregierenden Router bekannt.

```
R3:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.17.6.0      0.0.0.0         255.255.254.0   U     0      0        0 eth2
172.17.4.0      172.17.6.1      255.255.254.0   UG    2      0        0 eth2
172.17.2.0      0.0.0.0         255.255.254.0   U     0      0        0 eth3
172.17.0.0      172.17.2.1      255.255.254.0   UG    2      0        0 eth3
55.17.0.0       10.0.0.2        255.255.248.0   UG    2      0        0 eth1
R3:~#
```

**Abb. 3.10 Forwarding-Tabelle des Routers R3 bei Aggregation im gesamten Netz**

Die Abbildung 3.10 zeigt die Kernel-Forwarding-Tabelle des Routers R3, sie wird benutzt um das eigentliche Paket-Forwarding durchzuführen. Die Forwarding-Tabelle wird vom Zebra-Deamon der Quagga-Routingsuite verwaltet. Wie zu sehen ist, fehlen die Einträge der von R3 aggregierten Routen, ein Eintragen dieser wäre nicht sinnvoll, da auch hier longest Prefix-Match Anwendung findet und damit grundsätzlich immer der Kindknoten des Aggregats für das Forwarding benutzt wird. Außerdem kann sich ein Aggregat auch über mehrere Interfaces erstrecken, was einen Eintrag in der Forwarding-Tabelle problematisch werden lassen würde, da für jede Route nur ein Interfaceeintrag möglich ist.

# 4. Implementierung der Aggregation

Dieses Kapitel erläutert die vorgenommenen Änderungen und geht auf den aggregationspezifischen Quellcode ein.

```
01. void aggregate(struct route_node *nnew, struct route_node *nold,
02.         unsigned int ifindex) {
03.     struct route_node *parent;
04.
05.     int nnew_metric = get_infinity();
06.     int nold_metric = get_infinity();
07.
08.     if ((nnew->info) && (nold->info)) {
09.         struct rip_info *nnew_rinfo;
10.         struct rip_info *nold_rinfo;
11.         nnew_rinfo = nnew->info;
12.         nold_rinfo = nold->info;
13.
14.         nold_metric = nold_rinfo->metric;
15.         nnew_metric = nnew_rinfo->metric;
16.     }
17.     if (nnew)
18.         parent = nnew->parent;
19.
20.     if (parent && nnew && nold &&
21.         nnew->p.prefixlen == nold->p.prefixlen)
22.
23.         if ((parent->aggregator != 1 || !parent->info)
24.                 && ((parent->p.prefixlen + 1) == nnew->p.prefixlen)
25.                 && !prefix_same(&nold->p, &nnew->p) && (nold->info)
26.                 && (nnew_metric < get_infinity())
27.                 && (nold_metric < get_infinity())) {
28.
29.             parent->aggregator = 1;
30.             route_lock_node(parent);
31.             set_rinfo(parent,ifindex);
32.         }
33. }
```

Die dargestellte Funktion aggregate hat die Aufgabe zwei route_nodes darauf zu überprüfen, ob sie zu einem Aggregat zusammen gefasst werden können. In den Zeilen 5-16 werden die Metrikwerte beider Knoten erfasst. In den Zeilen 20 und 21 findet der Prefixlängenvergleich statt, es wird also geprüft, ob beide route_nodes die selbe Subnetzmaske aufweisen, was eine wichtige Voraussetzung für eine mögliche Aggregation ist. In Zeile 23 wird geprüft, ob es sich beim Elternknoten der beiden zu aggregierenden Knoten um einen virtuellen Knoten handelt, indem das Parent auf ein evtl. gesetztes Aggregator-Flag und eine vorhandene RIP-Info geprüft wird.

Wenn soweit alle Bedingungen erfüllt waren, wird untersucht, ob das Parent einen um eins kleineren Netzanteil als die beiden Kindknoten besitzt. In den nächsten beiden Zeilen wird ausgeschlossen, dass die Funktion versehentlich mit zwei absolut identischen Netzen aufgerufen wurde und das beide Routen eine Metrik kleiner RIP-Infinity besitzen. Wenn auch dieser letzte Test bestanden wurde, wird das Aggregator-Flag des Elternknoten gesetzt, der Semaphor des Parent erhöht und über den Funktionsaufruf von set_rinfo() eine RIP-Info für den Elternknoten erstellt.

Somit ist der Parentknoten nun von einem virtuellen Knoten ohne RIP-Info und ohne Aggregator-Flag, zu einem Aggregatknoten geworden.

```
01. void set_rinfo(struct route_node *rp, unsigned int ifindex)
02.     {
03.     struct rip_info *rinfo;
04.
05.     struct route_node *left;
06.     struct route_node *right;
07.     left = rp->l_left;
08.     right = rp->l_right;
09.     struct rip_info *left_rinfo;
10.     struct rip_info *right_rinfo;
11.     left_rinfo = left->info;
12.     right_rinfo = right->info;
13.     rinfo = rip_info_new();
14.     rinfo->rp = rp;
15.     if (left_rinfo->metric > right_rinfo->metric)
16.           rinfo->metric = left_rinfo->metric;
17.           else rinfo->metric = right_rinfo->metric;
18.     rinfo->tag = 0;
19.     rinfo->ifindex = ifindex;
20.     rinfo->mti_oldifindex = ifindex;
21.     rinfo->local_pref = DEFAULT_LOCAL_PREF;
22.     rinfo->mti_oldtimer = 0;
23.     rinfo->mti_oldmetric = rinfo->metric;
24.     memset(&rinfo->mti_oldfrom, 0, sizeof(struct in_addr));
25.     memset(&rinfo->nexthop, 0, sizeof(struct in_addr));
26.     memset(&rinfo->from, 0, sizeof(struct in_addr));
27.     rinfo->flags |= 2;
28.     rinfo->type = ZEBRA_ROUTE_CONNECT;
29.     rinfo->sub_type = RIP_ROUTE_STATIC;
30.
31.     rinfo->distance = 120;
32.     rp->info = rinfo;
33.
34.     rinfo->flags |= RIP_RTF_FIB;
35.
36.     if (rp->parent)
37.     {
38.         struct route_node *parent;
39.         parent = rp->parent;
40.         aggregate(parent->l_left,parent->l_right,ifindex);
41.
42.     }
43.
```

Die Funktion set_rinfo() erstellt für den aufgerufenen Knoten eine neue RIP-Info. Aufgerufen wird set_rinfo() aus der Funktion aggregate um einem Aggregatknoten eine RIP-Info zu erstellen. Zur Metrikberechung des Aggregats werden die beiden Kindknoten betrachtet und es wird ab Zeile 15 die höhere der beiden Metriken als Metrik des Aggregats festgelegt. Die Route wird in Zeile 28 als direkt verbunden eingetragen und als Subtyp wird RIP_ROUTE_STATIC gewählt, damit das Aggregat in der Routingtabelle direkt erkannt werden kann. Außerdem wird die administrative Distanz auf den Cisco-Standard für RIP-Routen gesetzt. In Zeile 32 wird dem Aggregatknoten dann die neue RIP-Info übergeben. Da

sich durch den neuen Aggregatknoten eventuell neue Möglichkeiten zur Aggregation aufgetan haben, wird falls ein Parent-Knoten für das Aggregat vorhanden ist aggregate() erneut aufgerufen mit dem linken und rechten Kind des Parent-Knoten vom neuen Aggregatknoten um eine eventuell mögliche Mehrfachaggregation durchzuführen.

```
01. void delete_aggregates(struct route_node *rp)
02. {
03.     int changed =0;
04.     struct route_node *parent;
05.     struct route_node *temp;
06.     parent = rp->parent;
07.     int loopbreak = 0;
08.
09.     if (parent && (parent->aggregator == 0) && (parent->info))
10.         loopbreak = 1;
11.
12.     while ((parent) && (loopbreak == 0)) {
13.         if (parent) {
14.             if (parent->aggregator == 1) {
15.                 parent->aggregator = 0;
16.                 parent->info = NULL;
17.                changed = 1;
18.             }
19.             temp = parent->parent;
20.             parent = temp;
21.             if (parent && (parent->aggregator == 0) && (parent->info))
22.                 loopbreak = 1;
23.         }
24.     }
25.     if(changed)rip_event(RIP_TRIGGERED_UPDATE, 0);
26. }
```

Um beim Wegfall eines realen route_nodes alle eventuell auf diesen aufbauende Aggregate zu löschen wurde die Funktion delete_aggregates() eingeführt. Sie durchläuft den Baum vom übergebenen Knoten bis zur Wurzel und beendet ihren Weg, sobald die Suche auf einen, eine reale Route repräsentierenden, Knoten trifft. Solange wie dies nicht der Fall ist, wird bei jedem passierten Aggregat die RIP-Info gelöscht und das Aggregator-Flag auf 0 gesetzt, alle besuchten Aggregatknoten werden also wieder zu virtuellen Knoten.

Sollte eine solche Änderung durchgeführt worden seien, wird am Ende der Suche ein Triggered-Update versendet um die Aggregate, die andere Router in ihre Routingtabellen aufgenommen haben ebenfalls zu löschen.

