Universität Koblenz
Fachbereich 4
Institut für Informatik
Arbeitsgruppe Rechnernetze

Diplomarbeit zur Erlangung des akademischen Grades
„Diplom-Informatiker“

Evaluation & Convergence-Analysis of RIP with metric based Topology Investigation

Manuel Kober
Matr.-Nr.: 201210270

Verantwortlicher Hochschullehrer: Prof. Dr. Christoph Steigner
Hochschulbetreuer: Diplom-Inform. Frank Bohdanowicz

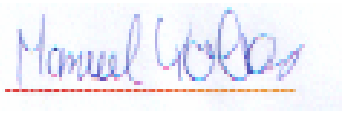
Koblenz, August 2009

Selbständigkeitserklärung

Ich, Manuel Kober, versichere an Eides statt durch eigenhändige Unterschrift, die vorliegende Arbeit selbständig und ohne Benutzung anderer als der aufgeführten Quellen und Hilfsmittel verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, habe ich als solche kenntlich gemacht. Ich weiß, dass bei Abgabe einer falschen Versicherung die Prüfung als nicht bestanden zu gelten hat. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mit der Einstellung dieser Diplomarbeit in die Bibliothek bin ich einverstanden. ☒ Ja ☐ Nein

Der Veröffentlichung dieser Diplomarbeit im Internet stimme ich zu. ☒ Ja ☐ Nein



Manuel Kober
Koblenz, den 24.08.2009

Inhaltsverzeichnis

Abbildungsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Das häufig eingesetzte Routing Information Protocol (RIP) zur Verwaltung autonomer Systeme im Internet existiert bereits seit den Anfängen des Internets. Mit dessen Ausbreitung und der einhergehenden dichteren Vernetzung des Internets, wurde die Einführung von Mechanismen zur Erkennung von Schleifeninformationen immer bedeutender. So wie der konkurrierende Algorithmus (OSPF) anderer Softwareentwickler, muss sich RIP mit Lösungen zu diesem Thema auseinandersetzen, da Schleifen in Netzwerken erhebliche Probleme verursachen können. Die eingeführten Mechanismen von RIP bergen einige Nachteile für die maximale Größe und die Konvergenz eines Netzwerkes. Der RIPMTI-Algorithmus wurde entwickelt um die Schleifenerkennung in Rechnernetzen zu verbessern. Ein Schleifenproblem, das als Count-To-Infinity-Problem bezeichnet wird, führt dazu, dass ein Netzwerk nur sehr langsam in einen konvergenten Zustand gelangt. Der Hopcount-Wert 16, der als künstliche Beschränkung des Count-To-Infinity-Effekts eingeführt wurde, beschränkt auch die maximale Topologie-Tiefe eines Netzwerkes. Diese Arbeit soll zeigen wie sich eine Erhöhung des Hopcount-Wertes und darüber hinaus die Manipulation der Kommunikationsintervalle jedes Routers auf die Konvergenzzeiten unterschiedlicher Netzwerk-Szenarien auswirkt. Ziel dieser Arbeit ist es, die Konvergenzeigenschaften des RIPMTI-Algorithmus durch umfangreiche Simulationen in vielen unterschiedlichen Netzwerk-Topologien zu untersuchen. Es gilt festzustellen, welchen Einfluss ein steigender Hopcount und beschleunigte Kommunikationsintervalle auf die Stabilität eines Netzwerkes haben und ob der RIPMTI-Algorithmus effektiv und effizient funktioniert.

1.1 Motivation

Zur Analyse einzelner Netzwerk-Topologien waren umfangreiche Vorbereitungen nötig. Da der Einsatz in der eigentlichen Zielumgebung aus kosten- und administrativen Gründen unpraktikabel ist, musste eine alternative und realitätsnahe Netzwerk-Simulationsumgebung geschaffen werden, die aus VNUML, , inklusive einem RIP-Routing-Daemon, und XT-Peer besteht. Die hohen Anschaffungspreise von realer Hardware lohnen sich zwar für den realen Betrieb, nicht aber wenn es ausschließlich um Testzwecke geht. Außerdem muss die Funktionsweise des RIPMTI-Algorithmus ausreichend geklärt werden, da dessen implementierte Erweiterungen über das Verständnis des ursprünglichen RIP-

Algorithmus hinausgehen. Eine Herausforderung bestand darin, kritische Router-Konstellationen und Verbindungsleitungen zu erstellen, in denen das Verhalten von RIPMTI getestet werden sollte. Konvergenzprobleme werden durch ungünstige Sendereihenfolgen von Updatenachrichten verursacht. Ein Netzwerkszenario wird daher von XT-Peer und automatisch ausgeführten Konfigurationsdateien gesteuert, um lange andauernde Tests unter immer gleichen Bedingungen durchzuführen, bei denen der Count-To-Infinity-Effekt mit hoher Wahrscheinlichkeit auftreten sollte. Aufgrund dieser Ausgangsbasis konnten mehrstündige Stress-Tests gestartet werden. Hintergrund ist, repräsentative Ergebnisse unter Variation verschiedener Parameter in einer Laborumgebung zu ermitteln und das Verhalten von RIP und RIPMTI zu vergleichen. Daraus sollen Erkenntnisse gewonnen werden, ob der RIPMTI-Algorithmus in kritischen Netzwerk-Topologien zum einen Routingschleifen zuverlässig verhindert und zum anderen eine Verbesserung der Konvergenzeigenschaften bewirkt. Die rein software-gesteuerte Simulation ermöglicht darüber hinaus eine rasche Implementation von Verbesserungen und einen unkomplizierten Einbau während der Testphasen.
Ziel dieser Diplomarbeit ist die Evaluation des RIPMTI-Algorithmus und eine umfangreiche Konvergenzanalyse. RIPMTI soll in vielen denkbaren Topologien auf seine Funktion überprüft werden. Dazu müssen XML-Dateien und Konfigurationsdateien erstellt werden. Anhand gesammelter Routinginformationen, die durch den XT-Peer im laufenden Testbetrieb erfasst werden, soll eine Analyse der Konvergenzzeiten stattfinden, indem der Count-To-Infinity-Effekt künstlich hervorgerufen wird. Die Auswertung dieser Zeiten gibt Aufschluss darüber ob ein Count-To-Infinity-Effekt entstanden ist und wie lange er gedauert hat bzw. ob er durch RIPMTI verhindert werden konnte. Darüber hinaus besteht die Möglichkeit, Topologien und Ursachen zu erkennen, in denen RIPMTI nicht funktioniert, um darauf aufbauend den Algorithmus zu verbessern. Weiterhin soll diese Diplomarbeit zeigen, welche Auswirkungen eine Erhöhung des Hop-Counts auf 64 und eine Beschleunigung der Kommunikationsintervalle von Routern untereinander haben.

1.2 Aufbau der Arbeit

Im zweiten Kapitel dieser Diplomarbeit werden die Grundlagen zur Netzwerksimulation erklärt. Hier findet sich eine Beschreibung des linuxbasierten Basissystems VNUML. Es werden außerdem die Vorgänge der Kommunikations-

Schnittstellen dargestellt. Zum einen die Testumgebung XT-Peer und zum anderen der Routing-Daemon Quagga. Das Kapitel 3 liefert eine ausführliche Präsentation der Funktionen des RIP-Algorithmus, mit all seinen Möglichkeiten und Einschränkungen. Im vierten Kapitel wird auf die Erweiterungen von RIP eingegangen, die unter der neuen Bezeichnung RIPMTI zu finden sind. Kapitel 5 beschreibt die Vorgehensweise jeder Testphase. Kapitel 6 enthält die ausführliche Beschreibung und Auswertung der Analysedaten aus den Testreihen und unterteilt sich in alle untersuchten Netzwerk-Topologien. Kapitel 7 soll abschließend ein Fazit über die Konvergenzzeiten des RIPMTI-Algorithmus abgeben und es soll eine Bewertung seiner Zuverlässigkeit und Leistungsfähigkeit mit einem Ausblick auf die Weiterentwicklung erfolgen.

1.3 Terminologie

- Netz: Ein Netz beschreibt hier einen zusammenhängenden IP-Adressbereich aus einem größeren Pool von IP-Adressen. Subnetzmasken teilen den Pool in Teilnetze auf, die dann einer eigenständigen Administration unterliegen können. Ein solches Teilnetz wird als Autonomes System bezeichnet.
- Netzwerk: Ein Netzwerk beschreibt die gesamte Topologie aller Netze, die über Knotenpunkte miteinander verbunden sind und darüber kommunizieren können. Knotenpunkte dienen zum Datenaustausch autonomer Systeme.
- Router: Router stellen die Knotenpunkte zwischen autonomen Systemen dar. Sie verwalten über Routingtabellen Einträge über die Wege, die ein Datenpaket vom Absender zum Empfänger gehen muss.
- Interface: Jeder Router besitzt ein oder mehrere Interfaces, die als Schnittstelle zwischen einem Netz und einem Router dienen. Typischerweise handelt es sich dabei um einen physikalischen Netzwerkanschluss, der i.d.R. per Kabelverbindung mit einem Netz verbunden ist.
- Konvergenz: Jeder Routingalgorithmus soll nach einer Änderung der Netzwerk-Topologie so schnell wie möglich in einen stabilen Zustand zurückkehren. Der Zustand gilt als stabil oder konvergent, wenn jeder Router eines Netzwerkes zu jedem angeschlossenen Netz eine gültige Route besitzt. Die laufende Zeit bis zu diesem Zustand wird als Konvergenzzeit bezeichnet. Sie ist abhängig von der Ausdehnung (Anzahl der Router) im Netzwerk.

2. Basiswissen

Die theoretischen Grundlagen des RIP- und RIPMTI-Algorithmus werden in späteren Kapiteln vorgestellt. Jetzt ist zu klären, wie eine geeignete Simulationsumgebung in der Praxis aussieht. Echte Router-Hardware ist aus Kostengründen und wegen des Administrationsaufwandes nicht akzeptabel. Das folgende Kapitel stellt die verwendete Software vor, mit der die Konvergenzanalyse durchgeführt werden kann. Als Grundlage dient die Virtualisierungssoftware VNUML, mit deren Hilfe virtuelle Linux-Rechner aufgebaut werden können. Die Routing-Software-Suite Quagga enthält den RIP-Routing-Daemon, der um die Funktionalitäten des RIPMTI erweitert wurde. Das Steuerungs- und Analyseprogramm XT-Peer dient dabei der Steuerung des RIP-Daemons und protokolliert die gemessenen Daten zur späteren Auswertung.

2.1 Virtual Network User Mode Linux (VNUML)

VNUML dient zur Simulation von Rechnernetzen und basiert auf der Virtualisierungstechnik *User Mode Linux (UML)*. Mit UML (s. [UML09]) können Linux-Kernels, Distributionen und beliebig andere Linux-Software auf einem physischen Hostrechner mit einem bereits laufenden Linux-Betriebssystem auf ihr Verhalten hin getestet werden, ohne dabei die Stabilität oder Funktion des existierenden Systems zu riskieren. Es ist möglich, mehrere virtuelle Instanzen eines Kernels parallel auf einem Linux-Hostrechner zu starten und jeder Instanz individuell Ressourcen zuzuweisen. Jeder UML-Rechner stellt eine virtuelle Maschine dar, die mit anderen UML-Rechnern über echte Verbindungsprotokolle kommuniziert. Sie besteht aus einem eigenen UML-Kernel und einem UML-Root-Filesystem, das in Form einer Image-Datei vorliegen muss. Jeder UML-Rechner soll einen Router repräsentieren. So entsteht ein realistisches Netzwerkszenario, in dem sich alle Teilnehmer wie echte autonome Rechner verhalten. In dieser Arbeit wurde die VNUML Version 1.8 eingesetzt. Als UML-Kernel dient *„linux-2.6.18.1-bb2-xt-4m“* und als Dateisystem *„ripmti-64-vnuml18.img“*. Ein UML-System verhält sich sehr realitätsnah und ist sehr gut zum Testen von Routingprotokollen geeignet. Es können komplexe Netzwerkumgebungen mit VNUML [VNU07] erstellt werden. In dieser Arbeit wurden mehrere unterschiedliche Netzwerk-Topologien erarbeitet und mit Hilfe von VNUML je ein XML-Szenario dazu konstruiert, um den RIPMTI-Daemon von Quagga darin zu testen. Die verwendete Beschreibungssprache ist XML [XML98]. Das Grundgerüst einer XML-Datei besteht aus drei Arten von

Tags. Strukturelle Tags dienen dem groben Aufbau der Datei. Darin werden zum einen Topologie Tags definiert, die die Topologie eines Netzwerkes beschreiben und zum anderen Simulations Tags, die als Kommando interpretiert und ausgeführt werden können. In einem XML-Szenario sind die Interfaces jedes Routers, die zugehörigen IP-Adressen und weitere Kommandoparameter für auszuführende Programme jedes UML-Rechners enthalten. Zusätzlich werden globale Konfigurationsdaten über das zu nutzende Dateisystem oder den Kernel und sämtliche Netze und Verbindungen definiert. Das folgende Beispiel beschreibt wie man eine Szenariodatei erstellt. Zwei Rechner sind mit je zwei Rechnernetzen verbunden, von denen sie eines gemeinsam haben.
Zuerst werden die globalen Einstellungen vorgenommen.

- Die verwendete VNUML-Versionsnummer, gefolgt von der Bezeichnung des Szenarios.
- Ein passwortgeschützter Zugang muss vorher noch über das Kommando

  ```
  sudo su ssh-keygen – t rsa
  ```

  angelegt werden. Das ist ein öffentlicher SSH-Schlüssel des Hostrechners. Er wird in einer eigenen Datei gespeichert, sodass eine ständige Abfrage des Passwortes entfällt.
- Aktiviert man das Tag <automac/> wird jedem Interface der UML-Rechner automatisch eine zufällige MAC-Adresse zugewiesen.
- Die nächste Einstellung dient der Kontrolle der virtuellen Maschinen vom Host-Rechner aus. Das Netz 192.168.0.0/24 ist das Managementnetz des eigenen Rechners, das den Zugriff auf die UML-Rechner gewährleistet. Jeder UML-Rechner sollte einen festen Adressbereich daraus erhalten.
- Das Tag <host_mapping> ist notwendig, damit die Hostnamen der virtuellen Maschinen in die Datei „/etc/host“ eingetragen werden und der XT-Peer die entsprechenden IP-Adressen zuordnen kann. Standardmäßig werden die UML-Rechner nur über die IP-Adresse angesprochen. Der Einfachheit halber sind die Hostnamen kurz und eindeutig gewählt.
- Die Pfade zum nutzenden Dateisystem und dem verwendeten Kernel sind ebenfalls notwendig.
- Das <Basedir> verweist auf einen globalen Erreichbarkeitspfad der Datei „ripd_conf“ des Routing-Daemons jedes UML-Rechners, sodass spezielle Routingeinstellungen geladen werden können.
- Das Xterm stellt ein Zugriffsterminal dar, mit dem man sich die laufende Konfiguration ansehen und Veränderungen vornehmen kann.

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

<vnuml>
  <global>
    <version>1.8</version>
   <simulation_name>testszenario</simulation_name>
   <ssh_version>2</ssh_version>
   <ssh_key>/root/.ssh/identity.pub</ssh_key>
   <automac/>
   <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</filesystem>
     <kernel>/usr/share/vnuml/kernels/linux</kernel>
     <basedir>/home/timbo/Diplom/</basedir>
     <console id="0">xterm</console>
     <!--xterm>gnome-terminal,-t,-x</xterm-->
     <forwarding/>
   </vm_defaults>
  </global>
```

Im nächsten Abschnitt werden die Netze zwischen UML-Rechnern beschrieben.

- Ein kurzer Bezeichner kennzeichnet das einzelne Netz.
- Der Modus und Typ beschreiben die Schnittstelle zwischen dem Hostrechner und den virtuellen Rechnern. Als „virtual bridge“ oder alternativ „uml-switch“ können alle Interfaces überwacht und ausgelesen werden. Die Auswahl ist abhängig davon, ob man die virtuellen Rechner über ein physisches Interface des Hostrechnern mit einem externen Netzwerk (z.b. Internet) verbinden will oder nicht.

```
<!-- Networks -->
  <net name="netA" mode="virtual_bridge" type="lan" />
  <net name="netB" mode="virtual_bridge" type="lan" />
  <net name="netC" mode="virtual_bridge" type="lan" />
```

Über den <vm>-Tag werden die virtuellen Maschinen konfiguriert.

- Jeder UML-Rechner erhält Interfaces <if id="x">, die seine Funktion als Router beschreiben sollen. Ein Interface stellt die Verbindung zwischen dem Router und einem Netz dar und umfasst die Schnittstellennummer (z.b. eth1), IP-Adresse (im IPv4-Format) sowie die Subnetzmaske und den Namen des Netzes, mit dem er verbunden werden soll. Die IP-Adresse muss sich vom Adressbereich des Management-Netzwerkes unterscheiden, da dieser Bereich nicht für die Analyse der Routinginformationen vorgesehen ist und vom XT-Peer ignoriert wird. So wird ein virtueller Rechner eindeutig einem Netz zugeordnet.
- Der <filetree>-Befehl kann optional aktiviert werden. Wenn alle Router die gleiche „ripd.conf"-Datei und „zebra.conf"-Datei verwenden sollen, bleibt der Befehl deaktiviert und es wird die Konfiguration aus dem <filesystem> übernommen. Für den Fall, dass ein Router spezielle Eigenschaften besitzen soll, verweist der Pfad in Abhängigkeit vom bereits erwähnten <basedir> auf eine individuelle Konfiguration. Im Rahmen dieser Arbeit wurden in der Konfigurationsdatei „ripd.conf" unter anderem die Timer-Einstellungen verändert, die für den Nachrichtenaustausch der Router untereinander verantwortlich sind.
- Alle Kommandos mit einem <exec>-Präfix dienen dem Starten und dem Abbruch der Routing-Daemons auf jedem UML-Rechner. Sie enthalten die Parameter zur Ausführung der Daemons.

```
<!-- Nodes -->

  <vm name="R1">

    <if id="1" net="netA">
      <ipv4 mask="255.255.255.0">10.0.1.1</ipv4>
    </if>
    <if id="2" net="netB">
      <ipv4 mask="255.255.255.0">10.0.2.1</ipv4>
```

```
    </if>
                <forwarding/>

              <filetree seq="start"
root="/etc/quagga">Modelle/testszenario/config/R1</filetree>

                <exec seq="start" type="verbatim">zebra -f /etc/quagga/zebra.conf -
d</exec>
                <exec seq="start" type="verbatim">ripd -f /etc/quagga/ripd.conf -x 5000 -
d</exec>
                <exec seq="stop" type="verbatim">killall zebra</exec>
                <exec seq="stop" type="verbatim">killall ripd</exec>
                <exec seq="start" type="verbatim">sysctl -w
net.ipv4.conf.all.rp_filter=0</exec>
</vm>
<vm name="R2">

    <if id="1" net="netB">
      <ipv4 mask="255.255.255.0">10.0.2.2</ipv4>
    </if>
    <if id="2" net="netC">
      <ipv4 mask="255.255.255.0">10.0.3.1</ipv4>
    </if>
                <forwarding/>

              <filetree seq="start"
root="/etc/quagga">Modelle/testszenario/config/R2</filetree>

                <exec seq="start" type="verbatim">zebra -f /etc/quagga/zebra.conf -
d</exec>
                <exec seq="start" type="verbatim">ripd -f /etc/quagga/ripd.conf -x 5000 -
d</exec>
                <exec seq="stop" type="verbatim">killall zebra</exec>
                <exec seq="stop" type="verbatim">killall ripd</exec>
                <exec seq="start" type="verbatim">sysctl -w
net.ipv4.conf.all.rp_filter=0</exec>
</vm>
</vnuml>
```

Neben der VNUML-Sprache besteht Virtual Network User Mode Linux aus dem VNUML-Parser, der sich um die Umsetzung einer Szenario-Datei in eine laufende Simulation kümmert. Der Parser vnumlparser.pl ist ein in Perl geschriebenes Skript, das optional mit mehreren Parametern aufgerufen wird. Das Konsolenkommando

```
vnumlparser.pl -t testszenario.xml -vB -u root
```

analysiert die XML-Datei „testszenario.xml“ auf syntaktische Korrektheit. Der Parameter -t sorgt für den Aufbau der Netztopologie. Alle virtuellen Maschinen werden nacheinander gestartet (je eine pro Router) und entsprechend der Angaben aus der XML-Datei verbunden. Die Performance des Hostrechners ist beim Aufbau der Netztopologie ausschlaggebend. Je nach Komplexität des Szenarios kann es eine Weile dauern bis es vollständig gestartet wurde. Der Verbose Mode -v lässt die ausgeführten Kommandos ausführlich auf dem Bildschirm anzeigen. Die Option erweist sich als hilfreich beim Erkennen von Fehlfunktionen, die sonst verborgen bleiben. Der Blocking Mode -B ist wichtig, damit der Parser nach dem Starten jeder virtuellen Maschine wartet bis der zugehörige SSH-Server online ist. Ohne ihn ist eine Verbindung vom Hostrechner aus unmöglich. Das Kommando -u root ist unter der Linuxdistribution Kubuntu notwendig um den VNUML-Parser mit Administrationsrechten zu starten.
Nachdem das Szenario anhand der Topologie Tags aufgebaut wurde, folgt das nächste Kommando zum Abarbeiten der Simulations Tags.

```
vnumlparser.pl -x start@testszenario.xml -vB -u root
```

Alle Befehle eines UML-Rechnern, die ein <exec>-Tag besizen, werden im Modus -x vom Parser ausgeführt. Im Beispielszenario wurde die Kommandosequenz „start“ definiert, um die Routing-Daemons zu starten
Um ein Szenario komplett vom Hostrechner zu entfernen wird

```
vnumlparser -P testszenario.xml -vB -u root
```

eingegeben. Dabei werden nicht nur die Szenarios heruntergefahren, sondern auch alle verwendeten Ressourcen inklusive Dateisystem der virtuellen Maschinen gelöscht. Alle darin enthaltenen Informationen, die nicht separat gespeichert wurden, gehen dabei verloren. Dieser Vorgang ist nötig, um den Hostrechner für ein weiteres Szenario vorzubereiten und seine Ressourcen effektiv einzusetzen.

Virtual Network User Mode Linux stellt im Vergleich zu alternativen Netzwerk-Simulationsprogrammen, wie z.B. dem *Network-Simulator ns-2*, ein mächtiges Werkzeug dar. Der Installationsaufwand von VNUML ist gering. Die Syntax der

Beschreibungssprache eines XML-Szenarios ist leicht verständlich und schnell erlernbar, sodass sich mit relativ geringem Aufwand komplexe Netzwerkszenarien entwickeln lassen. Jede virtuelle Maschine lässt sich frei konfigurieren und verwalten. Ihr realitätsnahes Verhalten, ähnlich einem physischen Rechner, erleichtert und beschleunigt das experimentelle Arbeiten und Simulieren im Bereich Rechnernetze, ohne die Anschaffung teurer Hardware. Die einzigen Kritikpunkte liegen in der fehlenden Konfigurationsmöglichkeit von Verbindungsleitungen zwischen UML-Rechnern bzgl. Latenz oder Bandbreite und in der abnehmenden Performance des Host-Rechners bei zunehmender Größe eines Szenarios.

2.2 Quagga-Software

VNUML ist die Simulationsumgebung. Im nächsten Schritt muss die Frage geklärt werden, wie der RIPMTI-Algorithmus innerhalb dieser Umgebung getestet werden kann.
Quagga [QUA07] ist eine Open-Source Routing-Software-Suite. Sie besteht aus zwei Teilen. Einmal aus mehreren eigenständigen Clients, die verschiedene TCP/IP-basierte Routingdienste (Routing Daemon) zur Verfügung stellen, wie das Routing Information Protocol (RIP), Open Shortest Path First (OSPF) oder das Border Gateway Protocol (BGP). Der andere Bestandteil ist der *zebra* genannte Core-Daemon. Zebra ist die Schnittstelle zwischen den Routingdaemons und der vom Linux-Kernel verwalteten Routingtabelle, über die der Austausch von Routen verwaltet wird. Die Clients veranlassen entsprechend ihrer Implementation den Core-Daemon zum Verschicken von Routing-Updates. Quagga ist bereits in die Dateisysteme von VNUML integriert und muss nicht extra eingebunden werden. In der aktuellen Version enthält Quagga für das OSPFv2-Routing den *ospfd*, für RIPv1 und RIPv2 den *ripd*, für BGPv4+ den *bgpd* und für IPv6-Adressierung zusätzliche Dienste. In dieser Arbeit liegt der Fokus auf der Benutzung des RIP-Daemons. Über die Netzwerkkonsole XTerm ist der RIP-Daemon vom Hostrechner aus via Telnet-Verbindung ansprechbar. Wie im Kapitel zu VNUML bereits erwähnt, existieren zu jeder virtuellen Maschine je eine *ripd.conf* und *zebra.conf*, die in einem entsprechenden Verzeichnis, wie in der oben vorgestellten XML-Datei beispielsweise unter

```
/home/timbo/Diplom/Modelle/testszenario/config/R1
```

abgelegt sind. Der Zebra-Daemon (zebra.conf) enthält üblicherweise nur den

Hostnamen der virtuellen Maschine und ein Passwort für den Zugriff auf den Daemon.

```
! -*- zebra -*-
!
! zebra sample configuration file
!
hostname R1
password zebra
enable password zebra
```

Der RIP-Daemon umfasst umfangreichere Angaben, die an das Cisco IOS angelehnt sind. Neben dem Hostnamen und den Zugangsdaten wird mit *rip router* der RIP-Routing-Daemon aktiviert. Das Element *timers basic* ändert die Werte für T_U , T_{TT} und T_{GC} , die standardmäßig immer auf das Verhältnis 30 Sekunden, 180 Sekunden und 120 Sekunden eingestellt sind. Der Parameter *network* beschreibt die Netze bzw. die zugehörigen Interfaces, für die RIP aktiviert wird. Das Element *redistribute* legt fest, welche Routinginformationen im Netzwerk verschickt werden. In diesem Fall werden neben den selbst gelernten Routen auch statische und direkt angeschlossene Netze propagiert.

```
! -*- rip -*-
! RIP-MTI example
!
hostname R3
password zebra
!
debug rip events
debug rip packet
!
router rip
 timers basic 5 18 12
 network 10.0.1.0/24
 network 10.0.2.0/24
!
redistribute static
redistribute connected
!
!
log file /tmp/ripd.log
```

Darüber hinaus sind noch weitere Optionen der *ripd.conf* möglich, die aber für die Testdurchläufe unbedeutend sind. Über das Konsolenterminal kann die *ripd.conf* einer laufenden virtuellen Maschine angezeigt werden und „on-th-fly“ bearbeitet

werden. Änderungen werden in Echtzeit auf den Rechner übertragen. Der Quagga RIP-Daemon fungiert als Grundgerüst für die Implementierung des RIPMTI-Algorithmus. Ein Vorteil dieses Vorgehens ist, dass nur die wesentlichen Teile des ursprünglichen RIP-Algorithmus in [BOH08] verändert werden mussten. Dadurch bleibt die Kompatibilität beider Algorithmen weitestgehend vorhanden. Außerdem kann mithilfe des XT-Peer leicht das Verhalten von RIP und RIPMTI analysiert und verglichen werden. Der neue Quagga RIPMTI-Daemon enthält neben der Erweiterung um die RIPMTI-Funktionalitäten eine Schnittstelle zum Analyse- und Steuerungsprogramm XT-Peer. Dazu wurde der XT-Server hinzugefügt, der die Steuerungsbefehle des XT-Peer empfängt und verarbeitet, und der SL-Client, der die protokollierten Routinginformationen an den XT-Peer sendet.

2.3 XT-Peer

VNUML baut das Szenario zusammen und Quagga kümmert sich um das eigentliche Routing. Das Programm XT-Peer (externally Triggered Peer) steuert den RIPMTI-Daemon von außen und liest die Routinginformationen der Router mit. „Von außen“ bedeutet hier, dass der XT-Peer auf dem Hostrechner gestartet wird. Für die Kommunikation mit der virtuellen Maschine muss der Hostrechner über eine Netzwerkverbindung zu jedem Knoten verfügen. Der XT-Peer ist eine in Java implementierte Software-Entwicklung, die über mehrere Diplomarbeiten hinweg an der Universität Koblenz entstanden ist und weiterentwickelt wurde. In dieser Arbeit wurde der XT-Peer an spezielle Umgebungsvariablen angepasst und im Laufe der Analyse von Testdurchläufen weiter verbessert.
Abbildung 1 stellt die Funktionsweise des XT-Peer schematisch dar. Entsprechend zum XT-Server und SL-Client aus dem Quagga RIP-Daemon existiert im XT-Peer das jeweils passende Gegenstück, der XT-Client und der SL-Server. Die Kommunikationskette zwischen dem Hostrechner, auf dem der XT-Peer gestartet wird, und Quagga wird durch folgenden Ablauf bestimmt. Das von VNUML gestartete Szenario wird mit dem XT-Peer aufgerufen und graphisch dargestellt. Die XML-Datei wird ausgelesen und entsprechend der Angaben wird das Netzwerk eingebunden. Damit verfügt XT-Peer über die korrekten Management-IP-Adressen, um jeden einzelnen Router anzusprechen. Der XT-Client auf dem Hostrechner kann Kontrollkommandos an den XT-Server des Quagga RIP-Daemon senden, um das Routing zu steuern. Währenddessen sammelt der SL-Client auf jedem Router kontinuierlich alle Routinginformationen und sendet diese über die SL-Datenleitung an den XT-Server.

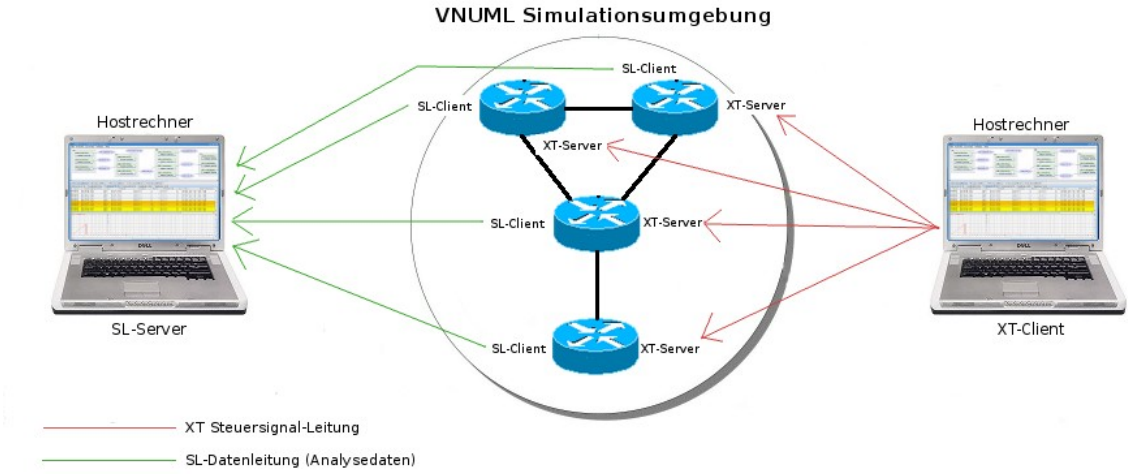
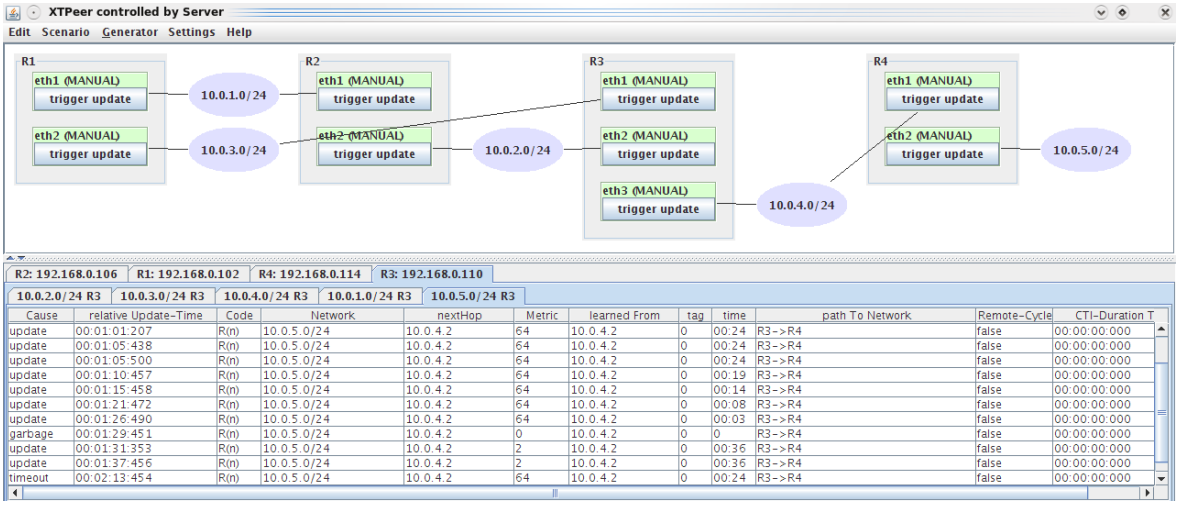


Abbildung 1: Funktionsweise des XT-Peer

Die Anzeige der XT-Peer-Oberfläche ist in zwei Teile gegliedert. Der obere Teil (siehe Abbildung 2 (oben)) zeigt die Topologie des Netzwerkes als graphisches Modell aller Router, Verbindungsleitungen und Netze. Jeder Router in diesem Modell kann „per Rechtsklick" konfiguriert werden. Das Konfigurationsmenü umfasst alle Einstellungen zum verwendeten Routingalgorithmus, z.B. die RIPMTI-Einstellungen. In der Graphik werden Schalter zu jedem Interface eines Routers angelegt, die einzeln „per Linksklick" aktiviert werden können. Ein aktivierter Schalter löst eine Aktion aus, die den XT-Client dazu veranlasst, dem XT-Server auf dem Router das entsprechende Steuersignal zu senden. So kann der Router z.B. dazu gezwungen werden, eine Updatenachricht zu versenden oder zu verhindern. Der XT-Server wertet die empfangenen Daten aus und stellt diese in tabellarischer Form im XT-Peer dar. Die Tabelle wird im unteren Teil



Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:01:01:207	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:24	R3->R4	false	00:00:00:000
update	00:01:05:438	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:24	R3->R4	false	00:00:00:000
update	00:01:05:500	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:24	R3->R4	false	00:00:00:000
update	00:01:10:457	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:19	R3->R4	false	00:00:00:000
update	00:01:15:458	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:14	R3->R4	false	00:00:00:000
update	00:01:21:472	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:08	R3->R4	false	00:00:00:000
update	00:01:26:490	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:03	R3->R4	false	00:00:00:000
garbage	00:01:29:451	R(n)	10.0.5.0/24	10.0.4.2	0	10.0.4.2	0	0	R3->R4	false	00:00:00:000
update	00:01:31:353	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	00:36	R3->R4	false	00:00:00:000
update	00:01:37:456	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	00:36	R3->R4	false	00:00:00:000
timeout	00:02:13:454	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:24	R3->R4	false	00:00:00:000

Abbildung 2: Screenshot der Programmoberfläche des XT-Peer

(siehe Abbildung 2 (unten) des XT-Peer angezeigt. Für jeden Router existiert ein eigener Registerkartenreiter.

Über einen „Rechtsklick“ auf ein Netz kann ein Netzgraph aufgerufen werden. Der Netzgraph zeigt die Werte der Tabelle in einem Routing-Diagramm. Dabei handelt es sich um ein Verlaufsdiagramm jedes Routers, das auf der X-Achse den zeitlichen Verlauf anzeigt. Dazu passend wird auf der Y-Achse die Metrik zum Netz (aus empfangenen Updatenachrichten) als Punkt markiert. Ein Count-To-Infinity-Effekt kann so im Diagramm (siehe Abbildung 3) anschaulich dargestellt werden. Über einen „Linksklick“ auf einen Punkt im Diagramm lassen sich detaillierte Informationen anzeigen, wie das Ergebnis der Schleifenüberprüfung und weitere Routinginformationen.

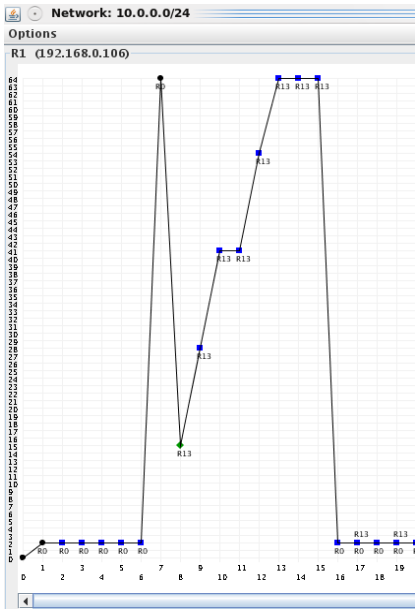


Abbildung 3: Netzgraph mit Erreichbarkeitsinformationen eines Routers zu einem Netz

Die in [KEU07] vorgestellte Erweiterung, automatisierte Testdurchläufe durchzuführen, ermöglicht ein vereinfachtes Testen einer Vielzahl von Szenarien in angemessener Zeit. Besonders bei komplexen Szenarien ist die Bedienung der einzelnen Router durch den Benutzer so umfangreich, dass eine Automatisierung der Steuersignale unbedingt erforderlich ist. Darüber hinaus haben die Testdurchläufe einzelner Szenarien eine Laufzeit von teilweise mehreren Stunden. Über Konfigurationsdateien (siehe Anhang) kann das Updateverhalten aller Router eines Szenarios gleichzeitig geregelt werden. Dieser Umstand gewährleistet eine wichtige Anforderung an die Testdurchläufe. Die Konfigurationsdatei wird immer gleich ausgeführt. Sie wird für jedes Szenario individuell angepasst, um einen Count-To-Infinity-Effekt kontrolliert und vor allem zuverlässig oft zu erzeugen. Änderungen der Updatereihenfolgen während einer Testphase sind nicht vorgesehen, sodass eine vollständige Testreihe aus 100 Durchläufen immer unter den gleichen Bedingungen ausgeführt werden kann. Nur auf diese Weise erhält man aussagekräftige Ergebnisse über die Konvergenzzeiten.

2.3.1 Konfigurationsdatei

Eine Konfigurationsdatei zum automatisierten Ansteuern der Quagga RIP-Daemons über den XT-Peer hat eine dreigeteilte Struktur. Zuerst werden alle RIP-Router mit der entsprechenden Bezeichnung aus der XML-Datei definiert, z.B. „Rip-Router R1“ wenn der Name der virtuellen Maschine „R1“ ist. Im nächsten Teil wird der periodische Update-Timer definiert, da der automatische Generator des XT-Peer regelmäßige Updatenachrichten der Router selbst unterdrückt. Der Parameter „periodic_update“ bekommt einen ganzzahligen Wert, angegeben in Millisekunden, zugeordnet. Die eigentliche Kontrolle der Router folgt im Update-Forecast. Jede Zeile ist stellvertretend für eine Update-Periode und beschreibt wie sich jeder einzelne Router des Netzwerkes verhalten soll. Hinter jedem Router stehen in Klammern Angaben, die sich, durch ein Komma getrennt, auf je ein Interface beziehen. Der Wert „0“ erlaubt ein sofortiges Update eines Routers. Das „x“ steht für ein blockiertes Update. Eine Zahl beschreibt die Wartezeit (in Sekunden), bevor der Router ein Update sendet. Während der Abarbeitung der Updatereihenfolge befindet sich die Simulation im „MANUAL“-Modus, da alle Updatenachrichten vom Generator des XT-Peer gesteuert werden. In der letzten Zeile wird jeder Routereintrag, mit Ausnahme des ausgefallenen Routers, mit einem „a“ ergänzt. Dieses Kommando setzt die Simulation in den „AUTO“-Modus, sodass die RIP-Router wieder selbständig Updatenachrichten versenden.