Die Funktion delete_aggregates() wird aus unterschiedlichen Teilen des Programms aufgerufen. Zum Beispiel aus rip_timeout(), diese Funktion wird aufgerufen sobald der Timeout für eine gelernte Route abgelaufen ist innerhalb dieser Funktion wird die Metrik der ausgefallenen Route auch auf RIP-Infinity gesetzt. Dies deckt also den Fall einer ausgefallenen gelernten Route ab. Außerdem erfolgt ein Aufruf aus rip_garbage_collect(), durch diesen Aufruf haben wir den Ausfall von direkt per Interface verbundene Routen abgedeckt, da es für diese keinen Timeout gibt und rip_timeout() für diese daher nicht ausgeführt wird. Schlussendlich kann delete_aggregates() auch aus rip_rte_process() aufgerufen werden. Rip_rte_process wird ausgeführt, wenn ein Routing-Update eintrifft.

Sollte nun ein Routing-Update mit Metrik RIP-Infinity für einen realen route_node eintreffen, werden die auf diesen basierenden Aggregate ebenfalls gelöscht.

```
01. if (ri->split_horizon == RIP_SPLIT_HORIZON)
02.       {
03.         if (rp->aggregator == 1) {
04.             ttmp = rp;
05.             rtmp = rp;
06.
07.             while (ttmp->l_left && ttmp != ttmp->l_left) {
08.                 ttmp = ttmp->l_left;
09.             }
10.
11.             while (rtmp->l_right && rtmp != rtmp->l_right) {
12.                 rtmp = rtmp->l_right;
13.             }
14.
15.             route_lock_node(ttmp);
16.             loopbreak = 0;
17.             while (ttmp != NULL && loopbreak == 0) {
18.                 if (ttmp->info) {
19.                     struct rip_info *tmpinfo = ttmp->info;
20.                     tmpPrefix = (struct prefix_ipv4 *) &ttmp->p;
21.                     if (
22.                          (tmpinfo->type == ZEBRA_ROUTE_RIP  &&
23.                       (tmpinfo->ifindex == ifc->ifp->ifindex) )
24.                          || (tmpinfo->type == ZEBRA_ROUTE_CONNECT &&
25.                       prefix_match((struct prefix *)p, ifc->address)  )
26.                          || (tmpinfo->type == ZEBRA_ROUTE_CONNECT &&
27.                    prefix_match((struct prefix *)tmpPrefix,
28.                          ifc->address)))
29.                     {
30.                         loopbreak = 1;
31.                     }
32.                 }
33.                 ttmp = next_leaf_until(ttmp, rtmp);
34.             }
35.
36.             if(loopbreak == 1) {
37.                 continue;
38.             }
39.         }
40.
41.         if(rp->aggregator == 1) {
42.             route_lock_node(rtmp);
43.             route_unlock_node(rp);
44.             rp = rtmp;
45.         }
46.         if (rinfo->type == ZEBRArip_split_ROUTE_RIP
47.          && rinfo->ifindex == ifc->ifp->ifindex) {
48.           continue;
49.         }
50.         if (rinfo->type == ZEBRA_ROUTE_CONNECT
51.          && prefix_match((struct prefix *)p, ifc->address)) {
52.           continue;
53.         }
54.       } else {
55.         while (rp->l_right && rp != rp->l_right) {
56.             rp = rp->l_right;
57.         }
58.       }
```

Der dargestellte Quelltext beschreibt den bereits von Milad Khojasteh-Samiei und Andreas Brandt in [KhBr11] eingeführten Split-Horizon für Aggregatknoten. Trotz der beschriebenen Veränderungen wie der implementierten Mehrfachaggregation, erfüllt dieser Programmteil seine Arbeit unverändert. Er ist Bestandteil der Funktion rip_output_process(), die für jedes Interface ein RIP-Update generiert und bei eingeschaltetem Split-Horizon überprüft, ob eine Route im Updatepaket enthalten sein darf. Dies ist der Fall, wenn die Route nicht zuvor vom entsprechenden Interface gelernt wurde. Für Aggregatknoten gilt, dass sie nur über ein Interface verbreitet werden dürfen, wenn keines der Teilnetze auf denen das Aggregat basiert vom entsprechenden Interface gelernt wurde. Zum besseren Verständnis sei erwähnt, dass sich der beschriebene Quellcode innerhalb einer Schleife über alle route_nodes der Routingtabelle befindet. Die im Quelltext enthaltenen Continue-Anweisungen beziehen sich auf diese Hauptschleife, ein Continue bedeutete also, dass die Überprüfung an der Stelle abgebrochen werden soll und das Netz des route_nodes kein Bestandteil des Update-Pakets ist und die Überprüfung mit dem nächsten route_node weiter geht.

Sollte es sich beim untersuchten Knoten um einen Aggregatknoten handeln, werden in den Zeilen 3-13 die route_nodes ttmp und rtmp so gesetzt, das ttmp der Kindknoten ganz links des Aggregats ist und rtmp der Kindknoten ganz rechts. Die Schleife in Zeile 17 prüft, ob der aktuell behandelte route_node eine RIP-Info besitzt, die das Interface enthält für das gerade das Update-Paket generiert wird und es sich gleichzeitig um eine gelernte Route (ZEBRA_ROUTE_RIP) handelt. Außerdem werden Routen die direkt verbunden sind (ZEBRA ROUTE CONNECT) geprüft, ob das Prefix des aktuellen Interfaces mit dem Prefix der Route übereinstimmt. Sollte eine Bedingung davon erfüllt sein, wird die Variable loopbreak auf eins gesetzt und die While-Schleife somit verlassen und die übergeordnete Schleife über alle route_nodes mit dem nächsten route_node gestartet (Zeile 37). Somit wird der Aggregatknoten nicht ins Updatepaket aufgenommen, da dies erst später im Quellcode passieren würde und wir bereits zum Anfang der Hauptschleife zurück gesprungen sind um den nächsten Knoten zu behandeln. Sollten die Bedingungen jedoch nicht zutreffen, also es existiert keine Interfacegleichheit, dann wird innerhalb der While-Schleife durch die Funktion next_leaf_until(ttmp,rtmp) der gesamte Baum unterhalb des Aggregats traversiert.

Sollte die While-Schleife über den Baum unterhalb des Aggregats ohne Abbruch durchlaufen, wird in Zeile 41-45 das Aggregat in das Paket mit aufgenommen, indem der aktuell untersuchte route_node auf das äußerste rechte Kind gesetzt wird und die äußere Schleife dadurch mit dem nächsten Knoten außerhalb des Aggregats fort fährt. Der Else-Zweig in Zeile 54-57 behandelt den Fall, wenn es sich in Zeile 01 nicht um einen Aggregatknoten gehandelt hat, dann wird dieser ebenfalls ins Update-Paket aufgenommen.

Betrachten wir nun erneut das bisher beschriebenen Netzwerkszenario mit Blick auf den Split-Horizon.

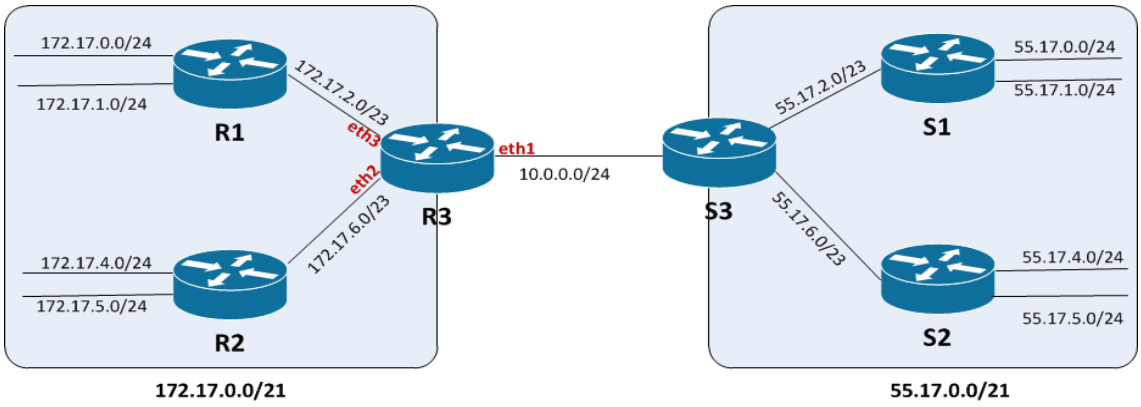



**Abb. 4.1 Netzwerkszenario mit gekennzeichneten Interfaces**

Die RIP-Routing-Tabelle baut sich für R3 folgendermaßen auf :

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

     Network          Next Hop        Metric From            Tag Time       Aggregator
C(i) 10.0.0.0/24      0.0.0.0              1 self              0  0
R(n) 55.17.0.0/21     10.0.0.2   eth1      3 10.0.0.2          0 00:26  0
C(s) 172.17.0.0/21    0.0.0.0              2 self              0  1
C(s) 172.17.0.0/22    0.0.0.0              2 self              0  1
R(n) 172.17.0.0/23    172.17.2.1 eth3      2 172.17.2.1        0 00:26  0
C(i) 172.17.2.0/23    0.0.0.0    eth3      1 self              0  0
C(s) 172.17.4.0/22    0.0.0.0              2 self              0  1
R(n) 172.17.4.0/23    172.17.6.1 eth2      2 172.17.6.1        0 00:26  0
C(i) 172.17.6.0/23    0.0.0.0    eth2      1 self              0  0
```

**Abb. 4.2 RIP-Tabelle des Routers R3 mit gekennzeichneten Interfaces**

Anschließend wurde mit Wireshark ein RIP-Update aller drei Interfaces des Routers R3 mitgeschnitten um zu überprüfen, ob die Split-Horizon-Implementation korrekt arbeitet.

R3-eth1

```
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 172.17.0.0, Metric: 2
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 172.17.0.0 (172.17.0.0)
      Netmask: 255.255.248.0 (255.255.248.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 2
```

R3-eth2

```
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 10.0.0.0, Metric: 1
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 10.0.0.0 (10.0.0.0)
      Netmask: 255.255.255.0 (255.255.255.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 1
  IP Address: 55.17.0.0, Metric: 3
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 55.17.0.0 (55.17.0.0)
      Netmask: 255.255.248.0 (255.255.248.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 3
  IP Address: 172.17.0.0, Metric: 2
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 172.17.0.0 (172.17.0.0)
      Netmask: 255.255.252.0 (255.255.252.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 2
```

R3-eth3

```
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 10.0.0.0, Metric: 1
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 10.0.0.0 (10.0.0.0)
      Netmask: 255.255.255.0 (255.255.255.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 1
  IP Address: 55.17.0.0, Metric: 3
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 55.17.0.0 (55.17.0.0)
      Netmask: 255.255.248.0 (255.255.248.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 3
  IP Address: 172.17.4.0, Metric: 2
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 172.17.4.0 (172.17.4.0)
      Netmask: 255.255.252.0 (255.255.252.0)
      Next Hop: 0.0.0.0 (0.0.0.0)
      Metric: 2
```

**Abb. 4.3 Wiresharkmitschnitt der RIP-Updates**

Über das Interface eth1 wird wie erwartet lediglich das Aggregat 172.17.0.0/21 gesendet. 55.17.0.0/21 fehlt, da dieses Netz über eth1 erlernt wurde. 10.0.0.0/24 ist direkt an eth1 angebunden und wird daher auch nicht weiter geleitet.
Über eth2 werden 10.0.0.0/24 und 55.17.0.0/21 unverändert gesendet das Aggregat 172.17.0.0/21 wird jedoch aufgespalten, da 172.17.4.0/23 über eth2 gelernt wurde und 172.17.6.0/23 direkt an eth2 angebunden ist. Somit wird über eth2 nur 172.17.0.0/22 bekannt gemacht.
Aufgrund der Topologie stellt es sich für eth3 ähnlich dar, 10.0.0.0/24 und 55.17.0.0/21 werden unverändert weitergereicht, da sie über eth1 gelernt wurden. Das Aggregat 172.17.0.0/21 wird erneut aufgespalten, und 172.17.4.0./22 wird bekannt gemacht, da die beiden Netze, die 172.17.0.0/22 bilden, von eth3 gelernt wurden.

Die bisherigen Änderungen führen zu einer funktionierenden Routenaggregation in Netzen die keine topologischen Schleifen aufweisen. Sobald es jedoch in einem aggregierbaren Bereich eine Netzwerkschleife gibt, stellt sich kein konvergenter Zustand mehr in den Routingtabellen ein. Es kommt zum Counting To Infinity der Aggregate, deren Metriken nach und nach hochzählen und sich regelmäßig gegenseitig aus den Routingtabellen löschen. Trotz allem funktioniert das Paket-Forwarding quer durchs Netz abgesehen von kurzen Unterbrechungen. Um diesen inkonsistenten Zustand der Routingtabellen zu beheben, wird eine weitere Bedingung eingeführt. Ab jetzt werden keine Knoten mehr direkt unterhalb gelernter Knoten eingefügt, somit nimmt ein Router beispielweise das Netz 172.17.0.0/24 nicht mehr entgegen, wenn er bereits das Netz 172.17.0.0/21 kennt. Um dies zu realisieren ist eine Änderung in der Funktion rip_rte_process() nötig. Diese Funktion wird aufgerufen, wenn eine Route der Routingtabelle hinzugefügt werden soll.

```
01.    struct route_node *rpparent;
02.    struct rip_info *parentinfo;
03.    if (rp && rp->parent) {
04.        rpparent = rp->parent;
05.        if (rpparent->info)
06.            parentinfo = rpparent->info;
07.        if ((rpparent->aggregator == 0) && (parentinfo)
08.                && (parentinfo->metric < rip->infinity_metric))
09.            return 0;
10.    }
```

Die Erweiterung wurde direkt zu Beginn der Funktion rip_rte_process() eingefügt. Wir prüfen ob ein Parent-Knoten für den einzufügenden Knoten(rp) existiert und wenn dieser ein gelernter Knoten ist, also das Aggregator-Flag auf 0 gesetzt ist, er eine RIP-Info besitzt und außerdem seine Metrik kleiner als RIP-Infinity ist, wird die Funktion verlassen und der Knoten wird nicht eingefügt.

Die letzte Version des Quellcodes ist unter https://userpages.uni-koblenz.de/~chenke/rmti verfügbar.

# 5. Evaluation der Routenaggregation

Um das Protokollverhalten bei vorhandenen Schleifen zu testen, verwenden wir das bisher benutzte Netzwerkszenario, welches allerdings nun durch einige Netzwerkschleifen erweitert wird.

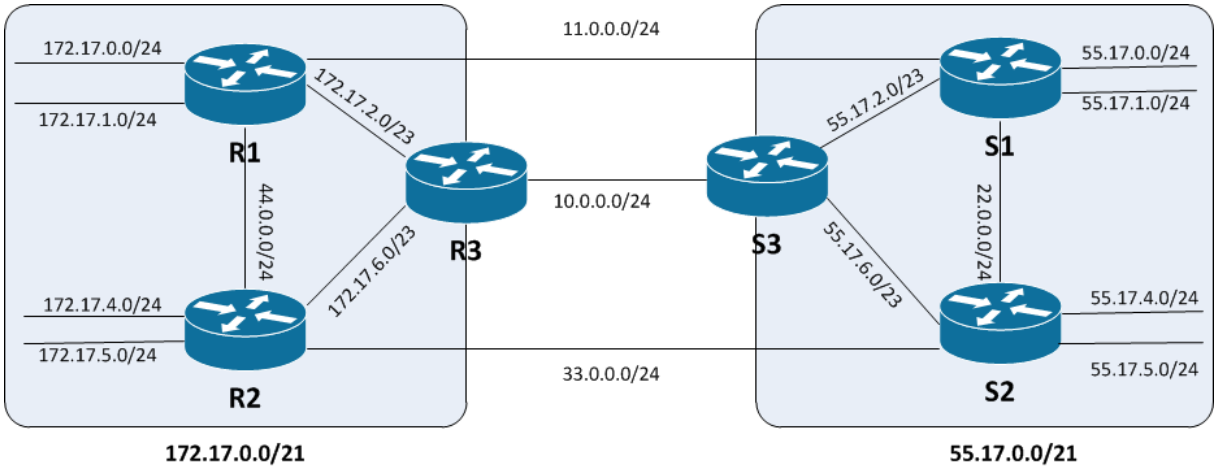



**Abb. 5.1 Topologie mit Aggregationsmöglichkeit und vorhandenen topologischen Schleifen**

Nachdem das Netz einen konvergenten Zustand erreicht hat, sieht die RIP-Routingtabelle von R3 abgesehen von den neu gelernten Netzen, die die Schleifen bilden nach wie vor so aus wie in der schleifenlosen Topologie.

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

     Network          Next Hop        Metric From           Tag Time       Aggregator
C(i) 10.0.0.0/24      0.0.0.0              1 self             0  0
R(n) 11.0.0.0/24      172.17.2.1           2 172.17.2.1       0 00:30  0
R(n) 22.0.0.0/24      10.0.0.2             3 10.0.0.2         0 00:28  0
R(n) 33.0.0.0/24      172.17.6.1           2 172.17.6.1       0 00:27  0
R(n) 44.0.0.0/24      172.17.2.1           2 172.17.2.1       0 00:30  0
R(n) 55.17.0.0/21     10.0.0.2             3 10.0.0.2         0 00:28  0
C(s) 172.17.0.0/21    0.0.0.0              3 self             0  1
C(s) 172.17.0.0/22    0.0.0.0              2 self             0  1
R(n) 172.17.0.0/23    172.17.2.1           2 172.17.2.1       0 00:30  0
C(i) 172.17.2.0/23    0.0.0.0              1 self             0  0
R(n) 172.17.4.0/22    172.17.2.1           3 172.17.2.1       0 00:30  0
C(i) 172.17.6.0/23    0.0.0.0              1 self             0  0
Router>
```

**Abb. 5.2 RIP-Tabelle von R3 in der Topologie von Abb. 5.1**

Nun senden wir einen Traceroute von R1 an S2-eth1, also an die IP 55.17.4.1, um zu sehen, ob die Pfadwahl die kürzeste ist.

```
R1:~# traceroute 55.17.4.1
traceroute to 55.17.4.1 (55.17.4.1), 30 hops max, 40 byte packets
 1  11.0.0.2 (11.0.0.2)  0.235 ms  0.142 ms  0.158 ms
 2  55.17.4.1 (55.17.4.1)  0.259 ms  0.208 ms  0.156 ms
R1:~#
```

**Abb. 5.3 Routenverfolgung von R1 nach 55.17.4.1**

Zu sehen ist, dass R1 einen der zwei für ihn günstigsten Pfade durchs Netz wählt.
Um nun die Reorganisationsfähigkeit des Netzes zu testen, deaktivieren wir das Interface eth4 an S1 unsere Topologie hat sich also folgendermaßen verändert.