```
// Generator Testdatei

Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4

// Parameter in ms
periodic_update 30000

// Update_Forecast
Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
// Ausfallphase
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
// Ausbreitungsphase
R1(0,0); R2(0,0); R3(x,0,0); R4(x,0);
// CTI-Phase
R1(0.5,0)a; R2(0,0)a; R3(x,0,0)a; R4(x,0);
```

Der Umfang des Update-Forecasts hängt von einzelnen Phasen ab, die zur Erzeugung eines Count-To-Infinity-Effekts erforderlich sind. Dabei werden vier Phasen unterschieden. Durch die Zeichen „//“ auskommentiert, kann man die einzelnen Phasen erkennbar machen.

- Die **Konvergenzphase** beschreibt den Vorgang, indem alle Router Routinginformationen ungehindert austauschen können. Damit jeder Router jede Route kennt, muss die Konvergenzphase lang genug sein. Die Anzahl der Zeilen hängt in dem Fall vom längsten Pfad zwischen ausfallendem Netz und dem am weitesten davon entfernten Router ab. Die Anzahl der benötigten Hops deutet auf die Anzahl der benötigten Zeilen in der Konvergenzphase hin. Sobald die Routingtabellen konsistent sind, kann die nächste Phase beginnen.

- Die **Ausfallphase** beschreibt den Verlust einer Route zu einem bestimmten Netz. Sie muss i.d.R. fünf Zeilen lang sein. Grund dafür ist das Verhältnis zwischen T_U zu T_{TT} von 1 zu 6. Das Verhältnis wurde ursprünglich so festgelegt. Beim nächsten Update nach den fünf Zeilen, wird eine Route als unerreichbar erklärt und sofort die neue Metrik verbreitet.
- Dieser Vorgang leitet die **Ausbreitungsphase** ein, in der die Verbreitung der neuen Metrik an bestimmte Router erlaubt ist und für andere Router unterdrückt werden muss. Das geschieht entweder durch das Blockieren eines Interfaces oder durch den Einbau einer zeitlichen Verzögerung, damit im nächsten Schritt ein Router dazu veranlasst wird, seine alten und ungültigen Routinginformationen weiterzugeben.
- Sobald das passiert, ist der Count-To-Infinity-Effekt entstanden und breitet sich in der **CTI-Phase** im Netzwerk aus. Damit der Count-To-Infinity-Effekt seine erwartete Entwicklung nimmt, wird das Netzwerk in den „AUTO"-Modus gesetzt, sodass alle RIP-Router autonom reagieren und durch eigene periodische Updatenachrichten die ungültige Routinginformation verbreiten.

Die Ausbreitungs- und CTI-Phase müssen individuell an jedes Szenario und die gewählten Timer-Einstellungen angepasst werden.
Der bereits angesprochene Generator ist Teil des XT-Peer-Programms und regelt nach dem Laden der Konfigurationsdatei deren Ausführung.

3. Das Routing Information Protocol

In diesem Kapitel wird der Distanzvektoralgorithmus beschrieben. Darauf aufbauend wird die Funktionalität und die speziellen Merkmale des Routing-Information-Protocols (RIP) erläutert. Im Speziellen werden dabei die Risiken und Auswirkungen des Count-To-Infinity-Effekts auf ein Netzwerk detailliert erklärt.

3.1 Distanzvektoralgorithmus

Der Distanzvektoralgorithmus gehört zu der Klasse der Routing-Algorithmen. Kennzeichnend dafür ist das Versenden von Routinginformationen unter Routern. Basierend auf dem graphentheoretischen Ansatz des Bellman-Ford-Algorithmus [BEL57], kann der Distanzvektoralgorithmus die kürzesten Wege in einem Netzwerk aus Knoten (Router) und Kanten (Verbindungsleitungen) ermitteln. Er funktioniert nach dem Prinzip *„Teile deinen Nachbarn mit, wie für dich die Welt aussieht“*. Dazu muss sich ein Router über den Austausch von Routinginformationen mit seinen Nachbarroutern ein Bild von der Topologie der direkt erreichbaren Netzwerke verschaffen. Die Datenpakete enthalten Adressierungs-Merkmale der registrierten physischen Netzwerke und die Entfernungen zu diesen Netzwerken [RIO09]. Es wird davon ausgegangen, dass jeder Knoten die Kosten der Verbindungsleitungen zu seinen direkten Nachbarn kennt. Als Maß (Metrik) für die Bestimmung der kürzesten Wege wird die geringste Anzahl von Hops (engl.: hop, dt.: Sprung, Teilstrecke) vom Startknoten zum Zielknoten verwendet. Es wäre zwar theoretisch auch möglich, Eigenschaften der Verbindungsleitungen als Metrik zu verwenden, beispielsweise die Bandbreite oder Latenz, praktisch ist das jedoch nicht implementiert. Es findet also keine Gewichtung bzgl. der Leistungsfähigkeit einer Verbindungsleitung statt. Daher haben alle Router einen gleichwertigen Status und werden nicht hinsichtlich ihrer Position im Netzwerk unterschieden [ITW09]. Der Hopcount beschreibt die minimale Entfernung zum Ziel über die Anzahl der Router, die ein Datenpaket bis zum Zielnetzwerk durchlaufen muss. Jeder Router verwaltet eine eigene Routingtabelle, in der er die Erreichbarkeits-Informationen festhält. Damit diese Tabelle stets optimal gehalten wird, müssen die Router in regelmäßigen Abständen über periodisch gesendete Updatenachrichten oder bei bemerkten Änderungen über getriggerte Updatenachrichten miteinander kommunizieren. Es können nämlich ständig Router ausfallen oder neue Router hinzukommen. Jeder Router erstellt seine Routingtabelle nach folgendem Schema [WIK09].

1. Bilde ein eindimensionales Array (Vektor) mit Erreichbarkeitsinformationen(Zielknoten, Metrik) zu den direkten Nachbarroutern.
2. Erstelle daraus eine Distanz-Tabelle und sende diese an alle Nachbarrouter.
3. Warte auf die Distanz-Tabelle der benachbarten Router und vergleiche diese mit der eigenen.
4. Sollte sich die minimale Metrik zu einem Router ändern, dann fahre mit Schritt 2 fort, sonst mit Schritt 3.

Abbildung 4 zeigt einen Graphen, der ein Netzwerk aus vier Knoten (Router A bis D) in einer möglichen Topologie darstellt. Die Kanten entsprechen Netzwerkverbindungen. Jeder Kante sind Kosten zugeordnet, die auf die Entfernung zum nächsten Router hinweisen. Die Distanz-Tabelle von Router A hat nun beispielsweise folgende Form (siehe Tabelle 1).

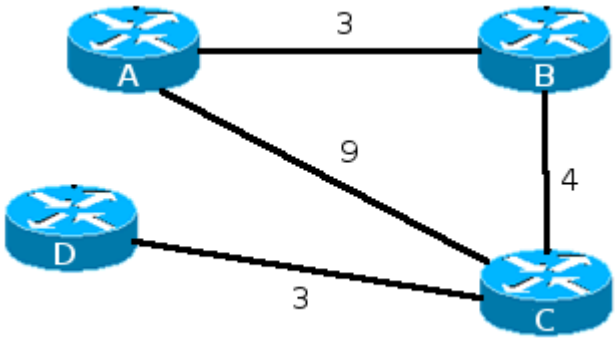


Abbildung 4: Beispiel-Topologie aus 4 Routern

T=0

Router A

Ziel	Next Hop	Metrik
B	B	3
C	C	9
D	--	∞

T=1

Router A

Ziel	Next Hop	Metrik
B	B	3
B	C	13
C	C	9
C	B	7
D	C	12

T=2

Router A

Ziel	Next Hop	Metrik
B	B	3
C	B	7
D	C	12
D	B	10

T=3

Router A

Ziel	Next Hop	Metrik
B	B	3
C	B	7
D	C	10

Tabelle 1: Routingtabelle von Router A

Erläuterung der Abläufe in Router A:

- T=0: Router A wird neu gestartet und kennt zu diesem Zeitpunkt nur seine direkten Nachbarn. B ist erreichbar mit Metrik 3 und C mit Metrik 9. D kann er noch überhaupt nicht erreichen. Diese Routinginformation sendet er weiter an seine Nachbarn .
- T=1: Router A fordert mittels Request-Nachricht von B und C alle Routinginformationen an. Er stellt fest, dass der Zielrouter C über B kürzere Pfadkosten hat (Metrik 7) und setzt die alte Route auf den Status „ungültig“, um sie aus der Tabelle zu entfernen. Die Alternativroute über C nach B bietet eine höhere Metrik an, als die aktuell eingetragene, und wird daher direkt verworfen. Zusätzlich wurde von C ein neuer Pfad zu Router D übertragen.

- T=2: Ungültige Routen wurden entfernt und Änderungen an die Nachbarn weitergeleitet. Von Router B erhält A einen neuen Pfad zu D mit geringerer Metrik und aktualisiert erneut seine Routingtabelle.
- T=3: Router A hat seine endgültige Routingtabelle zusammengestellt. Die Topologie des Netzwerkes hat sich nicht mehr verändert. Es werden also keine neuen Informationen verschickt. Die anderen Router haben den Algorithmus währenddessen in gleicher Weise abgearbeitet. Das Netzwerk befindet sich in einem konsistenten Zustand. Folgende periodische Updatenachrichten halten lediglich die Kommunikationsleitungen aufrecht. Diesen Zustand beschreibt man als Konvergenz.

An dieser Stelle muss betont werden, dass kein Router eine komplette Sicht auf die Topologie des gesamten Netzwerkes hat. Jeder Router kennt nur seine persönliche Routingtabelle und findet für sich einen optimalen Weg zum Nachbarn, um ein Datenpaket weiterzuleiten. D.h. die Entfernung ist nur aus der Sichtweise der eigenen Routingtabelle angegeben. Den weiteren Weg überlässt er dem nächsten Router.

3.2 Routing-Information-Protocol

Das Parade-Beispiel, in das der Distanz-Vektor-Algorithmus implementiert ist, ist das Routing-Information-Protocol (RIP). Es wird im Intra-Domain-Routing als Interior Gateway Protocol eingesetzt, um den Datenverkehr innerhalb eines Autonomen Systems zu verwalten. Ein Autonomes System beschreibt eine Ansammlung von IP-Netzen, die unter der Kontrolle einer administrativen Instanz stehen, beispielsweise einem Internet Service Provider. RIP gilt aufgrund seines dynamischen Lernverhaltens als selbstorganisierendes Protokoll, dass nahezu wartungsfrei eingesetzt werden kann. Es existiert jedoch ein geringfügiger Unterschied zwischen dem eben vorgestellten Distanzvektoralgorithmus und der Idee hinter RIP. Das Ziel besteht darin, Datenpakete an Hosts in bestimmten Netzen zu verschicken. Anstatt also die Kosten zur Erreichbarkeit bestimmter Router bekannt zu geben, teilen die Router Informationen über die Erreichbarkeit von Netzen mit. Das Grundprinzip dahinter hat sich aber dadurch nicht verändert. Jeder Router ermittelt für sich die beste Route mit den wenigsten Hops zum Zielnetz. Ein Router entspricht der Metrik 1. Der Hopcount entspricht der Anzahl der RIP-Router, die auf dem Weg zum Zielnetz passiert werden müssen. Empfängt

ein Router eine neue Route, zählt er sich selbst einmal zur Metrik dazu und trägt den Wert in seine Routingtabelle ein. Nur die Routen mit der niedrigsten Metrik werden in der Routingtabelle gespeichert und weitergeleitet. Das bedeutet, dass sich „gute“ Nachrichten mit günstigeren Pfadkosten schnell im Netzwerk ausbreiten können. Routinginformationen mit einer gleichwertigen oder höheren Metrik als die bereits vorhandene werden sofort verworfen. Damit ist die Wahl einer möglichen Alternativroute nach dem Ausfall einer benötigten Verbindungsleitung nicht verfügbar. Fällt eine Verbindung zu einem Netz aus, stellt der direkt daran angeschlossene RIP-Router dies durch das Ausbleiben der regelmäßigen Updatenachrichten über T_{TT} fest. Die Route zum Netz erhält dann eine Unerreichbarkeits-Metrik. Dieser Wert wird als RIP-Infinity bezeichnet und ist durch den Wert 16 festgelegt. RIP-Infinity beschränkt damit die maximale Ausdehnung eines Netzwerkes. Zwischen einem Router und seinem Zielnetz sind nur maximal 15 Router, inklusive ihm selbst, möglich. Infolgedessen kann ein von RIP-Routern geroutetes Netzwerk nicht beliebig groß skalieren. Der Grund für den relativ niedrigen RIP-Infinity-Wert liegt in dem schlechten Konvergenzverhalten von RIP und dem damit zusammenhängenden Count-To-Infinity-Problem (dt.: zähle ins Unendliche).
Bis der letzte Router in einer Kette von 15 Routern Routinginformationen des ersten Routers erhält, können mehrere Minuten vergehen. Erst dann ist das Netzwerk in einem konvergenten Zustand. Hinzu kommt das Routingschleifen-Problem, bei dem sich RIP-Router beispielsweise in einer ringförmigen Anordnung befinden. Dieses Problem wird bei RIP als Count-To-Infinity-Effekt bezeichnet und beschreibt das zyklische Hochzählen der Metrik einer eigentlich ausgefallenen Route. RIP kann diesen Effekt nicht verhindern und würde infolgedessen theoretisch bis ins Unendliche hochzählen. Datenpakete, die auf diese Route geschickt werden, bleiben ebenso lange in der Schleife bestehen und können die Leistungsfähigkeit von Verbindungsleitungen erheblich beeinträchtigen.

Die Kommunikation zwischen Routern, auf denen RIP läuft, ist intervallgesteuert. Alle 30 Sekunden sendet ein Router eine periodische Updatenachricht mit allen Informationen zu seiner Routingtabelle. Dies geschieht per Multicast, einer Nachrichtenübertragung, bei der gleichzeitig Nachrichten an mehrere Ziele gesendet werden können, ohne dabei die Bandbreite der Leitung proportional zur Zahl der Empfänger zu belasten. Parallel dazu gibt es sogenannte Triggered Updates, die ein Router versendet sobald er durch eine aktualisierte Nachricht zur Änderung seiner eigenen Routingtabelle veranlasst wurde. Um zu verhindern, dass

pausenlos eintreffende Aktualisierungen unkontrolliert verarbeitet werden, wurde ein spezieller Triggered Timer T_t implementiert. Eine Art Countdown-Zähler, der die Anzahl zu versendender Updatenachrichten dadurch reduziert, dass er für ein zufälliges Zeitfenster von bis zu fünf Sekunden alle ausgehenden Nachrichten blockiert und anschließend gesammelt weiterleitet. Für den Transport der Routingtabellen ist das User-Datagram-Protocol (UDP) verantwortlich. Da UDP ein verbindungsloses Protokoll ist, kann die zuverlässige Übertragung einer Updatenachricht nicht gewährleistet werden.

Eine Anforderung an dynamische Routingverfahren ist das flexible und schnelle Reagieren auf Veränderungen der Netztopologie. Um eine zuverlässige und vorhersagbare Verarbeitung von Routinginformationen zu gewährleisten, wurden in der RIP-Spezifikation [RFC95] drei unterschiedliche Timer eingeführt. Der bereits vorgestellte Update-Timer T_U sendet im 30 Sekunden-Takt periodische Updatenachrichten mit seiner vollständigen Routingtabelle. Um zu verhindern, dass alle Router eines Netzwerkes gleichzeitig das Netz mit Routinginformationen fluten und so eine punktuelle Mehrbelastung der Verbindungsleitungen zu vermeiden, ist ein Zufallsgenerator implementiert worden. Dieser variiert zwischen 0 und 5 Sekunden und verhindert Belastungsspitzen im Netzwerk. Ein weiterer Timer unterstützt den RIP-Router beim Erkennen der Gültigkeit einer Route in der Routingtabelle, der Timeout-Timer T_{TT} . Dieser Timer wird für jede bestätigte Route in einem eintreffenden Routing-Update reinitialisiert. Sollte eine Route nach 180 Sekunden nicht bestätigt werden, interpretiert der RIP-Router das als Ausfall der Verbindung und setzt das entsprechende Netz auf RIP_Infinity. Der T_{TT} ist um einiges länger als der T_U , um die Mängel des UDP und Latenzen von Verbindungsleitungen abzufangen. Wenn eine Updatenachricht vom Nachbarrouter auf dem Weg verloren geht oder durch hohe Belastung der Leitungsbandbreite verzögert wird, soll der RIP-Router dies nicht zu voreilig als Verbindungsausfall werten. Hierdurch wird ein Problem erzeugt, dass u.a. für die langen Konvergenzzeiten des Algorithmus verantwortlich ist. „Gute“ Nachrichten, die die Metrik zu einer Route verbessern, verbreiten sich schneller im Netz als „schlechte“ Nachrichten über den Ausfall einer Route. Nachdem eine Route nach Ablauf der 180 Sekunde als ausgefallen gilt, wird automatisch der Garbage-Collection-Timer T_{GC} gestartet, der den Löschvorgang dieser Route initiiert. Dabei verbleibt die Route mit der Unerreichbarkeitsmetrik 16 noch für 120 Sekunden in der Routingtabelle. Diese Phase wird genutzt, damit auch die Nachbarrouter über die Unerreichbarkeit des Netzes informiert werden können.

Zusätzlich ist gewährleistet, dass eine alternative Route mit kleinerer Metrik von einem anderen Router eintreffen kann und die Gültigkeit der Route wiederherstellt. In diesem Fall wird T_{GC} gestoppt und T_{TT} wieder initialisiert. Damit behandelt der Timer den Nachteil des RIP-Algorithmus, dass er beim Ausfall der besten Route keine Alternativroute als Ersatz vorhält. Treffen auch nach Ablauf des T_{GC} keine bestätigenden Updatenachrichten ein, wird die Route endgültig aus der Routingtabelle gelöscht.

3.3 Count-To-Infinity-Effekt

Dennoch kann es zu Fehlern kommen. Das größte Problem des Routing-Information-Protocol sind Routingschleifen in einem Netzwerk. Sie können auftreten wenn die einzige Verbindung zu einem Knoten ausfällt und ungünstige, zeitliche Sendereihenfolgen von Updatenachrichten auftreten, bei denen aktuellere Routinginformationen von älteren, nicht mehr gültigen Routinginformationen überschrieben werden.. Unter der Annahme, dass im Netzwerk (siehe Abbildung 5) die Verbindung zwischen Router C und D ausfällt, gibt C diese Information sofort an A und B weiter. Wenn A oder B im gleichen Moment ihre Routingtabelle mit einer Entfernung zu D von 3 Hops verschicken, dann kommt es zu einem Problem. A lernt beispielsweise von B, dass er D über 3 Hops erreichen kann, trägt in seiner Routingtabelle 4 ein und gibt die Aktualisierung in einem Triggered Update an C weiter. C folgert daraus, dass er D über A mit einer Metrik von 5 erreichen kann und gibt die bessere Route ebenfalls weiter. C kann in diesem Fall nicht feststellen, dass seine Route zu D über den Weg $C \rightarrow A \rightarrow B \rightarrow C \rightarrow D$ zweimal ihn selbst passiert. Die Distanz in seiner Routingtabelle steigt mit jedem Umlauf der Updatenachricht durch die Routingschleife. Er hat also eine ungültige Routeninformation erhalten, die im weiteren Text auch als Falschnachricht bezeichnet wird. Die Art der Kommunikation von RIP-Routern, die sich lediglich auf die Nachbarrouter beschränkt, macht eine globale Sicht auf die Topologie des Netzwerkes unmöglich und fördert damit die Entstehung eines Count-To-Infinity-Effekts. Tabelle 2

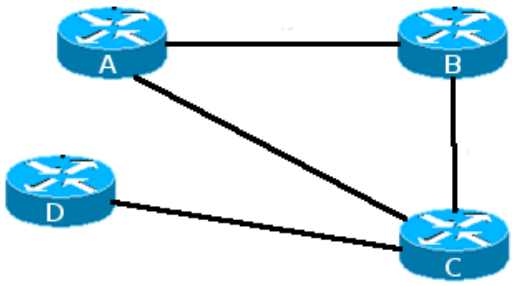


Abbildung 5: CTI-begünstigende Topologie

beschreibt den kompletten Vorgang des Count-To-Infinity-Effekts in einer Schritt-für-Schritt-Abfolge von ausgetauschten Routinginformationen.

Schritt	Metrik Router				Abfolge der Updatenachrichten	Schritt	Metrik Router				Abfolge der Updatenachrichten
	A	B	C	D			A	B	C	D	
1	∞	∞	∞	1	D wird neu ins Netzwerk integriert	13	4	6	5	x	B erhält ungültiges Update von C
2	∞	∞	2	1	Nach 1 Update	14	7	6	5	x	A erhält ungültiges Update von B
3	3	3	2	1	Nach 2 Updates (Konvergenz)	15	7	6	8	x	C erhält ungültiges Update von A
4	3	3	2	x	D fällt aus, A lässt T_{TT} durchlaufen	16	7	9	8	x	B erhält ungültiges Update von C
5	3	3	2	x		17	10	9	8	x	A erhält ungültiges Update von B
6	3	3	2	x		18	10	9	11	x	C erhält ungültiges Update von A
7	3	3	2	x		19	10	12	11	x	B erhält ungültiges Update von C
8	3	3	2	x		20	13	12	11	x	A erhält ungültiges Update von B
9	3	3	∞	x	180s-Timeout bei C endet	21	13	12	14	x	C erhält ungültiges Update von A
10	∞	3	∞	x	A erhält das Update, B erhält das Update nicht	22	13	15	14	x	B erhält ungültiges Update von C
11	4	3	∞	x	A erhält ungültiges Update von B	23	∞	15	14	x	A erhält ungültiges Updates von B und erreicht RIP_Infinity
12	4	3	5	x	C erhält ungültiges Update von A (CTI entsteht)	24	∞	15	∞	x	A erreicht RIP_Infinity
						25	∞	∞	∞	x	B erreicht RIP_Infinity

Tabelle 2: Abfolge von Updatenachrichten zu einer ausgefallenen Route

Erklärung zu Tabelle 2:

- Router D wird in das bestehende Netzwerk aus A, B, C integriert. Zu diesem Zeitpunkt hat noch kein Router Kontakt zu D.
- D ist direkt an C angeschlossen. C lernt D als erster Router kennen und trägt in seine Routingtabelle die Metrik 2 ein, da er D mit zwei Hops erreichen kann.
- C leitet die neue Routeninformation an A und B weiter und diese ermitteln die Metrik 3 zu Router D. Das Netzwerk befindet sich im konvergenten Zustand.
- Router D fällt jetzt aus. Der Timeout-Timer von C wird für die folgenden sechs Updateperioden nicht reinitialisiert.
- Router C macht die Route zu D mit RIP_Infinity ungültig, wobei ∞ stellvertretend ist für die maximal mögliche Metrik 16. C sendet seine Routingtabelle an A und B weiter.
- Aufgrund eines Paketverlustes erhält B das Routing-Update nicht und behält seine alte Metrik.
- Router B schickt seine nicht mehr aktuelle Metrik an A. Router A vergleicht seine eigene Routingtabelle mit dem Update von B und stellt eine bessere Route über B mit der Metrik 4 fest.
- Router C erhält diese Aktualisierung in einem Update von A und trägt Metrik 5 zu dieser Route ein.
- Router B erhält im nächsten Schritt von A die neue Routeninformation und zählt seine Metrik wieder hoch. Damit ist die Routingschleife einmal durch den Zyklus B, C, A gelaufen und setzt sich in den folgenden Schritten fort.
- Als erster Router stellt A durch RIP_Infinity die Unerreichbarkeit von Router D fest und leitet diese Information an C und dieser an B weiter.
- Auf allen Routern startet der Garbage-Collection-Timer und die Routeninformation wird gelöscht

Die Routingschleife existiert ab dem ersten ungültigen Update und so lange bis der erste Router RIP_Infinity erreicht hat. Sämtlicher Datenverkehr, der an ein Zielnetz hinter Router D adressiert ist, wird durch die Routingschleife geleitet, obwohl das Zielnetz unerreichbar ist.

Der RIP-Algorithmus bietet mehrere Lösungsansätze, die das Auftreten eines Count-To-Infinity-Effekts erschweren, ihn aber nicht erkennen und abbrechen können.

3.2.1 Hopcount

Zum einen gibt es den Hopcount. In der Routingschleife wird die Entfernungsmetrik theoretisch bis ins Unendliche hochgezählt, wobei praktisch eine Beschränkung bei 16 implementiert wurde. Ein Datenpaket, das bereits über 15 Router weitergeleitet wurde, wird verworfen wenn es den 16. Router erreicht, da dieser Wert RIP_Infinity (RIP_∞) repräsentiert. RIP_∞ deklariert eine Route als ungültig. Dadurch wird zwar unnötig viel Datenverkehr in der Routingschleife verhindert, aber gleichzeitig auch die maximale Anzahl an Hops auf 15 begrenzt, was der Ausdehnung des Internets nicht mehr gerecht wird.

3.2.2 Split Horizon

In Übereinstimmung mit dem Distanzvektoralgorithmus, verschickt jeder Router seine vollständige Routingtabelle auf allen Interfaces an seine Nachbarn. Dadurch wird unnötiger Datenverkehr produziert. Wenn Router A eine Route von Router B gelernt hat, muss er diese nicht im nächsten periodischen Update wieder an B weiterleiten, da der die Route bereits kennt. Die Split-Horizon-Funktion sorgt dafür, dass ein RIP-Router eine Routeninformation nicht mehr an den Router zurücksendet, von dem sie gelernt wurde. Damit wird nicht nur die Menge an zu übermittelnden Routinginformationen reduziert, sondern auch ein Count-To-Infinity-Effekt zwischen zwei Routern verhindert. In komplexeren Netzwerk-Topologien, in denen häufig ringförmige Anordnungen von Routern oder Mehrfachverbindungen vorkommen, verliert Split Horizon seinen Wert.

3.2.3 Split Horizon with Poison Reverse

Diese Erweiterung der Split-Horizon-Funktion blockiert die Rückroute, von der eine Routeninformation gelernt wurde. Dazu wird die empfangene Routeninformation als unerreichbar (Metrik 16) markiert und auf dem gleichen Interface an den Sender zurückgeschickt.
Router A lernt eine Route zu einem Netz hinter Router D von Router B. Er kennzeichnet das Netz als unerreichbar und sendet diese Information an B zurück. So wird verhindert, dass zwei Router dieselbe Route voneinander lernen. Die Anzahl an Routing-Updates nimmt allerdings wieder zu.

3.2.4 Holddown-Timer

Der Holddown-Timer ist ein Maß für ein Ablehnungs-Zeitfenster, in dem Routinginformationen ignoriert werden können. Wenn die Entfernungsmetrik zu einem Zielnetz steigt, beispielsweise von 4 auf 7, initialisiert der RIP-Router den Holddown-Timer (i.d.R. 60 Sekunden). In dieser Zeit werden keine neuen Routeninformationen zu diesem Netz akzeptiert, ob sie nun gültig sind oder nicht. Dadurch wird verhindert, dass sich Falschnachrichten ungehindert in einer Routingschleife ausbreiten können. Es wird davon ausgegangen, dass während dieser Phase alle Router die Unerreichbarkeit des Netzes erkannt haben. Allerdings wirkt sich dieser Timer negativ auf die Konvergenz des Netzwerks aus. Je länger der Holddown-Timer aktiv ist, desto länger dauert es bis sich mögliche Alternativrouten im Netzwerk ausbreiten können. Ein zu kurzer Holddown-Timer wird auf der anderen Seite ineffektiv.

3.2.5 Triggered Updates

Triggered Updates funktionieren recht einfach. Sobald sich die Entfernung zu einer Route verbessert oder verschlechtert, wird ein Routingupdate generiert, das ausschließlich die Änderungen enthält. Falls ein Count-To-Infinity-Effekt oder starke Veränderungen der Topologie des Netzwerkes auftreten, wird die Konvergenz und die Dauer eines Count-To-Infinity-Effekts erheblich beschleunigt. Problematisch ist allerdings der starke Anstieg an Routing-Updates, die dadurch erzeugt werden. Damit nicht jede einzelne Änderung ein eigenes Update erhält, wartet ein RIP-Router zwischen 0 - 5 Sekunden und sendet mehrere Änderungen gesammelt in einer Updatenachricht. Dadurch steigt aber wiederum die Gefahr, dass eine periodische Updatenachricht eines anderen Routers mit ungültigen Routinginformationen den Nachbarrouter erreicht und dessen gültige Route überschreibt.

Die vorgestellten Mechanismen können laut [RFC95] einzeln aktiviert und deaktiviert werden. Die Funktionen Hopcount, Split Horizon und Triggered Updates müssen unbedingt aktiviert sein und die übrigen können bei Bedarf eingeschaltet werden. Alle Maßnahmen dienen jedoch nur als „Notlösungen“, um den Count-To-Infinity-Effekt irgendwie beherrschen zu können. Doch weder einzeln noch im Zusammenspiel sind sie in der Lage, einen Count-To-Infinity-Effekt zu verhindern, geschweige denn ihn zu erkennen oder zu stoppen.

4. RIPMTI – RIP with metric based topology investigation

So wie viele andere Routingalgorithmen vom Auftreten von Routingschleifen betroffen sind, hat das Routing Information Protocol das Problem des Count-To-Infinity-Effekts. Die vorgestellten Mechanismen und Lösungsansätze begegnen dem Problem nur, indem sie die Wahrscheinlichkeit seiner Entstehung herabsetzen bzw. seine Laufzeit einschränken. Der RIPMTI-Algorithmus stellt eine Erweiterung des konventionellen RIP-Algorithmus dar und soll das Problem erkennen und verhindern.

4.1 Grundlagen

RIPMTI erkennt Schleifen in einem Netzwerk über Routinginformationen zu einem Netz, die er über verschiedene Interfaces empfangen hat und verhindert den Count-To-Infinity-Effekt, indem Routinginformationen, die eine Schleife auslösen, verworfen werden. RIPMTI bleibt jedoch stets kompatibel zum Format des ursprünglichen Routing Information Protocols, sodass ein paralleler Einsatz von RIP- und RIPMTI-Routern in einem gemeinsamen Netzwerk möglich ist. Die zusätzlichen Funktionen von RIPMTI beziehen ihre benötigten Daten aus dem vorhandenen RIP-Format. Der Grundgedanke ist, aus dem Wissen über Netzwerkschleifen gültige Alternativrouten (Simple-Loops) von ungültigen Routen (Source-Loops) zu unterscheiden. Zum weiteren Verständnis der Berechnung von Schleifen müssen einige Parameter definiert werden, nach denen der RIPMTI-Algorithmus arbeitet. Ein Pfad oder die Route P zu einem Netz d über ein Interface A von Router i $P_A^{(i,d)}$ ist eine Kombination aller Hops vom Ursprungsrouter i zum Zielnetz d und wird beschrieben als Metrik $m_A^{(i,d)}$.

4.2 Simple-Loops

Gibt es in einem Netzwerk eine Netzwerkschleife und existieren zu Netz d innerhalb dieser Schleife, wie in Abbildung 6, zwei alternative Routen (blaue Pfeile) von Router i aus über zwei unterschiedliche Interfaces (A und B), so handelt es sich um einen Simple-Loop.

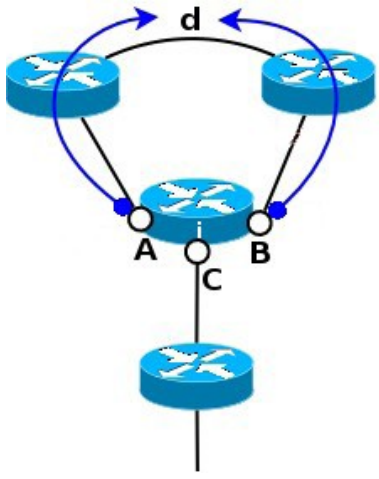


Abbildung 6: Simple-Loop

Router i kann das Netz d über Interface A und über Interface B erreichen. Eine der beiden Routen kann bei Ausfall der anderen Route als Alternativroute ausgewählt werden. Hierbei muss noch einmal betont werden, dass ein RIP-Router keine alternativen Routen in seiner Routingtabelle vorhält. Wenn die Route zu Netz d über Interface A ausfällt und die Metrik auf RIP_Infinity gesetzt wird, bekommt er in einem periodischen Update über Interface B eine niedrigere Metrik angeboten und übernimmt diese.

Router i erkennt die Netzwerkschleife über die Routingupdates der beiden direkt an Netz d angeschlossenen Router, die ihm jeweils einen Weg vorschlagen.
Theoretisch können mehr als zwei redundante Routen in einem komplexen Netzwerk vorkommen. In einem voll vermaschten Netzwerk, in dem jeder Router mit jedem anderen Router direkt verbunden ist, gibt es eine sehr große Anzahl redundanter Verbindungen und dementsprechend viele Netzwerkschleifen. Interessant ist, dass in diesem Fall durch die Redundanz die Verfügbarkeit von Netzen verbessert wird. Simple-Loops stellen für RIP erst einmal kein Problem dar. So lange die ausfallende Verbindung ein Teil der Schleife ist, kann kein Count-To-Infinity-Effekt entstehen. Der Simple-Loop bringt dem RIPMTI-Algorithmus auf Router i einen besonderen Nutzen. Er kann anhand der verschiedenen Metriken zu Netz d beider RIP-Nachrichten, die er über Interface A und B erhalten hat, den Umfang der Netzwerkschleife berechnen, indem er die Routen $P_A^{(i,d)}$ und $P_B^{(i,d)}$ kombiniert. Folgende Formel beschreibt die Kombination beider Pfade der Netzwerkschleife zu $P_{(A,B)}^{(i,d,i)}$, die im weiteren auch als Zyklus bezeichnet wird.

$$P_{(A,B)}^{(i,d,i)} = P_A^{(i,d)} + P_B^{(i,d)} - 1$$

Zur Berechnung des Umfangs des Zyklus U_Z wird die Metrik der Pfade verwendet. Die Metrik über Interface A von Router i zu Netz d $m_A^{(i,d)}$ und die Metrik über Interface B von Router i zu Netz d $m_B^{(i,d)}$ ergeben zusammen den Umfang der Netzwerkschleife bzw. des Zyklus.

$$m_{(A,B)}^{(i,d,i)} = m_A^{(i,d)} + m_A^{(i,d)} - 1 = U_Z$$

In Anlehnung an Abbildung 6 ergibt die Formel folgende Berechnung. Die Metrik über Interface A zu Netz d hat den Wert 2, da zwei Hops benötigt werden, um von Router i das Netz d zu erreichen. Die gleiche Metrik hat Router i über Interface B. In der Summe beider Metriken ergibt das 4. Das Ergebnis muss um 1 reduziert werden, da Router i sich bei jedem Pfad ein Mal selbst mit zählt und so im Ergebnis doppelt vorhanden wäre. Die Berechnung von U_Z liefert als

Umfangswert 3, was der Anzahl der Router im Zyklus entspricht. Da die Routingtabelle von Router i nur die kleinsten Metriken enthält, wird erkennbar, dass für RIPMTI immer nur die kleinsten Zyklen relevant sind. Tatsächlich baut RIPMTI darauf auf, dass er die Metrik des kleinsten Simple-Loops zwischen zwei Interfaces A und B (MSILM) und die Metrik des kleinsten Simple-Loops zwischen Interface A und allen anderen Interfaces des Router i (MRPM) in einer eigenen Tabelle gespeichert werden.

- MSILM steht für „Minimal-Simple-Loop-Metric“ und wird beim Start des RIPMTI-Algorithmus mit RIPMTI_Infinity initialisiert, wobei die Metrik der Kombination von zwei ausgefallenen Routen (RIP_Infinity) entspricht. Der Startwert für RIPMTI_Infinity berechnet sich aus
$$msilm^{i}_{(A,B)} = 2 * RIP_{\infty} - 1$$
und ergibt mit der RIP_Infinity-Metrik 16 einen MSILM-Wert von 31. Dieser Wert repräsentiert, dass kein Simple-Loop zwischen zwei Interfaces erkannt wurde. In dieser Arbeit wird RIP_Infinity auf 64 erweitert, sodass sich unter den Testbedingungen ein MSILM-Wert von 127 ergeben wird.
- MRPM bedeutet „Minimal-Return-Path-Metric“. Dieser Wert kennzeichnet den kleinsten Simple-Loop eines einzigen Interfaces A und kann sich vom MSILM-Wert unterscheiden, wenn das Interface A Teil mehrerer Simple-Loops ist, die in der MSILM-Tabelle jeweils einen eigenen Eintrag bekommen. Beispielsweise beschreibt $msilm^{i}_{(A,B)} = 3$ eine kleinste Schleife zwischen Interface A und B, und $msilm^{i}_{(A,C)} = 7$ eine kleinste Schleife zwischen Interface A und C. In beiden Fällen ist das Interface A vorhanden. Für die MRPM-Tabelle eines RIPMTI-Routers wird ausgehend von $msilm^{i}_{(A,B)} < msilm^{i}_{(A,C)}$ die kleinste aller existierenden Simple-Loop-Metriken $mrpm^{i}_{A} = msilm^{i}_{(A,B)} = 3$ aufgenommen.

Nach dem Ausfall einer Route kann RIPMTI nur über die Interfaces entscheiden ob eine gültige oder ungültige Updatenachricht zu einem Netz empfangen wurde. Indem der MSILM-Wert an zwei Interfaces gekoppelt wird, kann eine gültige Alternativroute nur von einem dieser Interfaces akzeptiert werden. Eine Routinginformation, die über ein anderes Interfaces empfangen wurde, muss mit hoher Wahrscheinlichkeit eine ungültige Route darstellen, deren alte Erreichbarkeitsinformation zuvor vom Router selbst auf diesem Interface in eine Netzwerkschleife verbreitet wurde. Ungünstige Updatereihenfolgen lösten den Count-To-Infinity-Effekt aus, sodass der Router seine eigene Updatenachricht

empfangen hat.
Netzwerkschleifen sind also nicht nur vorteilhaft zur Ermittlung alternativer Pfade, sondern sind auch verantwortlich für die Entstehung des Count-To-Infinity-Effekts. Inwiefern Die Berechnung des Schleifenumfangs für die Erkennung eines Count-To-Infinity-Effekts wichtig, zeigt sich anhand der Source Loops.

4.3 Source-Loops

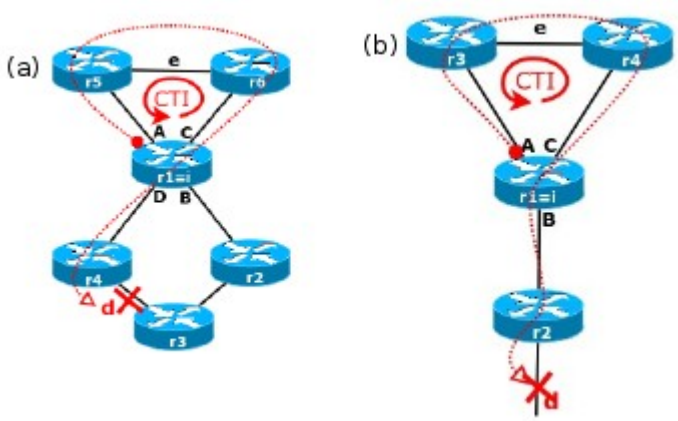


Abbildung 7: Beispiele für Source-Loops

In Abbildung 7 (b) liegt das Netz d an einer anderen Stelle im Netzwerk. In diesem Fall existiert keine alternative Route. Fällt die einzige Verbindung aus, ist das Netz nicht mehr erreichbar. Da sich an der Topologie des Netzwerkes nichts geändert hat, kann die Netzwerkschleife das Problem der Routingschleife verursachen. Fällt Netz d aus, stellt Router i die Unerreichbarkeit über Interface B fest. Ungünstige Updatereihenfolgen können dazu führen, dass die gültige Unerreichbarkeitsinformation beispielsweise über Interface A vom Nachbarrouter überschrieben wird. Da der Nachbarrouter noch nicht über den Ausfall des Netzes d informiert wurde, enthält dessen Routingtabelle noch den alten, ungültigen Wert, der als Falschnachricht bezeichnet wird. Die darin enthaltene Metrik ist niedriger als RIP_Infinity und wird demzufolge von Router i fälschlicherweise als gültig gewertet und auf Interface C weitergegeben. Dieses Verhalten ist identisch mit dem im Simple-Loop. Router i vergleicht die Metrik seiner Routingtabelle mit eingehenden Updatenachrichten und aktualisiert seine Routingtabelle mit dem besseren Pfad. Da die neue Route keine Alternativroute darstellt, sondern der gesamte Pfad (roter Pfeil) Router i zweimal passiert, bezeichnet man die Netzwerkschleife als Source Loop. An diesem Punkt entwickelt sich der Count-To-Infinity-Effekt. Der beschriebene Vorgang, dass eine Routeninformation Router i über Interface A erreicht und über Interface C verlässt, um ihn dann wieder über A zu erreichen, wiederholt sich ständig. Dies gilt es zu vermeiden. In der

Diplomarbeit von [SCH99] wurde gezeigt, dass unter Verwendung von Split Horizon nur zwei Arten von Source-Loops verhindert werden müssen. Zum einen X-Kombinationen (s. Abbildung 7(a)) mit beliebiger Länge und zum anderen Y-Kombinationen (s. Abbildung 7(b)) mit zwei verbundenen Schleifen. Derjenige Router, der Teil der Netzwerkschleife ist und dem ausgefallenen Netz am nächsten liegt, wird als Source Router bezeichnet. Er nimmt eine Schlüsselposition im Netzwerk ein. Da er als Verbindungsglied zwischen Zyklus und ausgefallenem Netz fungiert, ist er verantwortlich für die Erkennung von Schleifeninformationen. Auf diesem Router muss der RIPMTI-Algorithmus zwingend eingesetzt werden, um den Count-To-Infinity-Effekt zu verhindern. Es kommt jetzt darauf an, Simple-Loops zu erkennen und dadurch Source-Loops zu verhindern. Dazu wurden zwei Testverfahren entworfen, der sogenannte X-Test und Y-Test. In der Diplomarbeit von [BOH08] und dem Paper [StDiKe08] wird darauf hingewiesen, dass der X-Test im Gegensatz zum Y-Test kein hinreichendes Werkzeug darstellt. Der X-Test erkennt zwar Simple-Loops genauso wie der Y-Test und schließt darauf auf die Existenz eines Source-Loops, allerdings wurde bewiesen, dass der X-Test auch gültige Alternativrouten zu einem Netz ablehnt, sofern die alternative Metrik zufällig mit der ungültigen Metrik aus einem Source-Loop übereinstimmt oder größer ist als es der minimal zugelassene Simple-Loop erlaubt. Im RIPMTI-Algorithmus wird der X-Test daher nicht mehr angewendet. Darüber hinaus konnte gezeigt werden, dass der Y-Test auch in X-Kombinationen erfolgreich Source-Loops verhindert. Im Folgenden wird der Y-Test analog zur Formulierung in den genannten Quellen als Simple-Loop-Test bezeichnet. Er bildet den Kern des RIPMTI-Algorithmus zur Vermeidung des Count-To-Infinity-Effekts.

4.4 Der Simple-Loop-Test

Die Idee hinter dem RIPMTI-Algorithmus geht davon aus, dass der Count-To-Infinity-Effekt eine Netzwerkschleife voraussetzt, in der ein Routing-Loop entsteht, wenn sich ungültige Routingupdates ausbreiten. Ein Routing-Loop ist definiert als eine Route, die einen Router i zum Erreichen eines Zielnetzes d mehr als ein Mal passiert. Bevor die Route den Router i das zweite Mal passiert, handelt es sich noch um einen Source-Loop. Wenn dieser verhindert wird, in dem der Router die angebotene ungültige Routinginformation mithilfe eines geeigneten Prüfverfahrens ablehnt, kann sich kein Routing-Loop entwickeln und damit auch kein Count-To-Infinity-Effekt entstehen.

Daraus lassen sich drei Schlussfolgerungen ableiten:

1. Der Count-To-Infinity-Effekt setzt die Ausprägung eines Routing-Loops voraus. Wird der Routing-Loop verhindert, entsteht kein Count-To-Infinity-Effekt.
2. Der Routing-Loop ist abhängig von der Existenz eines Source-Loops. Wird der Source-Loop verhindert, kann kein Routing-Loop entstehen.
3. Der Source-Loop muss durch ein Prüfverfahren ermittelt und verhindert werden, den Simple-Loop-Test.

Der Simple-Loop-Test umfasst mehrere Arbeitsschritte und wird für jede Route einer Routingtabelle aktiv. Er entscheidet per Ausschlussverfahren, ob eine angebotene Route gültig ist oder nicht.

Zwei Situationen können für die Metrik und ihre Gültigkeit einer Route innerhalb einer Routingtabelle unterschieden werden. Enthält die Routingtabelle von Router i eine gültige Metrik zur Erreichbarkeit eines Netzes d über Interface D, $m_D^{(i,d)} = 2$ in Abbildung 7 (a), und treffen bei Router i über die übrigen Interfaces Routinginformationen mit gleichwertiger oder schlechterer Metrik zu diesem Netz ein, beginnt der Simple-Loop-Test seine Überprüfung der eingehenden Routinginformation. Besteht die eingehende Route den Test, wurde ein Simple-Loop zwischen dem Interface, auf dem diese Route angekommen ist und dem Interface, über das Router i mit dem Netz d verbunden ist, erkannt. Das ist bei der Updatenachricht von Router r2 der Fall. Router i ermittelt aus dem eingehenden Update über Interface B eine Erreichbarkeit von Netz d mit $m_B^{(i,d)} = 4$. Ein RIP-Router würde diesen Wert, da er eine schlechtere Route anbietet, direkt verwerfen. Der RIPMTI-Router errechnet daraus und seiner eingetragenen Route einen Simple-Loop zwischen Interface B und D mit dem Umfang 5. Die neue Routinginformation hat also den Simple-Loop-Test bestanden und der Umfang wird als $msilm_{(B,D)}^{i} = 5$ als kleinste Simple-Loop-Metrik in die MSILM-Tabelle und parallel dazu $mrpm_D^i = 5$ in die MRPM-Tabelle eingetragen. An der eigentlichen Metrik von Router i zum Netz d ändert sich hier nichts, da der angebotene Wert größer und damit schlechter ist.

Nach Ausfall der eingetragenen Route, werden die Werte aus der MSILM-Tabelle als Vergleichswerte herangezogen. Zur Berechnung der Gültigkeit dient die allgemeine Gleichung:

$$m_A^{(i,d)} + m_B^{(i,d)} - 1 \geq msilm_{(A,B)}^{i}$$

Ein Routing-Update von r2 liefert eine gültige Route. In diesem Fall bestätigt die Gleichung $4 + 2 = 6 \geqslant 5(w)$ die neue Route als Simple-Loop und wird als Alternativroute akzeptiert. Trifft bei Router i ein Falschupdate von Router r5 mit alter, ungültiger Metrik ein, erkennt der RIPMTI-Router den Source-Loop sofort. Router i erhält die Metrik über Interface A mit $m_A^{(i,d)} = 5$. Seine eigene alte und gültige $m_D^{(i,d)} = 2$ wird nun mit der neu angebotenen falschen Metrik verrechnet. Da zuvor zwischen Interface A und D kein Simple-Loop ermittelt werden konnte steht $msilm^i_{(A,D)} = 31$. Die Berechnung der Gültigkeit der Route ergibt nach obiger Gleichung $5+2=7 \geqslant 31(f)$ und wird damit abgelehnt. Die Metrik aus der alten und neuen Route kann nur dann kleiner sein als der MSILM-Wert, wenn dieser noch bei RIPMTI_Infinity steht. Aufgrund der Tatsache, dass ein Simple-Loop ausgeschlossen wurde, bestimmt der RIPMTI-Algorithmus die Existenz eines Source-Loops.
Der Simple-Loop-Test kann in verkürzter Form wie folgt dargestellt werden. Die Überprüfung auf einen Simple-Loop einer Routinginformation wird für beide Kombinationen aus Abbildung 7 abgelehnt. Zu beachten sind jeweils die „roten Pfeile“, die stellvertretend eine falsche Routinginformation darstellen.
Für Abbildung 7(a) heißt das:

$$m_A^{(i,d)} = 5$$
$$m_D^{(i,d)} = 2$$
$$msilm^i_{(A,D)} = 31 = mrpm^i_A$$

$$Simple-Loop-Test = false \Leftrightarrow m_A^{(i,d)} - m_D^{(i,d)} \geqslant mrpm^i_A$$

$$(5-2) \geqslant 31 = false \Rightarrow kein\ Simple-Loop\ erkannt$$
$$\Rightarrow Source-Loop\ muss\ vorliegen$$

$$\Rightarrow angebotene\ Route\ ist\ ungültig\ und\ muss\ blockiert\ werden$$

Analog dazu sieht die Berechnung für Abbildung 7(b) aus. Die Berechnung kann auch alternativ über die Learned-From-IP-Adress durchgeführt werden. Für die Implementation des Algorithmus ändert sich dadurch nichts, außer dass die Positionen mit den Interfaces durch die IP-Adressen der Router, von denen eine Route empfangen wurde, ersetzt wird. Der Hintergedanke dabei ist, dass es in geswitchten Netzwerken durchaus Verbindungen geben kann, in denen ein Router über eine Verbindungsleitung mit mehreren anderen Routern verbunden ist. Um

dies zu unterscheiden, reicht das Interface nicht mehr aus, sodass die Berechnung in Zukunft ausschließlich über die Learned-From-IP-Adress erfolgen soll. Der Einfachheit halber soll in dieser Arbeit noch an der Formulierung mit dem Begriff „Interfaces“ festgehalten werden.

Mit anderen Worten besagt der Simple-Loop-Test nun folgendes. Wenn ein Simple-Loop erkannt wird, kann ein Source-Loop ausgeschlossen und die neue Route als gültiger Wert in die Routingtabelle übernommen werden. Wenn kein Simple-Loop erkannt wird, wird daraus ein Source-Loop geschlossen. Diese Aussagen sind allerdings nur zum Teil korrekt, denn die eigentliche Aussage ist, dass die Wahrscheinlichkeit für eine Routinginformation aus einer anliegenden Schleife sehr groß ist. In der Diplomarbeit von [KLE01] wurde gezeigt, dass ein erkannter Simple-Loop nicht unbedingt einen Source-Loop ausschließt, aber ein nicht existierender Simple-Loop zwischen zwei Interfaces mit hoher Wahrscheinlichkeit für die Existenz eines Source-Loops spricht. In verschachtelten Netzwerktopologien, in denen ein ausfallendes Netz nicht nur über einen Router mit einer Netzwerkschleife verbunden ist, sondern über mindestens einen weiteren Router, der selbst Teil der Netzwerkschleife ist, verbunden ist, kann der Count-To-Infinity-Effekt allein mithilfe des Simple-Loop-Tests nicht verhindert werden. In Abbildung 8 führt Router r2 erwartungsgemäß einen MSILM-Eintrag über Interface A und B bzgl. eines vorhandenen Simple-Loops. Erhält der Router eine Updatenachricht von Router r3 mit Erreichbarkeitsinformationen zum Netz d, wird diese Route als alternativer, gültiger Pfad akzeptiert, obwohl die Route (siehe roter Pfeil) in dem speziellen Fall aus einer benachbarten Netzwerkschleife stammt und damit einen Source-Loop darstellt, der Router r2 zwei Mal passiert. Aus diesem Grund wurde der RIPMTI-Strict-Mode eingeführt, der während der Arbeit von [BOH08] aufgrund zu strenger Bewertung von Routinginformationen in den RIPMTI-Careful-Mode überführt wurde. Im aktuell eingesetzten RIPMTI-Careful-Mode markiert der Algorithmus eine ausgefallene Route, wenn die angebotene, neue Route den Simple-Loop-Test nicht besteht. Zusätzlich löst der fehlgeschlagene Test eine Updatenachricht aus, in der RIP_Infinity zum betroffenen Netz an die Nachbarrouter gesendet wird. Dadurch kann ein möglicher Source-Loop über eine

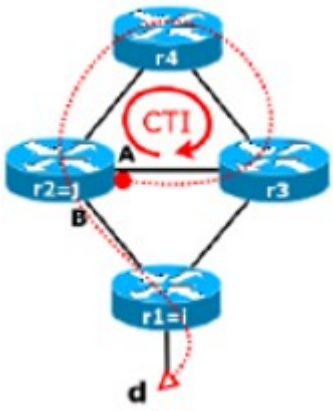


Abbildung 8: Verschachtelte Netzwerktopologie

Route bei den Nachbarroutern sofort überschrieben werden. Da der Simple-Loop jetzt über eine gewisse Zeitspanne aktiv sein soll, kann er mehrmals hintereinander Source-Loop-Informationen erkennen und jedes Mal seine RIP_Infinity-Nachricht versenden. Die Idee dahinter ist, dass ein Source-Loop meistens nicht lange genug existieren kann, um einen Router mehrmals zu erreichen, da sich RIP_Infinity selbständig im Netzwerk ausbreitet und ungültige Routeninformationen überschreibt. Der eingeführte Request-Timer T_R des Careful-Mode beschreibt dabei die Zeitspanne, in der der Careful-Mode aktiv sein soll. Diese wird in Sekunden angegeben. T_R Wird initialisiert sobald der erste Simple-Loop-Test fehlgeschlagen ist und verwirft während seiner Dauer alle Routingupdates, die er zur ausgefallenen Route empfängt. Nach Ablauf dieses Zeitfensters fragt der Router bei seinen Nachbarn die Route neu an und übernimmt diese falls vorhanden als gültig an.

Eine weitere Verbesserung bezieht sich auf den Garbage-Collection-Timer T_{GC} . Bisher musste dieser Wert grundsätzlich vor dem Start des RIP-Daemons fest angegeben werden. In den ursprünglichen RIP-Spezifikationen ist er mit 120 Sekunden angegeben. Im Laufe der Testphasen hat sich herausgestellt, dass ein fixer T_{GC} Probleme mit kürzeren Zeitintervallen hat. Die genauen Details werden in den entsprechenden Anwendungsbeispielen in Kapitel 6 genau beschrieben. Das Hauptproblem besteht darin, dass der Garbage-Collection-Timer in einem bestimmten Zahlenverhältnis zum Timeout-Timer T_{TT} und periodischen Update-Timer T_U steht, nämlich $T_U : T_{TT} : T_{GC}$ im Verhältnis 1:6:4, sodass abhängig von der periodischen Updatezeit alle anderen Timer vorgegeben sind. Mit den Timer-Einstellungen, die im aktuellen RIP-Algorithmus verwendet werden, läuft RIPMTI korrekt. Sobald man die Timer auf $T_U=3$, $T_{TT}=18$ und $T_{GC}=12$ heruntersetzt und damit die Konvergenz eines Netzwerkes beschleunigen will, entsteht in verschiedenen Szenarien folgendes Phänomen. Ein Count-To-Infinity-Effekt benötigt in einem Zyklus mit drei Routern bis zu maximal 15 Sekunden für einen Durchlauf, da der Triggered Timer T_t bis zu 5 Sekunden lang Updatenachrichten verzögern kann. In dieser Situation ist es möglich, dass der Garbage-Collection-Timer auf dem Source-Router eine als ungültig markierte Route nach 12 Sekunden aus seiner Routingtabelle entfernt, während sich noch eine Falschnachricht im Zyklus befindet. Trifft diese verzögert beim Source-Router ein, der für die angebotene Route jetzt keinen Eintrag mehr hat, wird die ungültige Route als gültige Alternativroute akzeptiert und neu in die Routingtabelle eingetragen. Um dieses Phänomen, das häufig bei größeren Zyklen aufgetreten ist, zu beherrschen, wird

der Garbage-Collection-Timer dynamisch an den Schleifenumfang angepasst. Die Berechnung basiert auf dem bereits genannten Triggered Timer und dem MSILM-Wert über den Schleifenumfang. Der neue Garbage-Collection-Timer ergibt sich durch Multiplikation der beiden Werte:

$T_{GCneu} = msilm^{i}_{(A,B)} * T_t$. Das Ergebnis wird mit dem voreingestellten Wert $T_{GCalt} = T_{GC}$ verglichen. Ist der neue Wert größer als der alte, dann wird der neue Wert als effektiver Garbage-Collection-Timer gewählt, im anderen Fall ändert sich nichts an der Voreinstellung:

$$T_{GC} = \begin{cases} T_{GCneu} \Leftrightarrow T_{GCneu} > T_{GCalt} \\ T_{Galt} \Leftrightarrow T_{GCneu} < T_{GCalt} \end{cases}$$

5. Anforderungsliste

In diesem Abschnitt wird erläutert, wie eine einzelne Testphase durchgeführt werden muss. Die genau spezifizierte Vorgehensweise zur Erfassung der Daten ist entscheidend für die Qualität der Ergebnisse. Folgende Arbeitsschritte umfassen eine Testphase:

- Konfigurieren von RIP_Infinity auf 64 im RIP-Daemon
- Anlegen einer korrekten XML-Datei, die das vorliegende Netzwerkszenario in allen Aspekten korrekt wiedergibt.
  - (a) die vollständige Anzahl an Routern
  - (b) die vollständige Anzahl der Interfaces
  - (c) sämtliche execute-Befehle
- Anlegen einer Konfigurationsdatei mit allen benötigten Daten
  - (a) Auflistung aller Router
  - (b) Abstimmung des periodischen Update-Timers auf die Konfiguration der einzelnen Router
  - (c) Abstimmung der Sendereihenfolge von Updatenachrichten auf die Timer-Einstellungen, um den Count-To-Infinity-Effekt zuverlässig auszulösen.
- Abstimmung der Konfigurationsdateien jedes Routers auf die gewünschten Timer-Einstellungen
- Starten der Simulationsumgebung durch den VNUML-Parser
- Starten der XML-Szenariodatei mithilfe des XT-Peer

- Einstellen der RIP/RIPMTI-Eigenschaften aller Router
- Durchführung der Testphase mit 100 Durchläufen durch den AUTO-Generator
- Nach Abschluss der Testphase sind die Ergebnisse zwecks Nachvollziehbarkeit zu speichern
- Auswertung der Ergebnisse

  (a) Bestimmung der Konvergenzzeiten und Berechnung von repräsentativen Durchschnittswerten, indem für jeden aufgetretenen Count-To-Infinity-Effekt die Dauer erfasst wird. Interessant ist hier die Zeit zwischen abgelaufenem T_{TT} und dem Anspringen des T_{GC} .

  (b) Beschreibung des Verhaltens aller relevanten Ereignisse, die mit dem Count-To-Infinity-Effekt in Verbindung stehen bzw. Beschreibung des Verhaltens aller Router, die im RIPMTI-Modus arbeiten.

  (c) Bewertung der gewonnenen Erkenntnisse hinsichtlich der Leistungsfähigkeit und Effizienz des RIPMTI-Algorithmus im Gegensatz zu RIP bzw. Aufzeigen von Problemsituationen zur Fehlerbehebung.

Alle gewonnenen Daten befinden sich auf einer DVD, die dieser Diplomarbeit beiliegt. Dadurch ist eine rasche Nachvollziehbarkeit der gewonnenen Daten möglich. Außerdem lassen sich damit zukünftige Testdurchläufe mit einem weiterentwickelten RIPMTI-Algorithmus schnell vergleichen und so die Softwareentwicklung beschleunigen.

6. Anwendungsbeispiele

6.1 Y-Szenario

Das Y-Szenario in Abbildung 9 stellt eine grundlegende Topologie dar. Hierauf aufbauend wurde der Y-Test entwickelt, der mittlerweile zum Simple-Loop-Test erweitert und verändert wurde.

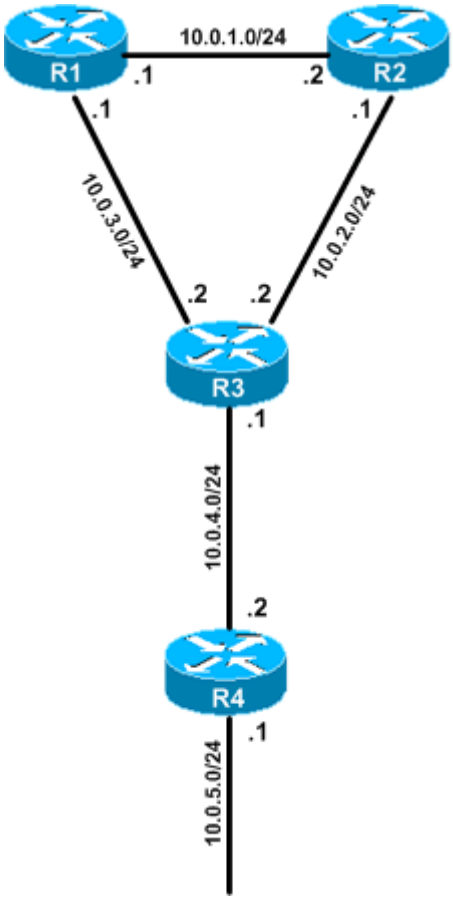


Abbildung 9: Topologie Y-Szenario

Modellbeschreibung

Dieses Szenario stellt die kleinste Y-Topologie dar, in der der Count-To-Infinity-Effekt auftreten kann. Es besteht nur aus einem Zyklus über R1, R2 und R3 und einem weiteren Router R4, der mit R3 verbunden ist.

Erzeugung des Count-To-Infinity-Effekts

Der Count-To-Infinity-Effekt soll erzeugt werden, wenn das Netz 10.0.5.0/24 ausfällt und im oberen Zyklus das Versenden einer Falschnachricht provoziert wird. Die entsprechende Konfigurationsdatei „**Y-Szenario1**“ befindet sich in Anhang A. Die darin beschriebene Updatereihenfolge lässt nach der Konvergenz des Netzwerkes das Netz 10.0.5.0/24 ausfallen, indem die Verbindung von R4 auf Interface 1 (10.0.4.2) zu R3 blockiert wird. R3 setzt nach dem Timeout die Metrik zu diesem Netz auf den Wert 64 und benachrichtigt anschließend R2. Das Interface 1 (10.0.3.2) von R3 blockiert zu diesem Zeitpunkt eine Weiterleitung der Updatenachricht an R1. R2 lernt die neue Routeninformation. Damit nicht R1 sofort von R2 über den Ausfall des Netzes informiert wird, blockiert R2 ebenfalls kurzzeitig die Verbindung auf Interface 1 (10.0.1.2) zu R1. R1 (erhält dadurch den Status: False-Router) sendet sein regelmäßiges Update mit der alten ungültigen Metrik an R2, der diese ungültige Information als Alternativroute akzeptiert, in seine Routingtabelle

einträgt und anschließend an R1 weiterleitet. R1 reagiert genauso wie R2 und damit läuft die Falschnachricht im Zyklus ringsum. Zunächst bestand das Problem darin, eine Konfiguration zu ermitteln, bei der auch hinreichend sicher ein Count-To-Infinity-Effekt entsteht. In mehreren Anläufen mit verschiedenen Einstellungen führte die Konfiguration zu selten zu einem Count-To-Infinity-Effekt. Ursache dafür sind die CTI-Phase, in der das Falsch-Update erzwungen wird und die Ausbreitungsphase, in der diese Nachricht von den Routern durchgeschleift wird. Man muss in der CTI-Phase darauf achten, dass der False-Router besonders bei niedrigen Timer-Einstellungen genug Zeit hat, seine Falschnachricht zu verschicken. Da der Zyklus minimal ist, darf der False-Router darüber hinaus nicht gleichzeitig mit den anderen Routern triggern. In diesem Fall kann es vorkommen, dass das Falsch-Update augenblicklich von einer gültigen Nachricht überschrieben wird. Er darf aber auch nicht zu lange warten. Stellt man die Verzögerung für den False-Router zu hoch, dann verliert der Source-Router das Netz 10.0.5.0/24 nach Ablauf seines Garbage-Collection-Timers und erhält erst anschließend das Falsch-Update, welches er natürlich als neue Route akzeptiert und in seine Routingtabelle einträgt. Es galt folgendes Routerverhalten zu erzeugen:

- R1 darf die Metrik 64 nicht von R3 bekommen: R3(**x**,0,0)
- R2 soll die Metrik 64 von R3 bekommen
- R1 muss genug Zeit haben, um ein Falsch-Update zu erzeugen: R2(**x**,0)
- R1 muss sein Falsch-Update etwas verzögert an R2 schicken, damit die falsche Metrik nicht direkt vom gleichzeitigen Update von R3 überschrieben wird: R1(0.x,0)a, wobei x hier ein Wert zwischen 0 und 9 sein kann abhängig von den gewählten Timer-Einstellungen
- Die Wartezeit darf aber auch nicht zu lang sein, da R3 sonst das Netz 10.0.5.0/24 komplett verlieren kann

Ergebnis des RIPMTI-Algorithmus

Im konvergenten Zustand erreicht R3 das ausgefallene Netz mit der Metrik 2, R1 und R2 jeweils mit der Metrik 3. Nach Ausfall des Netzes und Abarbeitung der Konfigurationsdatei, sendet R1 die Metrik 3 als Erreichbarkeitsinformation an R2. R2 hat zu diesem Zeitpunkt die Metrik 64 in seiner Routingtabelle und übernimmt die vermeintlich kürzere Route für sich mit der Metrik 4. Da R3 nun von R2 lernt, trägt er bei sich entsprechend die Metrik 5 ein und im Folgenden wandert diese Falschnachricht in Dreierschritten im Zyklus umher.

R3 zählt wie Abbildung 10 und 11 zeigen von 5, 8, 11, 14, 17 bis 62, 64 hoch, bevor der Garbage-Collection-Timer anspringt und die Routeninformation verwirft. Es sind 21 Updatenachrichten notwendig bevor R3 konvergieren kann.

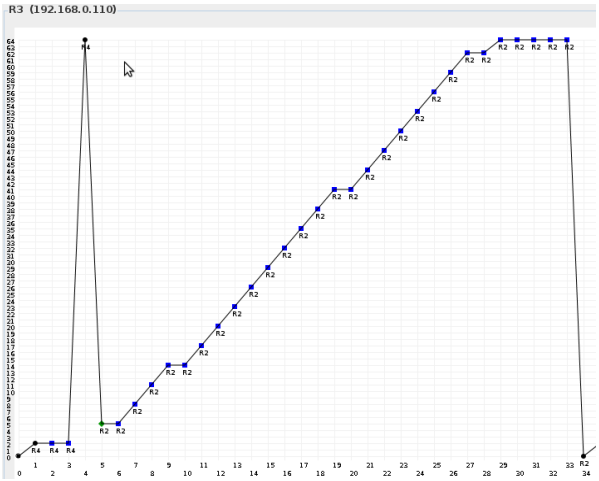


Abbildung 10: Netzgraph: CTI im Y-Szenario

10-60-40-CTI

graph

R4 R1 R3 R2

10.0.5.0/24 10.0.4.0/24 10.0.3.0/24 10.0.2.0/24 10.0.1.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network
initial	00:00:00:000	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	00:35	R3->R4
update	00:00:00:671	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	01:00	R3->R4
update	00:00:10:671	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	01:00	R3->R4
timeout	00:01:10:682	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:40	R3->R4
update	00:01:21:479	R(n)	10.0.5.0/24	10.0.2.1	5	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:25:161	R(n)	10.0.5.0/24	10.0.2.1	5	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:31:148	R(n)	10.0.5.0/24	10.0.2.1	8	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:33:149	R(n)	10.0.5.0/24	10.0.2.1	11	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:34:161	R(n)	10.0.5.0/24	10.0.2.1	14	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:36:176	R(n)	10.0.5.0/24	10.0.2.1	14	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:38:214	R(n)	10.0.5.0/24	10.0.2.1	17	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:42:226	R(n)	10.0.5.0/24	10.0.2.1	20	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:47:201	R(n)	10.0.5.0/24	10.0.2.1	23	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:47:241	R(n)	10.0.5.0/24	10.0.2.1	26	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:48:282	R(n)	10.0.5.0/24	10.0.2.1	29	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:50:255	R(n)	10.0.5.0/24	10.0.2.1	32	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:52:263	R(n)	10.0.5.0/24	10.0.2.1	35	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:54:170	R(n)	10.0.5.0/24	10.0.2.1	38	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:56:173	R(n)	10.0.5.0/24	10.0.2.1	41	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:56:225	R(n)	10.0.5.0/24	10.0.2.1	41	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:01:57:313	R(n)	10.0.5.0/24	10.0.2.1	44	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:01:328	R(n)	10.0.5.0/24	10.0.2.1	47	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:06:244	R(n)	10.0.5.0/24	10.0.2.1	50	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:06:383	R(n)	10.0.5.0/24	10.0.2.1	53	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:11:388	R(n)	10.0.5.0/24	10.0.2.1	56	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:14:392	R(n)	10.0.5.0/24	10.0.2.1	59	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:15:414	R(n)	10.0.5.0/24	10.0.2.1	62	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:17:259	R(n)	10.0.5.0/24	10.0.2.1	62	10.0.2.1	0	01:00	R3->R2->R1->R3
update	00:02:19:432	R(n)	10.0.5.0/24	10.0.2.1	64	10.0.2.1	0	00:40	R3->R2->R1->R3
update	00:02:29:264	R(n)	10.0.5.0/24	10.0.2.1	64	10.0.2.1	0	00:30	R3->R2->R1->R3
update	00:02:39:271	R(n)	10.0.5.0/24	10.0.2.1	64	10.0.2.1	0	00:20	R3->R2->R1->R3
update	00:02:48:280	R(n)	10.0.5.0/24	10.0.2.1	64	10.0.2.1	0	00:11	R3->R2->R1->R3
update	00:02:57:292	R(n)	10.0.5.0/24	10.0.2.1	64	10.0.2.1	0	00:02	R3->R2->R1->R3
garbage	00:02:59:451	R(n)	10.0.5.0/24	10.0.2.1	0	10.0.2.1	0	0	R3->R2->R1->R3
update	00:03:00:736	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	01:00	R3->R4
update	00:03:10:753	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	01:00	R3->R4

Abbildung 11: Analysetabelle: CTI im Y-Szenario

Die Konvergenzzeiten des RIP-Algorithmus in Tabelle 3 zeigen, wie langsam das Netz auf den Ausfall des Netzes 10.0.5.0/24 reagiert.

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	43,2s
T_1	6s / 36s / 24s	47,5s
T_2	10s / 60s / 40s	57,3s
T_3	30s / 180s / 120s	83,1s

Tabelle 3: Ergebnisse versch. T-Einstellungen beim CTI im Y-Szenario

Mit den langsamen Timer-Einstellungen T_3 (30s) dauert es mit 83,1 Sekunden deutlich über einer Minuten bis das Netz in einen konvergenten Zustand gelangt. Eine Beschleunigung der Timer-Einstellungen wirkt sich positiv auf die Konvergenzzeit aus. Kommunizieren alle Router mit T_2 dreimal so schnell, dann reagiert das Netzwerk um 31% schneller darauf. Um den Faktor 5 gesenkte Updatezeiten (T_1) lassen das Netz um ca. 43% schneller konvergieren und bei 3 Sekunden Update-Timern sogar knapp 50% schneller. Allerdings steht der Geschwindigkeitsbonus nicht mehr im Verhältnis zur Verringerung der Timer-Einstellungen. Sie lassen sich zwar deutlich heruntersetzen, aber der Leistungszuwachs wird dabei immer geringer.

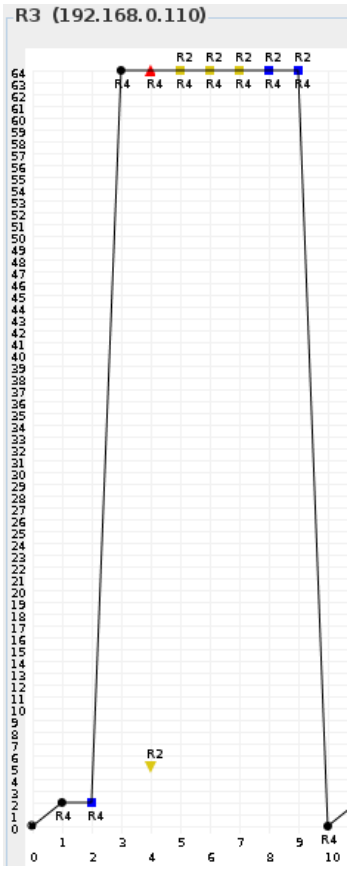
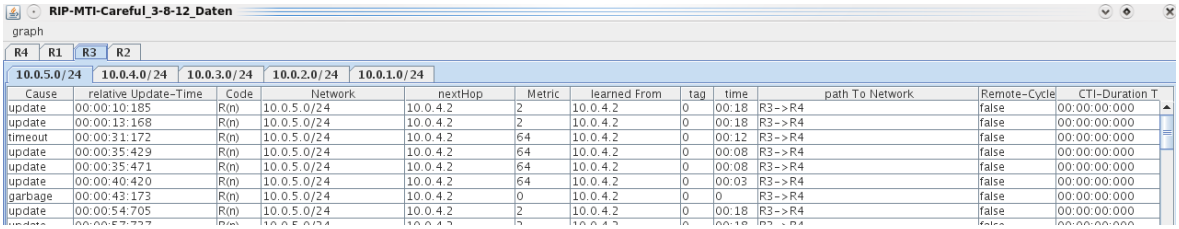


Abbildung 12: Netzgraph: RIPMTI im Y-Szenario

Ergebnis des RIPMTI-Algorithmus

Zu betrachten ist wieder R3 als Schlüsselposition im gesamten Netzwerk. Nachdem R1 die Falschnachricht losgeschickt hat und über R2 zu R3 gelangt ist, wird sie sofort als Schleifeninformation erkannt und abgefangen. Es entstehen nur 2 Falschnachrichten, die von R3 sofort korrigiert werden. Diese stehen im extremen Gegensatz zu 63 Updatenachrichten, die der Count-To-Infinity-Effekt auslösen würde, da jeder der drei Router im

Zyklus jeweils 21 Updatenachrichten versenden muss bis das Netzwerk konvergiert. Abbildung 12 und 13 zeigen den Graphen und die tabellarische Auflistung der gemessenen Werte wenn R3 im RIPMTI-Modus arbeitet. Zu beachten ist, dass der Request-Timer auf einen Wert größer als 15 Sekunden gestellt werden muss, da sich bis maximal 15 Sekunden nach Ablauf des Timeout-Timers eine Falschnachricht im Zyklus befinden kann, die nach Entfernen der Route aus der Routingtabelle von R3 verspätet eintreffen kann. Der daraufhin angepasste RIPMTI-Algorithmus, berechnet den Garbage-Collection-Timer entsprechend des Schleifenumfangs genau auf diesen Wert.


RIP-MTI-Careful_3-8-12_Daten

graph

R4 | R1 | R3 | R2

10.0.5.0/24 | 10.0.4.0/24 | 10.0.3.0/24 | 10.0.2.0/24 | 10.0.1.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:00:10:185	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000
update	00:00:13:168	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000
timeout	00:00:31:172	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:12	R3->R4	false	00:00:00:000
update	00:00:35:429	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:08	R3->R4	false	00:00:00:000
update	00:00:35:471	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:08	R3->R4	false	00:00:00:000
update	00:00:40:420	R(n)	10.0.5.0/24	10.0.4.2	64	10.0.4.2	0	00:03	R3->R4	false	00:00:00:000
garbage	00:00:43:173	R(n)	10.0.5.0/24	10.0.4.2	0	10.0.4.2	0	0	R3->R4	false	00:00:00:000
update	00:00:54:705	R(n)	10.0.5.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000

Abbildung 13: Analysetabelle: RIPMTI im Y-Szenario

Erkannt wird der Simple-Loop anhand folgender Berechnung:

$$m_{IF1}^{(R3,10.0.5.0/24)} = Route\ über\ R3 \rightarrow R1 \rightarrow R2 \rightarrow R3 \rightarrow R4$$

$$m_{IF3}^{(R3,10.0.5.0/24)} = Route\ über\ R3 \rightarrow R4$$

$$m_{IF1}^{(R3,10.0.5.0/24)} + m_{IF3}^{(R3,10.0.5.0/24)} - 1 \geqslant msilm_{(IF1,IF3)}^{R3}$$

$$5 + 2 - 1 = 6 \geqslant 127\,(f)$$

$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$

$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Zusammenfassung

Der RIPMTI-Algorithmus funktioniert in diesem Szenario einwandfrei und führt zu einer deutlichen Verbesserung der Netzkonvergenz. Der Geschwindigkeitsgewinn in Abgrenzung zu RIP ist hoch. Nimmt man die komplette Zeitspanne vom Ausfallen der Route bis zum Entfernen der ungültigen Route aus der Routingtabelle vergehen bei RIP unter Verwendung der T_3-Einstellungen insgesamt 383,1 Sekunden und bei RIPMTI 300 Sekunden. Werden die T_0-Einstellungen verwendet, braucht RIP 76,2 Sekunden und RIPMTI 33 Sekunden, was für RIPMTI nicht nur im Gesamtvergleich extrem schneller ist, sondern auch bei diesen speziellen Einstellungen das Netzwerk in der Hälfte der Zeit konvergieren lässt wenn man die Zeiten für den Timeout- und Garbage-Collection-Timer mit einbezieht.

6.2 Y-Many

Dieses Szenario basiert auf dem Y-Szenario als Grundmodell und soll zeigen, ob sich die Konvergenzzeit eines Netzwerkes ändert, wenn gleichzeitig der Ausfall mehrerer Netze verarbeitet werden muss.

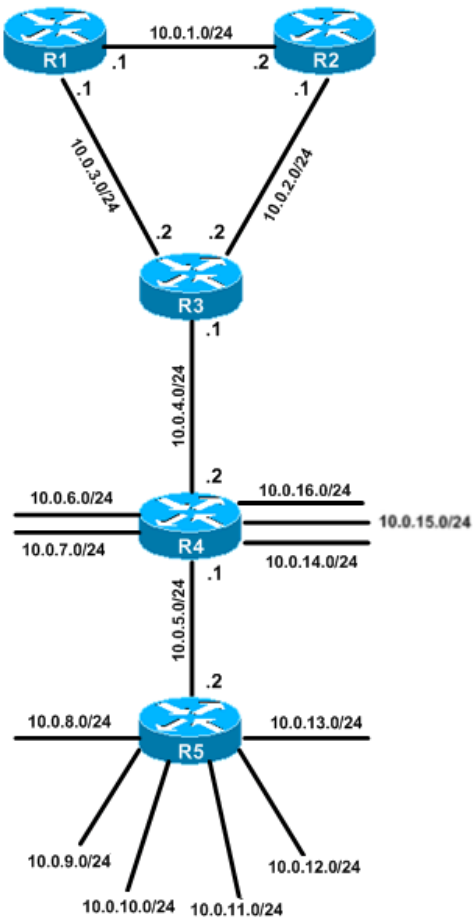


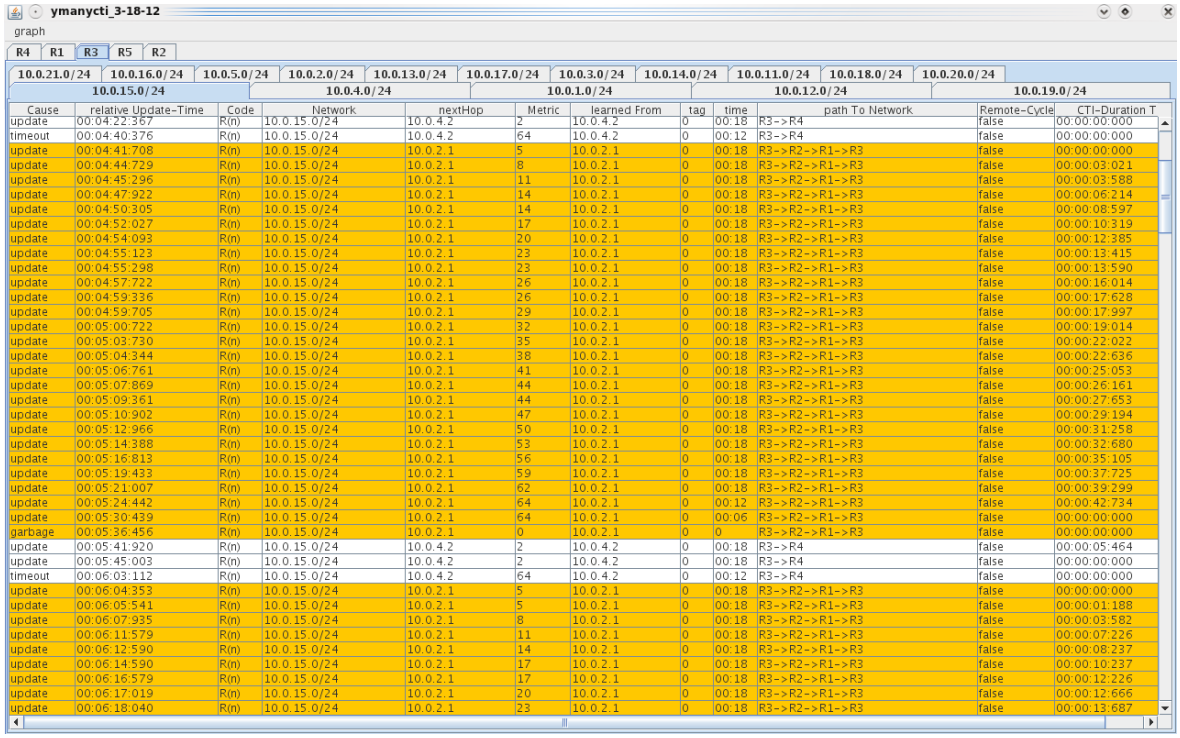
Abbildung 14: Topologie: Y-Many

Modellbeschreibung

Das Modell enthält fünf Router. R1, R2 und R3 bilden einen Zyklus. R3 besitzt eine zusätzliche Verbindung zu R4 und dieser wiederum zu R5. R4 und R5 bilden jeweils einen Knotenpunkt für mehrere Netze. Die Verbindung zwischen R3 und R4 wird künstlich unterbrochen, sodass das gesamte Teilnetz unterhalb von R4 für den Zyklus unerreichbar wird.

Erzeugung des Count-To-Infinity-Effekts

Insgesamt sollen zwölf Netze ausfallen. Die Netze 10.0.5.0/24 bis 10.0.16.0/24. Im Grunde entspricht die Konfigurationsdatei aus Anhang B der des ursprünglichen Y-Szenarios. Mit R5 ist zwar ein weiterer Router hinzugekommen, aber die Ausfallsituation geschieht an der gleichen Stelle zwischen R3 und R4. R3 beschreibt den Source-Router als Bindeglied zwischen dem Zyklus und den ausfallenden Netzen. R1 wird per Konfigurationsdatei zum False-Router deklariert, der seine aktuelle ungültige Routinginformation zu allen Netzen unterhalb von R4 an R2 weiterleitet und damit den Anfang des Count-To-Infinity-Effekts beschreibt. Die im Anhang befindlichen Konfigurationsdateien unterscheiden sich nur darin, dass „Ymany1“ im Gegensatz zu „Ymany2“ über eine getrennte Ausbreitungs- und CTI-Phase verfügt, damit R1 bei T_0- und T_1-Einstellungen ausreichend Zeit hat, seine Falschnachricht zu versenden. Bei T_2 und T_3 konnten diese beiden Phasen zusammengelegt werden.


ymanycti_3-18-12

graph

R4 R1 R3 R5 R2

10.0.21.0/24 10.0.16.0/24 10.0.5.0/24 10.0.2.0/24 10.0.13.0/24 10.0.17.0/24 10.0.3.0/24 10.0.14.0/24 10.0.11.0/24 10.0.18.0/24 10.0.20.0/24

10.0.15.0/24 10.0.4.0/24 10.0.1.0/24 10.0.12.0/24 10.0.19.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:04:22:367	R(n)	10.0.15.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000
timeout	00:04:40:376	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:12	R3->R4	false	00:00:00:000
update	00:04:41:708	R(n)	10.0.15.0/24	10.0.2.1	5	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:00:000
update	00:04:44:729	R(n)	10.0.15.0/24	10.0.2.1	8	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:03:021
update	00:04:45:296	R(n)	10.0.15.0/24	10.0.2.1	11	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:03:588
update	00:04:47:922	R(n)	10.0.15.0/24	10.0.2.1	14	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:06:214
update	00:04:50:305	R(n)	10.0.15.0/24	10.0.2.1	14	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:08:597
update	00:04:52:027	R(n)	10.0.15.0/24	10.0.2.1	17	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:10:319
update	00:04:54:093	R(n)	10.0.15.0/24	10.0.2.1	20	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:12:385
update	00:04:55:123	R(n)	10.0.15.0/24	10.0.2.1	23	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:13:415
update	00:04:55:298	R(n)	10.0.15.0/24	10.0.2.1	23	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:13:590
update	00:04:57:722	R(n)	10.0.15.0/24	10.0.2.1	26	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:16:014
update	00:04:59:336	R(n)	10.0.15.0/24	10.0.2.1	26	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:17:628
update	00:04:59:705	R(n)	10.0.15.0/24	10.0.2.1	29	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:17:997
update	00:05:00:722	R(n)	10.0.15.0/24	10.0.2.1	32	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:19:014
update	00:05:03:730	R(n)	10.0.15.0/24	10.0.2.1	35	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:22:022
update	00:05:04:344	R(n)	10.0.15.0/24	10.0.2.1	38	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:22:636
update	00:05:06:761	R(n)	10.0.15.0/24	10.0.2.1	41	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:25:053
update	00:05:07:869	R(n)	10.0.15.0/24	10.0.2.1	44	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:26:161
update	00:05:09:361	R(n)	10.0.15.0/24	10.0.2.1	44	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:27:653
update	00:05:10:902	R(n)	10.0.15.0/24	10.0.2.1	47	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:29:194
update	00:05:12:966	R(n)	10.0.15.0/24	10.0.2.1	50	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:31:258
update	00:05:14:388	R(n)	10.0.15.0/24	10.0.2.1	53	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:32:680
update	00:05:16:813	R(n)	10.0.15.0/24	10.0.2.1	56	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:35:105
update	00:05:19:433	R(n)	10.0.15.0/24	10.0.2.1	59	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:37:725
update	00:05:21:007	R(n)	10.0.15.0/24	10.0.2.1	62	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:39:299
update	00:05:24:442	R(n)	10.0.15.0/24	10.0.2.1	64	10.0.2.1	0	00:12	R3->R2->R1->R3	false	00:00:42:734
update	00:05:30:439	R(n)	10.0.15.0/24	10.0.2.1	64	10.0.2.1	0	00:06	R3->R2->R1->R3	false	00:00:00:000
garbage	00:05:36:456	R(n)	10.0.15.0/24	10.0.2.1	0	10.0.2.1	0	0	R3->R2->R1->R3	false	00:00:00:000
update	00:05:41:920	R(n)	10.0.15.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:05:464
update	00:05:45:003	R(n)	10.0.15.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000
timeout	00:06:03:112	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:12	R3->R4	false	00:00:00:000
update	00:06:04:353	R(n)	10.0.15.0/24	10.0.2.1	5	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:00:000
update	00:06:05:541	R(n)	10.0.15.0/24	10.0.2.1	5	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:01:188
update	00:06:07:935	R(n)	10.0.15.0/24	10.0.2.1	8	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:03:582
update	00:06:11:579	R(n)	10.0.15.0/24	10.0.2.1	11	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:07:226
update	00:06:12:590	R(n)	10.0.15.0/24	10.0.2.1	14	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:08:237
update	00:06:14:590	R(n)	10.0.15.0/24	10.0.2.1	17	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:10:237
update	00:06:16:579	R(n)	10.0.15.0/24	10.0.2.1	17	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:12:226
update	00:06:17:019	R(n)	10.0.15.0/24	10.0.2.1	20	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:12:666
update	00:06:18:040	R(n)	10.0.15.0/24	10.0.2.1	23	10.0.2.1	0	00:18	R3->R2->R1->R3	false	00:00:13:687

Abbildung 15: Analysetabelle: CTI im Y-Many-Szenario

Ergebnisse des RIP-Algorithmus

Im Moment der Konvergenz hat R3 zu den Netzen 10.0.5.0/24 bis 10.0.16.0/24 eine Metrik von 2 bzw. 3 je nachdem ob das entsprechende Netz an R4 bzw. R5 hängt. Nach Ablauf des Timeout-Timers und der, durch die Konfigurationsdatei, provozierten Entstehung des Count-To-Infinity-Effekts, sendet R1 in seiner Falschnachricht ungültige Routinginformationen zu jedem ausgefallenen Netz. Jeder Router innerhalb des Zyklus, oder möglicherweise verbunden mit dem Zyklus, muss mit einem einzigen erhaltenen Update zwölf Routingtabelleneinträge aktualisieren und zusätzlich ein Update mit diesen Änderungen an seine Nachbarn verschicken. Es entsteht ein höherer Verarbeitungsaufwand als im Y-Szenario.

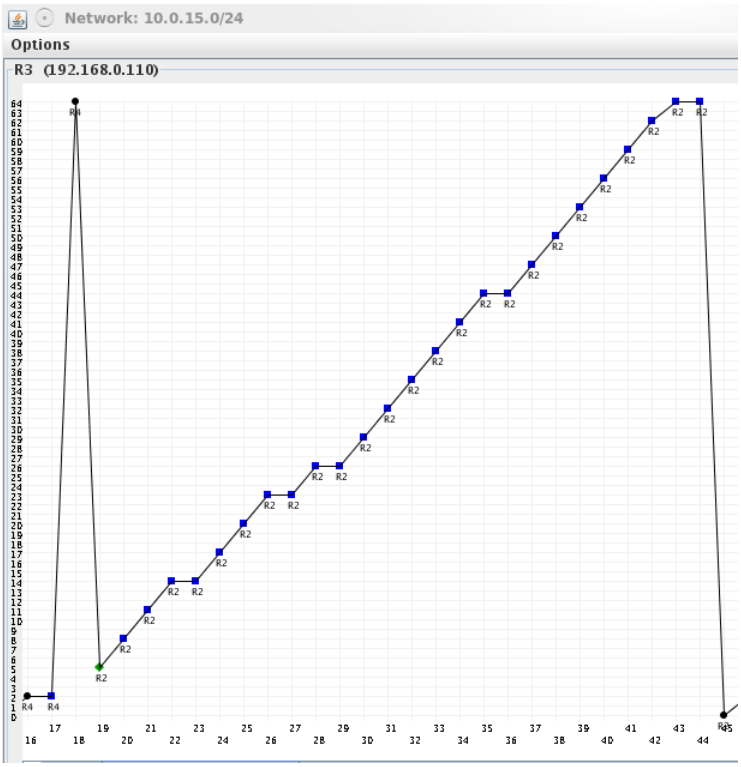


Abbildung 16: Netzgraph: CTI im Y-Many-Szenario

Die Abbildungen 15 und 16 zeigen stellvertretend für die Graphen und Tabellen zu den übrigen betroffenen Netzen und Router, dass die Menge der ausgefallenen Netze keinen Einfluss auf die Anzahl der zu versendenden Updatenachrichten und ebenso wenig auf die Dauer des gesamten Count-To-Infinity-Effekts hat. Die Verarbeitungsgeschwindigkeit für mehrere Änderungen in einer Routingtabelle hängt vielmehr von der Performance, also der Rechenleistung, des einzelnen Routers, ab und liegt im Millisekunden-Bereich. Ebenso wie beim Y-Szenario benötigt ein Router 21 Updatenachrichten bis zur Konvergenz, bedingt durch die Metrik 64 und einen Schleifenumfang von drei Routern. Dieses Szenario soll jedoch auch aufzeigen, welche Auswirkungen viele ausgefallene Netze in einem Netzwerk haben können. Da für jedes Netz ein eigener Count-To-Infinity-Effekt ausgelöst werden kann, steigt damit das Datenaufkommen auf den Verbindungsleitungen. Eine getriggerte Updatenachricht enthält in diesem Beispiel viel mehr Informationen über Metrik-Änderungen. In noch größerem Maßstab kann sich die Kommunikationsdatenlast auf die Übertragungskapazität einer Verbindungsleitung auswirken. Die Bandbreite jeder Leitung ist durch technische Spezifikationen beschränkt. Steigt der Anteil an Kommunikationsdatenverkehr zwischen Routern, schmälert das den Anteil von Nutzdaten, die versendet werden können. Darüber

hinaus kann das gesamte Netzwerk durch den Nutzdatenanteil komplett lahm gelegt werden. Existieren viele ungültige Routen, werden die Nutzdaten zu allen ausgefallenen Netzen durch den Zyklus geschleift und können damit die volle Kapazität der Leitungen ausschöpfen. Für die Dauer des Count-To-Infinity-Effekts können keine neuen Daten versendet werden und damit ist das Netzwerk in einem blockierten Zustand.

Tabelle 4 enthält die ermittelten Durchschnittswerte für die Konvergenzzeit unter Berücksichtigung der verschiedenen Timer-Einstellungen. Im Mittel ist der Count-To-Infinity-Effekt in 98% der Testdurchläufe aufgetreten.

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	37,8s
T_1	6s / 36s / 24s	45,5s
T_2	10s / 60s / 40s	57,3s
T_3	30s / 180s / 120s	72,6s

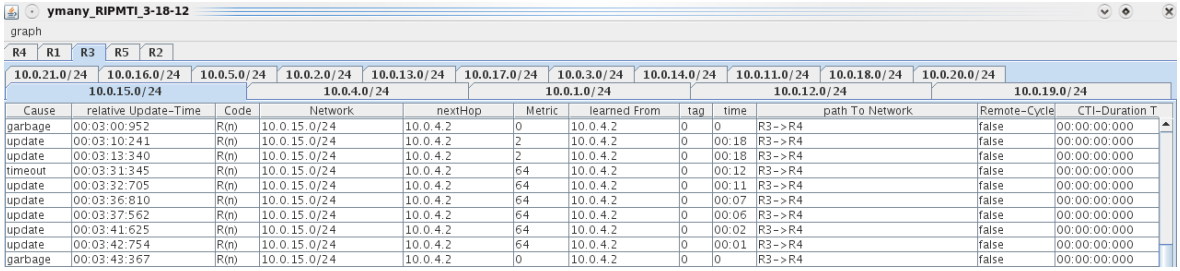
Tabelle 4: Ergebnisse versch. T-Einstellungen beim CTI im Y-Many-Szenario

Vergleicht man die Konvergenzzeiten mit der Tabelle zum Y-Szenario, wird deutlich, dass die Abweichungen nur geringfügig sind. Demzufolge ist der Geschwindigkeitsgewinn mit knapp 50% kürzerer Konvergenzzeit bei T_0 gegenüber T_3 ähnlich.

Ergebnisse des RIPMTI-Algorithmus

Der neue RIPMTI-Algorithmus funktioniert zu 100%. Trotz der gestiegenen Menge an Routinginformationen, erkennt RIPMTI jede Schleifeninformation zu allen ausgefallenen Netzen. R3 übernimmt als Source-Router die Ermittlung aller nötigen Informationen und ist daher ausreichend, um den Count-To-Infinity-Effekt zu erkennen und alle ungültigen Routen abzulehnen. Die Abbildungen 17 und 18 zeigen den entsprechenden Netzgraph und die Analysetabelle zu einem ausgefallenen Netz.

Abbildung 17: Netzgraph: RIPMTI im Y-Many-Szenario


ymany_RIPMTI_3-18-12

graph

R4 | R1 | R3 | R5 | R2

10.0.21.0/24 | 10.0.16.0/24 | 10.0.5.0/24 | 10.0.2.0/24 | 10.0.13.0/24 | 10.0.17.0/24 | 10.0.3.0/24 | 10.0.14.0/24 | 10.0.11.0/24 | 10.0.18.0/24 | 10.0.20.0/24

10.0.15.0/24 | 10.0.4.0/24 | 10.0.1.0/24 | 10.0.12.0/24 | 10.0.19.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