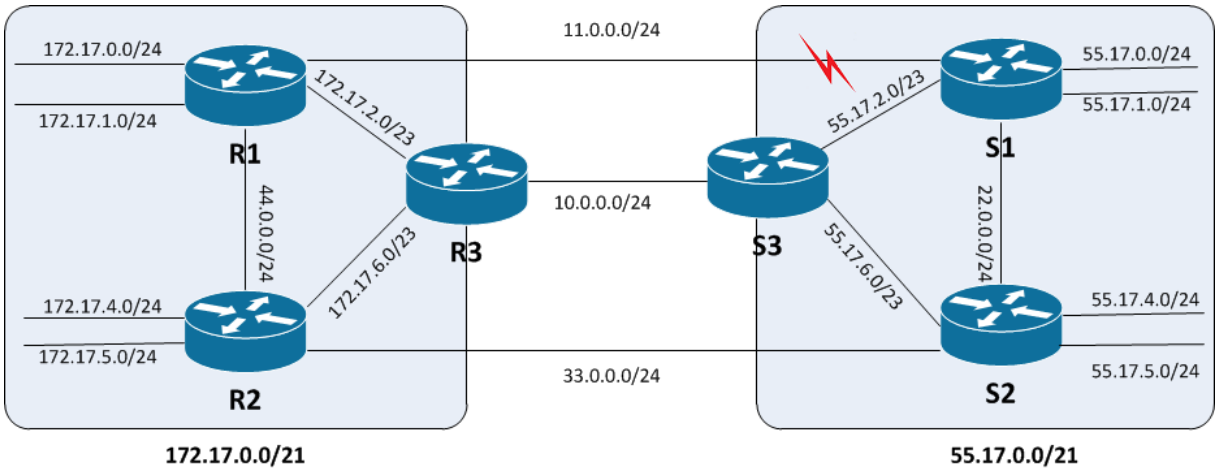



**Abb. 5.4 Verbindungsunterbrechung zwischen R1 und S1**

Die nächste Routenverfolgung zu 55.17.4.1 bringt nun das folgende Ergebnis :

```
R1:~# traceroute 55.17.4.1
traceroute to 55.17.4.1 (55.17.4.1), 30 hops max, 40 byte packets
 1  172.17.2.2 (172.17.2.2)  21.939 ms  0.193 ms  0.074 ms
 2  10.0.0.2 (10.0.0.2)  11.749 ms  0.360 ms  0.201 ms
 3  55.17.4.1 (55.17.4.1)  17.842 ms  0.528 ms  0.398 ms
R1:~#
```

**Abb. 5.5 Routenverfolgung von R1 nach 55.17.4.1 nach Reorganistation**

Die Reorganisation hat also funktioniert und die ausgefallene Route wurde durch eine Alternative ersetzt. Interessant ist hierbei, dass obwohl das eigentliche Ziel über R2 schneller zu erreichen ist, dennoch der längere Weg über R3 und S3 benutzt wird. Der Hintergrund dieser Wahl liegt in den Routingtabellen von R3 und R2, da diese beide nicht mehr das Teilnetz 55.17.4.0/24 kennen sondern nur noch das aggregierte Supernetz 55.17.0.0/21. Und dieses ist in R3 mit der Metrik 3 hinterlegt, während R2 für dieses Netz die Metrik 4 angibt.

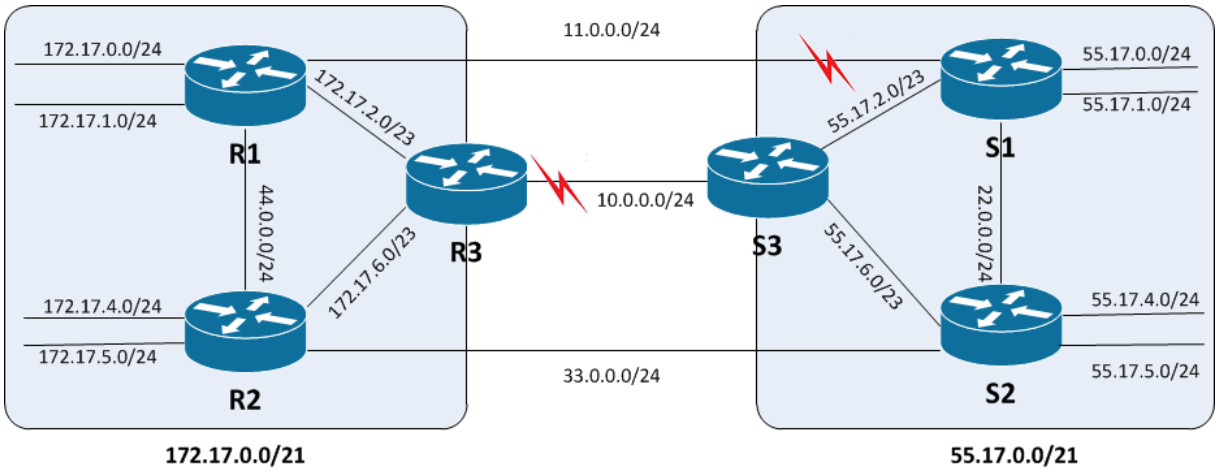



**Abb. 5.6 Verbindungsunterbrechung zwischen R3 und S3**

Um eine weitere Reorganisation zu erzwingen, wird nun wie in Abbildung 5.6 zu sehen das Interface eth1 von R3 deaktiviert, sodass die einzige noch zur Verfügung stehende Verbindung zwischen R1 und S2 über R2 führt.

```
traceroute to 55.17.4.1 (55.17.4.1), 30 hops max, 40 byte packets
 1  44.0.0.2 (44.0.0.2)  0.377 ms  2963.804 ms  1.642 ms
 2  55.17.4.1 (55.17.4.1)  30.819 ms  16.973 ms  0.859 ms
R1:~#
```

**Abb. 5.7 Routenverfolgung von R1 nach 55.17.4.1 nach erneuter Reorganistation**

Wie der Screenshot in Abbildung 5.7 zeigt, funktioniert auch die letzte mögliche Reorganisation und das Ziel bleibt somit erreichbar. Nachdem auch die letzte Verbindung zwischen R1 und S2 (R2-eth4 IP 33.0.0.1) deaktiviert wurde, verliert R1 (genau wie R2 und R3 (siehe Abbildung 5.8)) jegliche Erreichbarkeitsinformationen ins Netz 55.17.0.0/21.