garbage	00:03:00:952	R(n)	10.0.15.0/24	10.0.4.2	0	10.0.4.2	0	0	R3->R4	false	00:00:00:000
update	00:03:10:241	R(n)	10.0.15.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000
update	00:03:13:340	R(n)	10.0.15.0/24	10.0.4.2	2	10.0.4.2	0	00:18	R3->R4	false	00:00:00:000
timeout	00:03:31:345	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:12	R3->R4	false	00:00:00:000
update	00:03:32:705	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:11	R3->R4	false	00:00:00:000
update	00:03:36:810	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:07	R3->R4	false	00:00:00:000
update	00:03:37:562	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:06	R3->R4	false	00:00:00:000
update	00:03:41:625	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:02	R3->R4	false	00:00:00:000
update	00:03:42:754	R(n)	10.0.15.0/24	10.0.4.2	64	10.0.4.2	0	00:01	R3->R4	false	00:00:00:000
garbage	00:03:43:367	R(n)	10.0.15.0/24	10.0.4.2	0	10.0.4.2	0	0	R3->R4	false	00:00:00:000

Abbildung 18: Analysetabelle: RIPMTI im Y-Many-Szenario

Der Simple-Loop-Test ermittelt den Gültigkeitswert jeder Route separat. Für jedes Netz liefert die Berechnung folgende Ergebnisse:

- Netze, die über R4 direkt erreichbar sind:

$$m_{IF1}^{(R3,10.0.5.0/24)} = Route\ über\ R3 \rightarrow R1 \rightarrow R2 \rightarrow R3 \rightarrow R4$$
$$m_{IF3}^{(R3,10.0.5.0/24)} = Route\ über\ R3 \rightarrow R4$$
$$m_{IF1}^{(R3,10.0.5.0/24)} + m_{IF3}^{(R3,10.0.5.0/24)} - 1 \geqslant msilm_{(IF1,IF3)}^{R3}$$
$$5 + 2 - 1 = 6 \geqslant 127\,(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

- Netze, die über R5 direkt erreichbar sind:

$$m_{IF1}^{(R3,10.0.10.0/24)} = Route\ über\ R3 \rightarrow R1 \rightarrow R2 \rightarrow R3 \rightarrow R4 \rightarrow R5$$
$$m_{IF3}^{(R3,10.0.10.0/24)} = Route\ über\ R3 \rightarrow R4 \rightarrow R5$$
$$m_{IF1}^{(R3,10.0.10.0/24)} + m_{IF3}^{(R3,10.0.10.0/24)} - 1 \geqslant msilm_{(IF1,IF3)}^{R3}$$
$$6 + 3 - 1 = 6 \geqslant 127\,(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Dadurch verhindert RIPMTI, dass sich unkontrolliert viel Datenverkehr auf falschen Routen bewegt und das Netzwerk unnötig belastet.

Zusammenfassung

In diesem Szenario soll dargestellt werden, dass eine Metrikbegrenzung von 64 im Falle eines Count-To-Infinity-Effekts erheblichere Auswirkungen auf die Stabilität des Netzwerkes haben kann als mit der Metrikbegrenzung von 16. Hohes Nutzdatenaufkommen bleibt auf ungültigen Routen länger in Umlauf und verzögert weiteren Datenverkehr. Die hohe Anzahl ausgefallener Routen sorgt für ein linear wachsenden Datenaufkommen auf den Verbindungsleitungen und verstopft diese allmählich. Es ist also unbedingt erforderlich, diese Beeinträchtigung schnellstmöglich aufzulösen. Auch wenn beschleunigte Timer-Einstellungen die Situation schneller bewältigen können, muss RIPMTI unmittelbar nach Entstehen des Count-To-Infinity-Effekts eingreifen, um Verbindungsleitungen sofort wieder verfügbar zu machen.

Hier wirkt RIPMTI am effektivsten, weil er nicht zulässt, dass alle drei Router im Zyklus einen CTI zu jedem ausgefallenen Netz entwickeln und eine große Menge an fehlgeleitetem Datenverkehr entstehen kann. Der Geschwindigkeitsgewinn im Gesamtvergleich aller T-Einstellungen zeigt auch hier, dass T_0 mit RIPMTI in weniger als 50% der Zeit der vergleichbaren RIP-Werte inklusive T_{TT} und T_{GC} konvergieren kann.

6.3 Big-Loop

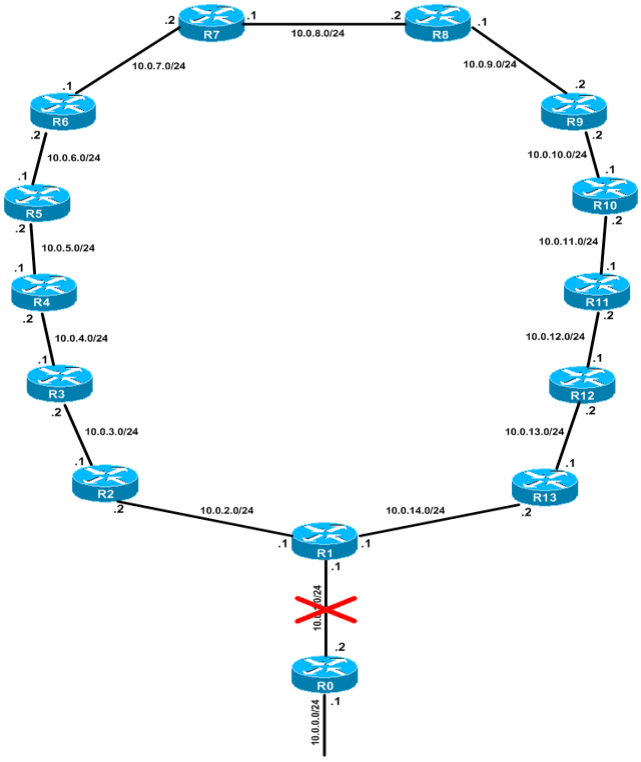


Abbildung 19: Topologie: Big-Loop

Das Big-Loop-Szenario in Abbildung 19 wurde entwickelt, um einen maximalen Schleifenumfang unter RIP_Infinity 16 abzubilden. In diesem Fall besteht der Zyklus aus einer endlichen Anzahl von Routern, sodass zu einem ausfallenden Netz bereits im ersten Schleifendurchlauf eine Metrik von 15 entstehen kann. Dieser Wert liegt einen Hop unterhalb des ursprünglichen Unerreichbarkeitswertes, der Metrik 16. Zum Zeitpunkt der Testphase dieses Szenarios war es technisch nicht möglich, den Schleifenumfang an die gewünschte Metrikerhöhung auf 64 anzupassen und 62 Router zu simulieren. Die Performance der zur Verfügung stehenden Rechner reichte nicht aus. Im Laufe der Testdurchläufe stellte sich heraus, dass das auch nicht notwendig war, da die gewonnenen Ergebnisse aus dem vorliegenden Szenario ausreichend sind.

Modellbeschreibung

Das Szenario entspricht grundsätzlich der Y-Topologie. In diesem Fall wurde die Anzahl der Router, die sich im Zyklus befinden, auf einen bestimmten Wert erhöht, der bei einer Count-To-Infinity-Metrik von 16 die theoretisch maximal mögliche Schleifengröße darstellt. Dadurch erhält man ein Netzwerk, in dem ein Count-To-Infinity-Effekt bereits nach einem Durchlauf von alleine endet, da R2 die Route zum ausfallenden Netz 10.0.0.0/24 als unerreichbar bewertet. In diesem Fall bringt der RIPMTI-Algorithmus keinen Gewinn. Inkrementiert man die Metrik von 16 auf 64 oder höher, entsteht der Count-To-Infinity-Effekt wieder. Es befinden sich 13 Router (R1 bis R13) im Zyklus und R1 ist zusätzlich an einen weiteren Router R0 angebunden.

Erzeugung des Count-To-Infinity-Effekts

In den Konfigurationsdateien aus Anhang C ist vorgesehen, dass das Netz 10.0.0.0/24 ausfallen soll. Dazu wird die Verbindung auf Interface 2 (10.0.0.1) von R0 in Richtung R1 unterbrochen, sodass R1 nach Ablauf des Timeout-Timers die Unerreichbarkeit des Netzes feststellt. Ab diesem Moment muss die Abfolge der Ausbreitungs- und CTI-Phase an die jeweiligen Timer-Einstellungen angepasst werden.

Anhang C **„Big-Loop1“** funktioniert bei kurzen Timer-Einstellungen (T_0 und T_1). Hier ist es ausreichend, während der CTI-Phase das Interface 2 (10.0.7.1) von R6 in Richtung R7 zu blockieren. Aufgrund der Symmetrie dieser Topologie kommt ein Update mit der Metrik 64 von R1 über $R2 \rightarrow R3 \rightarrow R4 \rightarrow R5 \rightarrow R6 \rightarrow R7$ schneller bei R7 an als über die Route $R13 \rightarrow R12 \rightarrow R11 \rightarrow R10 \rightarrow R9 \rightarrow R8 \rightarrow R7$. Daher ist eine Verzögerung auf der längeren Route nicht nötig. Diese Verzögerung von R6 bewirkt, dass R7 zum False-Router wird und an R8 über sein Interface 2 (10.0.8.1) eine Falschnachricht mit der alten ungültigen Metrik sendet. R8 akzeptiert die Metrik und leitet sie weiter an R9 bis das Update schließlich R1 erreicht und von diesem weiter zurück zu R7 gelangt. Somit ist der Count-To-Infinity-Effekt provoziert und durchläuft den Zyklus mehrere Male.

Anhang **„Big-Net2“** ist für die längeren Timer-Einstellungen geeignet (T_2 und T_3). Hier wird die Verzögerung von Updatenachrichten nicht von einem Router alleine sondern von allen hervorgerufen. Der Weg wird in beiden Richtungen von Router zu Router um eine halbe Sekunde verlängert, wobei der Weg von R1 über R2 zu R7 schneller läuft als über R13 zu R7. Schließlich blockieren jeweils R6 auf Interface 2 (10.0.7.1) und R8 auf Interface 1 (10.0.8.2) die Verbindung zu R7 für die Dauer von 2 Updateperioden. R7 wird dadurch wieder zum False-Router und sendet seine ungültige Updatenachricht an R8.

Ergebnis des RIPMTI-Algorithmus

Nachdem R1 aufgrund des Timeout-Timers seine Verbindung zu Netz 10.0.0.0/24 auf die Metrik 64 gesetzt hat, sendet er eine Nachricht an seine Nachbarn R2 und R13. Die Informationskette über R2 bis R7 wird jedoch verzögert, sodass R7 sein reguläres Update mit der alten und falschen Metrik 8 an R8 schickt und kurz danach erst die neue und richtige Metrik 64 über R6 erhält. Dadurch wird der CTI ausgelöst und verbreitet sich über R8, R9, R10, R11, R12, R13 und R1 im Uhrzeigersinn über den kompletten Zyklus. Jeder Sprung von einem Router zum nächsten erhöht die Metrik um 1. Die Abbildungen 20 und 21 zeigen den Count-To-Infinity-Effekt auf R1 über das ausgefallene Netz. Mit jedem Umlauf der Falschnachricht durch den Zyklus steigt die Metrik um 13 Hops, was der Anzahl der Router des Zyklus entspricht. Bis das Netz RIP_INFINITY erreicht, wandert der CTI fünf mal über jeden Router und erzeugt dabei 68 Updatenachrichten bis es konvergiert. Im Schnitt produziert jeder einzelne Router fünf Falschnachrichten.

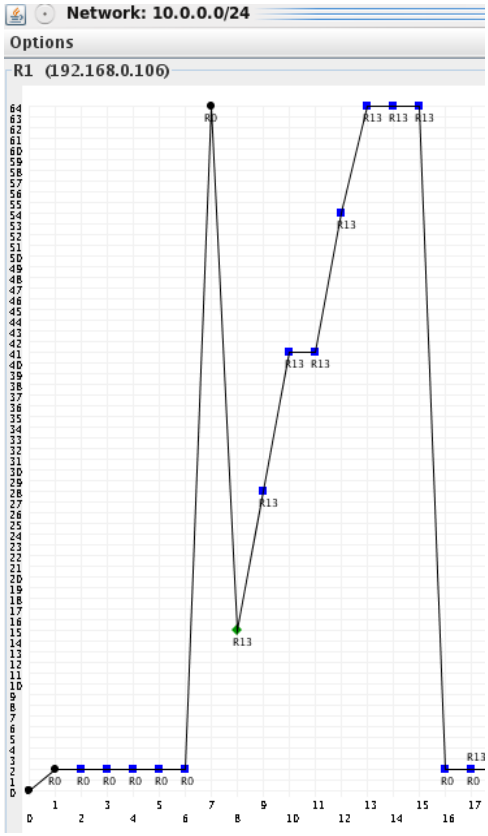


Abbildung 20: Netzgraph: CTI im Big-Loop-Szenario

bigloop_30-180-120_cti

graph

R9 R3 R8 R2 R13 R7 R1 R12 R6 R0 R11 R5 R10 R4

10.0.4.0/24 | 10.0.1.0/24 | 10.0.12.0/24 | 10.0.8.0/24 | 10.0.5.0/24 | 10.0.13.0/24 | 10.0.2.0/24 | 10.0.10.0/24 | 10.0.9.0/24 | 10.0.6.0/24 | 10.0.14.0/24 | 10.0.3.0/24 | 10.0.0.0/24 | 10.0.11

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration Time
update	00:00:06:144	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
update	00:01:08:113	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
update	00:01:51:297	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
update	00:02:42:041	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
update	00:03:13:916	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
update	00:03:44:877	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
timeout	00:06:44:874	R(n)	10.0.0.0/24	10.0.1.2	64	10.0.1.2	0	02:00	R1->R0	false	00:00:00:000
update	00:07:00:681	R(n)	10.0.0.0/24	10.0.14.2	15	10.0.14.2	0	03:00	R1->R13->R12->R11->R10->R9-...	false	00:00:00:000
update	00:07:05:692	R(n)	10.0.0.0/24	10.0.14.2	28	10.0.14.2	0	03:00	R1->R13->R12->R11->R10->R9-...	false	00:00:05:011
update	00:07:09:590	R(n)	10.0.0.0/24	10.0.14.2	41	10.0.14.2	0	03:00	R1->R13->R12->R11->R10->R9-...	false	00:00:08:909
update	00:07:11:578	R(n)	10.0.0.0/24	10.0.14.2	41	10.0.14.2	0	03:00	R1->R13->R12->R11->R10->R9-...	false	00:00:10:897
update	00:07:13:738	R(n)	10.0.0.0/24	10.0.14.2	54	10.0.14.2	0	03:00	R1->R13->R12->R11->R10->R9-...	false	00:00:13:057
update	00:07:18:747	R(n)	10.0.0.0/24	10.0.14.2	64	10.0.14.2	0	02:00	R1->R13->R12->R11->R10->R9-...	false	00:00:18:066
update	00:07:48:598	R(n)	10.0.0.0/24	10.0.14.2	64	10.0.14.2	0	01:30	R1->R13->R12->R11->R10->R9-...	false	00:00:00:000
update	00:08:24:612	R(n)	10.0.0.0/24	10.0.14.2	64	10.0.14.2	0	00:54	R1->R13->R12->R11->R10->R9-...	false	00:00:00:000
update	00:08:27:316	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000
update	00:08:27:369	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	03:00	R1->R0	false	00:00:00:000

Abbildung 21: Analysetabelle: CTI im Big-Loop-Szenario

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	13,3s
T_1	6s / 36s / 24s	14,1s
T_2	10s / 60s / 40s	15s
T_3	30s / 180s / 120s	16,4s

Tabelle 5: Ergebnisse versch. T-Einstellungen beim CTI im Big-Loop-Szenario

Der Count-To-Infinity-Effekt entstand in etwa 95% der Fälle und dauert in diesem Szenario trotz seiner Größe nur zwischen 13-16 Sekunden. Die Timer-Einstellungen spielen kaum eine Rolle. Obwohl T_3 zehn mal so lange Updatezeiten hat wie T_0, ist die Konvergenzzeit nur minimal länger. Fast schon vernachlässigbar gering. Ursache dafür ist die Größe des Zyklus und die daraus resultierende Tatsache, dass der Count-To-Infinity-Effekt den Zyklus nur fünf Mal durchlaufen muss. Beim ersten Durchlauf kann die Falschnachricht ohne Verzögerung einmal durch den gesamten Zyklus geschleust werden. Beim Absenden einer Nachricht wird zwar der Triggered Timer zurückgestellt und dadurch das Absenden einer weiteren Nachricht verzögert, aber nach dem ersten Durchlauf über 13 Router ist der Triggered Timer auf dem ersten Router fast wieder bereit für die nächste Updatenachricht. So wird ein Update zügig durch den Zyklus weitergereicht.

Ergebnis des RIPMTI-Algorithmus

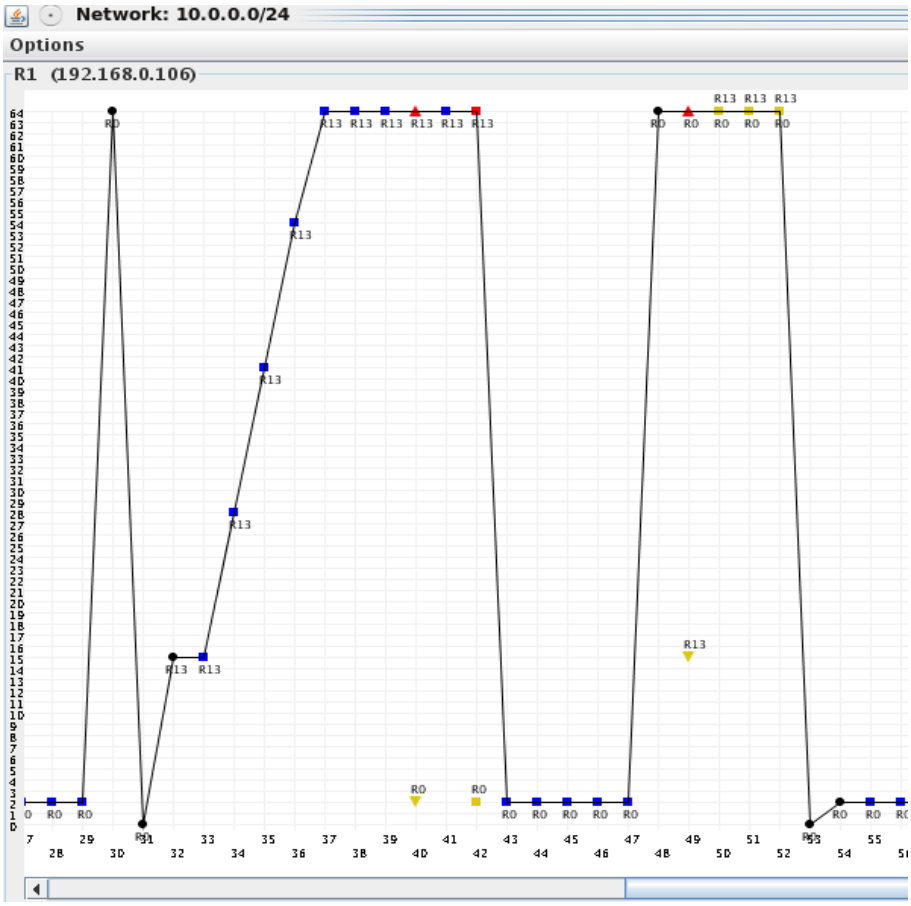


Abbildung 22: Netzgraph: RIPMTI und zu kurzer RT-Timer (links) und ausreichend langer RT-Timer (rechts) im Big-Loop-Szenario

Wenn R1 im RIPMTI-Modus läuft, unterbricht er die Schleife und verhindert damit den Count-To-Infinity-Effekt. Allerdings ist R1 einige Hops entfernt vom *False*-Router, sodass R8, R9, R10, R11, R12 und R13 einmal die Falsch-Nachricht in ihrer Routingtabelle übernehmen. In Abbildung 22 (rechts) wird gezeigt, dass ein Update von R13 abgelehnt wird, da der Simple-Loop-Test die Schleife erkannt hat.

$$m_{IF2}^{(R1,10.0.0.0/24)} = Route\ über\ R1 \rightarrow R2 \rightarrow R3 \rightarrow ..... \rightarrow R12 \rightarrow R13 \rightarrow R1 \rightarrow R0$$
$$m_{IF1}^{(R1,10.0.0.0/24)} = Route\ über\ R1 \rightarrow R0$$
$$m_{IF2}^{(R3,10.0.0.0/24)} + m_{IF1}^{(R3,10.0.0.0/24)} - 1 \geqslant msilm_{(IF1,IF2)}^{R3}$$
$$15 + 2 - 1 = 16 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

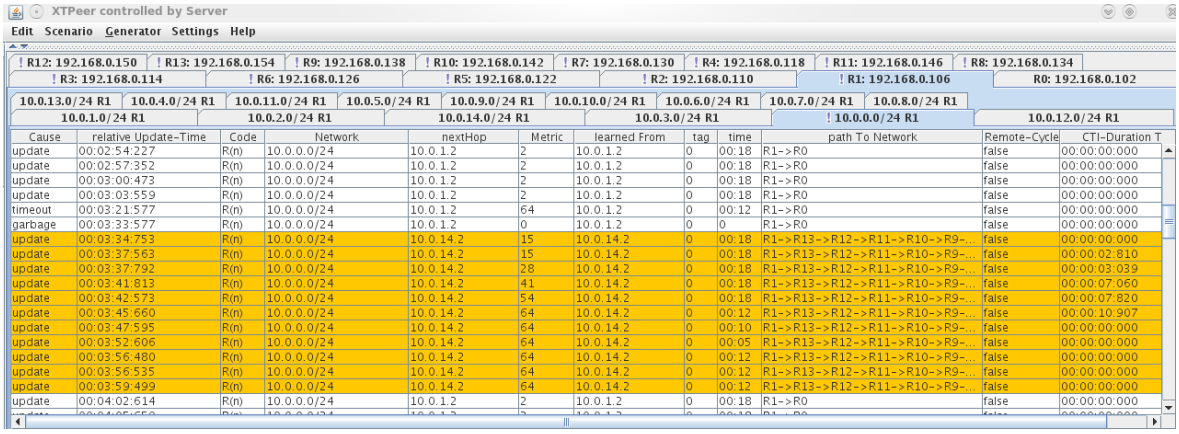


Abbildung 23: Analysetabelle: RIPMTI und zu kurzer RT-Timer im Big-Loop-Szenario

Abbildung 22 (links) und 23 beschreiben ein Phänomen, welches auftreten kann wenn kurze Timer-Einstellungen verwendet werden. Nachdem R1 die Route zu Netz 10.0.0.0/24 verloren hat, setzt er die Metrik auf 64 und verbreitet die Information im Netzwerk. Das Problem liegt beim jetzt anlaufenden Garbage-Collection-Timer. In den Testreihen wurde der Wert immer fest eingestellt. Im Fall der kurzen T_0-Einstellungen auf 12 Sekunden. Dieses Zeitfenster ist im Big-Loop-Szenario deutlich zu kurz. Im ungünstigen Fall, kann eine Updatenachricht einen Router erreichen, auf dem der Triggered Timer gerade zurückgesetzt wurde, sodass eine Nachricht auf jedem Router 5 Sekunden lang verzögert werden kann. Bei einem großen Schleifenumfang wie im Big-Loop-Szenario kann sich diese Verzögerung von Router zu Router linear vervielfachen. Die Abbildung beschreibt folgende Situation. Während R1 bereits den Garbage-Collection-Timer ablaufen lässt, sendet R7 seine Falschnachricht, die über R8 und schließlich R13 bei R1 ankommt. Wenn jetzt auf den Routern R8 bis R13 jeweils immer der Triggered-Timer zurückgesetzt wird, kommt die Falschnachricht erst nach etwa 30 Sekunden bei R1 an. Der Garbage-Collection-Timer von R1 ist aber bereits nach 12 Sekunden abgelaufen und damit die Route aus seiner Routingtabelle entfernt worden. Demzufolge wird eine später eintreffende Updatenachricht als neue gültige Route übernommen und daraus resultiert ein Count-To-Infinity-Effekt. Im Worst Case wird auf jedem Router in der Schleife der Triggered Timer reinitialisiert, sodass sich bei 13 Routern und 5 Sekunden Triggered Timer eine Umlaufzeit für eine Updatenachricht von 65 Sekunden ergeben kann. Der RIPMTI-Algorithmus muss an dieser Stelle verbessert werden. Die Idee ist, den Garbage-Collection-Timer dynamisch an den Schleifenumfang anzupassen, siehe Unter-Kapitel zu RIPMTI-Korrekturen.

Mit dieser Anpassung kann R1 mithilfe seiner Ermittlung des Schleifenumfangs den minimal benötigten Garbage-Collection-Timer neu berechnen und erhält den Wert von 65 Sekunden.

$$T_{GCneu} = (U_z = 13) * (T_t = 5) = 65\,Sekunden$$
$$T_{GCalt} = 12\,Sekunden$$

$$T_{GC} = \begin{cases} T_{GCneu} \Leftrightarrow T_{GCneu} > T_{GCalt} \\ T_{Galt} \Leftrightarrow T_{GCneu} < T_{GCalt} \end{cases}$$

Vergleicht er den Wert mit dem eingestellten alten Wert und ist dieser kleiner, so kann er ihn überschreiben. Diese Zeit reicht nun aus, damit R1 den Count-To-Infinity-Effekt erkennen und verhindern kann. Durch die Implementierung dieser neuen Funktion, besteht der RIPMTI-Algorithmus alle Testdurchläufe mit Erfolg.

Zusammenfassung

Im Big-Loop-Szenario wird deutlich wie stark sich die Abhängigkeit zwischen den verwendeten Timer-Einstellungen und der Größe des Zyklus ausprägt. Mit dem Hintergedanken, den Algorithmus beschleunigt arbeiten zu lassen, war eine Verbesserung nötig, die den Garbage-Collection-Timer an den maximalen Schleifenumfang des Szenarios anpasst. Eine Bezugsroute lange genug in der Routingtabelle zu behalten, damit sie als Vergleichsroute herhalten kann, um den Count-To-Infinity-Effekt zu erkennen und zu verhindern. Dank dieser Implementation funktioniert der RIPMTI-Algorithmus nun auch in einem großen Szenario einwandfrei. Es existieren weder Probleme mit der höheren Metrik noch mit den beschleunigten Timer-Einstellungen. Der Geschwindigkeitsgewinn zeigt sich etwas mager, da die Konvergenzzeiten selbst beim Entstehen eines Count-To-Infinity-Effekts sehr niedrig sind.

6.4 Y-Double

Hierbei handelt es sich grundsätzlich um ein Y-Szenario, das mit einer Doppelverbindung zweier Router ergänzt wurde.

Modellbeschreibung

Das Szenario besteht aus zwei Zyklen über den Routern R1, R2 und R3, wobei zwischen R2 und R3 eine weitere Verbindungsleitung eingerichtet wurde. Dadurch entstehen zum einen zwei redundante Zyklen. Beide Zyklen führen über die gleichen Router und besitzen den gleichen Umfang vom Wert 3. Die Doppelverbindung sorgt für einen weiteren Zyklus nur zwischen R2 und R3 mit dem Umfang 2.

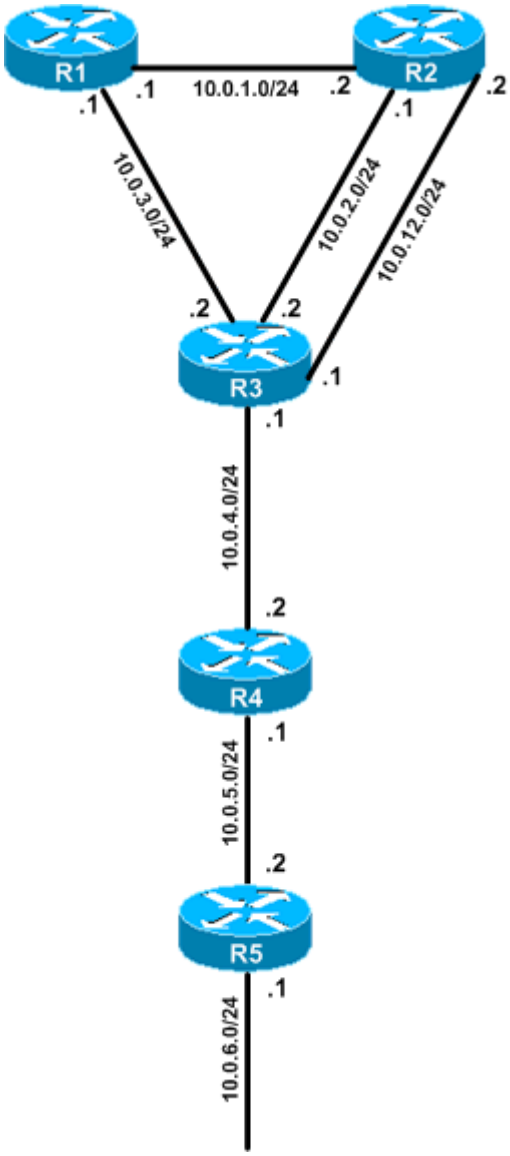


Abbildung 24: Topologie: Y-Double

Erzeugung eines Count-To-Infinity-Effekts

Im Anhang D befinden sich zwei Konfigurationsdateien mit verschiedenen Update-Reihenfolgen. **„Y-Double“** und **„Y-Double_R4“**. Beide haben die Testreihen durchlaufen. „Y-Double“ lässt das Netz 10.0.6.0/24 ausfallen und **„Y-Double_R4“** das Netz 10.0.5.0/24 und 10.0.6.0/24. Zum Verständnis der Ausfallsituation reicht es, wenn „Y-Double_R4“ erläutert wird, da „Y-Double“ keine zusätzlichen Erkenntnisse liefert. Der Count-To-Infinity-Effekt durch die Konfigurationsdatei entsteht für zwei Netze. Aus der Sicht von Router R3, der hier die Schlüsselrolle spielt, macht es keinen Unterschied ob ein oder mehrere Netze ausfallen. Da das Verhalten von R3 bei beiden Ausfallprozessen identisch ist, genügt es hier nur einen Prozess zu beschreiben. Die Netze 10.0.5.0/24 und 10.0.6.0/24 sollen ausfallen. Dazu wird die Verbindung von Interface 1 (10.0.4.2) auf R4 unterbrochen. Der Verlust des Netzes wird zuerst R3 bekannt und. R3 fungiert als Source-Router, da er als Teil des Zyklus dem verloren gegangenen Netz

am nächsten liegt. Bevor R3 die neue Metrik 64 im Zyklus propagiert, greift die Konfigurationsdatei ein und verzögert die Weiterleitung dieser Information. R3 blockiert sein Interface 1 (10.0.3.2) in Richtung R1. Updatenachrichten auf den übrigen Interfaces werden um jeweils eine Sekunde verzögert. Dadurch wird erreicht, dass R1 ausreichend Zeit erhält, um seine Falschnachricht zu versenden, ohne dass R2 ihn frühzeitig über die neue Metrik 64 informieren kann. R2 erhält anschließend die Falschnachricht von R1 und gibt diese Information auf der inneren Verbindungsleitung zuerst und kurz darauf auf der äußeren Verbindungsleitung an R3 weiter. Der Count-To-Infinity-Effekt wandert nun in Dreierschritten über R1, R2 und R3 und zusätzlich in Zweierschritten zwischen R2 und R3 hin und her. Da R3 die Schlüsselposition in diesem Szenario hat, hängt es von ihm ab wie der RIPMTI-Algorithmus damit umgeht.

Ergebnisse des RIP-Algorithmus

Der ursprüngliche RIP-Algorithmus verhält sich erwartungsgemäß langsam. In den meisten Fällen wird der Count-To-Infinity-Effekt auf R3 über den größeren Zyklus eingeleitet und unregelmäßig vom Count-To-Infinity-Effekt in der Doppelverbindung abgelöst. Abhängig von den Triggered Timern kommen Updatenachrichten unterschiedlich schnell an, sodass das eine Mal die äußere Schleife schneller bei R3 ankommt und ein anderes Mal Updatenachrichten aus der Doppelverbindung. Dadurch schwankt die Konvergenzzeit. Je länger R3 Nachrichten über die Doppelverbindung akzeptiert, desto länger dauert es bis die Metrik auf 64 hochgezählt hat. Ist der Anteil des 3er-Zyklus häufiger vertreten, konvergiert R3 deutlich schneller. Die Abbildungen 25 und 26 zeigen die unterschiedliche Auswirkung des größeren Zyklus im Gegensatz zur Doppelverbindung. Zu Beginn des Tests verläuft der Updateaustausch unter den Routern so wie es ein stabiles Netz erwarten lässt. Die Router konvergieren zügig und alle Route haben Kenntnis von z.b. Netz 10.0.6.0/24.

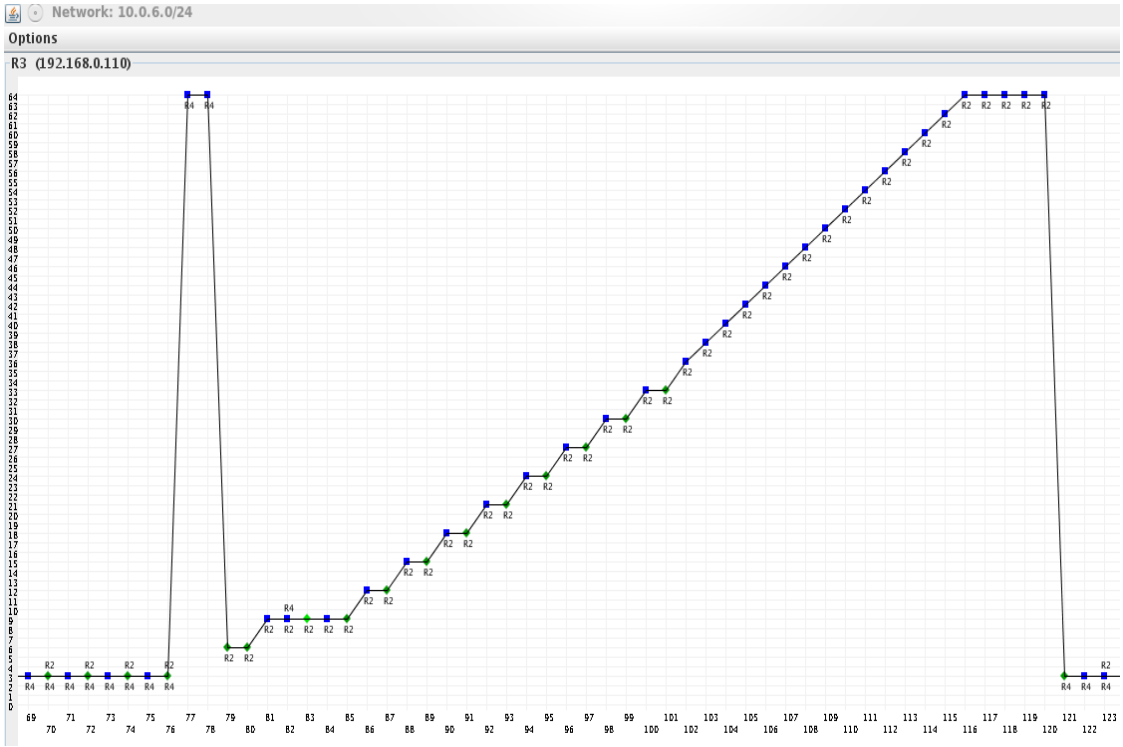


Abbildung 25: Netzgraph: CTI im Y-Double-Szenario

30-180-120_CTI

graph

R4 | R1 | R3 | R5 | R2

10.0.12.0/24 | 10.0.6.0/24 | 10.0.5.0/24 | 10.0.4.0/24 | 10.0.3.0/24 | 10.0.2.0/24 | 10.0.1.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration Time
update	00:11:28:315	R(n)	10.0.6.0/24	10.0.4.2	3	10.0.4.2	0	03:00	R3->R4->R5	false	00:00:00:000
update	00:11:58:437	R(n)	10.0.6.0/24	10.0.4.2	3	10.0.4.2	0	02:30	R3->R4->R5	false	00:00:00:000
update	00:11:58:508	R(n)	10.0.6.0/24	10.0.4.2	64	10.0.4.2	0	02:00	R3->R4->R5	false	00:00:00:000
update	00:12:28:599	R(n)	10.0.6.0/24	10.0.4.2	64	10.0.4.2	0	01:30	R3->R4->R5	false	00:00:00:000
update	00:12:30:236	R(n)	10.0.6.0/24	10.0.12.2	6	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:00:000
update	00:12:31:317	R(n)	10.0.6.0/24	10.0.12.2	6	10.0.12.2	0	02:59	R3->R2->R1->R3	false	00:00:01:081
update	00:12:31:321	R(n)	10.0.6.0/24	10.0.12.2	9	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:01:085
update	00:12:37:664	R(n)	10.0.6.0/24	10.0.12.2	9	10.0.12.2	0	02:54	R3->R2->R1->R3	false	00:00:07:428
update	00:12:44:925	R(n)	10.0.6.0/24	10.0.12.2	9	10.0.12.2	0	02:46	R3->R2->R1->R3	false	00:00:14:689
update	00:12:44:968	R(n)	10.0.6.0/24	10.0.12.2	9	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:14:732
update	00:13:01:530	R(n)	10.0.6.0/24	10.0.12.2	9	10.0.12.2	0	02:44	R3->R2->R1->R3	false	00:00:31:294
update	00:13:01:534	R(n)	10.0.6.0/24	10.0.12.2	12	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:31:298
update	00:13:05:754	R(n)	10.0.6.0/24	10.0.12.2	12	10.0.12.2	0	02:56	R3->R2->R1->R3	false	00:00:35:518
update	00:13:05:760	R(n)	10.0.6.0/24	10.0.12.2	15	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:35:524
update	00:13:10:729	R(n)	10.0.6.0/24	10.0.12.2	15	10.0.12.2	0	02:55	R3->R2->R1->R3	false	00:00:40:493
update	00:13:10:733	R(n)	10.0.6.0/24	10.0.12.2	18	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:40:497
update	00:13:12:966	R(n)	10.0.6.0/24	10.0.12.2	18	10.0.12.2	0	02:57	R3->R2->R1->R3	false	00:00:42:730
update	00:13:12:999	R(n)	10.0.6.0/24	10.0.12.2	21	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:42:763
update	00:13:15:801	R(n)	10.0.6.0/24	10.0.12.2	21	10.0.12.2	0	02:58	R3->R2->R1->R3	false	00:00:45:565
update	00:13:15:808	R(n)	10.0.6.0/24	10.0.12.2	24	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:45:572
update	00:13:20:823	R(n)	10.0.6.0/24	10.0.12.2	24	10.0.12.2	0	02:55	R3->R2->R1->R3	false	00:00:50:587
update	00:13:20:827	R(n)	10.0.6.0/24	10.0.12.2	27	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:50:591
update	00:13:23:825	R(n)	10.0.6.0/24	10.0.12.2	27	10.0.12.2	0	02:57	R3->R2->R1->R3	false	00:00:53:589
update	00:13:23:834	R(n)	10.0.6.0/24	10.0.12.2	30	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:53:598
update	00:13:27:821	R(n)	10.0.6.0/24	10.0.12.2	30	10.0.12.2	0	02:56	R3->R2->R1->R3	false	00:00:57:585
update	00:13:27:829	R(n)	10.0.6.0/24	10.0.12.2	33	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:00:57:593
update	00:13:30:846	R(n)	10.0.6.0/24	10.0.12.2	33	10.0.12.2	0	02:56	R3->R2->R1->R3	false	00:01:00:610
update	00:13:30:854	R(n)	10.0.6.0/24	10.0.12.2	36	10.0.12.2	0	03:00	R3->R2->R1->R3	false	00:01:00:618
update	00:13:33:876	R(n)	10.0.6.0/24	10.0.12.2	38	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:03:640
update	00:13:36:912	R(n)	10.0.6.0/24	10.0.12.2	40	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:06:676
update	00:13:39:928	R(n)	10.0.6.0/24	10.0.12.2	42	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:09:692
update	00:13:41:996	R(n)	10.0.6.0/24	10.0.12.2	44	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:11:760
update	00:13:42:034	R(n)	10.0.6.0/24	10.0.12.2	46	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:11:798
update	00:13:44:079	R(n)	10.0.6.0/24	10.0.12.2	48	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:13:843
update	00:13:46:094	R(n)	10.0.6.0/24	10.0.12.2	50	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:15:858
update	00:13:48:112	R(n)	10.0.6.0/24	10.0.12.2	52	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:17:876
update	00:13:50:120	R(n)	10.0.6.0/24	10.0.12.2	54	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:19:884
update	00:13:53:122	R(n)	10.0.6.0/24	10.0.12.2	56	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:22:886
update	00:13:55:134	R(n)	10.0.6.0/24	10.0.12.2	58	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:24:898
update	00:13:58:154	R(n)	10.0.6.0/24	10.0.12.2	60	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:27:918
update	00:14:00:142	R(n)	10.0.6.0/24	10.0.12.2	62	10.0.12.2	0	03:00	R3->R2->R3	false	00:01:29:906
update	00:14:03:166	R(n)	10.0.6.0/24	10.0.12.2	64	10.0.12.2	0	02:00	R3->R2->R3	false	00:01:32:930
update	00:14:06:187	R(n)	10.0.6.0/24	10.0.12.2	64	10.0.12.2	0	01:57	R3->R2->R3	false	00:00:00:000
update	00:14:07:021	R(n)	10.0.6.0/24	10.0.12.2	64	10.0.12.2	0	01:56	R3->R2->R3	false	00:00:00:000
update	00:14:37:041	R(n)	10.0.6.0/24	10.0.12.2	64	10.0.12.2	0	01:26	R3->R2->R3	false	00:00:00:000
update	00:15:11:718	R(n)	10.0.6.0/24	10.0.12.2	64	10.0.12.2	0	00:52	R3->R2->R3	false	00:00:00:000
update	00:15:11:720	R(n)	10.0.6.0/24	10.0.4.2	3	10.0.4.2	0	03:00	R3->R4->R5	false	00:00:00:000
update	00:15:41:812	R(n)	10.0.6.0/24	10.0.4.2	3	10.0.4.2	0	03:00	R3->R4->R5	false	00:00:00:000

Abbildung 26: Analysetabelle: CTI im Y-Double-Szenario

Die Routingtabelle verweist auf das Netz mit der Metrik 3. Nach Ablauf des Timeout-Timers setzt er die Metrik auf 64 zur Verdeutlichung der Unerreichbarkeit des verloren gegangenen Netzes. Da an dieser Stelle die

reibungslose Kommunikation der Router untereinander aufgrund der Konfigurationsdatei unterbrochen wird, kann der Count-To-Infinity-Effekt entstehen. Der False-Router R1 versendet seine vermeintlich gültige Metrik 4 an R2, R2 trägt bei sich Metrik 5 in die Routingtabelle ein und sendet weiter an R3, der dann schließlich als Metrik zum ausgefallenen Netz eine 6 einträgt. Ursprünglich besaß R3 Metrik 3 und durch den ersten Schleifendurchlauf mit dem Umfang 3 wurde der Wert um drei Hops inkrementiert. In der bereits vorgestellten Y-Topologie rechnet R3 linear in Dreierschritten hoch und erreicht nach 21 Updatenachrichten RIP_INFINITY. Laut Abbildung 25 werden 25 Updatenachrichten bis zur Konvergenz des Netzes benötigt. Ursache dafür ist ein zufälliges Ereignis. Grundsätzlich sollte eine Updatenachricht in der Doppelverbindung aufgrund des geringeren Schleifenumfangs schneller ringsum laufen als im 3er-Zyklus. Tatsächlich geschieht folgendes. R2 bekommt von R3 stets eine kleinere Metrik geschickt als von R1. Da ein Update von R1 meist etwas später bei R2 eintrifft, wird auf R2 die kleinere Metrik regelmäßig überschrieben, sodass R3 schließlich wieder ein Dreierschritt erreicht. Hier sind wieder die Triggered Timer ausschlaggebend. Die Verzögerung von Updatenachrichten wirkt sich im größeren Zyklus mit drei Routern und 15 Sekunden stärker aus als in der Doppelverbindung mit nur zwei Routern und 10 Sekunden. Dieser zeitliche Vorteil führt nach zehn Updatenachrichten dazu, dass R3 die kleineren Metriken aus der Doppelverbindung übernimmt und infolgedessen nur noch in Zweierschritten weiterzählt.

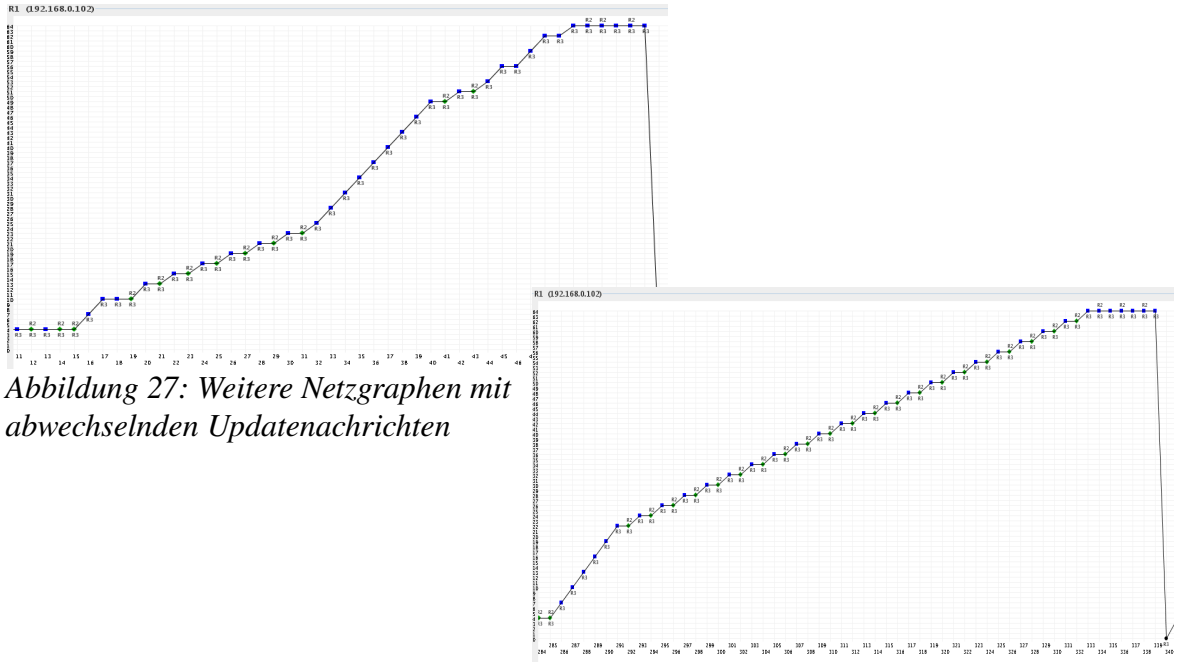


Abbildung 27: Weitere Netzgraphen mit abwechselnden Updatenachrichten

An der Abbildung 27 ist nochmals der bereits beschriebene Wechsel zwischen dem Zyklus und der Doppelverbindung anschaulich gemacht. Die unterschiedliche Steigung der Graphen beschreibt die wechselnden Zyklen

Die Tabelle enthält die durchschnittlichen Messergebnisse aus 100 Durchläufen des Testprogramms XT-Peer bei unterschiedlichen Timer-Einstellungen.

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	52,8s
T_1	6s / 36s / 24s	59,2s
T_2	10s / 60s / 40s	75,6s
T_3	30s / 180s / 120s	138,6s

Tabelle 6: Ergebnisse versch. T-Einstellungen beim CTI im Y-Double-Szenario

Laufen alle Router mit T_3, dann dauert der Count-To-Infinity-Effekt 138,6 Sekunden. In einem Netzwerk, in dem die Kommunikationsleitungen im Millisekunden-Bereich Daten übertragen, sind mehr als zwei Minuten der Inkonsistenz nicht mehr akzeptabel. Deutlich erkennbar ist bereits der Vorteil von T_2. Ein dreimal so schneller Nachrichtenaustausch bewirkt bereits eine beinahe Halbierung der Konvergenzzeit. Sobald die Timer-Einstellungen wie bei T_1 und T_0 in die Größenordnung des Triggered-Timers von 5 Sekunden gelangen nimmt der Konvergenzbonus ab. Bei T_1 ist zwar noch eine Verbesserung der Konvergenzzeit im Gegensatz zu T_2 erkennbar, aber bei T_0 ist sie vernachlässigbar gering. Zwischen der langsamsten und schnellsten Einstellung ist der Geschwindigkeitsbonus deutlich. T_0 konvergiert in nur knapp 40% der ursprünglichen Timer von T_3

Ergebnisse des RIPMTI-Algorithmus

In allen Testläufen hat der RIPMTI-Algorithmus zuverlässig funktioniert. R3 erkennt den Zyklus über die Doppelverbindung und verwirft die erste Falschnachricht von R2 und erkennt den Zyklus über $R3 \rightarrow R1 \rightarrow R2 \rightarrow R3$, von dem die zweite Falschnachricht R3 erreicht.

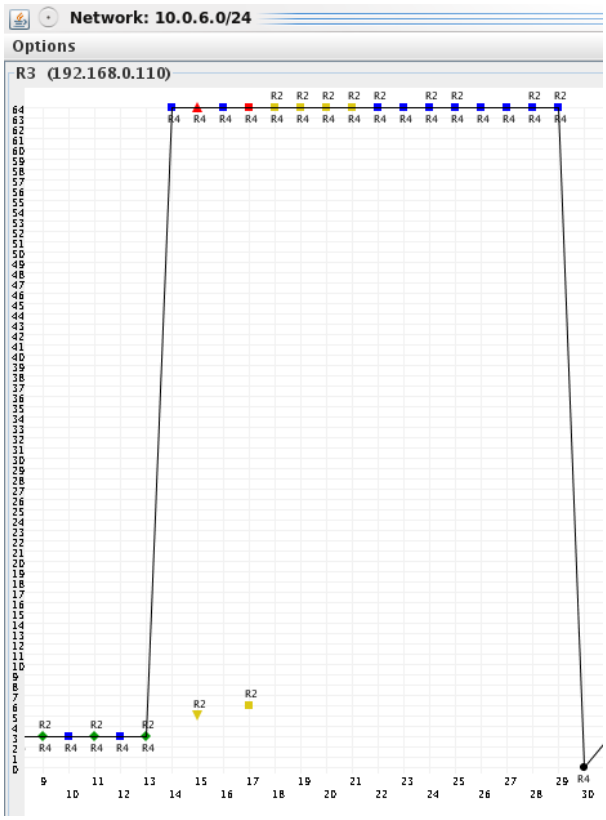


Abbildung 28: Netzgraph: RIPMTI im Y-Double-Szenario

Der Simple-Loop-Test ermittelt für eingehende alternative Routingpfade sowohl aus der Doppelverbindung als auch aus dem 3er-Zyklus folgende Ergebnisse.

- Erkannt wird der Simple-Loop in der Doppelverbindung anhand folgender Berechnung:

$$m_{IF2}^{(R3,10.0.6.0/24)} = Route\ über\ R3 \rightarrow R2 \rightarrow R3 \rightarrow R4 \rightarrow R5$$
$$m_{IF4}^{(R1,10.0.6.0/24)} = Route\ über\ R3 \rightarrow R4 \rightarrow R5$$
$$m_{IF2}^{(R3,10.0.6.0/24)} + m_{IF4}^{(R3,10.0.0.0/24)} - 1 \geq msilm_{(IF1,IF2)}^{R3}$$
$$5 + 3 - 1 = 7 \geq 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

- Erkannt wird der Simple-Loop im 3er-Zyklus anhand folgender Berechnung:

$$m_{IF1}^{(R1,10.0.6.0/24)} = Route\ über\ R3 \rightarrow R1 \rightarrow R2 \rightarrow R3 \rightarrow R4 \rightarrow R5$$
$$m_{IF4}^{(R1,10.0.6.0/24)} = Route\ über\ R3 \rightarrow R4 \rightarrow R5$$
$$m_{IF1}^{(R3,10.0.6.0/24)} + m_{IF4}^{(R3,10.0.0.0/24)} - 1 \geq msilm_{(IF1,IF2)}^{R3}$$
$$6 + 3 - 1 = 8 \geq 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Der Netzwerkgraph aus Abbildung 28 kann stellvertretend für alle durchgeführten Messreihen angesehen werden, da er sich nicht wesentlich vom Rest unterscheidet. Alle Router des Netzwerkes konvergieren sofort. Nur für die Dauer von wenigen Sekunden, nämlich in der Phase, in der R1 die Falschnachricht an R2 senden soll, befindet sich R2 im Rahmen von 1-2 Updateperioden in einem inkonsistenten Zustand. Der Garbage-Collection-Timer läuft auf R3 auch unter Verwendung der T_0-Einstellungen lange genug, um in diesem kleinen Szenario einen Count-To-Infinity-Effekt zu verhindern. RIPMTI vergleicht den eingestellten Garbage-Collection-Timer der Konfiguration mit dem selbst errechneten Wert aus der maximalen Schleife. Im Y-Double-Szenario ermittelt RIPMTI aus dem Umfang des 3er-Zyklus U_Z und dem Umfang der Doppelverbindung U_D $(U_Z = 3) > (U_D = 2)$ einen minimalen Garbage-Collection-Timer von 15 Sekunden, was nur im Falle von T_0 größer ist als der eingestellte Wert (12 Sekunde). Diese dynamische Anpassung unter Berücksichtigung des Schleifenumfangs wirkt sich nicht auf die Konvergenzzeit aus. Sie dient R3 dazu, eine Bezugsroute lange genug aufrecht zu erhalten, um damit eine Falschnachricht vergleichen zu können. Ist der Garbage-Collection-Timer zu kurz, dann kann eine verlorengegangene Route aus der Routingtabelle verschwinden während noch eine Falschnachricht in Umlauf ist. RIPMTI könnte nicht mehr eingreifen.

Zusammenfassung

Da die Beschleunigung der Timer-Einstellungen keinen erkennbaren Nachteil bringt, ist T_0 immerhin deutlich schneller als die ursprünglichen T_3-Einstellungen und führt ebenso zuverlässig zur Konvergenz des Netzwerkes.

Der RIPMI-Algorithmus funktioniert unabhängig von den gewählten Timer-Einstellungen zu 100%. Der Unterschied zwischen RIP und RIPMTI ist erkennbar. RIPMTI schneidet im Test besser ab und erfüllt die Bedingung, den Algorithmus auf eine größere Metrik (64) und kürzere Kommunikationsintervalle einzustellen. Im Endeffekt bewirkt RIPMTI einen großen Geschwindigkeitsgewinn. Nimmt man wieder zum Vergleich die komplette Dauer von Verlust einer Route bis zum entfernen aus der Routingtabelle, dann benötigt RIPMTI nur noch 38% der ursprünglichen Zeit, die entstanden wäre, wenn ein Count-To-Infinity-Effekt aufgetreten wäre.

6.5 Double-Con

In diesem Szenario wurde das ursprüngliche Y-Szenario um einen Router des Zyklus reduziert und eine Doppelverbindung zwischen zwei Routern hergestellt.

Modellbeschreibung

Das Versuchsmodell besteht aus nur 4 Routern. R1 und R2 sind über eine Doppelverbindung miteinander verknüpft. Es soll gezeigt werden, dass der RIPMTI-Algorithmus auch funktioniert, wenn der Zyklus minimal ist und ein ausgefallenes Netz weit vom Zyklus entfernt ist. Dazu wurde an R2 eine Reihe von Routern mit R3 und R4 angefügt.

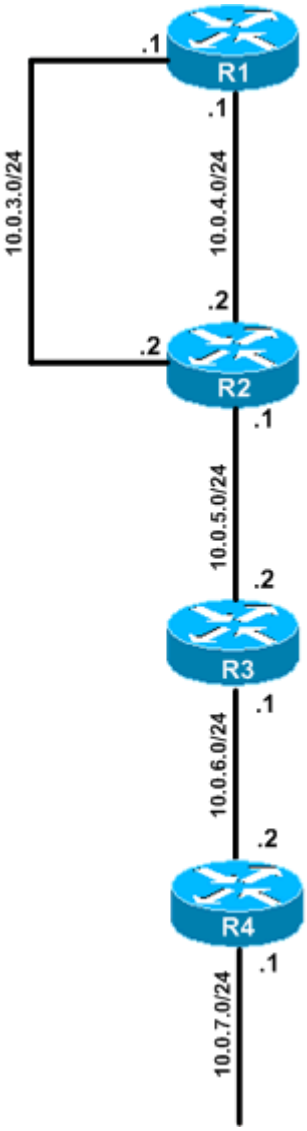


Abbildung 29: Topologie: Double-Con

Erzeugung des Count-To-Infinity-Effekts

Es ist seltener, dass in einem solchen Szenario ein Count-To-Infinity-Effekt entsteht. Deutlich wurde das durch den hohen Anspruch an die Konfigurationsdatei für jeden Testdurchlauf. Unter Berücksichtigung der Tatsache, dass sowohl die Router R3 und R4 ein Ausfallszenario darstellen können als auch vier abweichende Timer-Einstellungen möglich sind, ergaben sich mehrere Konfigurationsdateien, um einen Count-To-Infinity-Effekt zuverlässig zu erzeugen. Im Anhang E befinden sich die Konfigurationen „DoubleCon1“ bis „DoubleCon3“. In der Konfiguration selbst ist, in Klammern, die Timer-Einstellung, für die diese geeignet ist, notiert. Die aufgelisteten Konfigurationsdateien beschreiben den Ausfall von R4 und damit den Verlust des Netzes 10.0.7.0/24. Um die gleichen Tests für den Ausfall von R3 zu simulieren, muss das x-Symbol anstatt bei R4 in R3 eingetragen werden. Der Count-To-Infinity-Effekt wird folgendermaßen ausgelöst. Nachdem sich alle Router in der Konvergenzphase über ihre Updatenachrichten kennen gelernt haben, blockiert R4 auf Interface 1 (10.0.6.2) die Verbindung zu R3. Auf diesem signalisiert der ablaufende Timeout-Timer den Verlust des Netzes 10.0.7.0/24 und schickt an R2 ein Update mit der neuen Erreichbarkeitsinformation 64. R2 blockiert jetzt die

Verbindung auf Interface 2 (10.0.4.2) zu R1, sendet aber auf Interface 1 (10.0.3.2). Aufgrund der Doppelverbindung hat R1 jetzt nur ein kurzes Zeitfenster, um seine Falschnachricht an R2 zu senden, das dieser Router im Anschluss als neue gültige Route akzeptiert und damit den Count-To-Infinity-Effekt auslöst. R1 sendet die ungültige Routeninformation über Interface 1 (10.0.4.1) an R2

Ergebnis des RIP-Algorithmus

R3 erfährt als erster Router von der Unerreichbarkeit des verloren gegangenen Netzes. R4 wird im Anschluss von R3 darüber informiert und editiert seine Routinginformation ebenfalls. R1 erfährt niemals von der neuen Metrik 64. Für ihn gilt immer noch der letzte Wert 4. Durch die Falschnachricht von R1 bekommt R2 jetzt die Metrik 5 zugeschickt und überschreibt damit die eigentlich korrekte 64. Da der Zyklus aus zwei Routern besteht, zählt das System nun in Zweierschritten bis zur 64 hoch und benötigt dazu bis zu 31 Updatenachrichten auf R1 bis R3. Die Abbildungen 30 und 31 zeigen den entsprechenden Graphen und die dazugehörige Metriktabelle.

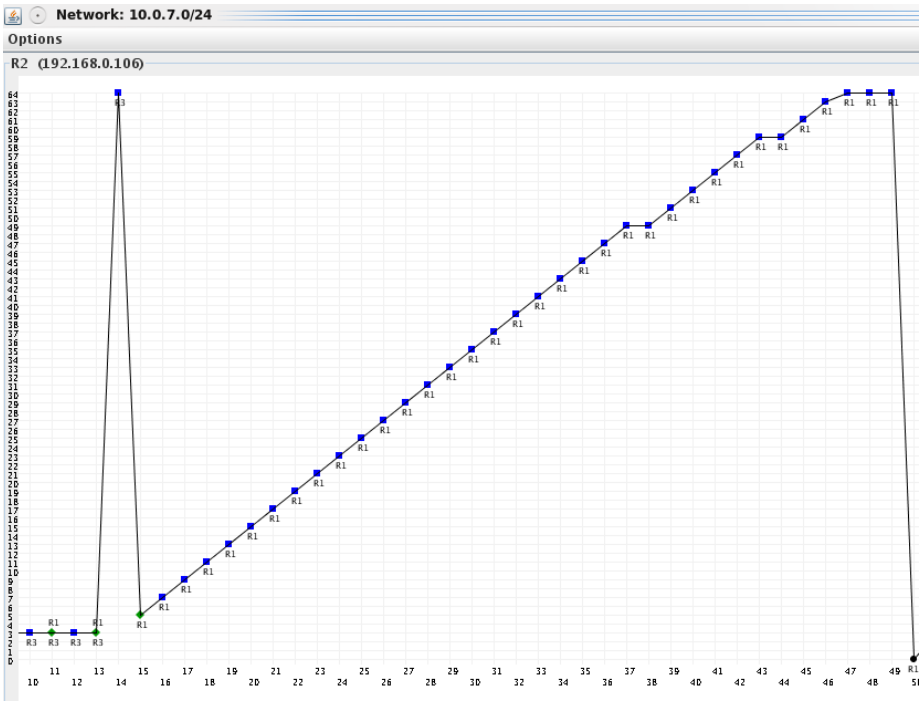


Abbildung 30: Netzgraph: CTI im Double-Con-Szenario

3-18-12_doublecon_CTI_R4.sdf

graph

R3 | R1 | R4 | R2

10.0.6.0/24 | 10.0.5.0/24 | 10.0.4.0/24 | 10.0.3.0/24 | 10.0.7.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:00:33:395	R(n)	10.0.7.0/24	10.0.5.2	3	10.0.5.2	0	00:18	R2->R3->R4	false	00:00:00:000
update	00:00:36:299	R(n)	10.0.7.0/24	10.0.5.2	3	10.0.5.2	0	00:18	R2->R3->R4	false	00:00:00:000
update	00:00:36:410	R(n)	10.0.7.0/24	10.0.5.2	3	10.0.5.2	0	00:18	R2->R3->R4	false	00:00:00:000
update	00:00:39:776	R(n)	10.0.7.0/24	10.0.5.2	64	10.0.5.2	0	00:12	R2->R3->R4	false	00:00:00:000
update	00:00:39:874	R(n)	10.0.7.0/24	10.0.4.1	5	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:00:000
update	00:00:43:147	R(n)	10.0.7.0/24	10.0.4.1	7	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:03:273
update	00:00:45:157	R(n)	10.0.7.0/24	10.0.4.1	9	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:05:283
update	00:00:48:240	R(n)	10.0.7.0/24	10.0.4.1	11	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:08:366
update	00:00:48:268	R(n)	10.0.7.0/24	10.0.4.1	13	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:08:394
update	00:00:50:274	R(n)	10.0.7.0/24	10.0.4.1	15	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:10:400
update	00:00:51:298	R(n)	10.0.7.0/24	10.0.4.1	17	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:11:424
update	00:00:52:840	R(n)	10.0.7.0/24	10.0.4.1	19	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:12:966
update	00:00:53:260	R(n)	10.0.7.0/24	10.0.4.1	21	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:13:386
update	00:00:56:866	R(n)	10.0.7.0/24	10.0.4.1	23	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:16:992
update	00:00:58:276	R(n)	10.0.7.0/24	10.0.4.1	25	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:18:402
update	00:00:58:922	R(n)	10.0.7.0/24	10.0.4.1	27	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:19:048
update	00:01:01:941	R(n)	10.0.7.0/24	10.0.4.1	29	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:22:067
update	00:01:03:296	R(n)	10.0.7.0/24	10.0.4.1	31	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:23:422
update	00:01:03:330	R(n)	10.0.7.0/24	10.0.4.1	33	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:23:456
update	00:01:05:932	R(n)	10.0.7.0/24	10.0.4.1	35	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:26:058
update	00:01:07:320	R(n)	10.0.7.0/24	10.0.4.1	37	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:27:446
update	00:01:07:970	R(n)	10.0.7.0/24	10.0.4.1	39	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:28:096
update	00:01:11:329	R(n)	10.0.7.0/24	10.0.4.1	41	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:31:455
update	00:01:11:338	R(n)	10.0.7.0/24	10.0.4.1	43	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:31:464
update	00:01:14:378	R(n)	10.0.7.0/24	10.0.4.1	45	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:34:504
update	00:01:16:343	R(n)	10.0.7.0/24	10.0.4.1	47	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:36:469
update	00:01:16:379	R(n)	10.0.7.0/24	10.0.4.1	49	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:36:505
update	00:01:20:390	R(n)	10.0.7.0/24	10.0.4.1	49	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:40:516
update	00:01:20:407	R(n)	10.0.7.0/24	10.0.4.1	51	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:40:533
update	00:01:24:402	R(n)	10.0.7.0/24	10.0.4.1	53	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:44:528
update	00:01:24:436	R(n)	10.0.7.0/24	10.0.4.1	55	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:44:562
update	00:01:28:435	R(n)	10.0.7.0/24	10.0.4.1	57	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:48:561
update	00:01:29:445	R(n)	10.0.7.0/24	10.0.4.1	59	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:49:571
update	00:01:30:430	R(n)	10.0.7.0/24	10.0.4.1	59	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:50:556
update	00:01:32:017	R(n)	10.0.7.0/24	10.0.4.1	61	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:52:143
update	00:01:35:452	R(n)	10.0.7.0/24	10.0.4.1	63	10.0.4.1	0	00:18	R2->R1->R2	false	00:00:55:578
update	00:01:35:486	R(n)	10.0.7.0/24	10.0.4.1	64	10.0.4.1	0	00:12	R2->R1->R2	false	00:00:55:612
update	00:01:39:468	R(n)	10.0.7.0/24	10.0.4.1	64	10.0.4.1	0	00:08	R2->R1->R2	false	00:00:00:000
update	00:01:44:487	R(n)	10.0.7.0/24	10.0.4.1	64	10.0.4.1	0	00:03	R2->R1->R2	false	00:00:00:000
garbage	00:01:47:492	R(n)	10.0.7.0/24	10.0.4.1	0	10.0.4.1	0	0	R2->R1->R2	false	00:00:00:000
update	00:01:58:574	R(n)	10.0.7.0/24	10.0.5.2	3	10.0.5.2	0	00:18	R2->R3->R4	false	00:00:11:082
update	00:02:02:023	R(n)	10.0.7.0/24	10.0.5.2	3	10.0.5.2	0	00:18	R2->R3->R4	false	00:00:00:000

Abbildung 31: Analysetabelle: CTI im Double-Con-Szenario

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	46,5s
T_1	6s / 36s / 24s	66,3s
T_2	10s / 60s / 40s	77,9s
T_3	30s / 180s / 120s	136,2s

Tabelle 7: Ergebnisse versch. T-Einstellungen beim CTI im Double-Con-Szenario

Trotz der relativ einfachen Struktur des Netzwerkes dauert es lange bis das Netzwerk einen konvergenten Zustand erreicht. Verantwortlich dafür ist wieder der Triggered Timer. Die Falschnachricht muss 32 Mal durch den Zyklus wandern bis RIP_INFINITY erreicht ist. In der Doppelverbindung verhindert dieser Timer regelmäßig ein sofortiges Weiterleiten der Nachricht, da es sich immer um Routinginformationen zur gleichen Route handelt. Daher weichen die gemessenen Werte aus Tabelle 7 auch tendenziell nach oben ab von den Konvergenzzeiten des

Y-Szenarios. Es ist jedoch erkennbar, dass auch hier T_0 die vorteilhafteste von allen Timer-Einstellungen ist. T_0 konvergiert etwa dreimal so schnell wie T_3, wobei anzumerken ist, dass die große Menge an Schleifendurchläufen das Datenaufkommen auf den Verbindungsleitungen stark erhöht. Im Test entstand der Count-To-Infinity-Effekt in 90 von 100 Durchläufen. Die Eigenschaften des Count-To-Infinity-Effekts ändern sich nicht wenn anstelle von R4 R3 ausfällt. Es entsteht dabei nur ein eigener Count-To-Infinity-Effekt für das Netz 10.0.6.0/24.

Ergebnis des RIPMTI-Algorithmus

Der RIPMTI-Algorithmus auf R2 erkennt zuverlässig alle Falschnachrichten und lehnt diese ab. Er macht dabei keinen Unterschied ob bei R3 oder R4 eine Ausfallsituation eintritt. Abbildung 32 zeigt die Analyseergebnisse von R2 unter

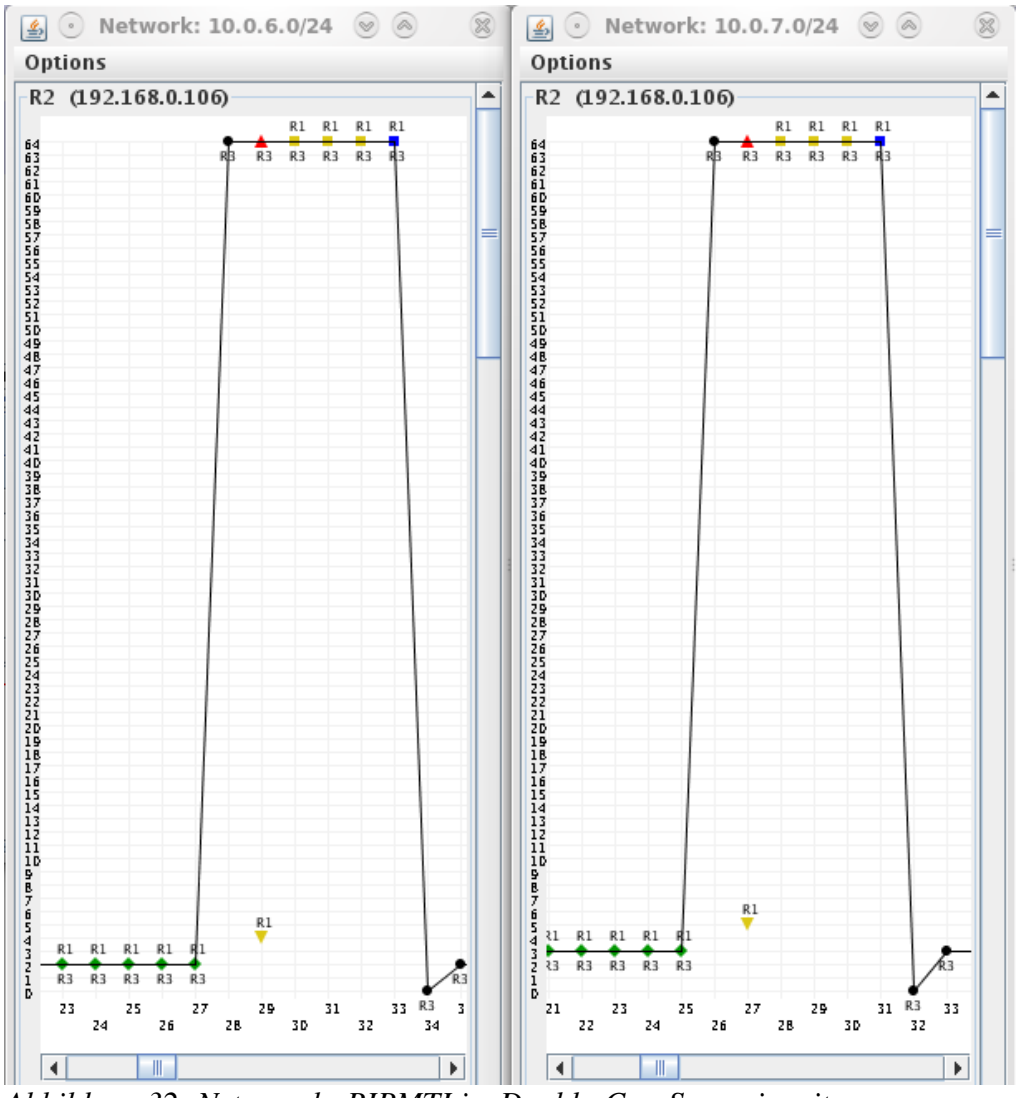


Abbildung 32: Netzgraph: RIPMTI im Double-Con-Szenario mit KonfigA(links) und KonfigB(rechts)

Verwendung der Konfiguration **„DoubleCon2“**, in der R3 ausfällt und dadurch das Netz 10.0.6.0/24 (links) und 10.0.7.0/24 (rechts) unerreichbar werden.
Der Garbage-Collection-Timer, der sich auch hier an den Schleifenumfang anpassen will, ermittelt zunächst bei zwei Routern in der Doppelverbindung einen Wert, der niedriger ist als die voreingestellte Zeit und wird damit nicht geändert. Speziell bei To-Einstellungen ist diese Berechnung entscheidend.

$$T_t = 5\text{s}$$
$$msilm^{R2}_{(IF1,\,IF2)} = 2$$
$$T_{GCalt} = 12\text{s}$$
$$T_{GCneu} = T_t * msilm^{R2}_{(IF1,\,IF2)} = 10\text{s}$$
$$T_{GCalt} > T_{GCneu} \Rightarrow T_{GC} = T_{GCalt}$$

Der Algorithmus stellt fest, dass der selbst berechnete Garbage-Collection-Timer kleiner wäre als der voreingestellte Wert. Das Überschreiben wird über einen Vergleich beider Werte verhindert, sodass der größere Zeitraum verwendet wird. So wird verhindert, dass auf R2 eine ausgefallene Routen bei kurzen Timer-Einstellungen zu voreilig gelöscht wird und danach eine Falschnachricht als gültig akzeptiert wird.
Der Y-Test erkennt die Count-To-Infinity-Effekte zu beiden verloren gegangenen Netzen.

Berechnung für R4 und Netz 10.0.7.0/24

$$m^{(R2,10.0.7.0/24)}_{IF1} = Route\ über\ R2 \rightarrow R1 \rightarrow R2 \rightarrow R3 \rightarrow R4$$
$$m^{(R2,10.0.7.0/24)}_{IF3} = Route\ über\ R2 \rightarrow R3 \rightarrow R4$$
$$m^{(R2,10.0.7.0/24)}_{IF1} + m^{(R2,10.0.7.0/24)}_{IF3} - 1 \geqslant msilm^{R2}_{(IF1,\,IF3)}$$
$$5 + 3 - 1 = 7 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Berechnung für R3 und Netz 10.0.6.0/24

$$m^{(R2,10.0.6.0/24)}_{IF1} = Route\ über\ R2 \rightarrow R1 \rightarrow R2 \rightarrow R3$$
$$m^{(R2,10.0.6.0/24)}_{IF3} = Route\ über\ R2 \rightarrow R3$$
$$m^{(R2,10.0.6.0/24)}_{IF1} + m^{(R2,10.0.6.0/24)}_{IF3} - 1 \geqslant msilm^{R2}_{(IF1,\,IF3)}$$
$$4 + 2 - 1 = 5 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Es werden die Falschnachrichten zu beiden Netzen, die R2 über R1 jeweils mit Metrik 4 bzw. 5 erreichen, abgelehnt und der Count-To-Infinity-Effekt damit verhindert.

Zusammenfassung

RIPMTI funktioniert zu 100%. Das Double-Con-Szenario zeigt, dass RIPMTI und speziell die neue Implementierung zur Berechnung des Garbage-Collection-Timers sowohl bei maximal großen Zyklen (siehe Big-Loop) als auch minimal kleinen Doppelverbindungen zuverlässig funktioniert. RIPMTI liefert eine um knapp 32% schnellere Konvergenzzeit unter T_3-Einstellungen und mit T_0-Einstellungen bis zu 70% schnellere Konvergenz, da bei RIPMTI der Timeout-Timer und anschließend direkt der Garbage-Collection-Timer ablaufen kann.

6.6 Y-Mod

Das Y-Mod-Szenario ist eine andere Erweiterung des Y-Szenarios. Sie enthält eine große Netzwerkschleife, innerhalb der drei Router über je eine zusätzliche Verbindungsleitung zu einem Nachbarrouterverfügen.

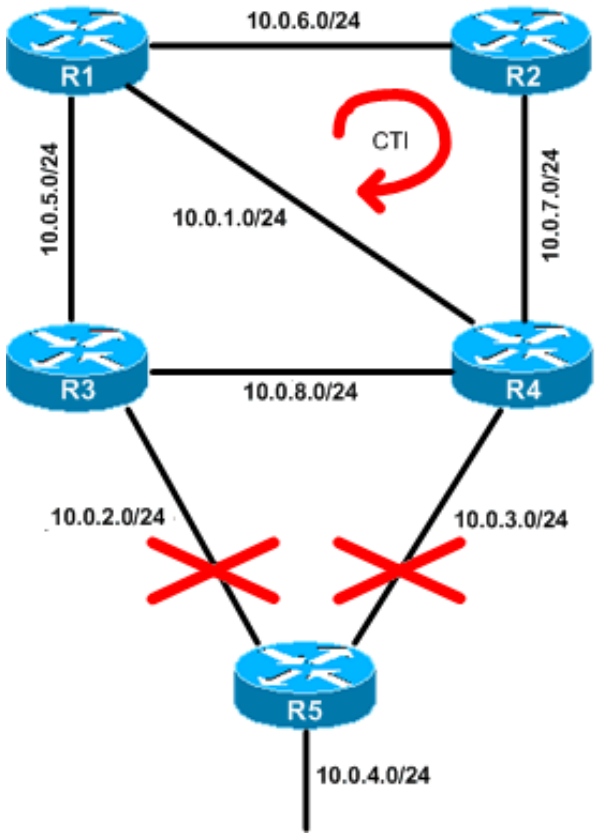


Abbildung 33: Topologie: Y-Mod

Modellbeschreibung

Das Netzwerk besteht aus fünf Routern, die in einer großen Netzwerkschleife miteinander verbunden sind. R1 und R4 bzw. R3 und R4 besitzen eine weitere Verbindung untereinander, sodass sich eine verschachtelte Topologie ergibt, in der mehrere Zyklen entstehen. $R5 \rightarrow R3 \rightarrow R4 \rightarrow R5$, $R5 \rightarrow R3 \rightarrow R1 \rightarrow R4 \rightarrow R5$, $R5 \rightarrow R3 \rightarrow R1 \rightarrow R2 \rightarrow R4 \rightarrow R5$, $R1 \rightarrow R2 \rightarrow R4 \rightarrow R1$, $R1 \rightarrow R2 \rightarrow R4 \rightarrow R3 \rightarrow R1$ und $R1 \rightarrow R4 \rightarrow R3 \rightarrow R1$. Jeder dieser Zyklen stellt ein Risiko zur Entstehung von Schleifeninformationen dar.

Erzeugung des Count-To-Infinity-Effekts

Die Konfigurationsdatei aus Anhang F **„Y-Mod1“** blockiert Interface 1 (10.0.2.1) und Interface 2 (10.0.3.1) auf R5. Den Ausfall von R5 bzw. die Unerreichbarkeit

des Netzes 10.0.4.0/24 registrieren R3 und R4 annähernd gleichzeitig. Diese wollen anschließend ein Routingupdate an die übrigen Router weitergeben. R1 lernt von R3 über Interface 1 (10.0.5.1) oder R4 über Interface 3 (10.0.1.1) RIP_Infinity. R1 trägt also in seine Routingtabelle den Wert 64 ein. Die Verbindung zwischen R4 und R2 wird blockiert, damit R2 die neue gültige Metrik erhält. R2 wird daraufhin zum False-Router und sendet an R1 seine ungültige Falschnachricht, die R1 übernimmt, weil sie kleiner ist als RIP_Infinity. R1 informiert seinerseits R3 und R4 und verbreitet damit die Falschnachricht in mehreren Netzwerkschleifen. R3 lernt von R1 und sendet die Schleifeninformation an R4 weiter, R4 lernt von R1 und sendet die Schleifeninformation an R2 weiter.

Ergebnisse des RIP-Algorithmus

Nach dem Verlust des Netzes 10.0.4.0/24 setzen R3 und R4 mehr oder weniger zeitgleich die Erreichbarkeit zu diesem Netz auf RIP_Infinity. Im nächsten Schritt sollte R1 ebenfalls davon erfahren. Die Verbindung von R4 zu R2 ist zu diesem Zeitpunkt blockiert. In einigen Fällen geschieht bereits an dieser Stelle ein Count-To-Infinity-Effekt. Wenn R3 RIP_Infinity an R1 weiter gibt bevor R4 RIP_Infinity festgestellt hat und R4 seine periodische Updatenachricht mit der alten, ungültigen Metrik verschickt, kann es vorkommen, dass R1 eine Falschnachricht von R4 akzeptiert. Demzufolge erhält R3 wiederum eine Routinginformation mit der Metrik 4 über die Route $R3 \rightarrow R1 \rightarrow R4 \rightarrow R5$. Genauso wie R1 die Falschnachricht erhalten hat, kann auch R3 direkt von R4 lernen und darüber einen Source-Loop verursachen. Wenn R4 eine ungültige Nachricht von R1 erhält bevor R1 von R3 RIP_Infinity lernen konnte, dann akzeptiert R4 diese neue Route. Die Konfigurationsdatei verursacht zusätzlich das Versenden einer Updatenachricht mit alten, ungültigen Routinginformationen bei R2. All diese Falschnachrichten können sich gegenseitig beeinflussen. Während R3 über R1 seine Metrik von 64 auf 4 herunterreduziert hat und ein weitere Updatenachricht im Dreierschritt die Metrik hochzählt, gelangt die Falschnachricht von R2 zu R1. R2 hat zu diesem Zeitpunkt eine Erreichbarkeitsmetrik von 3 in seiner Routingtabelle, die niedriger ist als die aktuell eingetragene. Somit wird die ungültige Routinginformation auf R1 weiter zurückgesetzt und das verzögert den Count-To-Infinity-Effekt in diesem Netzwerk. Jeder Router zählt abhängig von der Route, die ihm angeboten wurde, unterschiedlich schnell bis RIP_Infinity, sodass sich schlechtere und bessere Metriken abwechseln können. Abbildung 34 und 35 zeigen wie sich dieser Effekt zu Beginn der Routingschleifen auswirkt.

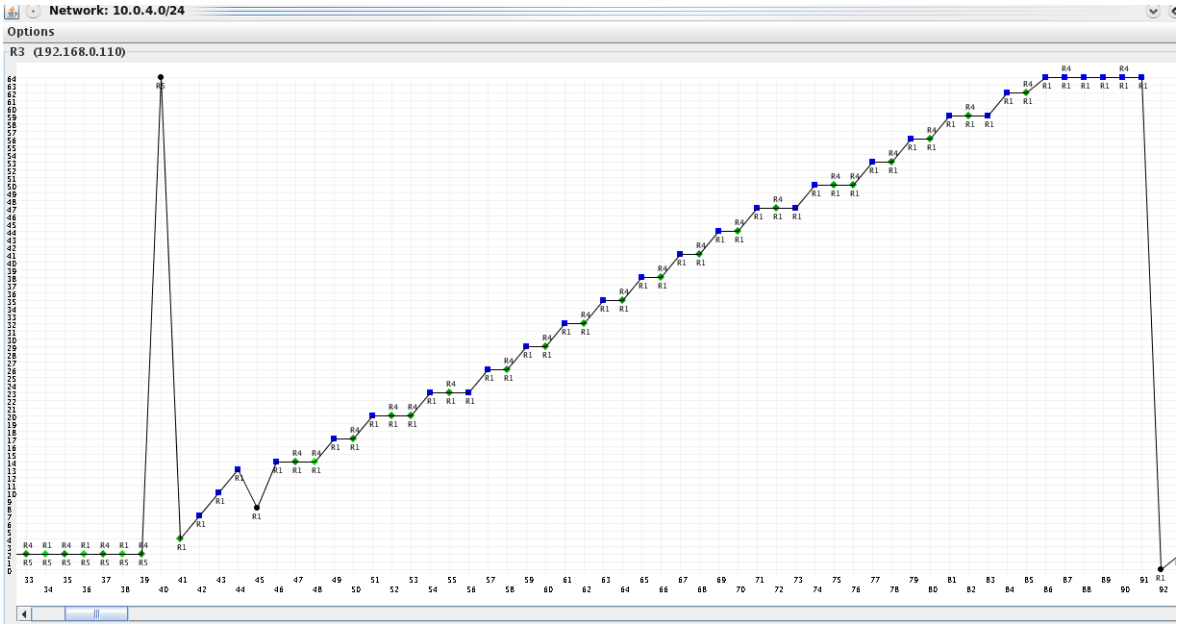


Abbildung 34: Netzgraph: CTI im Y-Mod-Szenario

3-18-12_CTI

graph

R4 | R1 | R3 | R5 | R2

10.0.6.0/24 | 10.0.5.0/24 | 10.0.4.0/24 | 10.0.3.0/24 | 10.0.2.0/24 | 10.0.1.0/24 | 10.0.8.0/24 | 10.0.7.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration Time
update	00:03:15:969	R(n)	10.0.4.0/24	10.0.2.2	2	10.0.2.2	0	00:07	R3->R5	false	00:00:00:000
update	00:03:16:096	R(n)	10.0.4.0/24	10.0.2.2	2	10.0.2.2	0	00:07	R3->R5	false	00:00:00:000
update	00:03:19:844	R(n)	10.0.4.0/24	10.0.2.2	2	10.0.2.2	0	00:03	R3->R5	false	00:00:00:000
update	00:03:20:013	R(n)	10.0.4.0/24	10.0.2.2	2	10.0.2.2	0	00:03	R3->R5	false	00:00:00:000
timeout	00:03:22:987	R(n)	10.0.4.0/24	10.0.2.2	64	10.0.2.2	0	00:12	R3->R5	false	00:00:00:000
update	00:03:23:555	R(n)	10.0.4.0/24	10.0.5.1	4	10.0.5.1	0	00:18	R3->R1->R4->R5	false	00:00:00:000
update	00:03:26:576	R(n)	10.0.4.0/24	10.0.5.1	7	10.0.5.1	0	00:18	R3->R1->R4->R3	false	00:00:00:000
update	00:03:29:959	R(n)	10.0.4.0/24	10.0.5.1	10	10.0.5.1	0	00:18	R3->R1->R4->R3	false	00:00:03:383
update	00:03:30:199	R(n)	10.0.4.0/24	10.0.5.1	13	10.0.5.1	0	00:18	R3->R1->R4->R3	false	00:00:03:623
update	00:03:31:425	R(n)	10.0.4.0/24	10.0.5.1	8	10.0.5.1	0	00:18	R3->R1->R2->R4->R3	false	00:00:04:849
update	00:03:31:458	R(n)	10.0.4.0/24	10.0.5.1	14	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:04:882
update	00:03:32:402	R(n)	10.0.4.0/24	10.0.5.1	14	10.0.5.1	0	00:17	R3->R1->R2->R4->R1	true	00:00:05:826
update	00:03:36:353	R(n)	10.0.4.0/24	10.0.5.1	14	10.0.5.1	0	00:13	R3->R1->R2->R4->R1	true	00:00:09:777
update	00:03:36:450	R(n)	10.0.4.0/24	10.0.5.1	17	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:09:874
update	00:03:36:475	R(n)	10.0.4.0/24	10.0.5.1	17	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:09:899
update	00:03:36:484	R(n)	10.0.4.0/24	10.0.5.1	20	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:09:908
update	00:03:39:486	R(n)	10.0.4.0/24	10.0.5.1	20	10.0.5.1	0	00:15	R3->R1->R2->R4->R1	true	00:00:12:910
update	00:03:41:369	R(n)	10.0.4.0/24	10.0.5.1	20	10.0.5.1	0	00:13	R3->R1->R2->R4->R1	true	00:00:14:793
update	00:03:41:441	R(n)	10.0.4.0/24	10.0.5.1	23	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:14:865
update	00:03:41:452	R(n)	10.0.4.0/24	10.0.5.1	23	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:14:876
update	00:03:41:467	R(n)	10.0.4.0/24	10.0.5.1	23	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:14:891
update	00:03:41:471	R(n)	10.0.4.0/24	10.0.5.1	26	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:14:895
update	00:03:45:277	R(n)	10.0.4.0/24	10.0.5.1	26	10.0.5.1	0	00:14	R3->R1->R2->R4->R1	true	00:00:18:701
update	00:03:45:279	R(n)	10.0.4.0/24	10.0.5.1	29	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:18:703
update	00:03:46:436	R(n)	10.0.4.0/24	10.0.5.1	29	10.0.5.1	0	00:17	R3->R1->R2->R4->R1	true	00:00:19:860
update	00:03:46:491	R(n)	10.0.4.0/24	10.0.5.1	32	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:19:915
update	00:03:46:504	R(n)	10.0.4.0/24	10.0.5.1	32	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:19:928
update	00:03:46:506	R(n)	10.0.4.0/24	10.0.5.1	35	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:19:930
update	00:03:50:273	R(n)	10.0.4.0/24	10.0.5.1	35	10.0.5.1	0	00:14	R3->R1->R2->R4->R1	true	00:00:23:697
update	00:03:50:531	R(n)	10.0.4.0/24	10.0.5.1	38	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:23:955
update	00:03:51:444	R(n)	10.0.4.0/24	10.0.5.1	38	10.0.5.1	0	00:17	R3->R1->R2->R4->R1	true	00:00:24:868
update	00:03:51:489	R(n)	10.0.4.0/24	10.0.5.1	41	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:24:913
update	00:03:51:512	R(n)	10.0.4.0/24	10.0.5.1	41	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:24:936
update	00:03:51:525	R(n)	10.0.4.0/24	10.0.5.1	44	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:24:949
update	00:03:55:533	R(n)	10.0.4.0/24	10.0.5.1	44	10.0.5.1	0	00:14	R3->R1->R2->R4->R1	true	00:00:28:957
update	00:03:55:543	R(n)	10.0.4.0/24	10.0.5.1	47	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:28:967
update	00:03:56:470	R(n)	10.0.4.0/24	10.0.5.1	47	10.0.5.1	0	00:17	R3->R1->R2->R4->R1	true	00:00:29:894
update	00:03:56:491	R(n)	10.0.4.0/24	10.0.5.1	47	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:29:915
update	00:03:58:551	R(n)	10.0.4.0/24	10.0.5.1	50	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:31:975
update	00:03:58:583	R(n)	10.0.4.0/24	10.0.5.1	50	10.0.5.1	0	00:17	R3->R1->R2->R4->R1	true	00:00:32:007
update	00:04:02:270	R(n)	10.0.4.0/24	10.0.5.1	50	10.0.5.1	0	00:14	R3->R1->R2->R4->R1	true	00:00:35:694
update	00:04:02:354	R(n)	10.0.4.0/24	10.0.5.1	53	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:35:778
update	00:04:02:381	R(n)	10.0.4.0/24	10.0.5.1	53	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:35:805
update	00:04:02:394	R(n)	10.0.4.0/24	10.0.5.1	56	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:35:818
update	00:04:04:400	R(n)	10.0.4.0/24	10.0.5.1	56	10.0.5.1	0	00:16	R3->R1->R2->R4->R1	true	00:00:37:824
update	00:04:06:314	R(n)	10.0.4.0/24	10.0.5.1	59	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:39:738
update	00:04:07:316	R(n)	10.0.4.0/24	10.0.5.1	59	10.0.5.1	0	00:17	R3->R1->R2->R4->R1	true	00:00:40:740
update	00:04:07:382	R(n)	10.0.4.0/24	10.0.5.1	59	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:40:806
update	00:04:07:393	R(n)	10.0.4.0/24	10.0.5.1	62	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:40:817
update	00:04:07:403	R(n)	10.0.4.0/24	10.0.5.1	62	10.0.5.1	0	00:18	R3->R1->R2->R4->R1	true	00:00:40:827
update	00:04:09:403	R(n)	10.0.4.0/24	10.0.5.1	64	10.0.5.1	0	00:12	R3->R1->R2->R4->R1	true	00:00:42:827
update	00:04:11:296	R(n)	10.0.4.0/24	10.0.5.1	64	10.0.5.1	0	00:10	R3->R1->R2->R4->R1	true	00:00:00:000
update	00:04:12:314	R(n)	10.0.4.0/24	10.0.5.1	64	10.0.5.1	0	00:09	R3->R1->R2->R4->R1	true	00:00:00:000
update	00:04:16:313	R(n)	10.0.4.0/24	10.0.5.1	64	10.0.5.1	0	00:05	R3->R1->R2->R4->R1	true	00:00:00:000
update	00:04:17:371	R(n)	10.0.4.0/24	10.0.5.1	64	10.0.5.1	0	00:04	R3->R1->R2->R4->R1	true	00:00:00:000
update	00:04:20:342	R(n)	10.0.4.0/24	10.0.5.1	64	10.0.5.1	0	00:01	R3->R1->R2->R4->R1	true	00:00:00:000
garbage	00:04:22:253	R(n)	10.0.4.0/24	10.0.5.1	0	10.0.5.1	0	0	R3->R1->R2->R4->R1	true	00:00:00:000

Abbildung 35: Analysetabelle: CTI im Y-Mod-Szenario

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	47s
T_1	6s / 36s / 24s	54,5s
T_2	10s / 60s / 40s	55,6s
T_3	30s / 180s / 120s	72,7s

Tabelle 8: Ergebnisse versch. T-Einstellungen beim CTI im Y-Mod-Szenario

Der Count-To-Infinity-Effekt entsteht mit einer Wahrscheinlichkeit von über 90% in allen Testdurchläufen. Die Variation der Timer-Einstellungen wirkt sich hier genauso begünstigend auf die Konvergenzzeiten aus wie in anderen Szenarien auch. Festzustellen ist, dass der Geschwindigkeitsbonus den größten Sprung von T_3 auf T_2 macht und ab da nur noch geringfügig zulegt. Der Geschwindigkeitsbonus liegt zwischen 24% und maximal 36% bei **T_0-Einstellungen.**

Ergebnisse des RIPMTI-Algorithmus

Der RIPMTI-Algorithmus verhindert den Count-To-Infinity-Effekt theoretisch zuverlässig. Auf R3 wird die Falschnachricht von R1 als Source-Loop erkannt. Der Grund dafür ist, dass zwischen Interface 1 und 2 kein Simple-Loop erkannt wurde. Der entsprechende Wert in der MSILM-Tabelle steht auf 127, da Interface 1 nicht mehr verbunden ist.

$$m_{IF2}^{(R3,10.0.4.0/24)} + m_{IF1}^{(R3,10.0.4.0/24)} - 1 \geqslant 127$$

$$4 + 2 - 1 = 5 \geqslant 127\,(f)$$

Das gleiche gilt für eine Falschnachricht von R4, denn auch über Interface 1 und 3 existiert kein Simple-Loop mehr. Auf R4 funktioniert der Simple-Loop-Test ebenfalls, da der Simple-Loop zwischen Interface 1 und den übrigen Interfaces ausgeschlossen werden kann. Worauf es bei diesem Szenario ankommt ist die Zeit. Der Request-Timer der Route muss sich an die maximale Schleifengröße anpassen, die die große Netzwerkschleife $R1 \rightarrow R2 \rightarrow R4 \rightarrow R3 \rightarrow R1$ umfasst. Vier Router ergeben ein Zeitfenster von 20 Sekunden, in denen auf R3 und R4 sämtliche Routinginformationen zu Netz 10.0.4.0/24 abgelehnt werden. In dieser Zeit verbreiten beide Router RIP_Infinity im Netzwerk und gewährleisten, dass bis

zum Ende des Zeitfensters alle Router des Netzwerkes die gültige Metrik 64 in ihrer Routingtabelle eingetragen haben. Abbildung 36 zeigt die erfolgreiche Arbeitsweise von RIPMTI.

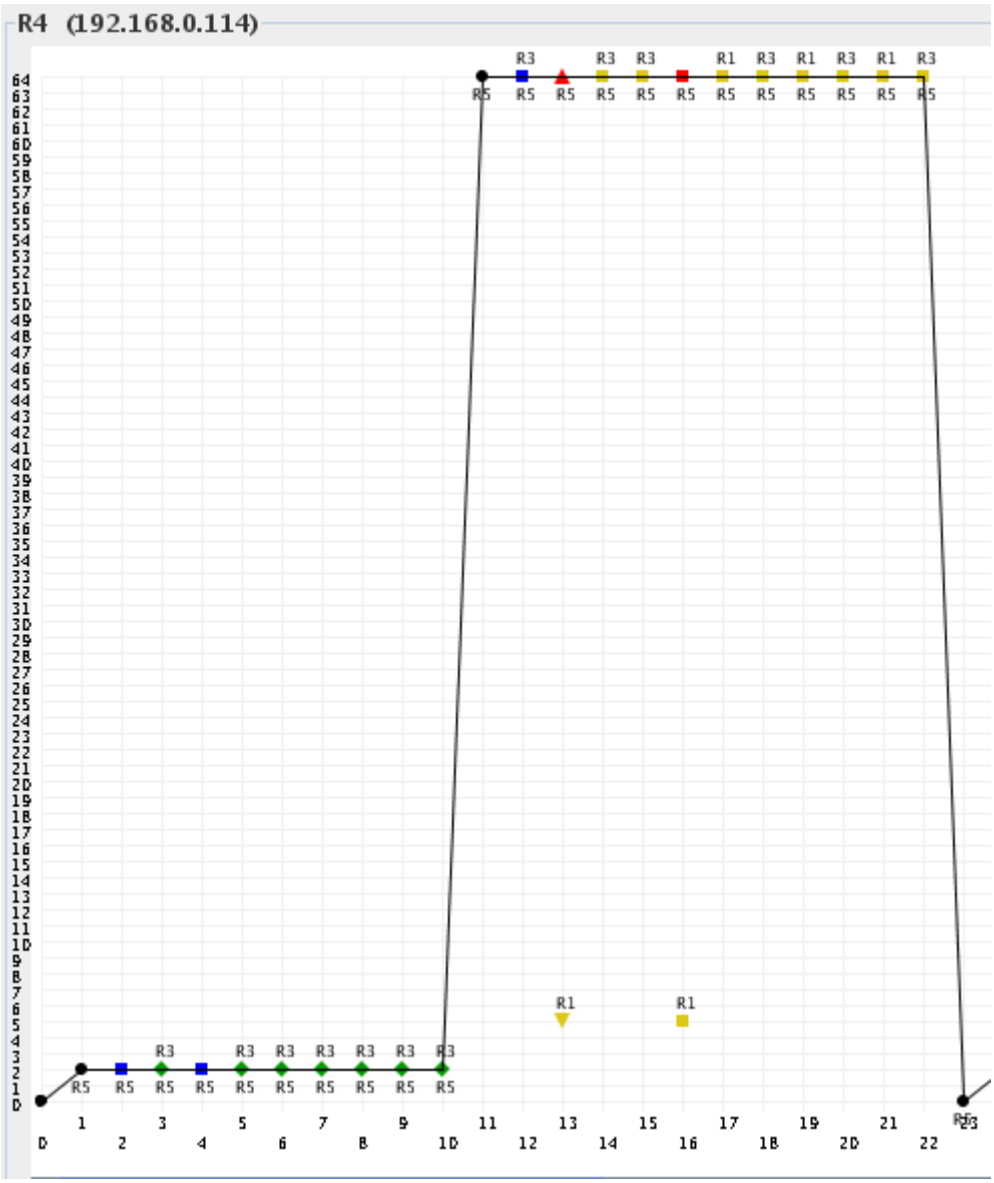


Abbildung 36: Netzgraph: RIPMTI im Y-Mod-Szenario

Allerdings hat sich ein Fehler in der Implementation herauskristallisiert wenn eine ganz bestimmte Sendereihenfolge von Updatenachrichten erfolgt. Folgender Ablauf ist entscheidend. Nach Ablauf des T_{TT} stellen R3 und R4 die Unerreichbarkeit des Netzes 10.0.4.0/24 fest. R1 soll zu diesem Zeitpunkt noch die alte Metrik besitzen und diese nun an R4 senden. R4 erkennt darin eine gültige Route mit der Metrik 4 über $R4 \rightarrow R1 \rightarrow R3 \rightarrow R5$ auf Interface 4. Währenddessen hat R2 seine Falschnachricht an R1 gesendet. Im Anschluss gelangt die Falschnachricht auf

genau diesem Interface mit der Metrik 5 über $R4 \rightarrow R1 \rightarrow R2 \rightarrow R4 \rightarrow R5$ an R4. Auch wenn auf R4 der Careful-RT-Mode läuft und die gewünschten Einstellungen konfiguriert werden, reagiert der RIPMTI-Algorithmus fehlerhaft. Wenn sich RIPMTI nicht an den Interfaces sondern der IP orientiert, über die ein Router die jeweilige Route gelernt hat, und darüber hinaus über die Konfigurationsoption „Oldmetric Delay“ die alte, ausgefallene Route anhand der Werte aus der MSILM-Tabelle lange genug vorhält, müsste R4 eigentlich eine Y-Topologie erkennen und den Source-Loop erkennen. Tatsächlich ergibt die fehlerhafte Implementation, dass die alte und neu angebotene Route, die aus unterschiedlichen Richtungen und damit über unterschiedliche Interfaces/IP gelernt wurden, vom gleichen Interface (Debug-Data: „EQUAL_INDICES“) stammen sollen. Aufgrund dieses Fehlers interpretiert R4 die angebotene Route als gültig und übernimmt die Metrik in seine Routingtabelle. Der Count-To-Infinity-Effekt breitet sich infolgedessen aus.

Zusammenfassung

Die Verschachtelung hat in diesem Szenario theoretisch keine Probleme verursacht. Wichtig für den RIPMTI-Algorithmus ist die Konfiguration auf die „MTI Orientation“ (IP, Multicast) und „MTI Oldmetric Delay“ (MSILM=5). Unter Verwendung der T_3-Einstellungen bewirkt RIPMTI im Falle eines Count-To-Infinity-Effektes ein um 20% schneller konvergierendes Netz und bei T_0 knapp über 70%.

In dem beschriebenen Fall, in dem die Routingschleife nicht verhindert wurde, konnte ein Implementierungsfehler des Algorithmus ausgemacht werden. Dessen Behebung und die sich daraus ergebende korrekte Verhaltensweise des RIPMTI-Routers bei eingehenden Routinginformationen auf unterschiedlichen Interfaces zum gleichen Zielnetz verbessert die Wirksamkeit des RIPMTI-Algorithmus.

6.7 Interloop

Dieses Szenario leitet sich aus einer Y-Struktur ab, die durch eine zusätzliche Verbindung zwischen Routern einer Netzwerkschleife zu einer verschachtelten Topologie erweitert wird.

Modellbeschreibung

Das Interloop-Szenario besteht aus fünf Routern. Router R1, R2, R3 und R4 bilden eine große Netzwerkschleife. Aufgrund einer zusätzlichen Verbindungsleitung zwischen R2 und R4 entsteht eine Verschachtelung kleinerer Zyklen innerhalb der großen Netzwerkschleife. Ein Zyklus A mit R1, R2 und R4 und ein Zyklus B mit R2, R4 und R3. Außerhalb der großen Netzwerkschleife existiert noch ein weiterer Router R0, der mit R1 verbunden ist. Diese Topologie spielt eine besondere Rolle zur Erkennung eines Count-To-Infinity-Effekts bei Routern, die über mehr als zwei Interfaces innerhalb eines Zyklus verfügen.

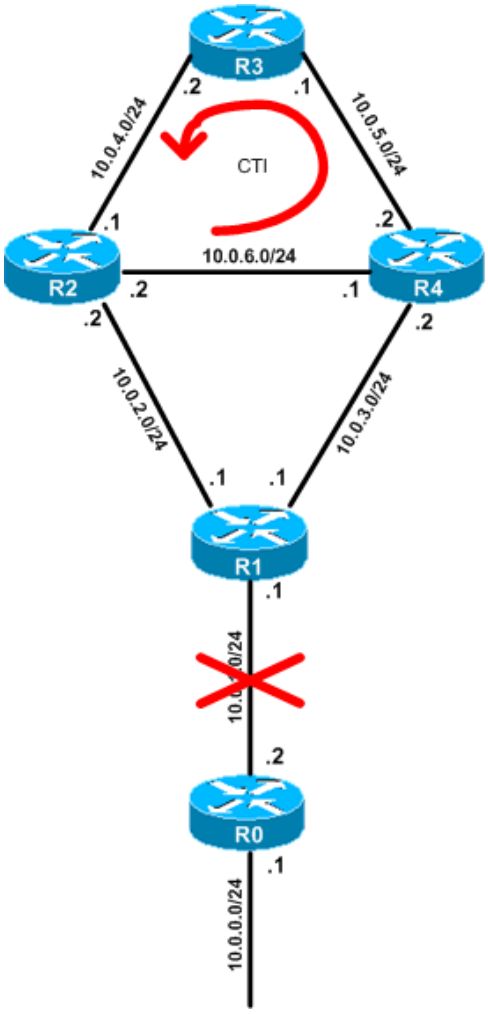


Abbildung 37: Topologie: Interloop-Szenario

Erzeugung eines Count-To-Infinity-Effekts

Es soll eine Falschnachricht im Zyklus B ausgelöst werden. Dazu blockiert die Konfigurationsdatei aus Anhang G „Interloop1“ Interface 1 (10.0.1.2) von R0, um den Verbindungsausfall zwischen R0 und R1 und den Verlust des Netzes 10.0.0.0/24 zu simulieren. R1 erkennt den Verlust nach Ablaufen von T_{TT} und propagiert die Unerreichbarkeit in der Netzwerkschleife. R1 sendet seine Routinginformation über Interface 2 (10.0.2.1) direkt an R2 und um eine halbe Sekunde verzögert über Interface 3 (10.0.3.1) an R4. Dadurch kann bereits bei R4 eine Falschnachricht ausgelöst werden, die er über Interface 3 (10.0.6.1) an R2 sendet und bei R2 RIP_Infinity überschreibt. R2 blockiert sein

Interface 2 (10.0.4.1) in Richtung R3, sodass R3 noch von der neuen Routinginformation ausgeschlossen ist. Durch diesen Vorgang wird R3 auch dazu veranlasst, in seiner regelmäßigen Updatenachricht die alte, ungültige Erreichbarkeitsmetrik zum Netz 10.0.0.0/24 zu verbreiten. R3 und R4 werden zum False-Router und senden ihre Falschnachricht jeweils ins Netzwerk. R2 gibt die Nachricht auf Interface 1 (10.0.2.2) an R1, auf Interface 2 an R3 und Interface 3 (10.0.6.2) an R4 weiter. R4 erhält die Falschnachricht von R3 und ändert RIP_Infinity hin zur vermeintlich alternativen Metrik und sendet diese ebenfalls an R1 und R2 weiter. Abhängig vom Interface, über das die Routinginformation gelernt wurde, verhindert Split Horizon das entsprechende eingehende Interface. Durch diese Falschnachrichten wird der Count-To-Infinity-Effekt erzeugt und setzt sich sowohl in Zyklus A als auch in Zyklus B fort und in der großen Netzwerkschleife fort..

Ergebnisse des RIP-Algorithmus

Im konvergenten Zustand hat R1 die Metrik 2 zum ausgefallenen Netz 10.0.0.0/24, R2 und R4 die Metrik 3 und R3 die Metrik 4. In den Abbildungen 38 und 39 lässt sich erkennen wie sich der Count-To-Infinity-Effekt in den verschachtelten Netzwerkschleifen bewegt. Die erste Schleifeninformation erreicht den Source-Router R1 über R2. Die Route führt dabei über $R1 \rightarrow R2 \rightarrow R4 \rightarrow R1$ und wurde durch die erste Falschnachricht von R4 ausgelöst, die R1 über R2 erreicht hat. Dieses Verhalten ist vergleichbar mit der typischen Y-Topologie, wenn man R3 zunächst außer Acht lässt. R1 verzeichnet daher die neue Metrik 5 in seiner Routingtabelle. Bereits im nächsten periodischen Update setzt sich der Count-To-Infinity-Effekt von Zyklus B durch, der parallel zu dem in Zyklus A entstanden ist. Da R2 und R4 mit drei Interfaces an beide Zyklen angebunden sind, gelangen beide Schleifeninformationen zu R1, der nun eine neue Route über $R1 \rightarrow R2 \rightarrow R4 \rightarrow R3 \rightarrow R2 \rightarrow R1$ mit Metrik 7 empfängt. Der Count-To-Infinity-Effekt von Zyklus B kann auch aus der anderen Richtung R1 erreichen, indem er über R4 eine Alternativroute zum ausgefallenen Netz angeboten bekommt. Die dazugehörige Route sind dann so aus: $R1 \rightarrow R4 \rightarrow R3 \rightarrow R2 \rightarrow R4 \rightarrow R1$. Diese und die vorige Route wandern beide im Netzwerk umher und verändern die Routingtabelle von R1 abhängig davon, welche Metrik zuerst ankommt. Wenn zuerst über R4 eine Metrik ankommt und über R2 kurz danach die gleiche, dann wird die zweite verworfen und die erste übernommen. Der Count-To-Infinity-Effekt zählt sowohl in Zyklus A wie auch in Zyklus B in Dreierschritten bis RIP_Infinity.

3-18-12_CTI

graph

R3 R0 R2 R4 R1

10.0.6.0/24 10.0.5.0/24 10.0.4.0/24 10.0.3.0/24 10.0.2.0/24 10.0.1.0/24 10.0.0.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:02:27:351	R(n)	10.0.0.0/24	10.0.1.2	2	10.0.1.2	0	00:18	R1->R0	false	00:00:00:000
timeout	00:02:45:382	R(n)	10.0.0.0/24	10.0.1.2	64	10.0.1.2	0	00:12	R1->R0	false	00:00:00:000
update	00:02:48:533	R(n)	10.0.0.0/24	10.0.2.2	5	10.0.2.2	0	00:18	R1->R2->R4->R1	false	00:00:00:000
update	00:02:49:537	R(n)	10.0.0.0/24	10.0.2.2	5	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:01:004
update	00:02:53:051	R(n)	10.0.0.0/24	10.0.2.2	7	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:04:518
update	00:02:53:144	R(n)	10.0.0.0/24	10.0.2.2	7	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:04:611
update	00:02:53:146	R(n)	10.0.0.0/24	10.0.2.2	10	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:04:613
update	00:02:55:009	R(n)	10.0.0.0/24	10.0.3.2	9	10.0.3.2	0	00:18	R1->R4->R3->R2->R4	true	00:00:06:476
update	00:02:55:121	R(n)	10.0.0.0/24	10.0.3.2	12	10.0.3.2	0	00:18	R1->R4->R3->R2->R4	true	00:00:06:588
update	00:02:55:157	R(n)	10.0.0.0/24	10.0.3.2	12	10.0.3.2	0	00:18	R1->R4->R3->R2->R4	true	00:00:06:624
update	00:02:59:036	R(n)	10.0.0.0/24	10.0.3.2	15	10.0.3.2	0	00:18	R1->R4->R3->R2->R4	true	00:00:10:503
update	00:02:59:147	R(n)	10.0.0.0/24	10.0.2.2	13	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:10:614
update	00:02:59:153	R(n)	10.0.0.0/24	10.0.2.2	16	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:10:620
update	00:02:59:168	R(n)	10.0.0.0/24	10.0.2.2	16	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:10:635
update	00:03:01:163	R(n)	10.0.0.0/24	10.0.2.2	19	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:12:630
update	00:03:01:194	R(n)	10.0.0.0/24	10.0.2.2	19	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:12:661
update	00:03:03:063	R(n)	10.0.0.0/24	10.0.2.2	19	10.0.2.2	0	00:16	R1->R2->R4->R3->R2	true	00:00:14:530
update	00:03:03:100	R(n)	10.0.0.0/24	10.0.2.2	22	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:14:567
update	00:03:03:143	R(n)	10.0.0.0/24	10.0.2.2	22	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:14:610
update	00:03:03:159	R(n)	10.0.0.0/24	10.0.2.2	25	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:14:626
update	00:03:05:162	R(n)	10.0.0.0/24	10.0.2.2	25	10.0.2.2	0	00:16	R1->R2->R4->R3->R2	true	00:00:16:629
update	00:03:05:175	R(n)	10.0.0.0/24	10.0.2.2	28	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:16:642
update	00:03:07:089	R(n)	10.0.0.0/24	10.0.2.2	28	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:18:556
update	00:03:07:188	R(n)	10.0.0.0/24	10.0.2.2	28	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:18:655
update	00:03:07:195	R(n)	10.0.0.0/24	10.0.2.2	31	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:18:662
update	00:03:08:089	R(n)	10.0.0.0/24	10.0.2.2	31	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:19:556
update	00:03:08:203	R(n)	10.0.0.0/24	10.0.2.2	34	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:19:670
update	00:03:08:237	R(n)	10.0.0.0/24	10.0.2.2	34	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:19:704
update	00:03:09:225	R(n)	10.0.0.0/24	10.0.2.2	37	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:20:692
update	00:03:10:258	R(n)	10.0.0.0/24	10.0.2.2	37	10.0.2.2	0	00:16	R1->R2->R4->R3->R2	true	00:00:21:725
update	00:03:10:268	R(n)	10.0.0.0/24	10.0.2.2	40	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:21:735
update	00:03:12:113	R(n)	10.0.0.0/24	10.0.2.2	40	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:23:580
update	00:03:13:128	R(n)	10.0.0.0/24	10.0.2.2	40	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:24:595
update	00:03:13:167	R(n)	10.0.0.0/24	10.0.2.2	43	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:24:634
update	00:03:13:188	R(n)	10.0.0.0/24	10.0.2.2	43	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:24:655
update	00:03:16:189	R(n)	10.0.0.0/24	10.0.2.2	46	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:27:656
update	00:03:16:230	R(n)	10.0.0.0/24	10.0.2.2	46	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:27:697
update	00:03:17:130	R(n)	10.0.0.0/24	10.0.2.2	49	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:28:597
update	00:03:18:127	R(n)	10.0.0.0/24	10.0.2.2	49	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:29:594
update	00:03:18:171	R(n)	10.0.0.0/24	10.0.2.2	52	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:29:638
update	00:03:18:191	R(n)	10.0.0.0/24	10.0.2.2	52	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:29:658
update	00:03:20:180	R(n)	10.0.0.0/24	10.0.2.2	55	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:31:647
update	00:03:21:132	R(n)	10.0.0.0/24	10.0.2.2	55	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:32:599
update	00:03:21:230	R(n)	10.0.0.0/24	10.0.2.2	55	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:32:697
update	00:03:21:247	R(n)	10.0.0.0/24	10.0.2.2	58	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:32:714
update	00:03:23:137	R(n)	10.0.0.0/24	10.0.2.2	58	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:34:604
update	00:03:25:158	R(n)	10.0.0.0/24	10.0.2.2	61	10.0.2.2	0	00:18	R1->R2->R4->R3->R2	true	00:00:36:625
update	00:03:25:243	R(n)	10.0.0.0/24	10.0.2.2	61	10.0.2.2	0	00:17	R1->R2->R4->R3->R2	true	00:00:36:710
update	00:03:25:268	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:12	R1->R2->R4->R3->R2	true	00:00:36:735
update	00:03:28:171	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:10	R1->R2->R4->R3->R2	true	00:00:00:000
update	00:03:28:265	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:09	R1->R2->R4->R3->R2	true	00:00:00:000
update	00:03:30:176	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:08	R1->R2->R4->R3->R2	true	00:00:00:000
update	00:03:33:179	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:05	R1->R2->R4->R3->R2	true	00:00:00:000
update	00:03:34:193	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:04	R1->R2->R4->R3->R2	true	00:00:00:000
update	00:03:37:205	R(n)	10.0.0.0/24	10.0.2.2	64	10.0.2.2	0	00:01	R1->R2->R4->R3->R2	true	00:00:00:000
garbage	00:03:37:284	R(n)	10.0.0.0/24	10.0.2.2	0	10.0.2.2	0	0	R1->R2->R4->R3->R2	true	00:00:00:000

Abbildung 38: Analysetabelle: CTI im Interloop-Szenario

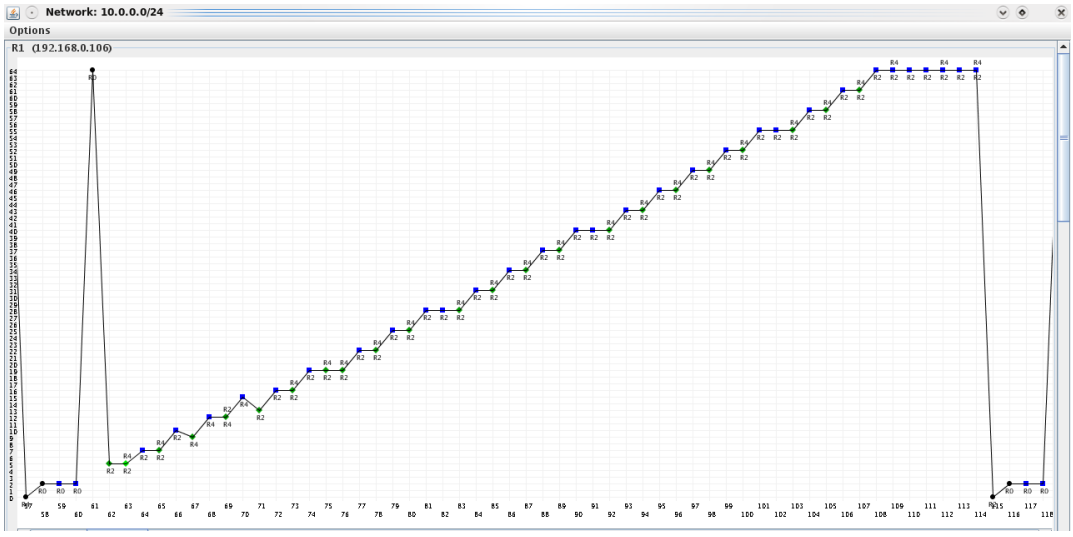


Abbildung 39: Netzgraph: CTI im Interloop-Szenario

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	42,3s
T_1	6s / 36s / 24s	47,2s
T_2	10s / 60s / 40s	56,4s
T_3	30s / 180s / 120s	81,7s

Tabelle 9: Ergebnisse versch. T-Einstellungen beim CTI im Interloop-Szenario

Im Durchschnitt zeigte sich der Count-To-Infinity-Effekt ca. 95 mal bei 100 Testdurchläufen. Auch hier ergibt sich beim Blick auf die Konvergenzzeiten ein Bild, welches vergleichbar ist mit bereits vorgestellten Szenarien. Die Konvergenzzeiten der T_3-Einstellungen sind mit 81,7 Sekunden im Mittelfeld der bereits vorgestellten Ergebnisse anzusiedeln. Beschleunigte Timer-Einstellungen wirken sich beschleunigend auf die Konvergenzzeiten aus, erreichen aber ebenso wie die anderen Modelle ganz klar einen Schwellenwert, was sich dadurch zeigt, dass sich der positive Effekt, verursacht durch die zunehmende Beschleunigung, sehr stark abschwächt. Mit den schnellsten Timer-Einstellungen konvergiert das Netzwerk nach Auftreten eines Count-To-Infinity-Effekts in nahezu der Hälfte der ursprünglichen Zeit.

Ergebnisse des RIPMTI-Algorithmus

Zu Beginn muss RIPMTI auf R1, R2 und R4 aktiviert werden. Aufgrund der Verschachtelung könnte RIPMTI auf R1 nur solche Schleifeninformationen erkennen, die über ihn selbst laufen, nicht aber solche, die sich nur im Zyklus B bewegen. R2 und R4 haben eine gleichwertige Position im Netzwerk, da sie in symmetrischer Anordnung beide Zyklen miteinander verbinden und somit beide für die Erkennung einer alten, ungültigen Routinginformation aus Zyklus B verantwortlich sind. Der RIPMTI-Algorithmus stößt bei diesem Modell auf einen Grenzfall. Nachdem R2 die Unerreichbarkeit des Netzes 10.0.0.0/24 registriert hat, erhält er eine periodische Updatenachricht von R4, der etwas verzögert über die neue Route informiert wurde und daher noch die alte, ungültige Metrik besitzt. Da der Zyklus A aus der Sicht von R2 einen Simple-Loop darstellt, wird die Falschnachricht von R4 als gültige Alternativroute angenommen und an R1