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

     Network            Next Hop         Metric From            Tag Time        Aggregator
R(n) 11.0.0.0/24        172.17.2.1            2 172.17.2.1        0 00:25  0
R(n) 44.0.0.0/24        172.17.6.1            2 172.17.6.1        0 00:25  0
C(s) 172.17.0.0/21      0.0.0.0               2 self              0  1
C(s) 172.17.0.0/22      0.0.0.0               2 self              0  1
R(n) 172.17.0.0/23      172.17.2.1            2 172.17.2.1        0 00:25  0
C(i) 172.17.2.0/23      0.0.0.0               1 self              0  0
C(s) 172.17.4.0/22      0.0.0.0               2 self              0  1
R(n) 172.17.4.0/23      172.17.6.1            2 172.17.6.1        0 00:25  0
C(i) 172.17.6.0/23      0.0.0.0               1 self              0  0
Router>
```

**Abb. 5.8 RIP-Tabelle von R3 nach Trennung aller Verbindungen zu 55.17.0.0/21**

Ebenso befindet sich in den Routingtabellen der Router S1 - S3 nach Ablauf des Timeouts keine Information mehr über 172.17.0.0/21 oder eines der Teilnetze.

# 6. Schleifenerkennung des RMTI bei aktivierter Aggregation

Bisher wurde gezeigt, dass die Routingfunktionalität des RMTI trotzt aktivierter Aggregation erhalten bleibt. Nach einigen Tests in unterschiedlichen Szenarien wurde allerdings klar, dass dies für die Schleifenerkennung, die zur Verhinderung des Counting To Infinity Problems benötigt wird nur in bestimmten Konstellationen gilt. Sobald alle Netze innerhalb einer topologischen Schleife Teil eines Aggregats sein können ist die Schleifenerkennung ausgehebelt.

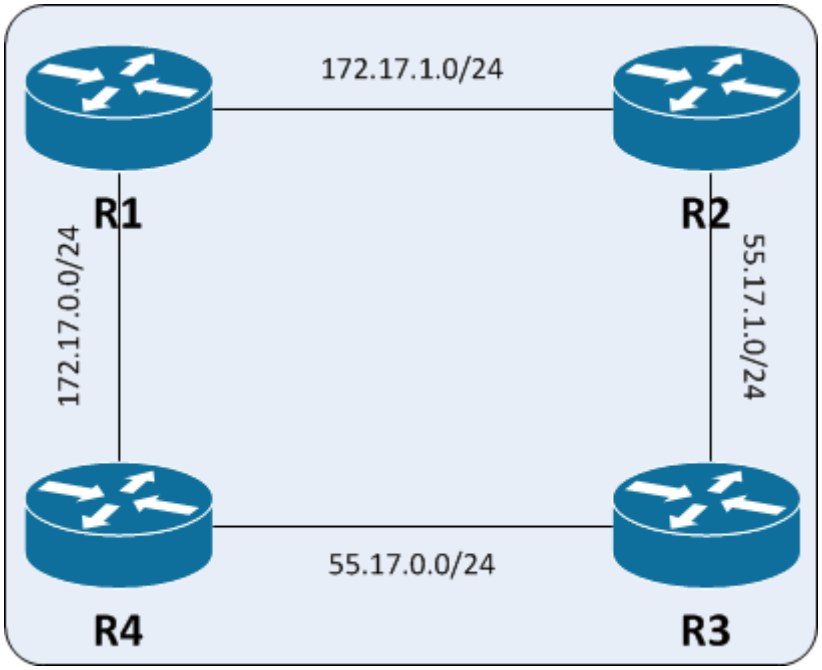



**Abb. 6.1 Ringtopologie mit Aggregationsmöglichkeiten**

Betrachten wir die Topologie in Abbildung 6.1, es handelt sich um eine Netzwerkschleife bestehend aus vier Netzen, die zu zwei Aggregaten zusammen gefasst werden können. Die RMTI-Routingtabellen bauen sich nach dem Erreichen der Konvergenz wie folgt auf :

**Routingtabelle von R1 :**

```
     Network          Next Hop        Metric From           Tag Time       Aggregator
R(n) 55.17.0.0/23     172.17.1.2           2 172.17.1.2       0 00:29  0
C(s) 172.17.0.0/23    0.0.0.0              1 self             0  1
C(i) 172.17.0.0/24    0.0.0.0              1 self             0  0
C(i) 172.17.1.0/24    0.0.0.0              1 self             0  0
Router>
```

**Routingtabelle von R2 :**

```
     Network          Next Hop        Metric From           Tag Time       Aggregator
C(s) 55.17.0.0/23     0.0.0.0              1 self             0  1
R(n) 55.17.0.0/24     55.17.1.2            2 55.17.1.2        0 00:29  0
C(i) 55.17.1.0/24     0.0.0.0              1 self             0  0
C(s) 172.17.0.0/23    0.0.0.0              1 self             0  1
R(n) 172.17.0.0/24    172.17.1.1           2 172.17.1.1       0 00:30  0
C(i) 172.17.1.0/24    0.0.0.0              1 self             0  0
Router>
```

**Routingtabelle von R3 :**

```
     Network          Next Hop        Metric From           Tag Time       Aggregator
C(s) 55.17.0.0/23     0.0.0.0              1 self             0  1
C(i) 55.17.0.0/24     0.0.0.0              1 self             0  0
C(i) 55.17.1.0/24     0.0.0.0              1 self             0  0
R(n) 172.17.0.0/23    55.17.1.1            2 55.17.1.1        0 00:24  0
Router>
```

**Routingtabelle von R4 :**