gesendet. Im Anschluss daran erhält R4 die neue Metrik 64 von R1. In diesem Zustand ist Zyklus B anfällig für eine Falschnachricht, die jetzt von R3 verschickt wird. R4 akzeptiert diese Nachricht als gültige Alternativroute und leitet sie an R2 weiter. R2 kann die Routinginformation nicht als Source-Loop erkennen, da sie über das gleiche Interface eintrifft wie die vorherige Nachricht und somit den Simple-Loop-Test besteht. Um sich gegen diesen ungünstigen Fall abzusichern, muss hier die „Oldmetric Delay"-Option eingeschaltet sein, damit die ursprüngliche Route zum ausgefallenen Netz lange genug vorgehalten wird, um sie als Vergleichsroute für neu angebotene Routen heranzuziehen.

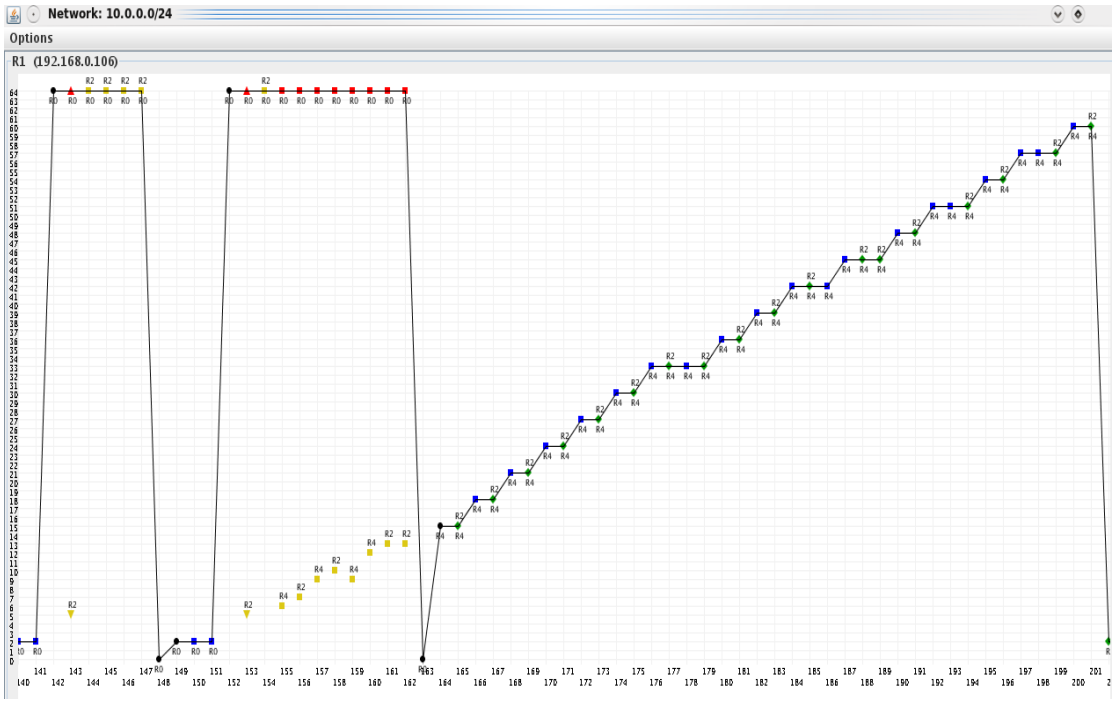


Abbildung 40: Netzgraph: RIPMTI mit ausreichend langem RT-Timer (links) und mit zu kurzem RT-Timer (rechts)

Abbildung 40 (rechts) zeigt das Verhalten des RIPMTI-Algorithmus auf Router 1. Router 1 kann den Count-To-Infinity-Effekt ohne „Oldmetric Delay" ebenso wenig verhindern wie R2. Folgendes geschieht hier. Erreicht R1 eine falsche Updatenachricht über R2, die über die Route des Zyklus A führt, erkennt R1 das anhand des Simple-Loop-Tests.

Die allgemeine Gleichung

$$m_A^{(i,d)} + m_B^{(i,d)} - 1 \geq msilm_{(A,B)}^{i}$$

schlägt auf R1 für das eingehende Update von R2 über die Route $R1 \rightarrow R2 \rightarrow R4 \rightarrow R1$ fehl.

$$m_{IF2}^{(R1,10.0.0.0/24)} + m_{IF1}^{(R1,10.0.0.0/24)} - 1 \geq msilm_{(IF1,IF2)}^{R1}$$

$$5 + 2 - 1 = 6 \geqslant 127\,(f)$$

Es wurde kein Simple-Loop zwischen Interface 1 und 2 erkannt. Der RIPMTI-Algorithmus lehnt jetzt für 15 Sekunden alle neuen Routinginformationen ab, da er aufgrund der MSILM-Tabelle für das Interface 2 einen kleinsten Schleifenumfang von 3 für Zyklus A ermittelt hat. Multipliziert man den Schleifenumfang mit dem Triggered Timer von 5 Sekunden, ergibt das eine maximale Umlaufdauer einer Falschnachricht von 15 Sekunden. Diese Zeit reicht für den „Oldmetric Delay Timer", damit R1 in dieser Phase RIP_Infinity im Netzwerk verbreiten kann. R4 erhält zeitweise sogar RIP_Infinity. Wäre da jetzt nicht Zyklus B, der seine eigene Falschnachricht verbreitet und auf diese Weise RIP_Infinity bei R4 wieder mit einer ungültigen Erreichbarkeitsinformation überschreibt. R1 empfängt nicht nur über R2, sondern auch über R4 falsche Routinginformationen. Eine ähnliche Rechnung mit dem gleichen Ergebnis ergibt sich für die Falschnachricht von R4, da auch zwischen Interface 1 und 3 kein Simple-Loop erkannt wurde. Die vorhandenen Source-Loops werden von R1 zwar erkannt, aber das Ausbreiten der ungültigen Metrik kann nicht verhindert werden. Das Warte-Zeitfenster von R1 müsste so lang sein wie der Count-To-Infinity-Effekt selbst. Der Mechanismus ist also in diesem Fall wirkungslos, da die Korrekturnachrichten von R1 direkt wieder durch die Falschnachricht aus dem oberen Zyklus überschrieben werden. Das wirkt sich auf die Konvergenzzeiten des Interloop-Szenarios nachteilig aus. Grundsätzlich dauert es mit dem RIPMTI-Algorithmus sogar länger bis zur Konvergenz des Netzwerkes als mit RIP. Der Malus bewegt sich zwischen ca. 26% verlangsamter Konvergenzzeit bei T_3-Einstellungen und ca. 14% schlechteren Werten bei T_0-Einstellungen. Der aktivierte „Oldmetric Delay Timer", der sich über die MSILM-Tabelle errechnet, bringt hier den gewünschten Effekt, sodass kein Count-To-Infinity-Effekt entstehen kann. Tendenziell wirken sich beschleunigte Timer-Einstellungen begünstigend aus. Abbildung 40 (links) zeigt, dass in allen Sendereihenfolgen der Updatenachrichten nur eine Schleifeninformation R1 erreicht und diese eine Updatenachricht von R1 mit RIP_Infinity auslöst. Die Korrekturnachricht verbreitet sich im Netzwerk.

Zusammenfassung

RIPMTI bestätigt sich auch in diesem Szenario mithilfe des „Oldmetric Delay Timers“. Dieser verhindert, dass eine Routeninformation, die über einen Source-Loop zurückkehrt, nicht vorher überschrieben wird, da die kleinste ursprüngliche Route im RIPMTI-Router länger vorgehalten wird. Der zeitliche Gewinn für die Konvergenzzeit liegt in einem ähnlichen Bereich wie bei den bereits vorgestellten Szenarien und macht deutlich, dass eine Beschleunigung der Timer-Einstellungen vorteilhaft ist.

6.8 X-Szenario

Das X-Szenario ist eines der grundlegenden Modelle zur Veranschaulichung eines Source-Loop.

Modellbeschreibung

Das vorliegende X-Szenario besteht aus sieben Routern, wobei jeweils 2 Zyklen über einen gemeinsamen Router verbunden sind. Der obere Zyklus „A“ besteht dabei aus drei Routern und der untere Zyklus „B“ aus fünf Routern. Das minimal möglichste X-Szenario besteht nur aus fünf Routern, wobei der untere und obere Zyklus aus jeweils drei Routern bestehen. In diesem Szenario ließe sich aufgrund der Symmetrie der Topologie nur eine Ausfallsituation simulieren. Es wären zwar mehrere Verbindungen potentiell als Ausfallkandidaten möglich, aber wegen der geringen Größe des Netzwerkes wären sie alle redundant. Das Ziel besteht darin, ein X-Szenario zu entwickeln, in dem mehrere voneinander verschiedene Ausfallsituationen simulierbar sind.

Das vorliegende X-Szenario ist deshalb ein vergrößertes X-Szenario und entsteht wenn zwei Netzwerkschleifen (Schleife A bestehend aus R1, R2 und R3 und Schleife B mit R1, R0, R4, R5 und R6) über einen Knotenpunkt, in diesem Fall R1, verbunden sind.

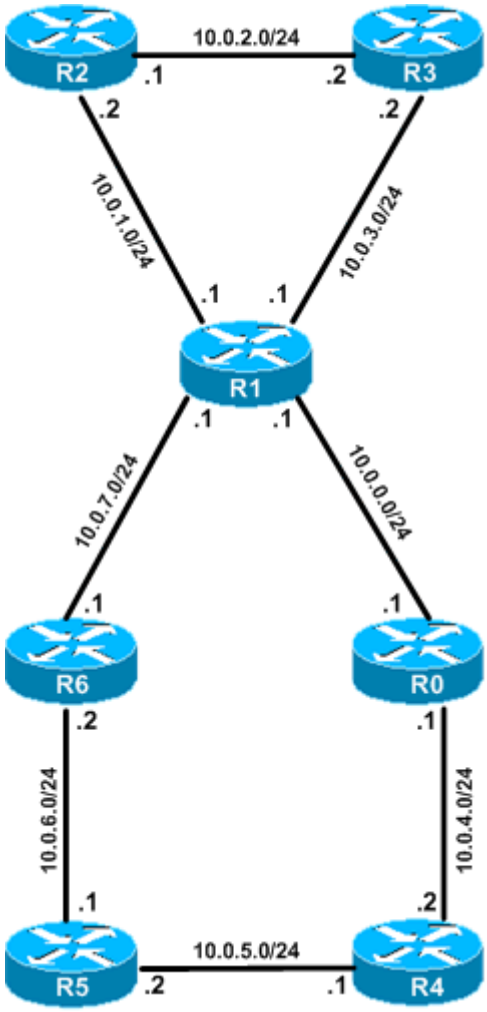


Abbildung 41: Topologie: X-Szenario

Erzeugung eines Count-To-Infinity-Effekts

Ein Count-To-Infinity-Effekt kann in zwei Situationen entstehen. Wenn wie in Anhang H **„X-Szenario1“** beschrieben, das Netz 10.0.5.0/24 ausfällt. Nachdem das Netz konvergiert ist, blockieren zwei Router ihrer Kommunikation. R4 blockiert auf Interface 1 (10.0.4.2) die Verbindung zu R0 und R5 blockiert auf Interface 1 (10.0.6.1) die Verbindung zu R6. Dadurch wird das Netz 10.0.5.0/24 als

unerreichbar deklariert. R6 und R0 stellen den Ausfall nach Ablauf ihres Timeout-Timers fest und benachrichtigen beide in ihrer Updatenachricht R1 über die neue Erreichbarkeitsinformation (Metrik 64). Je nach zeitlicher Abfolge erhält R1 entweder von R0 oder R6 die neue Metrik 64. Die Konfigurationsdatei sieht nun vor, dass R1 das neue Update nicht auf jeder Verbindungsleitung weiterleitet. Sein Interface 1 (10.0.1.1) blockiert den Nachrichtenaustausch zu R2, sodass R2 nichts vom Ausfall des Netzes 10.0.5.0/24 erfährt. Im nächsten periodischen Update schickt R2 seinerseits die alte ungültige Metrik 4 an R3 und dieser wiederum schickt die Information weiter an R1, der die Route als neu und gültig identifiziert und in seine Routingtabelle einträgt. Ab diesem Zeitpunkt bewegt sich der Count-To-Infinity-Effekt im Kreis.

Die Konfiguration aus Anhang H **„X-Szenario2“** lässt das Netz 10.0.4.0/24 ausfallen. R0 blockiert auf Interface 1 (10.0.0.1) die Verbindung zu R1. R1 stellt kurz darauf die Unerreichbarkeit des Netzes fest. Im oberen Zyklus A entsteht der Count-To-Infinity-Effekt von anschließend genauso wie im X-Szenario1 beschrieben. Damit sich der Count-To-Infinity-Effekt ausprägen kann, muss in den letzten beiden Update-Phasen der Konfigurationsdatei R6 kurzzeitig auf Interface 2 (10.0.7.1) die Verbindung zu R1 unterbrechen. Bleibt die Verbindung bestehen, lernt R1 die Route zu Netz 10.0.4.0/24 über R6 und der CTI entsteht nicht.

Ergebnis des RIPMTI-Algorithmus

Im „X-Szenario1“ lernt R1 seine Entfernung zum Netz 10.0.5.0/24 mit der Metrik 3. Nach Ausfall des Netzes lernt R1, wie in Abbildung 42 und 43 dargestellt, von R0 die neue Metrik 64. Der Garbage-Collection-Timer läuft an und wird aufgrund

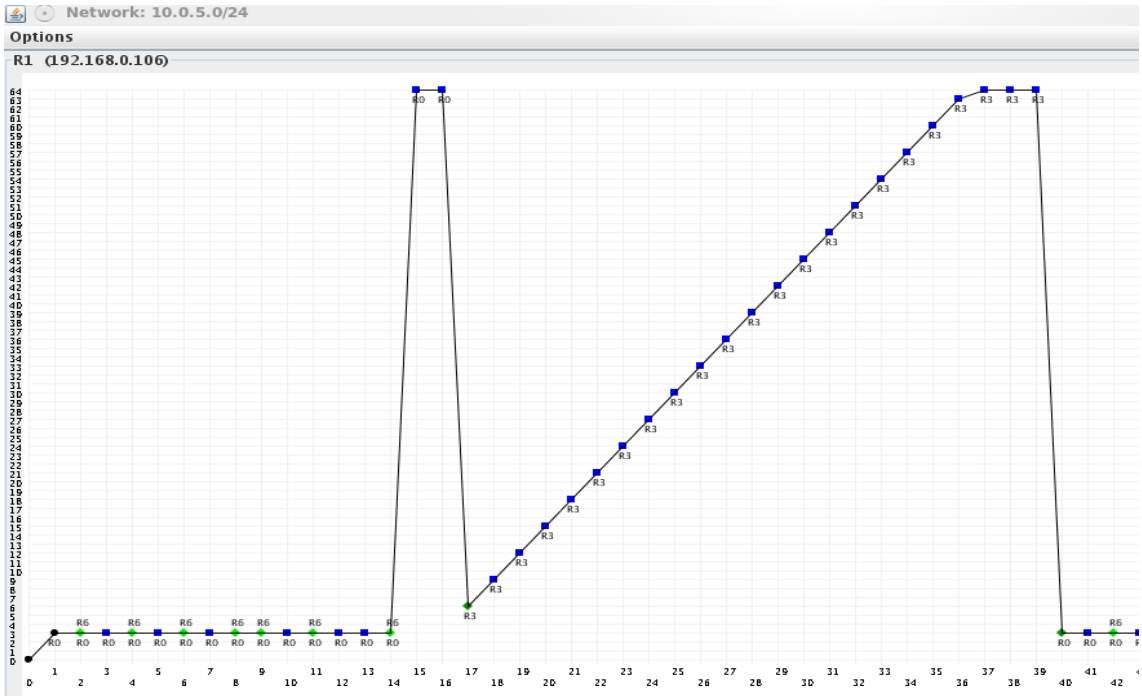


Abbildung 42: Netzgraph: CTI im X-Szenario

30-180-120_R4R5_CTI

graph

R3 | R0 | R5 | R2 | R4 | R1 | R6

10.0.6.0/24 | 10.0.5.0/24 | 10.0.4.0/24 | 10.0.3.0/24 | 10.0.2.0/24 | 10.0.1.0/24 | 10.0.0.0/24 | 10.0.7.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:02:38:233	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:02:38:251	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:03:08:332	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	02:30	R1->R0->R4	false	00:00:00:000
update	00:03:08:342	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:03:38:425	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	02:30	R1->R0->R4	false	00:00:00:000
update	00:03:38:465	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:04:08:591	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:04:08:626	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:04:38:668	R(n)	10.0.5.0/24	10.0.0.1	64	10.0.0.1	0	02:00	R1->R0->R4	false	00:00:00:000
update	00:05:08:773	R(n)	10.0.5.0/24	10.0.0.1	64	10.0.0.1	0	01:30	R1->R0->R4	false	00:00:00:000
update	00:05:09:719	R(n)	10.0.5.0/24	10.0.3.2	6	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:00:000
update	00:05:10:722	R(n)	10.0.5.0/24	10.0.3.2	9	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:01:003
update	00:05:14:788	R(n)	10.0.5.0/24	10.0.3.2	12	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:05:069
update	00:05:18:789	R(n)	10.0.5.0/24	10.0.3.2	15	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:09:070
update	00:05:22:808	R(n)	10.0.5.0/24	10.0.3.2	18	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:13:089
update	00:05:26:231	R(n)	10.0.5.0/24	10.0.3.2	21	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:16:512
update	00:05:30:248	R(n)	10.0.5.0/24	10.0.3.2	24	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:20:529
update	00:05:32:181	R(n)	10.0.5.0/24	10.0.3.2	27	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:22:462
update	00:05:34:296	R(n)	10.0.5.0/24	10.0.3.2	30	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:24:577
update	00:05:39:327	R(n)	10.0.5.0/24	10.0.3.2	33	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:29:608
update	00:05:43:333	R(n)	10.0.5.0/24	10.0.3.2	36	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:33:614
update	00:05:44:339	R(n)	10.0.5.0/24	10.0.3.2	39	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:34:620
update	00:05:49:342	R(n)	10.0.5.0/24	10.0.3.2	42	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:39:623
update	00:05:52:395	R(n)	10.0.5.0/24	10.0.3.2	45	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:42:676
update	00:05:56:399	R(n)	10.0.5.0/24	10.0.3.2	48	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:46:680
update	00:06:01:411	R(n)	10.0.5.0/24	10.0.3.2	51	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:51:692
update	00:06:05:204	R(n)	10.0.5.0/24	10.0.3.2	54	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:55:485
update	00:06:05:458	R(n)	10.0.5.0/24	10.0.3.2	57	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:55:739
update	00:06:08:474	R(n)	10.0.5.0/24	10.0.3.2	60	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:00:58:755
update	00:06:12:477	R(n)	10.0.5.0/24	10.0.3.2	63	10.0.3.2	0	03:00	R1->R3->R2->R1	false	00:01:02:758
update	00:06:17:490	R(n)	10.0.5.0/24	10.0.3.2	64	10.0.3.2	0	02:00	R1->R3->R2->R1	false	00:01:07:771
update	00:06:30:219	R(n)	10.0.5.0/24	10.0.3.2	64	10.0.3.2	0	01:47	R1->R3->R2->R1	false	00:00:00:000
update	00:06:48:872	R(n)	10.0.5.0/24	10.0.3.2	64	10.0.3.2	0	01:29	R1->R3->R2->R1	false	00:00:00:000
update	00:06:48:966	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000
update	00:07:19:027	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	03:00	R1->R0->R4	false	00:00:00:000

Abbildung 43: Analysetabelle: CTI im X-Szenario

der Falschnachricht von R3 unterbrochen. R3 sendet an R1 die Metrik 5 und daraufhin trägt R1 in seiner Routingtabelle die vermeintlich neue und bessere Erreichbarkeit mit der Metrik 6 (sich eingeschlossen) ein. Da der Zyklus A aus 3 Routern besteht, zählt der nun folgende Count-To-Infinity-Effekt in Dreierschritten von 6, 9, 12, 15 bis 63, 64 hoch. Die Schleife führt dann über R1->R3->R2->R1.

Das **„X-Szenario2“** zeigt häufiger als andere Szenarien einen Sonderfall, in dem der Count-To-Infinity-Effekt durch die Eigendynamik des Netzwerks verhindert werden kann. Anfangs hat R1 zum Netz 10.0.4.0/24 eine Entfernung von 2. Nachdem das Netz für R1 ausgefallen ist, setzt er die Metrik auf 64. Da aus Sicht von R1 eine Alternativ-Route zu diesem Netz existiert, nämlich über R6->R5->R4, muss die Verbindung von R6 nach R1 kurzzeitig unterbrochen werden. Andernfalls würde der Count-To-Infinity-Effekt im Zyklus A nicht auftreten. Sobald R1 die Metrik 64 hat, würde er im Anschluss von R6 die Metrik 4 lernen und dadurch den Count-To-Infinity-Effekt verhindern. Mit der hier verwendeten Konfigurationsdatei, soll sich der Count-To-Infinity-Effekt entwickeln und danach wird die Verbindung von R6 zu R1 wieder eingeschaltet. Im Folgenden durchläuft die von R2 erzeugte Falschnachricht einige Male den Zyklus A. Im nächsten Schritt sendet R6 seine Alternativ-Route an R1 und beendet damit den Count-To-Infinity-Effekt. Abbildung 44 zeigt wie diese spezielle Update-Reihenfolge den Count-To-Infinity-Effekt selbständig verhindert.

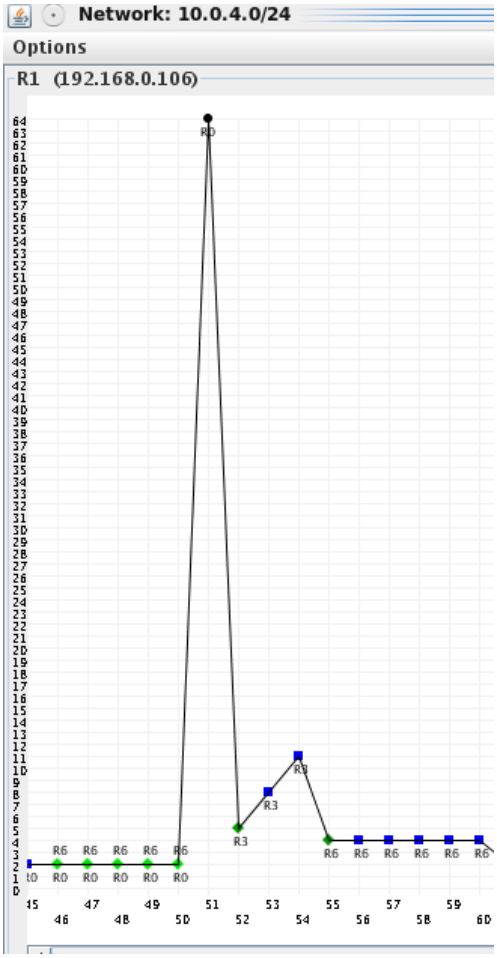
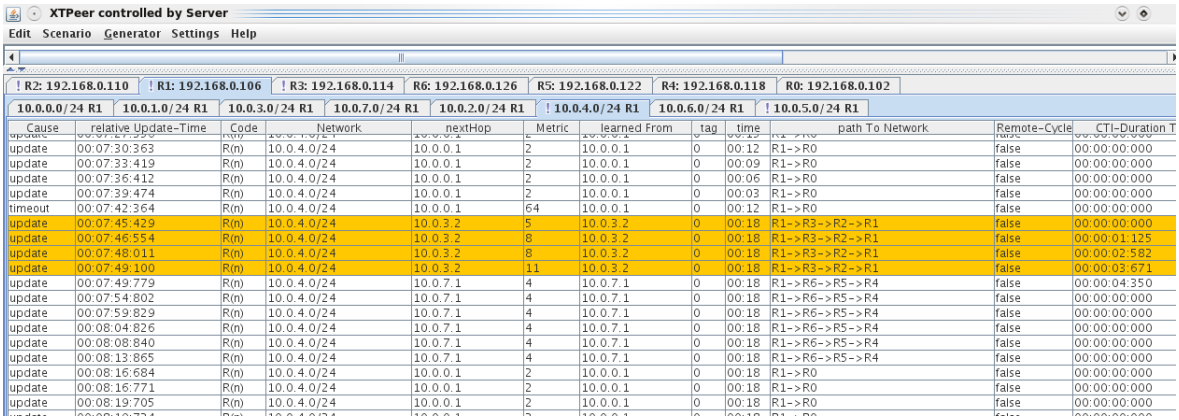


Abbildung 44: X-Szenario: CTI von R1 über Netz 10.0.4.0/24 (Ausfall R0) wird von R6 beendet

R1 hat in der Konvergenzphase das Netz 10.0.4.0/24 kennen gelernt und verliert dann die Verbindung. Im Anschluss daran wird der Count-To-Infinity-Effekt provoziert und zählt in Dreierschritten hoch. So lange bis R6 ein periodisches Update sendet, in dem R1 über die neue Router über R6 mit der Metrik 4 informiert wird. Da diese Router kürzer ist als der von R3 propagierte Weg, wird diese als gültig angenommen und der Count-To-Infinity-Effekt endet. Das Netz konvergiert wie Abbildung 45 zeigt innerhalb von 3 Sekunden. In diesem Fall sogar schneller als RIPMTI, da R6 im günstigsten Fall direkt R1 die neue Route zeigen könnte und RIPMTI zumindest den Garbage-Collection-Timer ablaufen lässt.
Bei der Berechnung der Konvergenzzeiten werden die gemessenen Daten der **X-**



Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:07:30:363	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:07:33:419	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:09	R1->R0	false	00:00:00:000
update	00:07:36:412	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:06	R1->R0	false	00:00:00:000
update	00:07:39:474	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:03	R1->R0	false	00:00:00:000
timeout	00:07:42:364	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:07:45:429	R(n)	10.0.4.0/24	10.0.3.2	5	10.0.3.2	0	00:18	R1->R3->R2->R1	false	00:00:00:000
update	00:07:46:554	R(n)	10.0.4.0/24	10.0.3.2	8	10.0.3.2	0	00:18	R1->R3->R2->R1	false	00:00:01:125
update	00:07:48:011	R(n)	10.0.4.0/24	10.0.3.2	8	10.0.3.2	0	00:18	R1->R3->R2->R1	false	00:00:02:582
update	00:07:49:100	R(n)	10.0.4.0/24	10.0.3.2	11	10.0.3.2	0	00:18	R1->R3->R2->R1	false	00:00:03:671
update	00:07:49:779	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:04:350
update	00:07:54:802	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:07:59:829	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:08:04:826	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:08:08:840	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:08:13:865	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:08:16:684	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:18	R1->R0	false	00:00:00:000
update	00:08:16:771	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:18	R1->R0	false	00:00:00:000
update	00:08:19:705	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:18	R1->R0	false	00:00:00:000

Abbildung 45: Analysetabelle X-Szenario: CTI von R1 über Netz 10.0.4.0/24 (Ausfall R0) wird von R6 beendet

Szenario2-Konfigurationsdatei nicht berücksichtigt, da die niedrigen Werte den Count-To-Infinity-Effekt unzureichend beschreiben und das Ergebnis von **X-Szenario1** zu sehr verfälschen. Tabelle 10 zeigt die durchschnittlichen Konvergenzzeiten aus 100 Durchläufen pro Timer-Einstellung.

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	41s
T_1	6s / 36s / 24s	44,6s
T_2	10s / 60s / 40s	53,3s
T_3	30s / 180s / 120s	64,1s

Tabelle 10: Ergebnisse versch. T-Einstellungen beim CTI im X-Szenario

Verwendet man schnellere Timer-Einstellungen, dann reagiert das Netzwerk dementsprechend zügiger auf Veränderungen der Topologie. Soll das Netzwerk dreimal so schnell Updates austauschen wie bisher (Updatezeiten von 30s auf 10s) vorgesehen, konvergiert es nach Auftreten eines Count-To-Infinity-Effekts doppelt so schnell. Fünfmal so schnelle Timer-Einstellungen (Updatezeiten von 30s auf 6s) lassen das Netzwerk knapp dreimal so schnell konvergieren und ein Zehntel der ursprünglichen Zeiten (Updatezeiten von 30s auf 3s) beschleunigt die Konvergenzzeit um den Faktor 3,5. Die Leistungssteigerung durch beschleunigte Timer-Einstellungen lässt sich nicht unendlich weiterführen, sondern fällt immer geringer aus.

Ergebnis des RIPMTI-Algorithmus

R1 spielt in diesem Szenario die zentrale Rolle. R1 verbindet die beiden Zyklen und stellt beim Count-To-Infinity-Effekt den Source-Router dar. Aus diesem Grund ist R1 dafür verantwortlich, den Count-To-Infinity-Effekt zu erkennen und zu verhindern, der in ungefähr 90% aller Tests provoziert werden konnte. RIPMTI hat ihn zu 100% erfolgreich verhindert. Abbildung 46 und 47 zeigen den Metrikgraph bzw. die numerische Anzeige der ausgewerteten Daten. R1 erhält die Metrik 64 und anschließend bekommt er eine Falschnachricht von R3. Der Y-Test erkennt einen Source-Loop in der Route und blockiert daraufhin jedes weitere Update von R3. In diesem Zeitraum lernen R2 und R3 ebenfalls die 64 als gültige Metrik und das Netz ist konvergent.

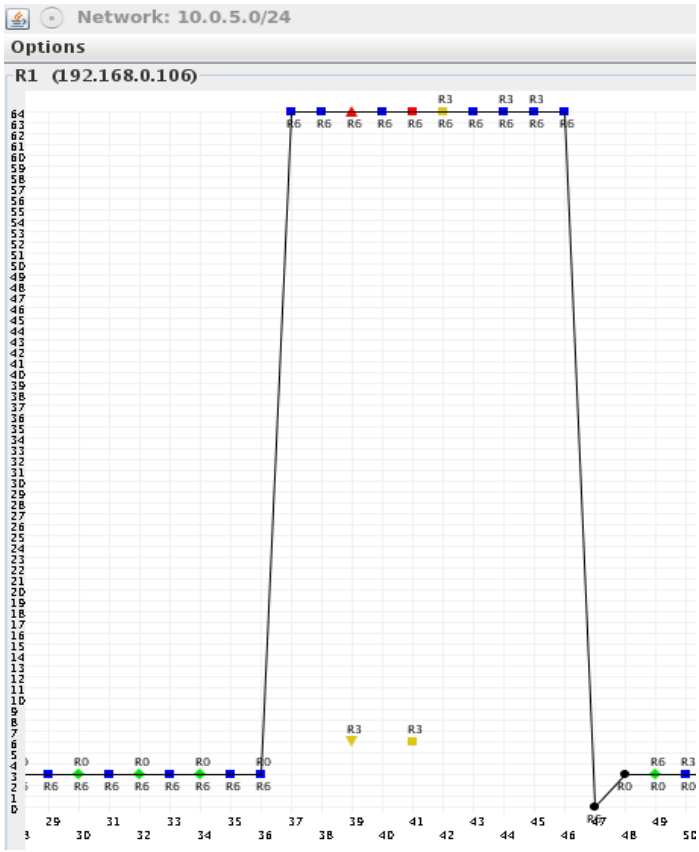


Abbildung 46: Netzgraph: RIPMTI im X-Szenario

6-36-24_RIPMTI

graph

R3 | R0 | R5 | R2 | R4 | R1 | R6

10.0.6.0/24 | 10.0.5.0/24 | 10.0.4.0/24 | 10.0.3.0/24 | 10.0.2.0/24 | 10.0.1.0/24 | 10.0.0.0/24 | 10.0.7.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration
update	00:02:28:626	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:30	R1->R6->R5	false	00:00:00:000
update	00:02:28:715	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:36	R1->R6->R5	false	00:00:00:000
update	00:02:34:634	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:30	R1->R6->R5	false	00:00:00:000
update	00:02:34:743	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:36	R1->R6->R5	false	00:00:00:000
update	00:02:40:677	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:30	R1->R6->R5	false	00:00:00:000
update	00:02:40:792	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:36	R1->R6->R5	false	00:00:00:000
update	00:02:46:737	R(n)	10.0.5.0/24	10.0.7.1	3	10.0.7.1	0	00:30	R1->R6->R5	false	00:00:00:000
update	00:02:46:818	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:24	R1->R6->R5	false	00:00:00:000
update	00:02:52:847	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:18	R1->R6->R5	false	00:00:00:000
update	00:02:53:794	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:17	R1->R6->R5	false	00:00:00:000
update	00:02:57:365	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:14	R1->R6->R5	false	00:00:00:000
update	00:02:59:183	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:12	R1->R6->R5	false	00:00:00:000
update	00:03:00:311	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:11	R1->R6->R5	false	00:00:00:000
update	00:03:03:382	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:08	R1->R6->R5	false	00:00:00:000
update	00:03:03:830	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:07	R1->R6->R5	false	00:00:00:000
update	00:03:05:197	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:06	R1->R6->R5	false	00:00:00:000
update	00:03:09:412	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:02	R1->R6->R5	false	00:00:00:000
garbage	00:03:10:824	R(n)	10.0.5.0/24	10.0.7.1	0	10.0.7.1	0	0	R1->R6->R5	false	00:00:00:000
update	00:03:20:247	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	00:36	R1->R0->R4	false	00:00:00:000
update	00:03:20:332	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	00:36	R1->R0->R4	false	00:00:00:000
update	00:03:20:672	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	00:35	R1->R0->R4	false	00:00:00:000
update	00:03:26:249	R(n)	10.0.5.0/24	10.0.0.1	3	10.0.0.1	0	00:30	R1->R0->R4	false	00:00:00:000

Abbildung 47: Analysetabelle: RIPMTI im X-Szenario

Berechnung des Simple-Loop-Tests für R1 und Netz 10.0.5.0/24

$$m_{IF1}^{(R1,10.0.5.0/24)} = Route\ über\ R1 \rightarrow R2 \rightarrow R3 \rightarrow R1 \rightarrow R6 \rightarrow R5$$

$$m_{IF4}^{(R2,10.0.5.0/24)} = Route\ über\ R1 \rightarrow R6 \rightarrow R5$$

$$m_{IF1}^{(R1,10.0.5.0/24)} + m_{IF4}^{(R1,10.0.5.0/24)} - 1 \geqslant msilm_{(IF1,IF4)}^{R1}$$

$$6 + 3 - 1 = 8 \geqslant 127(f)$$

$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$

$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Der Spezialfall aus **„X-Szenario2“** wird in den Abbildung 48 und 49 vorgestellt. Anfangs lernt R1 die Route zu Netz 10.0.4.0/24 von R0 mit der Metrik 2. Nach dem Ausfall dieser Route setzt R1 die Erreichbarkeit auf 64 und propagiert diese Information im Netz. Die Falschnachricht von R3 aus dem Zyklus A über die ungültige Erreichbarkeitsinformation wird von R1 blockiert. Der Request-Timer, der aufgrund der Größe des Zyklus 15 Sekunden beträgt, blockiert anschließend alle Updatenachrichten, sodass ein gültiges Update von R6 über eine Alternativ-Route abgelehnt wird und erst nach Ablauf des Request-Timers von R1 angenommen wird. Das zeigt, dass der Careful-RT-Mode zu streng für das X-Szenario arbeitet. Schaltet man in den MTI-Konfigurationseinstellungen „MTI AUTOCONFIG“ auf „Route“, dann schaltet R1 nach ablehnen der Falschnachricht von R3 in den Normal-Mode um. In dem Fall wird die gültige Route von R6 nicht mehr abgelehnt und das Netz kann schneller konvergieren.

Dieses Routerverhalten tritt sehr selten auf und die dadurch bewirkte Verzögerung bewegt sich im Bereich weniger Sekunden. Es soll daher bei der Bestimmung der Konvergenzzeiten keine große Aufmerksamkeit bekommen. Das X-Szenario konvergiert sofort, d.h. Nach Ablauf des Timeout-Timers erkennt der Algorithmus eine Schleife und lässt den Garbage-Collection-Timer ablaufen bis die Route wegen Unerreichbarkeit aus der Routingtabelle entfernt wird. Der Geschwindigkeitsgewinn durch einen beschleunigten Nachrichtenaustausch bewegt sich zwischen 35% bei den 30-Sekunden-Einstellungen und 77% wenn die 3-Sekunden-Einstellungen gewählt werden.

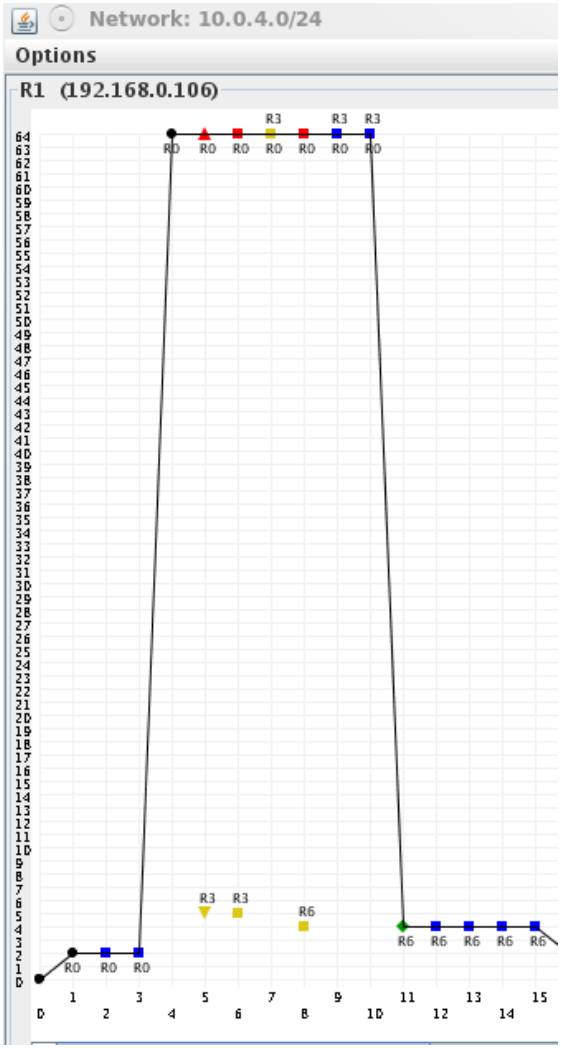
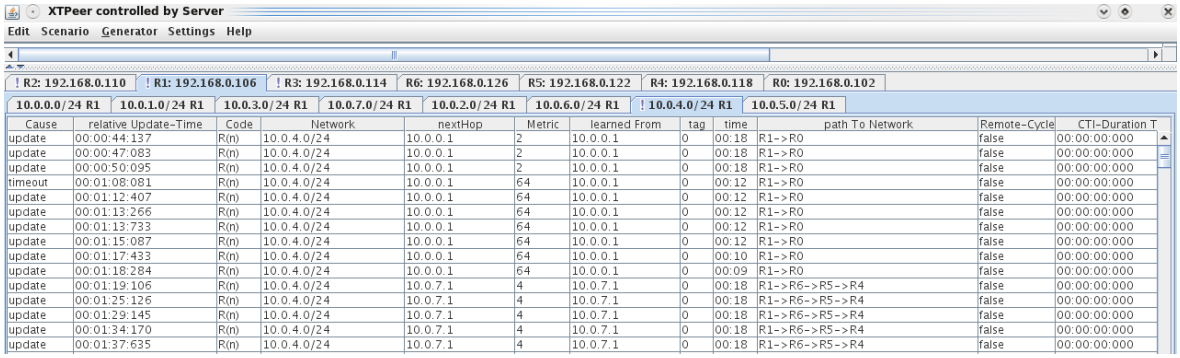


Abbildung 48: X-Szenario: RIPMTI auf R1 über Netz 10.0.4.0/24 nach Ausfall von R0



Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:00:44:137	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:18	R1->R0	false	00:00:00:000
update	00:00:47:083	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:18	R1->R0	false	00:00:00:000
update	00:00:50:095	R(n)	10.0.4.0/24	10.0.0.1	2	10.0.0.1	0	00:18	R1->R0	false	00:00:00:000
timeout	00:01:08:081	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:01:12:407	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:01:13:266	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:01:13:733	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:01:15:087	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:12	R1->R0	false	00:00:00:000
update	00:01:17:433	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:10	R1->R0	false	00:00:00:000
update	00:01:18:284	R(n)	10.0.4.0/24	10.0.0.1	64	10.0.0.1	0	00:09	R1->R0	false	00:00:00:000
update	00:01:19:106	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:01:25:126	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:01:29:145	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:01:34:170	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000
update	00:01:37:635	R(n)	10.0.4.0/24	10.0.7.1	4	10.0.7.1	0	00:18	R1->R6->R5->R4	false	00:00:00:000

Abbildung 49: Analysetabelle X-Szenario: RIPMTI auf R1 über Netz 10.0.4.0/24 nach Ausfall von R0

Berechnung des Simple-Loop-Tests für R1 und Netz 10.0.4.0/24

$m_{IF1}^{(R1,10.0.4.0/24)} = Route\ über\ R1 \rightarrow R2 \rightarrow R3 \rightarrow R1 \rightarrow R0$

$m_{IF3}^{(R1,10.0.4.0/24)} = Route\ über\ R1\ twoard\ R0$

$m_{IF1}^{(R1,10.0.4.0/24)} + m_{IF3}^{(R1,10.0.4.0/24)} - 1 \geqslant msilm_{(IF1,IF3)}^{R1}$

$5 + 2 - 1 = 6 \geqslant 127\,(f)$

$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$

$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$

Zusammenfassung

Der RIPMTI-Algorithmus funktioniert, mit Einschränkung, fehlerfrei. Er erkennt und verhindert erfolgreich den Count-To-Infinity-Effekt und arbeitet, unbeeindruckt von den Timer-Einstellungen, zuverlässig. Einzig der Spezialfall fällt in der Weise auf, dass sich der Algorithmus verzögernd auf die Konvergenz des Netzwerkes auswirken kann. In 100 Durchläufen entstand der Count-To-Infinity-Effekt in 95% der Fälle. Der Geschwindigkeitsgewinn mit beschleunigten Timer-Einstellungen und RIPMTI steigt auf bis zu 58% im direkten Vergleich mit dem RIP-Algorithmus unter Berücksichtigung aller Timer..

6.9 X-Mod

Hierbei handelt es sich um ein modifiziertes X-Szenario, indem im Gegensatz zum X-Szenario nicht zwei Teilnetze über einen Router miteinander verbunden sind, sondern über zwei geswitchte Verbindungsleitungen.

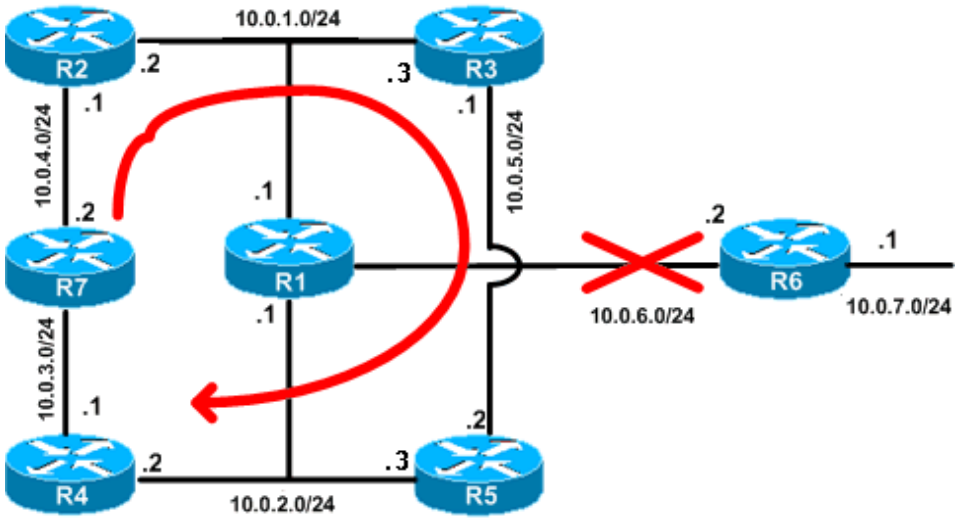


Abbildung 50: Topologie: X-Mod

Modellbeschreibung

Das Netzwerk besteht aus sieben Routern, die derart miteinander verbunden sind, dass drei Zyklen entstehen. Zunächst der äußere Zyklus A über die Router R2, R3, R4, R5 und R7. Zyklus B läuft über R1, R2, R4 und R7 und Zyklus C über R1, R3 und R5. Zu beachten ist, dass Zyklus A um einen Router größer ist als Zyklus B. Als Besonderheit hat dieses Netz zwei geswitchte Netze 10.0.2.0/24 und 10.0.1.0/24, die jeweils drei Router miteinander verbinden, wobei in jedem geswitchten Netz R1 vorhanden ist. R1 spielt hierbei eine zentrale Rolle. Er ist über R6 mit Netz 10.0.7.0/24 verbunden. Über diese Schnittstelle soll der Count-To-Infinity-Effekt erzeugt werden. Die Ausfallsituation soll sich auf alle Zyklen auswirken.

Erzeugung eines Count-To-Infinity-Effekts

Der Count-To-Infinity-Effekt wird durch die Konfigurationsdatei in Anhang I „X-Mod 1“ beschrieben. Nach der Konvergenzphase blockiert R6 auf Interface 1 (10.0.6.2) die Kommunikation mit R1. R1 erkennt dann nach Ablauf des Timeout-Timers den Verlust des Netzes 10.0.7.0/24. R1 propagiert die neue Metrik 64 auf Interface 1 (10.0.1.1) und Interface 2 (10.0.2.1) im gesamten Netz. Da Updatenachrichten per Multicast versendet werden, gelangen sie in den geswitchten Netzen an jeden angeschlossenen Nachbarn. Die neue Information wird also in beide Zyklen B und C gesendet. R1 sendet allerdings nicht synchron mit allen anderen Routern seine Updatenachrichten, sondern wartet kurz. Währenddessen blockieren R2 Interface 2 (10.0.4.1) und R4 Interface 1 (10.0.3.1), damit R7 keine Updatenachrichten erhält. So wird verhindert, dass sich die Metrik 64 durch sofortiges Triggern bis zu R7 ausbreiten kann. R7 wird zum False-Router, der im nächsten periodischen Update seine alte, ungültige Metrik 4 auf beiden Interfaces an R2 bzw. R4 sendet. An dieser Stelle ist die Falschnachricht im Umlauf und kann sich über alle drei Zyklen bewegen.

Ergebnis des RIP-Algorithmus

Betrachtet wird R1, der eine Schlüsselrolle in diesem Netzwerk besitzt. Er kann von R2, R3, R4 und R5 Falschnachrichten erhalten. Es existieren vier Richtungen von möglichen Falschnachrichten-Sendern, die im Zusammenwirken die Konvergenz

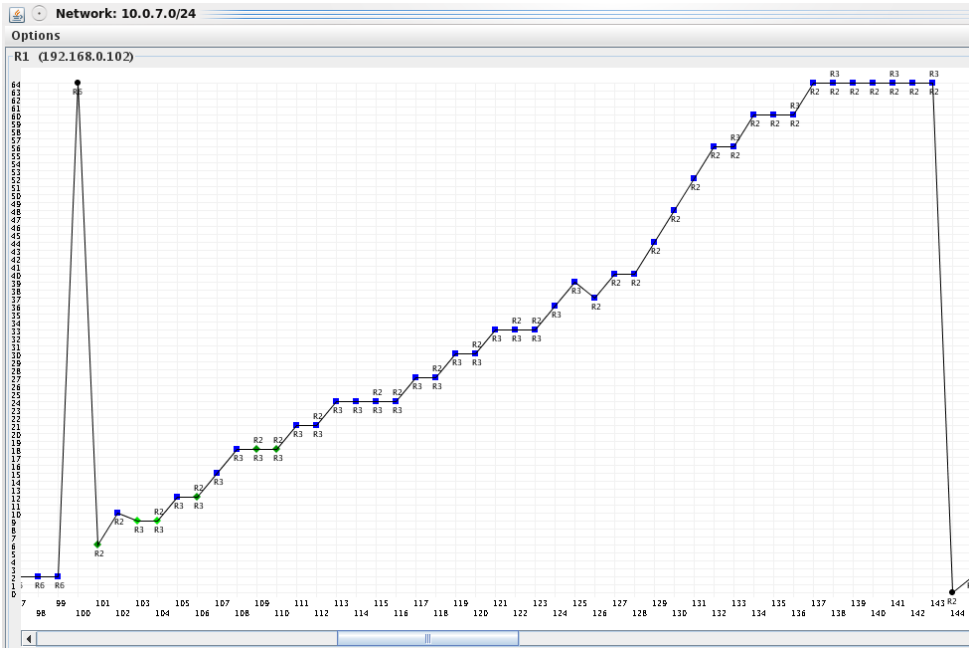
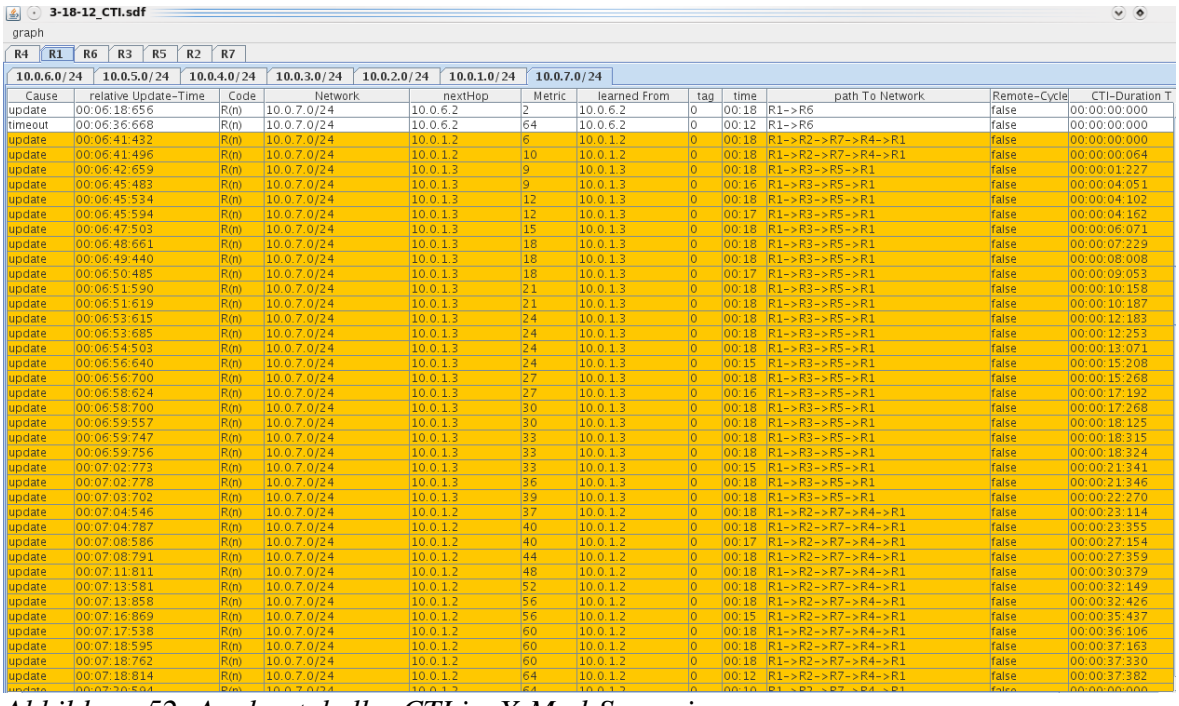


Abbildung 51: Netzgraph: CTI im X-Mod-Szenario

von R1 negativ beeinflussen können. Abbildung 51 und 52 beschreiben diese gegenseitige Beeinflussung.

Zu Beginn kennt R1 zu Netz 10.0.7.0/24 eine Metrik von 2 Hops. Nach Ablauf des Timeout-Timers stellt er die Metrik auf 64 und sendet diese Information weiter. R7, der False-Router, hat bis dahin noch die Metrik 4 zum ausgefallenen Netz über den Weg R7->R4->R1->R6 und aufgrund der Konfigurationsdatei entsteht bei diesem Router die Falschnachricht. R2 und R4 erhalten von R7 jeweils die ungültige Metrik und je nach zeitlicher Verzögerung auf der Verbindungsleitung erhält R1 von einem der beiden Router die Falschnachricht. In diesem Fall lernt R1 von R2 zuerst und die Metrik auf R1 steigt auf 6. Die Falschnachricht von R4 wird ohnehin verworfen, da die in ihr enthaltene Metrik nicht kleiner ist als die bereits eingetragene. Dieser Vorgang beschreibt den Zyklus B, der zweimal durchläuft und folglich die Metrik auf R1 bis auf den Wert 10 hochzählt. R3 und R5 haben ihrerseits jetzt auch diese Falschnachricht erhalten, sodass im Zyklus C ebenfalls der Count-To-Infinity-Effekt entsteht. Irgendwann erreicht R1 aus dieser Richtung eine weitere Falschnachricht über R3. Wie Abbildung 52 zeigt, ist die Metrik über Zyklus C (Wert 9) zufälligerweise kleiner ist als die, die über Zyklus B gelernt wurde. Ursächlich dafür ist die Geschwindigkeit, mit der die Falschnachrichten im jeweiligen Zyklus kursieren. Abhängig von den Triggered Timern jedes Routers und der unterschiedlichen Anzahl der Router in beiden


3-18-12_CTI.sdf

graph

R4 R1 R6 R3 R5 R2 R7

10.0.6.0/24 10.0.5.0/24 10.0.4.0/24 10.0.3.0/24 10.0.2.0/24 10.0.1.0/24 10.0.7.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:06:18:656	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
timeout	00:06:36:668	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:06:41:432	R(n)	10.0.7.0/24	10.0.1.2	6	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:00:000
update	00:06:41:496	R(n)	10.0.7.0/24	10.0.1.2	10	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:00:064
update	00:06:42:659	R(n)	10.0.7.0/24	10.0.1.3	9	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:01:227
update	00:06:45:483	R(n)	10.0.7.0/24	10.0.1.3	9	10.0.1.3	0	00:16	R1->R3->R5->R1	false	00:00:04:051
update	00:06:45:534	R(n)	10.0.7.0/24	10.0.1.3	12	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:04:102
update	00:06:45:594	R(n)	10.0.7.0/24	10.0.1.3	12	10.0.1.3	0	00:17	R1->R3->R5->R1	false	00:00:04:162
update	00:06:47:503	R(n)	10.0.7.0/24	10.0.1.3	15	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:06:071
update	00:06:48:661	R(n)	10.0.7.0/24	10.0.1.3	18	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:07:229
update	00:06:49:440	R(n)	10.0.7.0/24	10.0.1.3	18	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:08:008
update	00:06:50:485	R(n)	10.0.7.0/24	10.0.1.3	18	10.0.1.3	0	00:17	R1->R3->R5->R1	false	00:00:09:053
update	00:06:51:590	R(n)	10.0.7.0/24	10.0.1.3	21	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:10:158
update	00:06:51:619	R(n)	10.0.7.0/24	10.0.1.3	21	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:10:187
update	00:06:53:615	R(n)	10.0.7.0/24	10.0.1.3	24	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:12:183
update	00:06:53:685	R(n)	10.0.7.0/24	10.0.1.3	24	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:12:253
update	00:06:54:503	R(n)	10.0.7.0/24	10.0.1.3	24	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:13:071
update	00:06:56:640	R(n)	10.0.7.0/24	10.0.1.3	24	10.0.1.3	0	00:15	R1->R3->R5->R1	false	00:00:15:208
update	00:06:56:700	R(n)	10.0.7.0/24	10.0.1.3	27	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:15:268
update	00:06:58:624	R(n)	10.0.7.0/24	10.0.1.3	27	10.0.1.3	0	00:16	R1->R3->R5->R1	false	00:00:17:192
update	00:06:58:700	R(n)	10.0.7.0/24	10.0.1.3	30	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:17:268
update	00:06:59:557	R(n)	10.0.7.0/24	10.0.1.3	30	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:18:125
update	00:06:59:747	R(n)	10.0.7.0/24	10.0.1.3	33	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:18:315
update	00:06:59:756	R(n)	10.0.7.0/24	10.0.1.3	33	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:18:324
update	00:07:02:773	R(n)	10.0.7.0/24	10.0.1.3	33	10.0.1.3	0	00:15	R1->R3->R5->R1	false	00:00:21:341
update	00:07:02:778	R(n)	10.0.7.0/24	10.0.1.3	36	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:21:346
update	00:07:03:702	R(n)	10.0.7.0/24	10.0.1.3	39	10.0.1.3	0	00:18	R1->R3->R5->R1	false	00:00:22:270
update	00:07:04:546	R(n)	10.0.7.0/24	10.0.1.2	37	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:23:114
update	00:07:04:787	R(n)	10.0.7.0/24	10.0.1.2	40	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:23:355
update	00:07:08:586	R(n)	10.0.7.0/24	10.0.1.2	40	10.0.1.2	0	00:17	R1->R2->R7->R4->R1	false	00:00:27:154
update	00:07:08:791	R(n)	10.0.7.0/24	10.0.1.2	44	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:27:359
update	00:07:11:811	R(n)	10.0.7.0/24	10.0.1.2	48	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:30:379
update	00:07:13:581	R(n)	10.0.7.0/24	10.0.1.2	52	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:32:149
update	00:07:13:858	R(n)	10.0.7.0/24	10.0.1.2	56	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:32:426
update	00:07:16:869	R(n)	10.0.7.0/24	10.0.1.2	56	10.0.1.2	0	00:15	R1->R2->R7->R4->R1	false	00:00:35:437
update	00:07:17:538	R(n)	10.0.7.0/24	10.0.1.2	60	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:36:106
update	00:07:18:595	R(n)	10.0.7.0/24	10.0.1.2	60	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:37:163
update	00:07:18:762	R(n)	10.0.7.0/24	10.0.1.2	60	10.0.1.2	0	00:18	R1->R2->R7->R4->R1	false	00:00:37:330
update	00:07:18:814	R(n)	10.0.7.0/24	10.0.1.2	64	10.0.1.2	0	00:12	R1->R2->R7->R4->R1	false	00:00:37:382

Abbildung 52: Analysetabelle: CTI im X-Mod-Szenario

Zyklen, können sich die Updatenachrichten gegenseitig stören. Wenn die Nachricht erst einmal über den äußeren Zyklus A wandert und in dieser Zeit mehrmals durch Zyklus B oder C gelaufen ist, überschreiben sich die Falschnachrichten gegenseitig, wenn sie scheinbar kleiner sind. Dadurch wird die Konvergenzzeit negativ beeinflusst. Wenn Falschnachrichten sich gegenseitig überschreiben, zählt der Count-To-Infinity-Effekt auf der einen Seite in unterschiedlichen Intervallen hoch. Weil Zyklus B aus vier Routern und Zyklus C aus drei Routern besteht, nimmt die Metrik auf R1 entweder in 3er- oder 4er-Schritten zu. Andererseits verzögert sich die Konvergenz, wenn in unregelmäßigen Abständen bereits hochgezählte Metrikwerte wieder durch andere, kleinere Falschnachrichten herabgesetzt werden.

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	41,9s
T_1	6s / 36s / 24s	44,6s
T_2	10s / 60s / 40s	57,3s
T_3	30s / 180s / 120s	72,2s

Tabelle 11: Ergebnisse versch. T-Einstellungen beim CTI im X-Mod-Szenario

Die Auswirkungen unterschiedlicher Timer-Einstellungen ähneln denen des X-Szenarios. Die Abweichungen der gemessenen Konvergenzzeiten sind minimal. Das lässt darauf schließen, dass das modifizierte X-Szenario trotz der Verschachtelung der Schleifen keinen besonderen Einfluss hat. Die Konvergenzzeiten nach Entstehen eines Count-To-Infinity-Effekts bewegen sich zwischen 72,2 Sekunden bei **T_3-Einstellungen und 41,9 Sekunden bei T_0**-Einstellungen, was eine Konvergenzzeit beschreibt, die bei **T_0 nur 58% der ursprünglichen Zeit nur entspricht.**

Ergebnis des RIPMTI-Algorithmus

Der RIPMTI-Algorithmus funktioniert zuverlässig. Router für Zyklus B und C hauptverantwortlich für Loops über Netz 10.0.7.0/24 ist, muss auf ihm der RIPMTI-Modus aktiv sein. Er verhindert jede angebotene Alternativroute, die in einem Zyklus über ihn selbst eintrifft. Da der äußere Zyklus A Router R1 umgeht, könnte sich dort der Count-To-Infinity-Effekt ausbreiten. Auf den Routern in Zyklus A sollte daher auch RIPMTI eingesetzt werden. In verschiedenen Versuchen erreicht R3 die Falschnachricht von R2 über Interface 1. Über das gleiche Interface, über das R3 kurz zuvor RIP_Infinity von R1 gelernt hat. R3 erkennt, dass diese Nachricht aber von einer anderen IP und somit aus einer Schleife stammen muss und verwirft die angebotene Route. Der gleiche Effekt kann aufgrund der Symmetrie des Szenarios zwischen R4 und R5 eintreten. In allen Fällen wird der Count-To-Infinity-Effekt verhindert. Der angepasste RT-Timer blockiert ungültige Routen lange genug, damit sich RIP_Infinity im Netzwerk durchsetzen kann. Abbildung 53 und 54 zeigen, dass RIPMTI Falschnachrichten aus dem äußeren Zyklus erkennt.

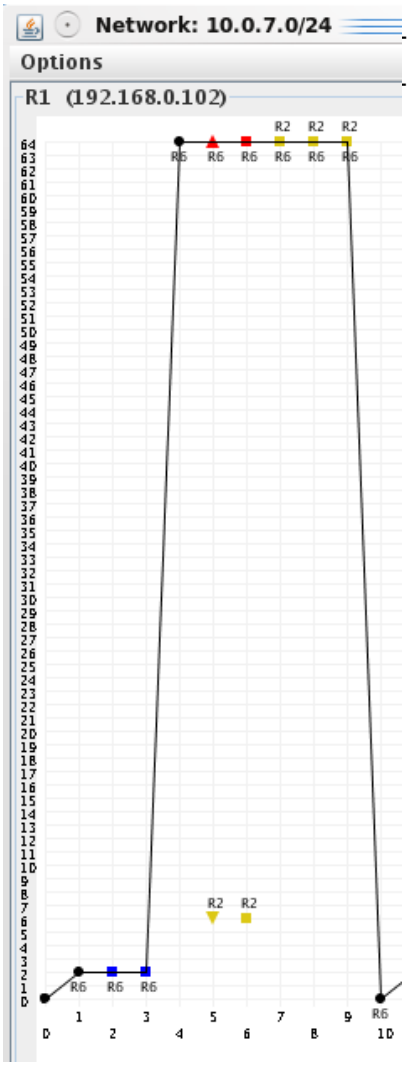
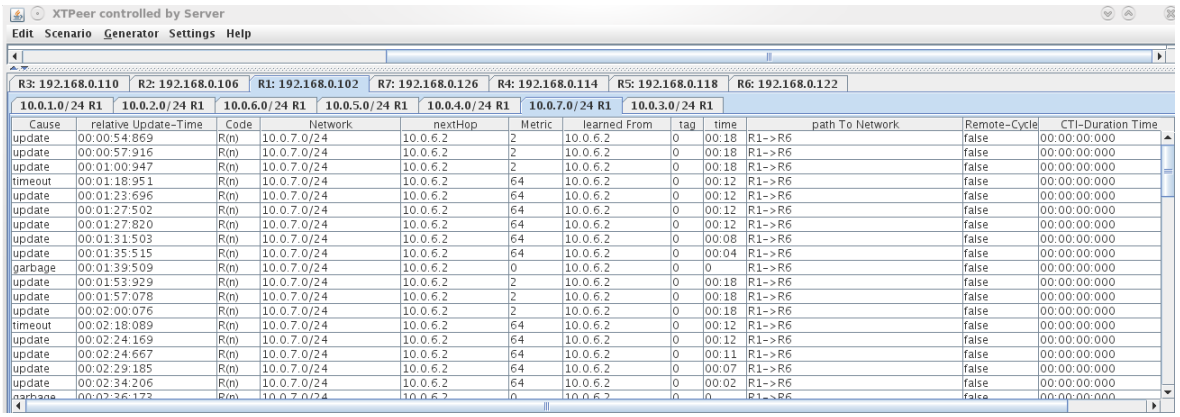


Abbildung 53: Netzgraph: RIPMTI im X-Mod-Szenario



Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration Time
update	00:00:54:869	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
update	00:00:57:916	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
update	00:01:00:947	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
timeout	00:01:18:951	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:01:23:696	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:01:27:502	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:01:27:820	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:01:31:503	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:08	R1->R6	false	00:00:00:000
update	00:01:35:515	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:04	R1->R6	false	00:00:00:000
garbage	00:01:39:509	R(n)	10.0.7.0/24	10.0.6.2	0	10.0.6.2	0	0	R1->R6	false	00:00:00:000
update	00:01:53:929	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
update	00:01:57:078	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
update	00:02:00:076	R(n)	10.0.7.0/24	10.0.6.2	2	10.0.6.2	0	00:18	R1->R6	false	00:00:00:000
timeout	00:02:18:089	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:02:24:169	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:12	R1->R6	false	00:00:00:000
update	00:02:24:667	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:11	R1->R6	false	00:00:00:000
update	00:02:29:185	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:07	R1->R6	false	00:00:00:000
update	00:02:34:206	R(n)	10.0.7.0/24	10.0.6.2	64	10.0.6.2	0	00:02	R1->R6	false	00:00:00:000

Abbildung 54: Analysetabelle: RIPMTI im X-Mod-Szenario

Folgende Berechnung des Simple-Loop-Tests läuft auf R1 und Netz 10.0.7.0/24:

$$m_{IF1}^{(R1,10.0.7.0/24)} = Route\ über\ R1 \rightarrow R3 \rightarrow R5 \rightarrow R4 \rightarrow R7 \rightarrow R2 \rightarrow R1 \rightarrow R6$$
$$m_{IF3}^{(R1,10.0.7.0/24)} = Route\ über\ R1 \rightarrow R6$$
$$m_{IF1}^{(R1,10.0.7.0/24)} + m_{IF3}^{(R1,10.0.7.0/24)} - 1 \geqslant msilm_{(IF1,IF3)}^{R1}$$
$$8 + 2 - 1 = 9 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Folgende Berechnung des Simple-Loop-Tests anhand der IP läuft auf R3 zu Netz 10.0.7.0/24 :

$$m_{10.0.1.2}^{(R3,10.0.7.0/24)} = Route\ über\ R3 \rightarrow R2 \rightarrow R7 \rightarrow R4 \rightarrow R1 \rightarrow R6$$
$$m_{10.0.1.1}^{(R3,10.0.7.0/24)} = Route\ über\ R3 \rightarrow R1 \rightarrow R6$$
$$m_{10.0.1.2}^{(R3,10.0.7.0/24)} + m_{10.0.1.1}^{(R3,10.0.7.0/24)} - 1 \geqslant msilm_{(10.0.1.1,10.0.1.2)}^{R3}$$
$$6 + 3 - 1 = 8 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Anzumerken ist, dass bei der Berechnung Interfaces mit IP-Adressen im Algorithmus ausgetauscht werden. Die Implementierung richtet sich nach den Positionsindizes der MRPM-Tabelle, sodass sich sonst nichts am Algorithmus ändert.

Zusammenfassung

RIPMTI funktioniert im X-Mod-Szenario. Die Konvergenzzeiten nach dem Entstehen eines Count-To-Infinity-Effekts werden durch beschleunigte Timer-Einstellungen verbessert. Ebenso reagiert RIPMTI. Schnellere Timer-Einstellungen ergeben eine bis zu 70%-ige Beschleunigung der Konvergenz des Netzwerkes.

6.10 Big-Net

Das Big-Net stellt ein komplexeres Szenario dar, indem unterschiedliche Szenarien in einem einzigen zusammengefasst werden. Das Big-Net-Szenario soll die Effektivität und Effizienz von RIP und RIPMTI in einem Szenario zeigen, das beliebig aus den bereits vorgestellten Szenarien kombiniert wird. Es sind also theoretisch unzählige Konstellationen vorstellbar. Das vorhandenen Big-Net beschränkt sich auf eine dieser Kombinationsmöglichkeiten.

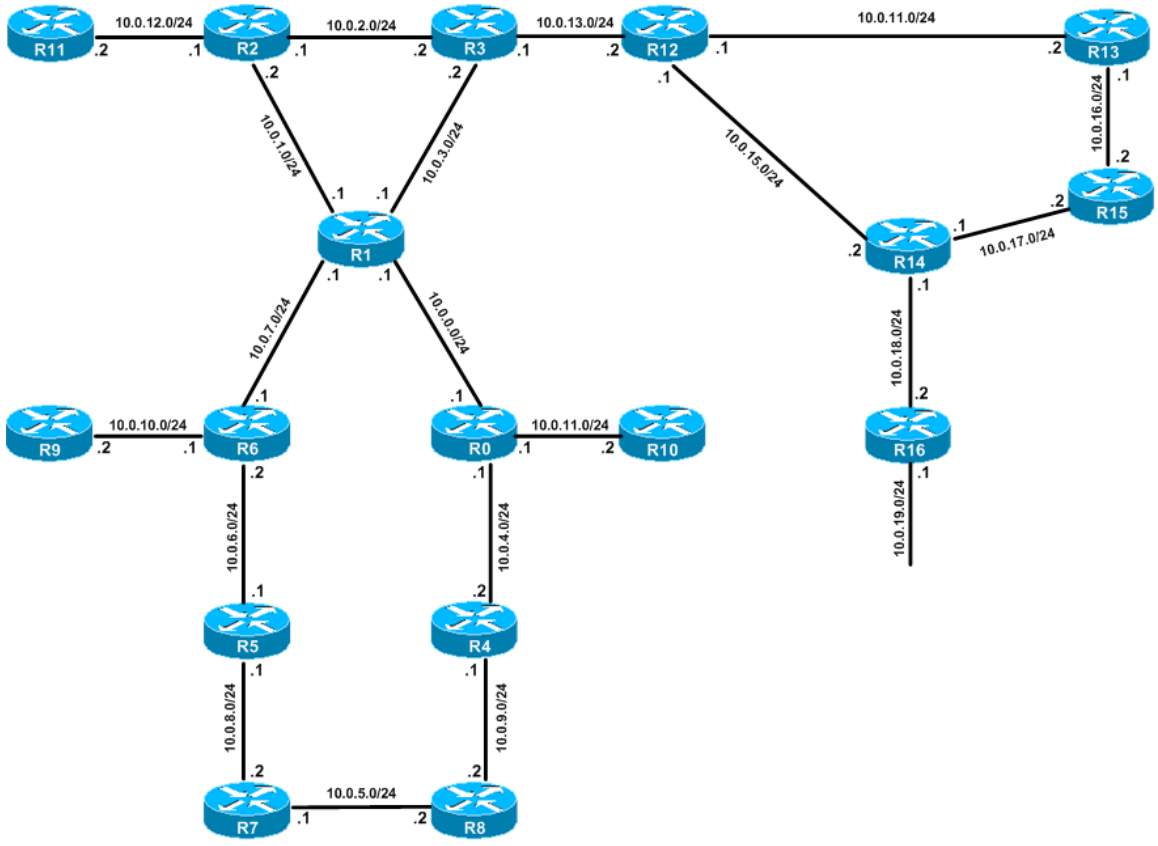


Abbildung 55: Topologie: Big-Net

Modellbeschreibung

Das Netzwerk besteht links aus einer X-Topologie, die auf einer Seite erweitert wurde, und einer Y-Topologie auf der rechten Seite. Der obere Zyklus der X-Topologie wurde über eine Verbindung von R3 mit R12 aus dem Zyklus der Y-Topologie zusammengefügt. Zusätzlich wurde das Netz um weitere Router an beliebigen Stellen (R9, R10, R11) ergänzt.

Erzeugung eines Count-To-Infinity-Effekts

Die Konfigurationsdatei aus Anhang J **„Big-Net1“** lässt die Verbindung zwischen R5 und R7 als auch R4 und R8 ausfallen. Nachdem das Netz konvergiert ist, blockiert R7 Updatenachrichten, die den Router auf seinem Interface 1 (10.0.8.2) verlassen wollen. R8 blockiert ausgehende Updatenachrichten auf seinem Interface 1 (10.0.9.2). Nach Ablauf des Erreichbarkeits-Timeout auf R4 und R5 bewerten diese das Netz 10.0.5.0/24 mit der Metrik 64 als unerreichbar. Die neue Information wird anschließend im Netz verbreitet. Eine zentrale Rolle bei der Entstehung des Count-To-Infinity-Effekts spielen R1, R2 und R3. R1 empfängt die neue Updatenachricht mit der Information über den Verlust des Netzes 10.0.5.0/24 und verbreitet die Nachricht auf seinen drei übrigen Interfaces. Unter Berücksichtigung des Split-Horizon-Mechanismus kann das zweierlei Bedeutung haben. Je nachdem von welchem Router (R0 oder R6) R1 lernt, schickt er die Updatenachricht entsprechend nicht an denjenigen, von dem er gelernt hat zurück. Das hat zwar keinen Einfluss auf den Count-To-Infinity-Effekt im oberen Zyklus, kann aber Auswirkungen auf weitere Netze haben, die zum Beispiel mit R0 bzw. R6 verbunden sind oder sein könnten. Um den Count-To-Infinity-Effekt zu provozieren betrachten wir nun den oberen Zyklus. R1 informiert R2 auf Interface 1 (10.0.1.1) über die Unerreichbarkeit des verloren gegangenen Netzes und blockiert gleichzeitig Interface 2 (10.0.3.1). R3 bekommt die Information nicht. Daraufhin sendet R3 auf Interface 1 (10.0.2.2) an R2 seine veraltete, ungültige Metrik. R2 bewertet diese Updatenachricht als gültig und aktualisiert seine Routingtabelle. Das Netz wird mittlerweile nicht mehr von der Konfigurationsdatei gesteuert, sondern befindet sich im Zustand des automatischen Routings und verwaltet sich selbst. R2 sendet auf Interface 1 (10.0.1.2) an R1 die ungültige Metrik und R1 sendet seinerseits auf Interface 2 an R3 die ungültige Metrik. Damit ist der Count-To-Infinity-Effekt erzeugt worden und durchläuft den Zyklus bis zum Metrikwert 64.

Die Konfigurationsdatei aus Anhang J **„Big-Net2“** lässt zusätzlich zum Netz 10.0.5.0/24 auch noch das Netz 10.0.19.0/24 ausfallen. Der Count-To-Infinity-Effekt in der X-Topologie entsteht genauso wie bei **„Big-Net1“**. Dazu kommt jetzt, dass R16 Updatenachrichten blockiert, die den Router auf Interface 1 (10.0.18.2) verlassen wollen, um den Verlust des darunter liegenden Netzes zu simulieren. Der Count-To-Infinity-Effekt wird in der rechten Y-Topologie erzeugt. Nachdem R14 festgestellt hat, dass das Netz 10.0.19.0/24 nicht mehr erreichbar ist,

sendet er ein Update über Interface 2 (10.0.17.1) an R15 und blockiert seine Verbindung zu R12. R15 und R13 geben bis dahin noch keine Updatenachrichten in Richtung R12 weiter. R12 wird somit zum False-Router , der auf Interface 1 (10.0.13.2) und Interface 2 (10.0.14.1) die alte, ungültige Metrik an R3 und R13 sendet. R13 gibt diese Information auf Interface 1 (10.0.16.1) an R15 weiter und dieser wiederum über sein Interface 2 (10.0.17.2) an R14. Da alle Router mittlerweile im automatischen Routingmodus laufen, sendet R14 wieder Updatenachrichten an R12, womit der Zyklus einmal durchlaufen ist und der Count-To-Infinity-Effekt eingetreten ist.

Ergebnis des RIPMTI-Algorithmus

Im ersten Teil wird beschrieben, wie sich das Netzwerk verhält, wenn **„Big-Net1“** benutzt wird. Der zweite Teil beschreibt die zusätzlichen Auswirkungen aus **„Big-Net2“**.

Wenn alle Router im Netzwerk im „Listen“-Modus laufen, reagieren sie auf Updatenachrichten wie das alte RIP-Protokoll. R1 ist in diesem Fall der Source-Router, über den der Zyklus mit dem ausgefallenen Netz erreichbar ist. R3 ist der False-Router, der die alte, ungültige Metrik weitergibt. Der CTI entsteht und breitet sich im gesamten Netzwerk aus. Die Abbildung 56 zeigt graphisch, wie sich der Count-To-Infinity-Effekt auf R1 entwickelt. Abbildung 57 stellt die ausgewerteten Daten entsprechend zum Graphen ausführlich dar.

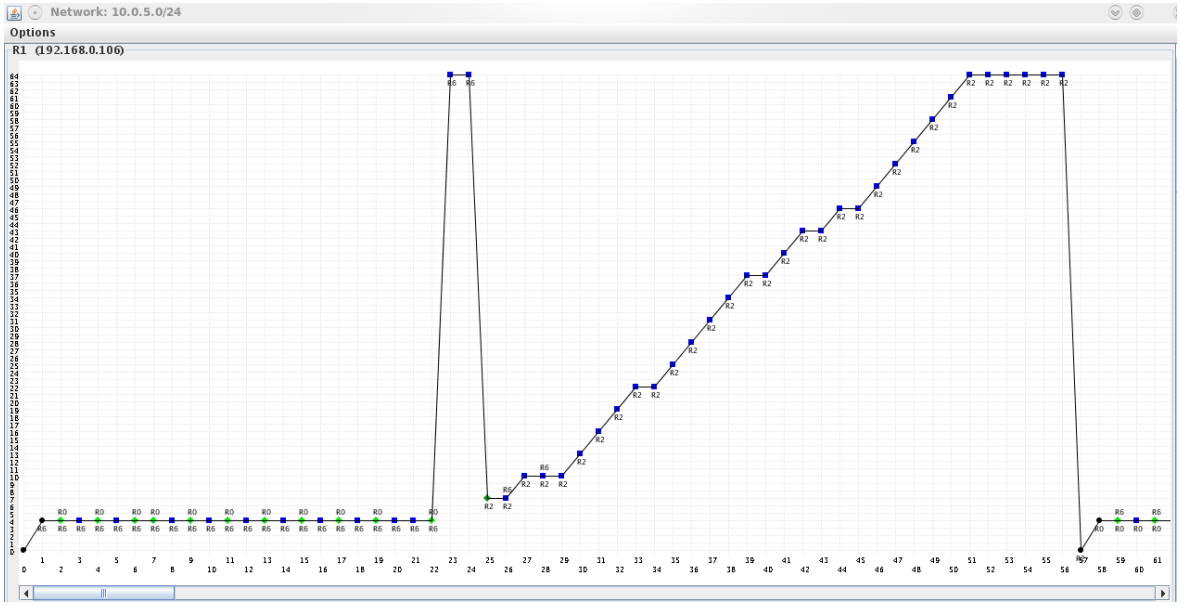
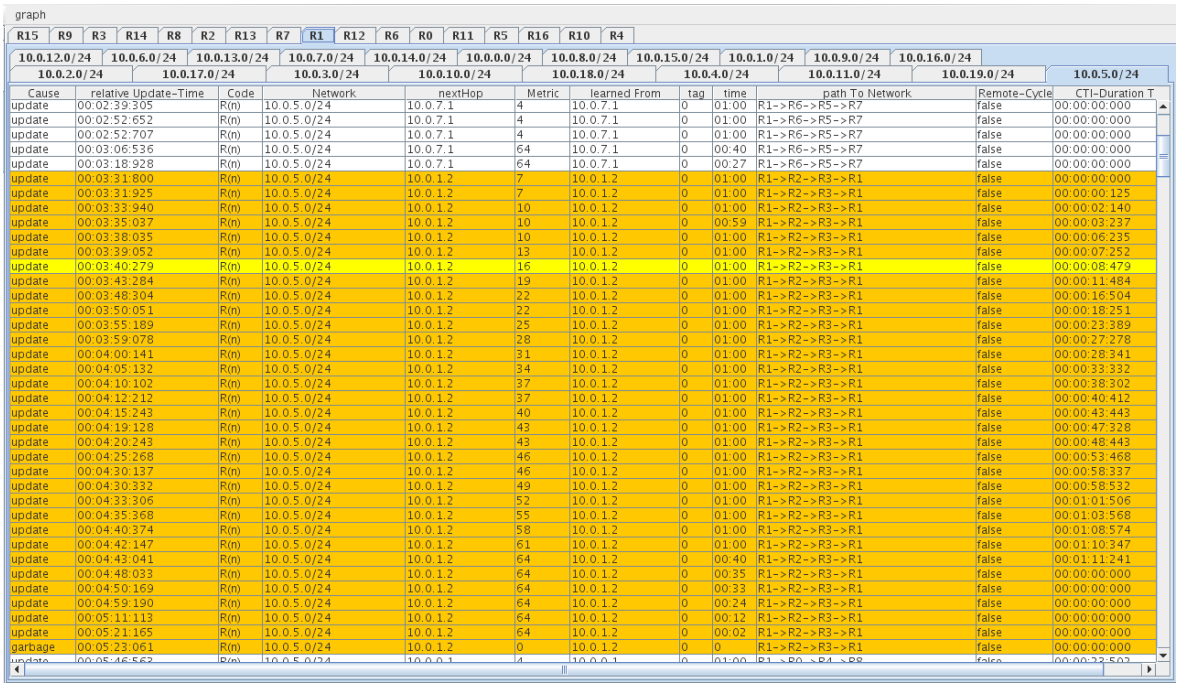


Abbildung 56: Netzgraph: CTI im Big-Net-Szenario auf R1

R1 befindet sich anfangs in einem konsistenten Zustand. Die Entfernung zum Netz 10.0.5.0/24 hat die Metrik 4. Nach Ablauf des Erreichbarkeits-Timeout setzt R1 die Metrik zum verloren gegangenen Netz auf 64. Kurz darauf erreicht R1 die Falschnachricht über R2 und aktualisiert seine Routingtabelle auf die Metrik 7. Dieser Wert ergibt sich aus seiner direkten Entfernung (Metrik 4) zum Zielnetz addiert mit dem ersten Schleifendurchlauf (Umfang der Schleife = 3). Im Folgenden steigt die Metrik regelmäßig um den Wert des Schleifenumfangs und endet mit dem Wert 64. Der Garbage-Collection-Timer läuft ab und die Route zum ausgefallenen Netz wird aus der Routingtabelle entfernt.


graph

R15 R9 R3 R14 R8 R2 R13 R7 R1 R12 R6 R0 R11 R5 R16 R10 R4

10.0.12.0/24 10.0.6.0/24 10.0.13.0/24 10.0.7.0/24 10.0.14.0/24 10.0.0.0/24 10.0.8.0/24 10.0.15.0/24 10.0.1.0/24 10.0.9.0/24 10.0.16.0/24

10.0.2.0/24 10.0.17.0/24 10.0.3.0/24 10.0.10.0/24 10.0.18.0/24 10.0.4.0/24 10.0.11.0/24 10.0.19.0/24 10.0.5.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:02:39:305	R(n)	10.0.5.0/24	10.0.7.1	4	10.0.7.1	0	01:00	R1->R6->R5->R7	false	00:00:00:000
update	00:02:52:652	R(n)	10.0.5.0/24	10.0.7.1	4	10.0.7.1	0	01:00	R1->R6->R5->R7	false	00:00:00:000
update	00:02:52:707	R(n)	10.0.5.0/24	10.0.7.1	4	10.0.7.1	0	01:00	R1->R6->R5->R7	false	00:00:00:000
update	00:03:06:536	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:40	R1->R6->R5->R7	false	00:00:00:000
update	00:03:18:928	R(n)	10.0.5.0/24	10.0.7.1	64	10.0.7.1	0	00:27	R1->R6->R5->R7	false	00:00:00:000
update	00:03:31:800	R(n)	10.0.5.0/24	10.0.1.2	7	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:00:000
update	00:03:31:925	R(n)	10.0.5.0/24	10.0.1.2	7	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:00:125
update	00:03:33:940	R(n)	10.0.5.0/24	10.0.1.2	10	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:02:140
update	00:03:35:037	R(n)	10.0.5.0/24	10.0.1.2	10	10.0.1.2	0	00:59	R1->R2->R3->R1	false	00:00:03:237
update	00:03:38:035	R(n)	10.0.5.0/24	10.0.1.2	10	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:06:235
update	00:03:39:052	R(n)	10.0.5.0/24	10.0.1.2	13	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:07:252
update	00:03:40:279	R(n)	10.0.5.0/24	10.0.1.2	16	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:08:479
update	00:03:43:284	R(n)	10.0.5.0/24	10.0.1.2	19	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:11:484
update	00:03:48:304	R(n)	10.0.5.0/24	10.0.1.2	22	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:16:504
update	00:03:50:051	R(n)	10.0.5.0/24	10.0.1.2	22	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:18:251
update	00:03:55:189	R(n)	10.0.5.0/24	10.0.1.2	25	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:23:389
update	00:03:59:078	R(n)	10.0.5.0/24	10.0.1.2	28	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:27:278
update	00:04:00:141	R(n)	10.0.5.0/24	10.0.1.2	31	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:28:341
update	00:04:05:132	R(n)	10.0.5.0/24	10.0.1.2	34	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:33:332
update	00:04:10:102	R(n)	10.0.5.0/24	10.0.1.2	37	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:38:302
update	00:04:12:212	R(n)	10.0.5.0/24	10.0.1.2	37	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:40:412
update	00:04:15:243	R(n)	10.0.5.0/24	10.0.1.2	40	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:43:443
update	00:04:19:128	R(n)	10.0.5.0/24	10.0.1.2	43	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:47:328
update	00:04:20:243	R(n)	10.0.5.0/24	10.0.1.2	43	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:48:443
update	00:04:25:268	R(n)	10.0.5.0/24	10.0.1.2	46	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:53:468
update	00:04:30:137	R(n)	10.0.5.0/24	10.0.1.2	46	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:58:337
update	00:04:30:332	R(n)	10.0.5.0/24	10.0.1.2	49	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:00:58:532
update	00:04:33:306	R(n)	10.0.5.0/24	10.0.1.2	52	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:01:01:506
update	00:04:35:368	R(n)	10.0.5.0/24	10.0.1.2	55	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:01:03:568
update	00:04:40:374	R(n)	10.0.5.0/24	10.0.1.2	58	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:01:08:574
update	00:04:42:147	R(n)	10.0.5.0/24	10.0.1.2	61	10.0.1.2	0	01:00	R1->R2->R3->R1	false	00:01:10:347
update	00:04:43:041	R(n)	10.0.5.0/24	10.0.1.2	64	10.0.1.2	0	00:40	R1->R2->R3->R1	false	00:01:11:241
update	00:04:48:033	R(n)	10.0.5.0/24	10.0.1.2	64	10.0.1.2	0	00:35	R1->R2->R3->R1	false	00:00:00:000
update	00:04:50:169	R(n)	10.0.5.0/24	10.0.1.2	64	10.0.1.2	0	00:33	R1->R2->R3->R1	false	00:00:00:000
update	00:04:59:190	R(n)	10.0.5.0/24	10.0.1.2	64	10.0.1.2	0	00:24	R1->R2->R3->R1	false	00:00:00:000
update	00:05:11:113	R(n)	10.0.5.0/24	10.0.1.2	64	10.0.1.2	0	00:12	R1->R2->R3->R1	false	00:00:00:000
update	00:05:21:165	R(n)	10.0.5.0/24	10.0.1.2	64	10.0.1.2	0	00:02	R1->R2->R3->R1	false	00:00:00:000
garbage	00:05:23:061	R(n)	10.0.5.0/24	10.0.1.2	0	10.0.1.2	0	0	R1->R2->R3->R1	false	00:00:00:000

Abbildung 57: Analysetabelle: CTI im Big-Net-Szenario auf R1

Obwohl der Zyklus nur einen Bruchteil des gesamten Netzwerks ausmacht, wirkt sich der Count-To-Infinity-Effekt auf sämtliche Router im Netzwerk aus. Abbildung 58 und 59 und 4 zeigen R16 als Metrikgraph und Tabelle. R16 ist vom ausgefallenen Netz 10.0.5.0/24 am weitesten entfernt (Metrik 8) und erhält Updatenachrichten ausschließlich über R14. Aufgrund der Verbindung über R12 mit dem Zyklus, erhält R16 das Falschupdate von R3. R16 ist 4 Hops vom Zyklus entfernt und erhält unregelmäßig Updatenachrichten. Das verursacht die 3er- oder 6er-Sprünge im Metrikgraphen. Der Triggered-Timer auf jedem Router kann das Weiterleiten eines Updates um bis zu 5 Sekunden verzögern. Je weiter ein Router zum Zyklus entfernt ist, desto größer ist die Wahrscheinlichkeit, dass auf

irgendeinem Router ungünstig der Triggered-Timer reinitialisiert wurde und desto weniger Updates erreichen ihn. Router dazwischen sammeln Updates, die kurz nacheinander eintreffen, und warten einige Sekunden bis zur Weiterleitung. Daher kann der Count-To-Infinity-Effekt in Vielfachen des Schleifenumfangs des Zyklus hochzählen. Der Triggered-Timer ist außerdem dafür verantwortlich, dass R14 nicht über die Metrik 64 informiert wird. Irgendein Router auf dem Weg zu R14 hat seine Metrik 64 bereits mit einer kurz dahinter folgenden Update-Nachricht mit der niedrigeren ungültigen Metrik aktualisiert.

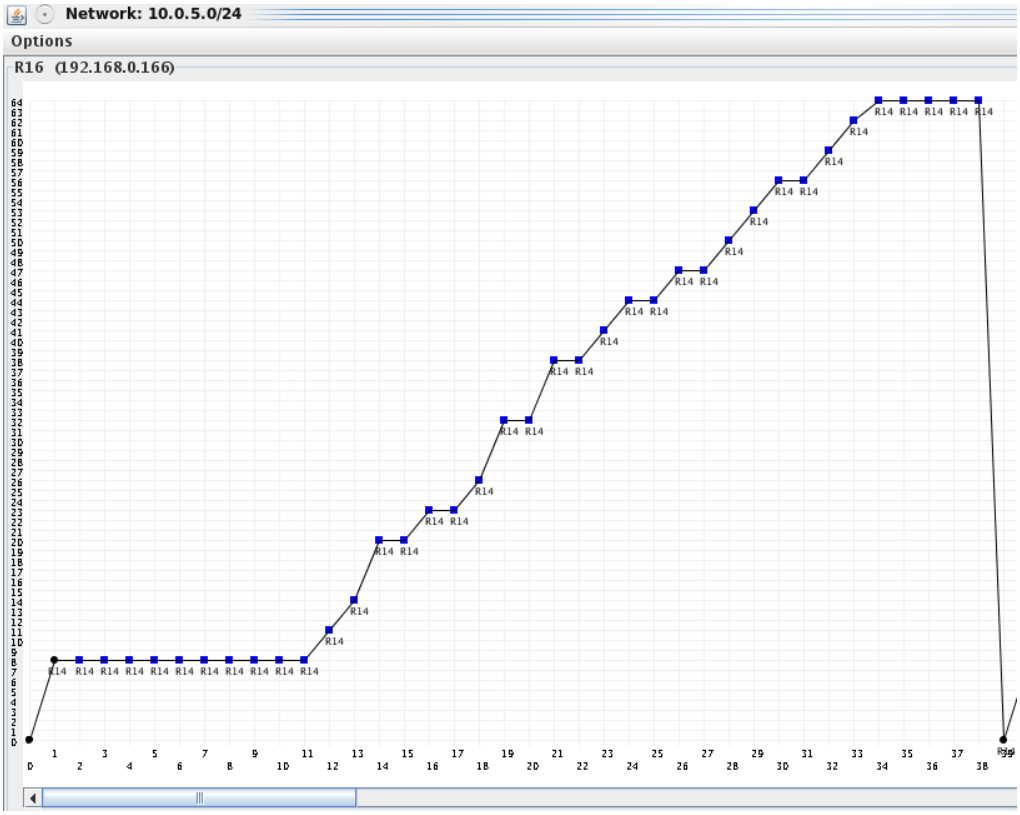


Abbildung 58: Netzwerkgraph: CTI im Big-Net-Szenario auf R16

graph

R15 R9 R3 R14 R8 R2 R13 R7 R1 R12 R6 R0 R11 R5 R16 R10 R4

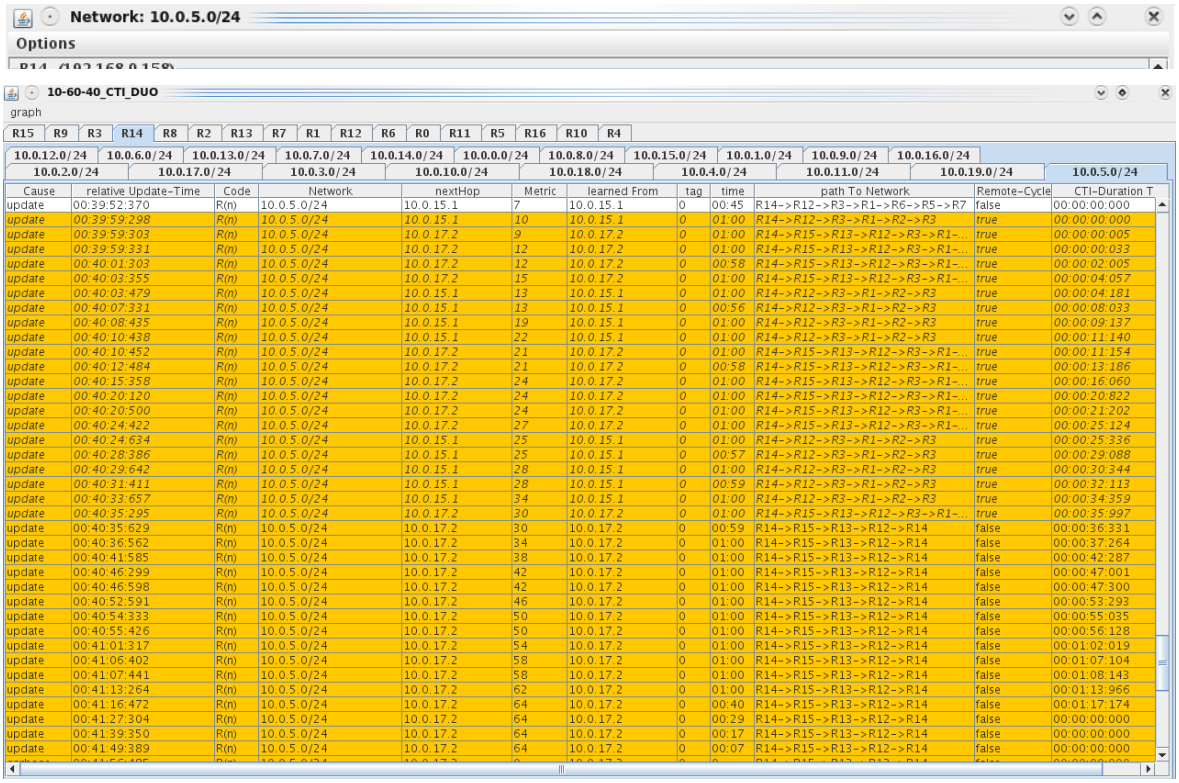
10.0.12.0/24 10.0.6.0/24 10.0.13.0/24 10.0.7.0/24 10.0.14.0/24 10.0.0.0/24 10.0.8.0/24 10.0.15.0/24 10.0.1.0/24 10.0.9.0/24 10.0.16.0/24
10.0.2.0/24 10.0.17.0/24 10.0.3.0/24 10.0.10.0/24 10.0.18.0/24 10.0.4.0/24 10.0.11.0/24 10.0.19.0/24 10.0.5.0/24

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:01:35:638	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:01:47:496	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:02:01:511	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:02:14:703	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:02:28:404	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:02:39:310	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:02:52:494	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:03:06:385	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:03:18:949	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:03:31:792	R(n)	10.0.5.0/24	10.0.18.1	8	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R6->...	false	00:00:00:000
update	00:03:36:005	R(n)	10.0.5.0/24	10.0.18.1	11	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:00:000
update	00:03:42:192	R(n)	10.0.5.0/24	10.0.18.1	14	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:06:187
update	00:03:43:190	R(n)	10.0.5.0/24	10.0.18.1	20	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:07:185
update	00:03:45:032	R(n)	10.0.5.0/24	10.0.18.1	20	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:09:027
update	00:03:47:186	R(n)	10.0.5.0/24	10.0.18.1	23	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:11:181
update	00:03:54:086	R(n)	10.0.5.0/24	10.0.18.1	23	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:18:081
update	00:03:55:249	R(n)	10.0.5.0/24	10.0.18.1	26	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:19:244
update	00:04:00:258	R(n)	10.0.5.0/24	10.0.18.1	32	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:24:253
update	00:04:03:107	R(n)	10.0.5.0/24	10.0.18.1	32	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:27:102
update	00:04:06:208	R(n)	10.0.5.0/24	10.0.18.1	38	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:30:203
update	00:04:11:109	R(n)	10.0.5.0/24	10.0.18.1	38	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:35:104
update	00:04:14:252	R(n)	10.0.5.0/24	10.0.18.1	41	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:38:247
update	00:04:20:264	R(n)	10.0.5.0/24	10.0.18.1	44	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:44:259
update	00:04:21:125	R(n)	10.0.5.0/24	10.0.18.1	44	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:45:120
update	00:04:26:326	R(n)	10.0.5.0/24	10.0.18.1	47	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:50:321
update	00:04:29:174	R(n)	10.0.5.0/24	10.0.18.1	47	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:53:169
update	00:04:31:263	R(n)	10.0.5.0/24	10.0.18.1	50	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:55:258
update	00:04:34:279	R(n)	10.0.5.0/24	10.0.18.1	53	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:00:58:274
update	00:04:36:296	R(n)	10.0.5.0/24	10.0.18.1	56	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:01:00:291
update	00:04:37:177	R(n)	10.0.5.0/24	10.0.18.1	56	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:01:01:172
update	00:04:40:419	R(n)	10.0.5.0/24	10.0.18.1	59	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:01:04:414
update	00:04:41:431	R(n)	10.0.5.0/24	10.0.18.1	62	10.0.18.1	0	01:00	R16->R14->R12->R3->R1->R2->...	true	00:01:05:426
update	00:04:46:415	R(n)	10.0.5.0/24	10.0.18.1	64	10.0.18.1	0	00:40	R16->R14->R12->R3->R1->R2->...	true	00:01:10:410
update	00:04:47:201	R(n)	10.0.5.0/24	10.0.18.1	64	10.0.18.1	0	00:39	R16->R14->R12->R3->R1->R2->...	true	00:00:00:000
update	00:04:57:238	R(n)	10.0.5.0/24	10.0.18.1	64	10.0.18.1	0	00:29	R16->R14->R12->R3->R1->R2->...	true	00:00:00:000
update	00:05:06:262	R(n)	10.0.5.0/24	10.0.18.1	64	10.0.18.1	0	00:20	R16->R14->R12->R3->R1->R2->...	true	00:00:00:000
update	00:05:18:302	R(n)	10.0.5.0/24	10.0.18.1	64	10.0.18.1	0	00:08	R16->R14->R12->R3->R1->R2->...	true	00:00:00:000
garbage	00:05:26:426	R(n)	10.0.5.0/24	10.0.18.1	0	10.0.18.1	0	0	R16->R14->R12->R3->R1->R2->...	true	00:00:00:000

Abbildung 59: Analysetabelle: CTI im Big-Net-Szenario auf R16

Im nächsten Schritt soll jetzt zusätzlich in der Y-Topologie auf der rechten Seite des Netzwerkes ein Count-To-Infinity-Effekt erzeugt werden. **„Big-Net2“** konfiguriert den Nachrichtenaustausch derart, dass R14 der Source-Router zum ausgefallenen Netz 10.0.19.0/24 wird, R12 wird zum False-Router deklariert und sendet über R13 und R15 sein Falsch-Update an R14, der dieses als gültig annimmt und in seine Routingtabelle einträgt. Zunächst einmal breitet sich diese Falschnachricht genauso im gesamten Netzwerk aus wie es auch schon **„Big-Net1“** verursacht hat. R12 sendet seine Falschnachricht an R3 und dieser verbreitet die ungültige Metrik im linken Teilnetz. Die Konfigurationen aus „Big-Net1“ und „Big-Net2“ verhalten sich demnach analog zueinander. Allerdings entsteht durch die Verbindung zweier möglicher Count-To-Infinity-Kandidaten ein gegenseitiger Beeinflussungseffekt. Als Beispiel soll hier R14 und das Netz 10.0.5.0/24 dienen. Abbildung 60 und 61 zeigen die entsprechenden Graphen und Tabellen. Im konvergenten Zustand führt die Router über R12 mit der Metrik 7. Die Route über R15 wird nicht angenommen, da sie eine Metrik von 9 besitzt und somit länger ist. Ein Count-To-Infinity-Effekt im linken oberen Zyklus führt dazu, dass auf R14 die Metrik in 3er- bzw. 6er-Schritten von 7 auf 10, 13, 19, 22, 25 bis zur 64 hochzählt. Entsteht nun gleichzeitig im rechten oberen Zyklus ein Count-To-Infinity-Effekt, dann bekommt R14 von R15 die Metrik 9, obwohl von R12 bereits die höhere Metrik 10 gelernt wurde. Im weiteren Verlauf lernt R14 dann über R15 die Metrik

9, 12, 15, 21, 24 bis zur 64. Auch hier sind 3er- bzw. 6er-Schritte vorhanden. Im Big-Net-Szenario treten beide Zählschritte kombiniert auf und Update-Nachrichten von R12 und R15 erreichen, wenn auch unregelmäßig, abwechselnd R14, sodass der Count-To-Infinity-Effekt nicht konstant hochzählt, sondern häufig durch niedrigere Metrik-Werte heruntergesetzt wird.



Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From	tag	time	path To Network	Remote-Cycle	CTI-Duration T
update	00:39:52:370	R(n)	10.0.5.0/24	10.0.15.1	7	10.0.15.1	0	00:45	R14->R12->R3->R1->R6->R5->R7	false	00:00:00:000
update	00:39:59:298	R(n)	10.0.5.0/24	10.0.15.1	10	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:00:000
update	00:39:59:303	R(n)	10.0.5.0/24	10.0.17.2	9	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:00:005
update	00:39:59:331	R(n)	10.0.5.0/24	10.0.17.2	12	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:00:033
update	00:40:01:303	R(n)	10.0.5.0/24	10.0.17.2	12	10.0.17.2	0	00:58	R14->R15->R13->R12->R3->R1-...	true	00:00:02:005
update	00:40:03:355	R(n)	10.0.5.0/24	10.0.17.2	15	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:04:057
update	00:40:03:479	R(n)	10.0.5.0/24	10.0.15.1	13	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:04:181
update	00:40:07:331	R(n)	10.0.5.0/24	10.0.15.1	13	10.0.15.1	0	00:56	R14->R12->R3->R1->R2->R3	true	00:00:08:033
update	00:40:08:435	R(n)	10.0.5.0/24	10.0.15.1	19	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:09:137
update	00:40:10:438	R(n)	10.0.5.0/24	10.0.15.1	22	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:11:140
update	00:40:10:452	R(n)	10.0.5.0/24	10.0.17.2	21	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:11:154
update	00:40:12:484	R(n)	10.0.5.0/24	10.0.17.2	21	10.0.17.2	0	00:58	R14->R15->R13->R12->R3->R1-...	true	00:00:13:186
update	00:40:15:358	R(n)	10.0.5.0/24	10.0.17.2	24	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:16:060
update	00:40:20:120	R(n)	10.0.5.0/24	10.0.17.2	24	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:20:822
update	00:40:20:500	R(n)	10.0.5.0/24	10.0.17.2	24	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:21:202
update	00:40:24:422	R(n)	10.0.5.0/24	10.0.17.2	27	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:25:124
update	00:40:24:634	R(n)	10.0.5.0/24	10.0.15.1	25	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:25:336
update	00:40:28:386	R(n)	10.0.5.0/24	10.0.15.1	25	10.0.15.1	0	00:57	R14->R12->R3->R1->R2->R3	true	00:00:29:088
update	00:40:29:642	R(n)	10.0.5.0/24	10.0.15.1	28	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:30:344
update	00:40:31:411	R(n)	10.0.5.0/24	10.0.15.1	28	10.0.15.1	0	00:59	R14->R12->R3->R1->R2->R3	true	00:00:32:113
update	00:40:33:657	R(n)	10.0.5.0/24	10.0.15.1	34	10.0.15.1	0	01:00	R14->R12->R3->R1->R2->R3	true	00:00:34:359
update	00:40:35:295	R(n)	10.0.5.0/24	10.0.17.2	30	10.0.17.2	0	01:00	R14->R15->R13->R12->R3->R1-...	true	00:00:35:997
update	00:40:35:629	R(n)	10.0.5.0/24	10.0.17.2	30	10.0.17.2	0	00:59	R14->R15->R13->R12->R14	false	00:00:36:331
update	00:40:36:562	R(n)	10.0.5.0/24	10.0.17.2	34	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:37:264
update	00:40:41:585	R(n)	10.0.5.0/24	10.0.17.2	38	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:42:287
update	00:40:46:299	R(n)	10.0.5.0/24	10.0.17.2	42	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:47:001
update	00:40:46:598	R(n)	10.0.5.0/24	10.0.17.2	42	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:47:300
update	00:40:52:591	R(n)	10.0.5.0/24	10.0.17.2	46	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:53:293
update	00:40:54:333	R(n)	10.0.5.0/24	10.0.17.2	50	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:55:035
update	00:40:55:426	R(n)	10.0.5.0/24	10.0.17.2	50	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:00:56:128
update	00:41:01:317	R(n)	10.0.5.0/24	10.0.17.2	54	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:01:02:019
update	00:41:06:402	R(n)	10.0.5.0/24	10.0.17.2	58	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:01:07:104
update	00:41:07:441	R(n)	10.0.5.0/24	10.0.17.2	58	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:01:08:143
update	00:41:13:264	R(n)	10.0.5.0/24	10.0.17.2	62	10.0.17.2	0	01:00	R14->R15->R13->R12->R14	false	00:01:13:966
update	00:41:16:472	R(n)	10.0.5.0/24	10.0.17.2	64	10.0.17.2	0	00:40	R14->R15->R13->R12->R14	false	00:01:17:174
update	00:41:27:304	R(n)	10.0.5.0/24	10.0.17.2	64	10.0.17.2	0	00:29	R14->R15->R13->R12->R14	false	00:00:00:000
update	00:41:39:350	R(n)	10.0.5.0/24	10.0.17.2	64	10.0.17.2	0	00:17	R14->R15->R13->R12->R14	false	00:00:00:000
update	00:41:49:389	R(n)	10.0.5.0/24	10.0.17.2	64	10.0.17.2	0	00:07	R14->R15->R13->R12->R14	false	00:00:00:000

Abbildung 61: Analysetabelle: Gegenseitige Beeinflussung beider Zyklen

Abbildung 60: Netzgraph: Gegenseitige Beeinflussung beider Zyklen

Auf die Konvergenzzeit jedes einzelnen Routern haben die oben genannten Effekte geringen Einfluss. R1 im Zyklus und R16 weit entfernt vom Zyklus konvergieren annähernd gleich schnell. Ebenso schnell konvergiert R14 trotz seines ungewöhnlichen Zählverhaltens. Die Tabelle 12 zeigt die Konvergenzzeiten in Abhängigkeit von den Einstellungen der Timer.

Timer-Einstellungen Update-Timer / Erreichbarkeits-Timeout / Garbage-Collection-Timer		Konvergenzzeit (CTI-Duration)
T_0	3s / 18s / 12s	41,3s
T_1	6s / 36s / 24s	42,9s
T_2	10s / 60s / 40s	56,4s
T_3	30s / 180s / 120s	67,2s

Tabelle 12: Ergebnisse versch. T-Einstellungen beim CTI im Big-Net-Szenario

Beschleunigt man den Nachrichtenaustausch der Router, wirkt sich das positiv auf die Dauer des Count-To-Infinity-Effekts aus. Bei den Standard-Timern (30/180/120), die zur Zeit verwendet werden, dauert es 187,2 Sekunden bis das Netz konvergiert. Wenn das Netz dreimal so schnell eingestellt wird, konvergiert das Netz bereits knapp doppelt so schnell. Ein Fünftel der ursprünglichen Timer-Einstellungen lässt das Netz ungefähr dreimal so schnell konvergieren. Bei einem Zehntel der ursprünglichen Timer-Einstellungen lässt die positive Wirkung sichtbar nach. Anstatt einer zu erwartenden mehr als sechs mal so schnellen Konvergenzzeit, sind die kürzesten Timer-Einstellungen nur noch etwa 3,5-mal so schnell wie der ursprüngliche Wert. Die Leistung der schnelleren Timer-Einstellungen stößt bedingt durch den Triggered-Timer an seine Grenzen.

Ergebnis des RIPMTI-Algorithmus

Um den Count-To-Infinity-Effekt erfolgreich zu verhindern, müssen nicht alle Router des Big-Net-Szenarios die Schleifenberechnung durchführen. Die Source-Router R1 und R14 sind ausreichend. Der Request-Timer muss für jede Timer-Einstellung angepasst werden. Er soll verhindern, dass eine Route zu schnell aus der Routingtabelle entfernt wird. Wenn auf einem Router der Garbage-Collection-Timer abgelaufen ist, wird die entsprechende Route gelöscht. Sollte sich bis zu diesem Zeitpunkt noch eine Falsch-Nachricht in Umlauf befinden und den Router erreichen, dann wird diese als gültig gewertet und neu in die Routingtabelle eingefügt. Wird im linken oberen Zyklus der Count-To-Infinity-Effekt von R1 verhindert, wird der entsprechende Eintrag in der Routingtabelle entfernt. Wenn sich im rechten oberen Zyklus noch Erreichbarkeitsinformationen über das verloren gegangene Netz befinden, können diese später wieder zu R1 gelangen. Um diesen

selten auftretenden Fall abzufangen, muss der Request-Timer auf einen Wert von 20-25 gestellt werden. Die Metrik 64 wird noch so lange vorgehalten wie der Request-Timer angibt bevor die Route gelöscht wird. RIPMTI konnte den Count-To-Infinity-Effekt in allen Fällen verhindern. Die in der Tabelle angegebenen Konvergenzzeiten beziehen sich auf die Dauer des Garbage-Collection-Timers der jeweiligen Timer-Einstellungen und sollen angeben, dass der Router sofort konvergiert.

Im Vergleich zu den Konvergenzzeiten des ursprünglichen RIP-Algorithmus ist der Geschwindigkeitsgewinn des Netzwerkes schneller, je kürzer die Timer-Einstellungen geregelt werden. Bei den ursprünglichen Timer-Einstellungen bringt RIPMTI einen Konvergenzbonus von 36%, wohingegen bei den kürzesten getesteten Timern der Bonus bereits bei 77% liegt.
Folgende Berechnung des Simple-Loop-Tests läuft für R1 und Netz 10.0.5.0/24:

$$m_{IF1}^{(R1,10.0.5.0/24)} = Route\ über\ R1 \rightarrow R2 \rightarrow R3 \rightarrow R1 \rightarrow R6 \rightarrow R5 \rightarrow R7$$
$$m_{IF4}^{(R1,10.0.7.0/24)} = Route\ über\ R1 \rightarrow R6 \rightarrow R5 \rightarrow R7$$
$$m_{IF1}^{(R1,10.0.5.0/24)} + m_{IF4}^{(R1,10.0.5.0/24)} - 1 \geqslant msilm_{(IF1,IF4)}^{R2}$$
$$7 + 4 - 1 = 10 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Folgende Berechnung des Simple-Loop-Tests läuft für R1 und Netz 10.0.5.0/24:

$$m_{IF1}^{(R1,10.0.5.0/24)}$$
$$= Route\ über\ R1 \rightarrow R2 \rightarrow R3 \rightarrow R12 \rightarrow R14 \rightarrow R15 \rightarrow R13 \rightarrow R12 \rightarrow R3 \rightarrow R1 \rightarrow R6 \rightarrow R5 \rightarrow R7$$
$$m_{IF4}^{(R1,10.0.7.0/24)} = Route\ über\ R1 \rightarrow R6 \rightarrow R5 \rightarrow R7$$
$$m_{IF1}^{(R1,10.0.5.0/24)} + m_{IF4}^{(R1,10.0.5.0/24)} - 1 \geqslant msilm_{(IF1,IF4)}^{R2}$$
$$13 + 4 - 1 = 10 \geqslant 127(f)$$
$$\Rightarrow angebotene\ Alternativroute\ ist\ kein\ Simple-Loop$$
$$\Rightarrow angebotene\ Alternativroute\ muss\ ein\ Source-Loop\ sein$$

Zusammenfassung

RIPMTI verhindert zuverlässig das Entstehen eines Count-To-Infinity-Effekts und führt zur sofortigen Konvergenz des Netzwerks und verhindert Inkonsistenzen bei allen Routern außerhalb des Netzwerkes. Im Big-Net-Szenario sind im Verhältnis zur Gesamtmenge der Router nur zwei Router in Schlüsselpositionen notwendig, um das Netzwerk stabil zu halten.
Der RIP-Algorithmus beeinträchtigt das Netzwerk enorm. Alle Router weisen für die Dauer des Count-To-Infinity-Effekts Inkonsistenzen auf und verursachen einen

hohen Update-Austausch auf den Verbindungsleitungen. RIPMTI konvergiert bei T_0-Einstellungen sofort und wenn man wieder alle Timer heranzieht dauert es nur 42% der vergleichbaren RIP-Zeit bis eine Route aus der Routingtabelle komplett entfernt werden kann.
Bei 100 Testdurchläufen trat der Count-To-Infinity-Effekt in 85% der Fälle auf und wurde von RIPMTI erfolgreich verhindert.

Fazit & Ausblick

In dieser Diplomarbeit wurden unterschiedliche Netzwerk-Szenarien vorgestellt, die in der Realität durchaus in dieser Form eingesetzt werden. Anhand automatisierter Tests konnte der RIPMTI-Algorithmus unter Verwendung unterschiedlicher Parameter auf seine Funktionalität hin getestet werden. Anhand der einzelnen Konfigurierbarkeit jedes virtuellen Routers konnte gezeigt werden, dass RIPMTI-Router in ein bestehendes Netzwerk aus RIP-Routern integriert werden können, ohne Probleme zu verursachen. Der Count-To-Infinity-Effekt konnte durch RIPMTI in allen Fällen verhindert werden, sodass ein Einsatz in komplexeren Netzwerk-Topologien möglich ist. Das X-Mod-Szenario und Y-Mod-Szenario konnten die Schwierigkeiten aufzeigen, die bei komplexen Szenarien auftreten können. Das zeigt die Schwierigkeit bei der Implementierung eines funktionierenden Algorithmus, der den Count-To-Infinity-Effekt verhindern und gleichzeitig die Konvergenzzeit eines Netzwerkes verbessern soll.
Zunächst ist feststellbar, dass eine Verringerung der periodischen Update-Timer,

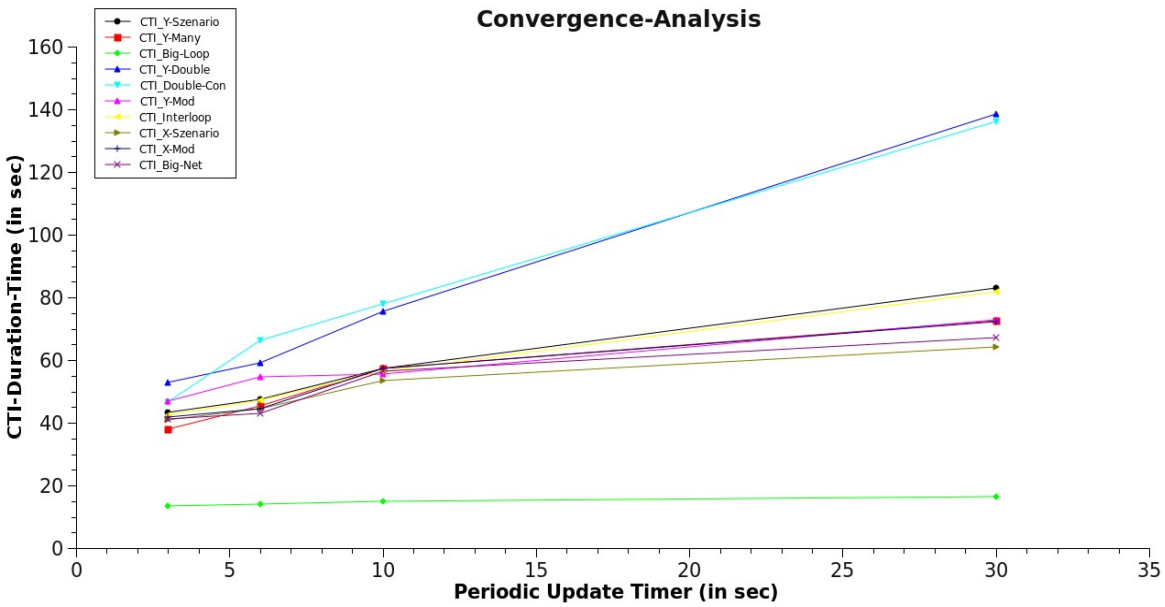


Abbildung 62: Konvergenzzeiten des CTI aller Szenarien in Abhängigkeit der Timer

und damit einhergehend die Verringerung des Timeout-Timers bzw. des Garbage-Collection-Timers grundsätzlich die Konvergenzzeit eines von RIP kontrollierten Netzwerkes deutlich verbessert. Die Graphik aus Abbildung 62 stellt alle getesteten Szenarien in einem Diagramm gegenüber. Tendenziell reduziert sich in den Tests die Konvergenzzeit nach dem Auftreten eines Count-To-Infinity-Effekts teilweise drastisch, in anderen Fällen etwas schwächer, was durch die Topologie selbst zu erklären ist. Die Modelle Double-Con und Y-Double haben bei den langsamen Timer-Einstellungen die schlechteste Konvergenzzeit im Vergleich zu allen anderen Modellen und bei einer Beschleunigung des Kommunikationsaustauschs der Router lässt sich hier auch der deutlichste Geschwindigkeitsgewinn feststellen. Das Big-Loop-Szenario schneidet am Besten ab. Seine Konvergenzzeiten bleiben über alle Tests hinweg annähernd konstant. Die übrigen Modelle bewegen sich relativ nah auf einem Level, da sich die Konvergenzzeiten trotz unterschiedlicher Topologien kaum unterscheiden. Alle Werte scheinen von den langsamen zu den schnelleren Timer-Einstellungen hin auf einen Ursprungspunkt hinzudeuten, da sich die Ergebnisse bei den kurzen 3-Sekunde-Timern sehr aneinander annähern, abgesehen vom ohnehin rasch konvergierenden Big-Loop-Szenario. Eine weitere Beschleunigung scheint nicht mehr nötig zu sein. Hinzu kommt, dass Nutzdaten eine gewisse Zeit in Anspruch nehmen, um eine Route zu durchlaufen und eine all zu schnelle Kommunikation der Router untereinander würde viel zu häufig zu Routenänderungen und enorm viel Kommunikationsdatenlast führen.
Der nächste zu klärende Sachverhalt ist die Anhebung des RIP_Infinity-Wertes von 16 auf 64, um eine maximale Größe eines Netzwerkes zu steigern. Zunächst fällt auf, dass eine höhere Metrik den Count-To-Infinity-Effekt entsprechend öfter durch eine Netzwerkschleife laufen lässt. Hier konnte bewiesen werden, dass RIPMTI im Gegensatz zu RIP einen Einsatz in größeren Netzwerken ermöglicht, da er keine Probleme damit hatte wenn ein Count-To-Infinity-Effekt aufgetreten war ein RIPMTI-Router ihn verhindern sollte. Die Schleife wurde grundsätzlich erkannt und als Unerreichbarkeitsmetrik der Wert 64 anstatt 16 verwendet.

Diese Arbeit konnte zeigen, dass eine Manipulation von RIP_Infinity und der Timer-Einstellungen für den Austausch von Routinginformationen, das Erkennen von ausgefallenen Routen und das Entfernen ungültiger Routen aus der Routingtabelle für den RIPMTI-Algorithmus keinen Unterschied mehr macht. Er hat in allen Szenarien mittels Simple-Loop-Test den Count-To-Infinity-Effekt zwischen zwei Interfaces eines Routers oder zwei IP-Adressen zweier Router auf einem Interface verhindert. Zukünftige Diplomarbeiten können hierauf aufbauend

komplexere Szenarien bearbeiten, die sich auf RIP_Infinity 64 einstellen und periodische Update-Timer von 3 Sekunden, Timeout-Timer von 18 Sekunden und den neuen dynamischen Garbage-Collection-Timer einsetzen, um das Verhalten des RIPMTI-Algorithmus zu ermitteln.

Anhang A

„Y-Szenario1“

// Konfigurationsdatei YSzenario1
// CTI 64 - kurze Timer 3/18/12 (yscenario_3-18-12.config)

Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
// Ausfallphase von Netz e
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
// CTI-Phase, Updates mit Metrik 64 kommen nicht an
R1(0,0); R2(x,0); R3(x,0,0); R4(x,0);
// Ausbreitungsphase: R1 versendet Falsch-Update
R1(0.2,0)a; R2(x,0)a; R3(x,0,0)a; R4(x,0);

„Y-Szenario2“

// Konfigurationsdatei YSzenario2
//CTI 64 - mittlere Timer 10/60/40 Auto-Timer 80000ms(yscenario_10-60-40.config)

Update-Forecast
// warten auf Konvergenz
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
// Netz e trennen
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);