```
     Network          Next Hop        Metric From           Tag Time       Aggregator
C(s) 55.17.0.0/23     0.0.0.0              1 self             0  1
C(i) 55.17.0.0/24     0.0.0.0              1 self             0  0
R(n) 55.17.1.0/24     55.17.0.1            2 55.17.0.1        0 00:28  0
C(s) 172.17.0.0/23    0.0.0.0              1 self             0  1
C(i) 172.17.0.0/24    0.0.0.0              1 self             0  0
R(n) 172.17.1.0/24    172.17.0.1           2 172.17.0.1       0 00:28  0
Router>
```

Um eine vorhandene Netzwerkschleife mit dem RMTI zu erkennen ist es nötig, dass auf mindestens zwei Netzwerkinterfaces das selbe Netz über ein Routingupdate empfangen wird.

Betrachtet wird im folgenden der Router R4, der die IP-Adressen 172.17.0.2 und 55.17.0.2 an seinen beiden Netzwerk-Interfaces besitzt. R4 erhält seine Routingupdates von R1 und R3. Aus der Tabelle von R1 ist ersichtlich, dass R1 das Netz 55.17.0.0/23 und 172.17.1.0/24 an R4 schicken wird. Es wird nicht das komplette Aggregat an R4 gesendet, denn 172.17.0.0/24 ist mit dem Interface, über das R1 das Update versendet, direkt verbunden, weshalb das Split-Horizon-Verfahren dies verhindert. Auf dem Netzwerkinterface von R4, welches mit R3 verbunden ist treffen Updates mit Informationen über 172.17.0.0/23 und 55.17.1.0/24 ein, da R4 eine direkte Verbindung mit 55.17.0.0/24 hat und somit nur einen Teil des Aggregates erhält. Somit kommen an beiden Interfaces jeweils unterschiedliche Netze an, die vom RMTI nicht für die Schleifenerkennung benutzt werden können. Eine Überprüfung der MSILM-Tabelle belegt dies, da hier keine Einträge zu finden sind.

Die Überlegung, die Schleifenerkennung so abzuändern, dass eine Schleife erkannt wird, sobald an einem Interface ein Netz wie beispielweise 172.17.0.0/23 und auf einem anderen ein Teil dieses Netzes, wie 172.17.0.0/24 anliegt, würde dazu führen, dass es in einer Schleife

passieren kann, dass zwischen zwei Interfaces die nicht in der selben Schleife liegen eine vermeintliche Verbindung erkannt wird, obwohl diese Schleife gar nicht existiert.

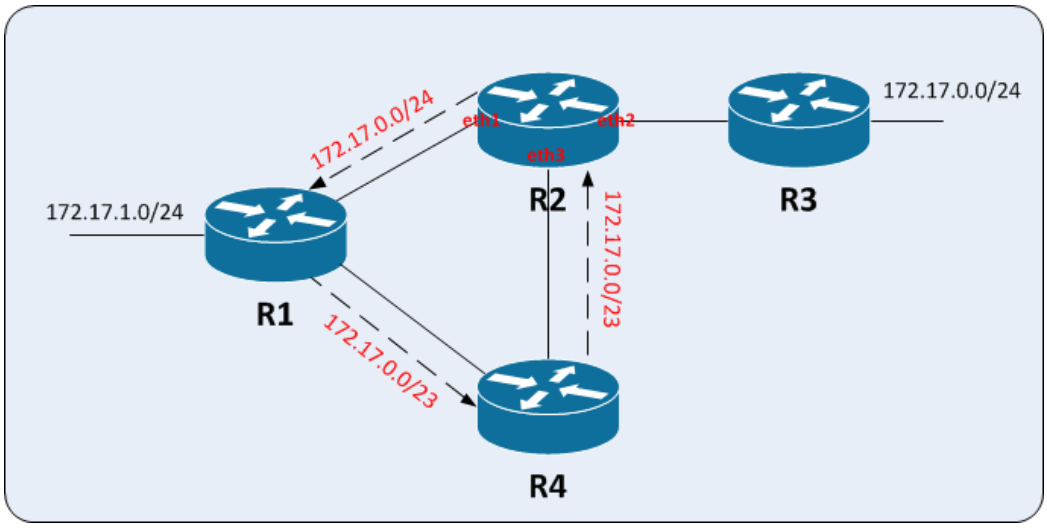



**Abb. 6.2 Topologie mit Problemen bei der Schleifenerkennung**

Dies würde in der Topologie in Abbildung 6.2 passieren. Der Router R2 sendet das Netz 172.17.0.0/24 an R1, welcher nun in der Lage ist zu aggregieren. Das Aggregat 172.17.0.0/23 wird an R4 versendet und anschließend an den Router R2 weiter geleitet, der es auf dem Interface eth2 empfängt. Wäre die Schleifenerkennung auf Teilnetze ausgeweitet, würde R2 nun eine Schleife zwischen eth2 und eth3 vermuten, da 172.17.0.0/24 ein Teilnetz von 172.17.0.0/23 ist. Daher wurde dieser Ansatz in dieser Arbeit nicht weiter verfolgt.

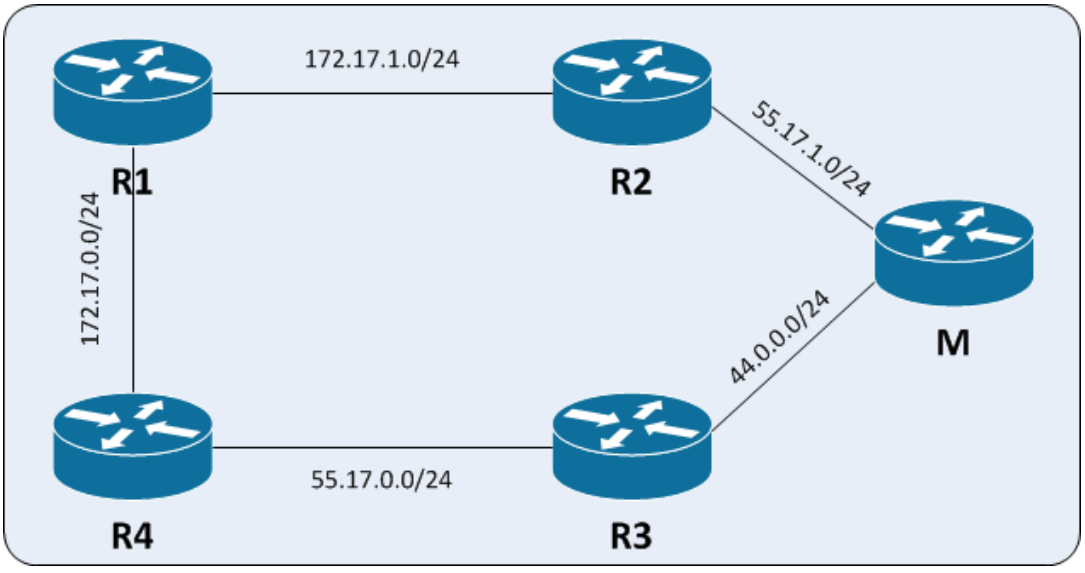



**Abb. 6.3 Netzwerkschleife mit einem nicht aggregierbarem Netz**

Sobald wir nun jedoch wie in Abbildung 6.3 ein Netz, welches nicht aggregierbar ist in die Schleife mit aufnehmen, funktioniert die Erkennung wieder wie aus der ursprünglichen Variante des RMTI gewohnt. Dies gilt allerdings nur, wenn für die Metrikbestimmung der Aggregate jeweils die Metrik des Teilnetzes mit der größeren Metrik benutzt wird. Nur so ist gewährleistet, dass sich das nicht aggregierte Netz gegenüber dem Aggregat durchsetzen

kann. Bei gemittelter Metrik oder der kleinsten aus beiden Teilnetzen, wird die Schleife unter Umständen als zu klein erkannt und es werden gültige Alternativrouten abgelehnt. Ebenso kann es passieren, dass bei der kleinsten oder der gemittelten Metrik Problemen bei der Schleifenverhinderung entstehen, da beide Methoden die Metrik der Ursprungsnetze mitunter stark verzerren können.

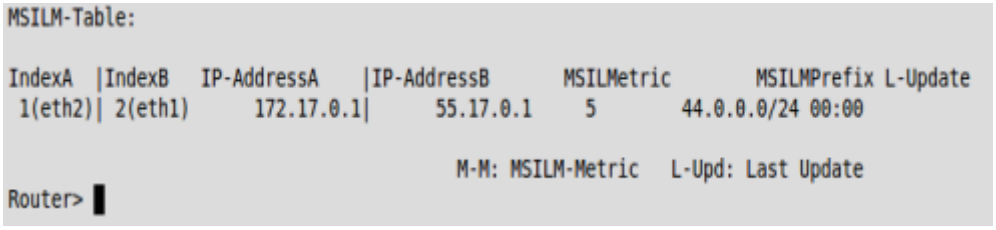



**Abb. 6.4 MSILM-Tabelle des Routers R4 in der Topologie aus Abb. 6.2**

Abbildung 6.4 zeigt die MSILM-Tabelle von R4, auf Basis des Netzes 44.0.0.0/24 wurde die Schleifenlänge korrekt erkannt. Da ein Routingalgorithmus jedoch mit den minimal möglichen Einschränkungen funktionieren soll, wäre es auch denkbar, den Zustand der nicht aggregierbaren Netze in topologischen Schleifen durch den RMTI selbst herbei zu führen. Es wäre möglich jedem Router die IP seines ersten Netzwerk-Interfaces als Subnetz mit der Maske /30 einzutragen und diese aktiv im Netz zu verbreiten. Gleichzeitig wäre es nötig eine Aggregation von Netzen mit der Netzmaske von /30 zu unterbinden und die Schleifenerkennung des RMTI nur noch auf eben diese Netze anzuwenden. Allerdings würde auf diese Art der Routingtraffic wieder ansteigen und es käme zum Wachstum der Routingtabellen, da für jeden Router im Netz ein neuer Eintrag pro Updatepaket und ein Eintrag in die Routingtabelle nötig ist. Dies würde dem Konzept der Router-ID von Link-State-Verfahren nahe kommen, da so jeder Router im Netz eindeutig identifizierbar ist.

VNUML bietet eine Möglichkeit diesen Gedankengang zu verfolgen ohne eine vollständige Implementierung vornehmen zu müssen. Jeder UML-Client eines VNUML-Netzwerkes verfügt über ein Management-Netz beginnend mit 192.168.100.0/30 mit dem er an den Host-Computer angebunden ist. Bisher wurden diese Netze vom Routing durch eine Änderung in der ripd.conf ausgenommen, da sie für die Bearbeitung der Fragestellung keinerlei Relevanz besaßen. Im Nachfolgenden werden die Management-Netze fürs Routing wieder frei gegeben und der Aggregationsprozess wird so abgeändert, dass sie von der Aggregation ausgenommen werden und die Schleifenerkennung nur noch auf Basis dieser Netze geschieht. Bei der MSILM-Berechnung muss nun noch eins von der Metrik abgezogen werden, da sich die Netze für die Berechnung der Schleifen jetzt nicht mehr innerhalb der Schleife, sondern einen Hop außerhalb befinden.

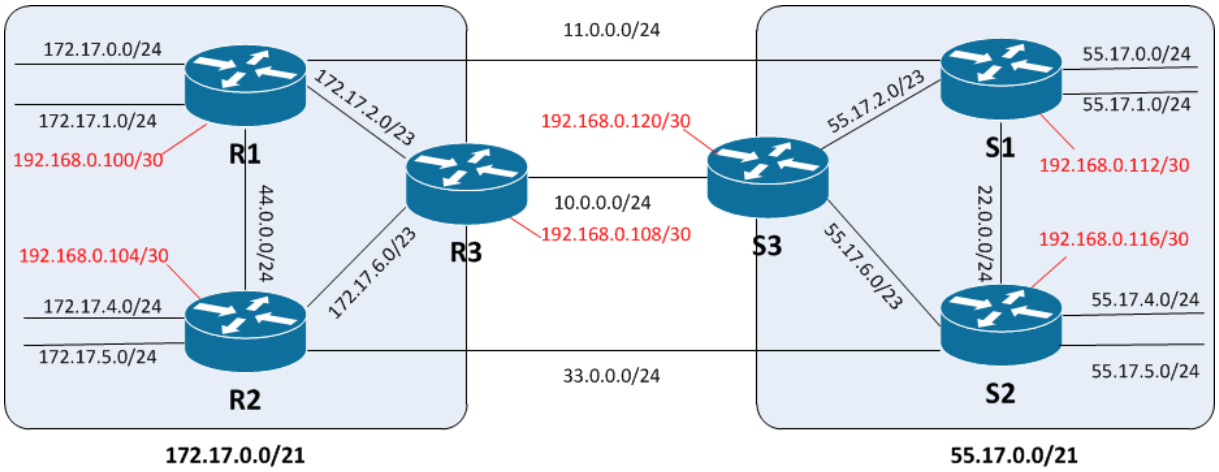



**Abb. 6.5 schleifenbehaftete, aggregierbare Netztopologie mit gekennzeichneten Management-Netzen**

Die in Abbildung 6.5 dargestellte Topologie wurde um die aktivierten Management-Netze erweitert und die Schleifenberechnung des RMTI wie zuvor erwähnt angepasst.

```
MSILM-Table:

IndexA  |IndexB   IP-AddressA    |IP-AddressB      MSILMetric       MSILMPrefix L-Update
 1(eth3)| 2(eth5)      172.17.2.2|       44.0.0.2     3   192.168.0.108/30 00:00
 1(eth3)| 3(eth4)      172.17.2.2|       11.0.0.2     4   192.168.0.120/30 00:02
 2(eth5)| 3(eth4)       44.0.0.2|       11.0.0.2     4   192.168.0.116/30 00:00

                                    M-M: MSILM-Metric   L-Upd: Last Update

Router>
```

**Abb. 6.6 MSILM-Tabelle berechnet auf Basis der Management-Netze**

Die MSILM-Tabelle in Abbildung 6.6 zeigt, dass die Schleifenerkennung trotz Aggregation durch den Umweg über die Management-Netze funktioniert. Die Schleife zwischen 172.17.2.1 und 44.0.0.1 wurde als Schleife R1→R3→R2→R1 mit Metrik 3 erkannt. Ebenso wurde zwischen 172.17.2.1 und 11.0.0.1 die kleinste mögliche Schleife mit Metrik 4 und dem Verlauf R1→R3→S3→S1→R1 korrekt erkannt und auch die Verbindung zwischen 44.0.0.1 und 11.0.0.1 wurde über R1→R2→S2→S1→R1 mit Metrik 4 richtig berechnet. Ein Counting to Infinity ließ sich auf diese Weise trotz vieler Versuche nicht mehr erzeugen.

```
▽ Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  ▷ IP Address: 10.0.0.0, Metric: 2
  ▷ IP Address: 11.0.0.0, Metric: 1
  ▷ IP Address: 22.0.0.0, Metric: 2
  ▷ IP Address: 33.0.0.0, Metric: 2
  ▷ IP Address: 44.0.0.0, Metric: 1
  ▷ IP Address: 55.17.0.0, Metric: 3
  ▷ IP Address: 172.17.1.0, Metric: 1
  ▷ IP Address: 172.17.2.0, Metric: 1
  ▷ IP Address: 172.17.4.0, Metric: 2
  ▷ IP Address: 192.168.0.100, Metric: 1
  ▷ IP Address: 192.168.0.104, Metric: 2
  ▷ IP Address: 192.168.0.108, Metric: 2
  ▷ IP Address: 192.168.0.112, Metric: 2
  ▷ IP Address: 192.168.0.116, Metric: 3
  ▷ IP Address: 192.168.0.120, Metric: 3
```

**Abb. 6.7 RIP-Update über eth1 von R1**

Durch die Verwendung der Management-Netze zur Schleifenerkennung wird der Grundgedanke der Aggregation jedoch unterlaufen. Denn das Ziel dahinter war es den Netzwerktraffic zu reduzieren und die Routingtabellen möglichst klein zu halten. Mit der Einführung der Management-Netze wachsen die RIP-Updates nun aber um jeweils einen zusätzlichen Netzeintrag je verwendetem Router im Netz, wie in Abbildung 6.6 dargestellt. Es stellt sich nun die Frage, ob

es wirklich nötig ist zur Schleifenerkennung alle Management-Netze in den RIP-Updates mitzuführen, oder ob sich Bedingungen finden lassen unter denen ein Management-Netz ab einem gewissen Punkt nicht mehr weiter propagiert werden muss und die Schleifen-erkennung dennoch funktioniert. Aus diesem Grund wird die Schleifenerkennung ohne Blick auf die Aggregats- und Transitnetze betrachtet. Für die Schleifenerkennung werden nur die Netze benötigt, die auch selbst innerhalb der Schleife liegen.

Bezogen auf die Topologie in Abbildung 6.8, benötigen die Router R1, R2, R3 und R4 die Managementnetze 192.168.0.116/30 und 192.168.0.120/30 nicht, da diese zu R5 und R6 gehören und außerhalb der Schleife liegen. Ebenso sind für R5 und R6 die Management-Netze von R1,R2 , R3 für eine Schleifenerkennung nicht nutzbar und bilden somit unnötigen Overhead.

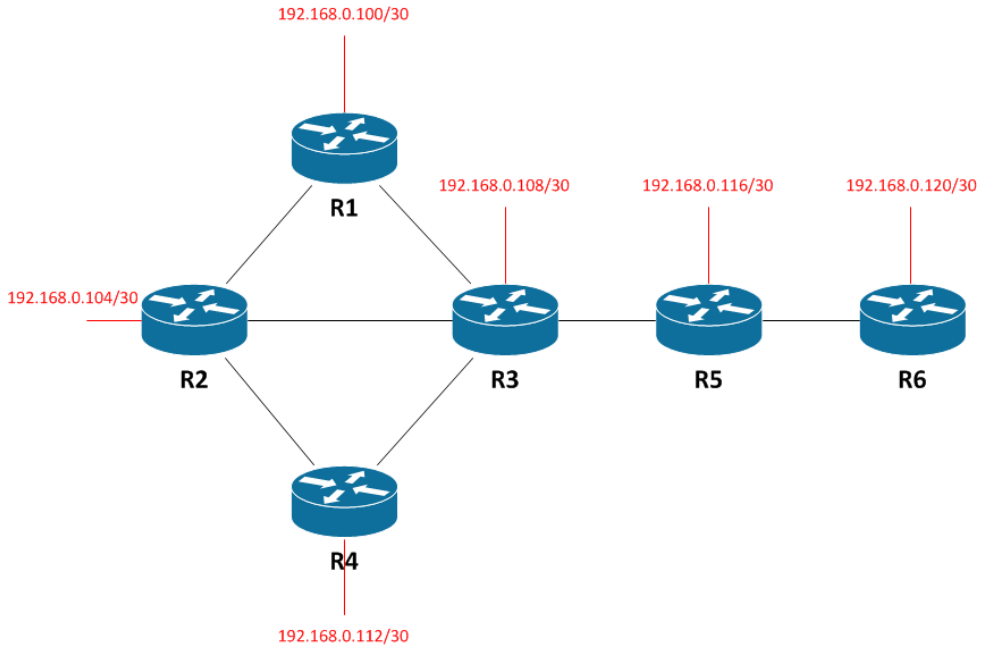



**Abb. 6.8 Schleifentopologie mit Management-Netzen**

Ein Management-Netz, welches an zwei oder mehr Interfaces empfangen wird, soll nicht über Interfaces versendet werden, über das es nicht ebenfalls empfangen wurde, da dieses Interface nicht Bestandteil der Schleife sein kann die durch eben dieses Netz erkannt wird. Bezogen auf die Topologie in Abbildung 6.8 würde der Router R3 auf diesem Weg die Netze 192.168.0.100/30, 192.168.0.104/30 und 192.168.0.112/30 nicht an R5 weiter reichen, da diese mehrfach an seinen eigenen Interfaces anliegen. Um die Netzwerklast durch die Managementnetze weiter zu reduzieren, wäre es zudem möglich, die Weiterleitung für Managementnetze nach einer gewissen Konvergenzzeit für alle Interfaces zu unterbinden, für die keine topologische Schleife durch den RMTI erkannt wurde. Durch diese Änderung würden die Router R1, R2, R3, R4 keine Informationen über 192.168.0.116/30 und 192.68.0.120/30 bekommen. Sollte der Fall eintreten, dass ein neuer Router ins Netz integriert wurde und sich eventuell durch diesen eine neue topologische Schleife gebildet hat, muss der Konvergenz-Timer zurück gesetzt werden und jeder Router muss kurzfristig erneut alle Management-Netze aktiv im Netz veröffentlichen, bis davon auszugehen ist, dass die Schleifenerkennung wieder konvergent ist. Auf diese Weise lässt sich trotz aggregierter Netze eine Schleifenerkennung über die Management-Netze realisieren ohne zu viel zusätzlichen

Routing-Overhead zu produzieren. So werden nach der Konvergenzphase nur noch die Management-Netze innerhalb der zu erkennenden Schleife propagiert, die zur Schleifenerkennung eben dieser Schleife benötigt werden.

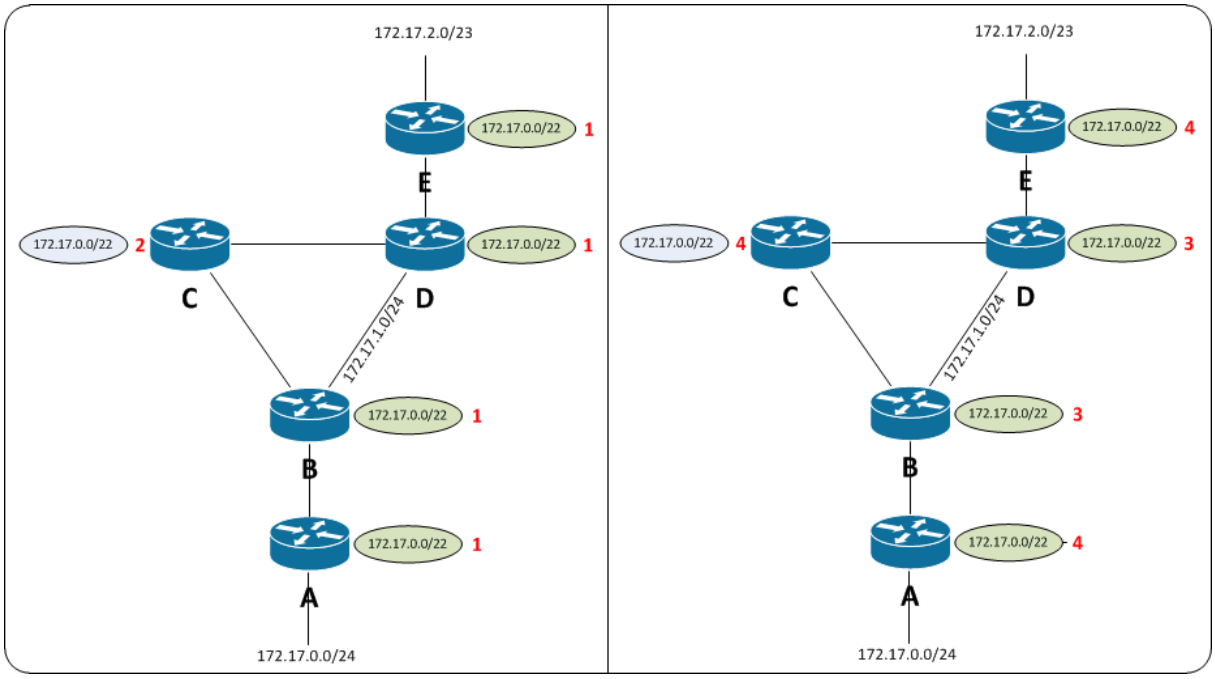



**Abb. 6. 9 Topologie mit minimaler/maximaler Metrik**

Zuletzt soll an dieser Stelle auf die Wahl der Metrik für die Aggregate eingegangen werden. Auf den ersten Blick scheint es drei ähnlich sinnvolle Möglichkeiten zu geben, die kleinste, die gemittelte und die größte Metrik der Teilnetze dem erzeugten Aggregat zu zuweisen. In Abbildung 6.9 ist die selbe Topologie einmal mit kleinster und einmal mit größter Metrik dargestellt. Sollte nun das Netz 172.17.0.0/24 ausfallen, spielt sich unter Umständen im linken Szenario der Abbildung, bei Benutzung der kleinsten Metrik folgendes ab. Router A setzt B,D und E darüber in Kenntnis, dass 172.17.0.0/24 unerreichbar ist, woraufhin diese ihre Aggregate auflösen. Router C erreicht diese Information jedoch nicht rechtzeitig, sodass er das geroutete Netz 172.17.0.0/22 welches er von D gelernt hat, an Router B verschickt. Router B überprüft nun ob er 172.17.0.0/22 annehmen kann, indem er alle Teilnetze, die in seiner Routingtabellenstruktur unter 172.17.0.0/22 eingruppiert sind und mit einer RIP-Infinity-Metrik versehen sind den MTI-Tests unterzieht. Die Schleifenerkennung hat für das Interface auf dem das Routingupdate empfangen wird, dank der Managementnetze eine kleinste Schleife mit Metrik 3 erkannt. Bei der Überprüfung zwischen dem Aggregat und dem ausgefallenem Netz 172.17.0.0/24, welches mit Metrik 16 in der Tabelle geführt ist und einen old_metric-Wert von 2 besitzt, wird folgende Rechnung vollzogen :

$$mrpm(3) + metric\_old(2) > metric\_new(3) \quad \textbf{\textit{wahr}}$$

Somit wird Router B 172.17.0.0/22 annehmen, wenn für die Metrikbestimmung der Aggregate der kleinere Wert der beiden Teilnetze übernommen wird. Ebenso ist es möglich Szenarien zu entwerfen, in denen dies auch bei der gemittelten Metrik geschieht. Wenn Router B im rechten Teil der Abbildung 6.9. nach einem Ausfall von 172.17.0.0/24 das Netz 172.17.0.0/22 angeboten bekommt, sieht der Test folgendermaßen aus :

*mrpm(3) + metric_old(2) > metric_new(5)* ***falsch***

Router B lehnt dieses Netz also ab und verbreitet es somit auch nicht aktiv weiter im Netzwerk. Somit kann kein Counting to Infinity entstehen.

# 7. Fazit & Ausblick

Die ursprüngliche Vermutung, dass die Routenaggregation ohne tiefergehende Änderungen am RMTI implementiert werden kann stellte sich im Laufe der Arbeit als nicht haltbar heraus. Dennoch scheint ein Weg gefunden worden zu sein, der die Vorteile der Routingschleifen-verhinderung und der Routenaggregation vereint. Die besprochene Möglichkeit der Erkennung von topologischen Schleifen über die Zuhilfenahme von Management-Netzen ermöglicht es auch in aggregierten Netzen Schleifen effektiv zu erkennen. Dadurch das die Weiterleitung dieser Netze nur unter bestimmten Gesichtspunkten nötig ist, entsteht nach der Konvergenzzeit lediglich in einer topologischen Schleife ein etwas erhöhter Routing-Traffic, welcher in Abhängigkeit zum Schleifenumfang steht. Durch die Wahl der größten Teilnetz-Metrik für das erzeugte Aggregat kann das CTI-Problem auch bei Aggregaten verhindert werden. Der RMTI war bisher bereits in der Lage einen der schwerwiegendsten Nachteile des RIPv2-Protokolls, nämlich das CTI-Problem, zufriedenstellend aufzuheben. Dadurch ist es ebenso möglich die Limitierung von maximal 15 Hops im Netz aufzuheben und somit größere Netze zu ermöglichen. Die Vorteile die RIPv2 bietet, nämlich ein robustes und leicht nachvollziehbares Protokollverhalten und einen ressourcenschonenden Einsatz ohne viel Konfigurationsaufwand kann der RMTI in gleichem Maße bieten. Die Erweiterung um die automatische Routenaggregation stattet den RMTI mit einem Feature aus, welches in dieser Art in moderne Routingprotokolle, zum jetzigen Zeitpunkt, noch keinen Eingang gefunden hat. Die Vorteile der dynamischen Aggregation, wie schnelleres Paket-Forwarding, schlankere und damit besser nachvollziehbare Routingtabellen und weniger Routing-Traffic liegen allerdings klar auf der Hand, sodass davon auszugehen ist, dass auch andere Protokolle wenn es ihre Architektur zulässt, dieses Feature in der Zukunft bieten werden. Die durchgeführten Änderungen am RMTI zur Einführung der dynamischen Routenaggregation bei gleichzeitiger Aufrechterhaltung der Routingschleifenverhinderung lassen das ursprünglich leicht verständliche, von Andreas Schmid 1999, vorgestellte Konzept hinter dem RMTI zunehmend komplexer werden. Dennoch ist ein mit dem RMTI betriebener Router nach wie vor in der Lage in einem Netz bestehend aus RIPv2-Routern reibungslos zu funktionieren, da immer noch das in RFC2453 spezifizierte RIPv2-Nachrichtenformat benutzt wird. Ob der RMTI in der Zukunft durch weitere Features ergänzt wird und sich noch stärker als bisher zu einem ernsthaften Konkurrenten der modernen Protokolle entwickelt wird die Zeit zeigen. Für zukünftige Erweiterungen sollte allerdings eine Änderung des Nachrichten-formats erwogen werden, um beispielsweise die maximale Netzgröße von 15 Hops aufzuheben und beispielweise Informationen über Routenauslastungen zu transportieren.

# Literaturverzeichnis


**[Schmid99]** Andreas Schmid, RIP-MTI: Minimum-effort loop-free distance vector routing algorithm, Diplomarbeit, 1999, Universität Koblenz

**[Bohdan08]** Frank Bohdanowicz, Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons, Diplomarbeit, 2008, Universität Koblenz

**[Klee01]** Thomas Kleemann, RIPeval - Evaluierung und Weiterentwicklung des RIP-MTI-Algorithmus,Diplomarbeit, 2001, Universitat Koblenz

**[FrGeHu10]** Franck Ley, Geoffrey G., Xie Hui Zhangz, Understanding Route Aggregation, 2010, School of Computer Science Carnegie Mellon University Pittsburgh

**[Allision99]** Allision Loyd, PATRICIA, 1999, Monash University http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/PATRICIA/

**[KhBr11]** Milad Khojasteh-Samiei, Andreas Brandt, Automatisierte Testumgebung für Routingprotokolle und Route-Aggregation in der Quagga Software Routing Suite, Forschungsbericht , 2011, Universität Koblenz

**[RFC1058]** C. Hedrick, RFC 1058 - Routing Information Protocol, 1988, IETF

**[RFC1338]** V. Fuller, T. Li, J. Yu, K. Varadhan, RFC 1338 - Supernetting: an Address Assignment and Aggregation Strategy, 1992, IETF

**[RFC1519]** V. Fuller, T. Li, J. Yu, K. Varadhan, RFC 1519 - Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy, 1993, IETF

**[RFC2453]** G. Malkin, RIP Version 2, RFC 2453 - RIP Version 2, 1998, IETF

**[RFC4632]** V. Fuller, T. Li, RFC 4632 - Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan, 2006, IETF

**[wiki1]** Routing Information Protocol, Webseite, 2012. verfügbar online auf http://de.wikipedia.org/wiki/Routing_Information_Protocol besucht am 09. Juni 2013.

**[wiki2]** BGP Table growth, Webseite, 2012. verfügbar online auf http://en.wikipedia.org/wiki/File:BGP_Table_growth.svg besucht am 09. Juni 2013.

**[wiki3]** administrative Distanz, Webseite, 2012. verfügbar online auf http://de.wikipedia.org/wiki/Administrative_Distanz besucht am 09. Juni 2013.

**[cisco]** What Is Administrative Distance?, Webseite, 2013. verfügbar online auf http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094195.shtml besucht am 09. Juni 2013.