```
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
// Updates mit neuer Metrik 64 kommen nicht an
R1(0,0); R2(0,0); R3(x,0,0); R4(x,0);
// R1 versendet Falsch-Update
R1(0.5,0)a; R2(0,0)a; R3(x,0,0)a; R4(x,0);
```

Anhang B

„Ymany1“

```
// Konfigurationsdatei 1

// Hostname des RIP-Router
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5


// Parameter in ms
periodic_update 3000


// Update_Forecast
// 0 fuer normales Update
// x fuer kein Update
// n fuer Update n Sekunden spaeter
// Postfix a fuer MODE=AUTO

Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
```

```
// Ausfallphase
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
// Ausbreitungsphase
R1(1,0); R2(0,0); R3(x,0,0); R4(x,0); R5(0,0);
// CTI-Phase
R1(0,0)a; R2(0,0)a; R3(x,0,0)a; R4(x,0); R5(0,0)a;
```

„YMany2“

```
// Konfigurationsdatei 2

// Hostname des RIP-Router
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5

// Parameter in ms
periodic_update 10000

// Update_Forecast
// 0 fuer normales Update
// x fuer kein Update
// n fuer Update n Sekunden spaeter
// Postfix a fuer MODE=AUTO

Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
// Ausfallphase
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
```

R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
R1(0,0); R2(0,0); R3(0,0,0); R4(x,0); R5(0,0);
// Ausbreitungs- und CTI-Phase
R1(1,0)a; R2(0,0)a; R3(x,0,0)a; R4(x,0); R5(0,0)a;

Anhang C

„Big-Loop1“

Big-Loop1

```
// autogenerated config file
Rip-Router r5
Rip-Router r4
Rip-Router r3
Rip-Router r2
Rip-Router r1
Rip-Router r0
Rip-Router r13
Rip-Router r12
Rip-Router r11
Rip-Router r10
Rip-Router r9
Rip-Router r8
Rip-Router r7
Rip-Router r6


// update_time in ms
periodic_update 3000

Update-Forecast
// Konvergenzphase
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,0); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
// Ausfallphase (R0 wird vom Netz getrennt)
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
```

r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0.0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,0);
// CTI-Phase (R7 wird zum False-Router)
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0.0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,x);
r5(0,0); r4(0,0); r3(0,0); r2(0,0); r1(0,0,0.0); r0(0,x); r13(0,0); r12(0,0); r11(0,0); r10(0,0); r9(0,0); r8(0,0); r7(0,0); r6(0,x);
// Ausbreitungsphase (CTI breitet sich im UZS aus)
r5(0,0)a; r4(0,0)a; r3(0,0)a; r2(0,0)a; r1(0,0,0)a; r0(0,x); r13(0,0)a; r12(0,0)a; r11(0,0)a; r10(0,0)a; r9(0,0)a; r8(0,0)a;
r7(0,0)a; r6(0,x)a;

„Big-Loop2"

Big-Loop2

// Generator Testdatei

// Hostname
Rip-Router R0
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5
Rip-Router R6
Rip-Router R7
Rip-Router R8
Rip-Router R9
Rip-Router R10
Rip-Router R11
Rip-Router R12
Rip-Router R13

// Parameter in ms
periodic_update 10000

Update-Forecast
// Konvergenzphase
R0(0,0); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,0); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,0); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,0); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,0); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,0); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);

// Ausfallphase (R0 wird vom Netz getrennt)
R0(0,x); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);

```
R13(0,0);
R0(0,x); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,x); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,x); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);
R0(0,x); R1(0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0); R7(0,0); R8(0,0); R9(0,0); R10(0,0); R11(0,0); R12(0,0);
R13(0,0);

// CTI-Phase (alle Router verzögern ihre Updates in Richtung R7 => R7 wird zum False-Router)
R0(0,x); R1(0,1,1); R2(0,1.5); R3(0,2); R4(0,2.5); R5(0,3); R6(0,x); R7(0,0); R8(x,0); R9(3.5,0); R10(3,0); R11(2.5,0);
R12(2,0); R13(1.5,0);

// Ausbreitungsphase (CTI breitet sich im UZS aus)
R0(0,x); R1(0,0,0)a; R2(0,0)a; R3(0,0)a; R4(0,0)a; R5(0,0)a; R6(2,x)a; R7(1,1)a; R8(x,2)a; R9(0,0)a; R10(0,0)a;
R11(0,0)a; R12(0,0)a; R13(0,0)a;
```

Anhang D

„Y-Double“

```
// Konfigurationsdatei „Y-Double“

Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5
// Parameter in ms
periodic_update 3000

// Update_Forecast
// 0 fuer normales Update
// x fuer kein Update
// n fuer Update n Sekunden spaeter
// Postfix a fuer MODE=AUTO

Update-Forecast
// Konvergenzphase
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(0,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(0,0);
```

```
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(0,0);
// Ausfallphase
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
// Ausbreitungsphase
r3(x,1,1,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
// CTI-Phase
r3(2.4,1,0,0); r2(0,0,1.6)a; r1(0.8,0)a; r4(0,0)a; r5(x,0);
r3(2.4,1,0,0)a;
```

„Y-Double_R4“

```
// Konfigurationsdatei „Y-Double_R4“

Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5


// Parameter in ms
periodic_update 3000

// Update_Forecast
// 0 fuer normales Update
// x fuer kein Update
// n fuer Update n Sekunden spaeter
// Postfix a fuer MODE=AUTO

Update-Forecast
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(0,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(0,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(0,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(x,0);
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(x,0);
```

```
r3(0,0,0,0); r2(0,0,0); r1(0,0); r5(0,0); r4(x,0);

r3(x,1,1,0); r2(0,0,0); r1(0,0); r5(0,0); r4(x,0);
r3(2.4,1,0,0); r2(0,0,1.6)a; r1(0.8,0)a; r5(0,0)a; r4(x,0);
r3(2.4,1,0,0)a;
```

Anhang E

„Double-Con1“

```
// Konfigurationsdatei 1 (T0)
// Hostname
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4

// Parameter in ms
periodic_update 3000

Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
// Ausfallphase
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
// Ausbreitungsphase
```

```
R1(0,0); R2(0,x,0); R3(0,0); R4(x,0);
// CTI-Phase
R1(0,1)a; R2(2,x,0)a; R3(0,0)a; R4(x,0);
```

„Double-Con2“

```
//  Konfigurationsdatei 2 (T1)
// Hostname
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4

// Parameter in ms
periodic_update 6000

Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
// Ausfallphase
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(x,0)
// Ausbreitungsphase
R1(0,0); R2(0,x,0); R3(0,0); R4(x,0);
//CTI-Phase
R1(0,0)a; R2(1,x,0)a; R3(0,0); R4(x,0)a;
```

„Double-Con3“

```
//  Konfigurationsdatei 3 (T2 und T3)
```

```
// Hostname
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4

// Parameter in ms
periodic_update 10000

Update-Forecast
// Konvergenzphase
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(0,0); R4(0,0);
// Ausfallphase
R1(0,0); R2(0,0,0); R3(x,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(x,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(x,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(x,0); R4(0,0);
R1(0,0); R2(0,0,0); R3(x,0); R4(0,0);
// Ausbreitungsphase
R1(0,0); R2(0,x,0); R3(0,0); R4(x,0);
// CTI-Phase
R1(0,1)a; R2(2,x,0)a; R3(0,0); R4(x,0)a;
```

Anhang F

„Y-Mod1“

```
// Konfigurationsdatei Y-Mod
Rip-Router r3
Rip-Router r2
Rip-Router r1
Rip-Router r5
```

```
Rip-Router r4


// update_time in ms
periodic_update 3000

Update-Forecast
r3(0,0,0); r2(0,0); r1(0,0,0); r5(0,0,0); r4(0,0,0,1);
r3(0,0,0); r2(0,0); r1(0,0,0); r5(0,0,0); r4(0,0,0,1);
r3(0,0,0); r2(0,0); r1(0,0,0); r5(0,0,0); r4(0,0,0,1);

r3(0,0,0); r2(0,0); r1(0,0,0); r5(x,x,0); r4(0,0,0,0);
r3(0,0,0); r2(0,0); r1(0,0,0); r5(x,x,0); r4(0,0,0,0);
r3(0,0,0); r2(0,0); r1(0,0,0); r5(x,x,0); r4(0,0,0,0);
r3(0,0,0); r2(0,0); r1(0,0,0); r5(x,x,0); r4(0,0,0,0);
r3(0,0,0); r2(0,0); r1(0,0,0); r5(x,x,0); r4(0,0,0,0);

r3(0,0,0); r2(0,0); r1(0,0,0); r5(x,x,0); r4(0,x,0,0);
r3(0,0,0)a; r2(1,0)a; r1(0,0,0)a; r5(x,x,0); r4(0,x,0,0)a;
```

Anhang G

„Interloop1“

```
// Interloop1
// Konfigurationsdatei

Rip-Router R0
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4

// Parameter in ms
periodic_update 3000
```

Update-Forecast
// Konvergenzphase
r0(0,0); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
r0(0,0); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
r0(0,0); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
// Ausfallphase
r0(x,x); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
r0(x,x); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
r0(x,x); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
r0(x,x); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
r0(x,x); r1(0,0,0); r2(0,0,0); r3(0,0); r4(0,0,0);
// Ausbreitungsphase
r0(x,x); r1(0,0,0.5); r2(0,x,0); r3(0,0); r4(0,0,0);
// CTI-Phase
r0(x,x); r1(0,0,0)a; r2(0,x,0)a; r3(1,1)a; r4(0,0,0)a;

Anhang H

„X-Szenario1“

Update-Forecast (**xscenariocti1**)
Konvergenzphase
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
Ausfallphase
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(x,0); R5(x,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(x,0); R5(x,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(x,0); R5(x,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(x,0); R5(x,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(x,0); R5(x,0); R6(0,0);
CTI-Phase
R0(0,0); R1(x,0,0,0); R2(0,0); R3(0,0); R4(x,0); R5(x,0); R6(0,0);
Ausbreitungsphase

R0(0,0)a; R1(x,0,0,0)a; R2(0,1)a; R3(0,0)a; R4(x,0); R5(x,0); R6(0,0)a;

„X-Szenario2“

Update-Forecast (xscenario2)
Konvergenzphase
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(0,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
Ausfallphase
R0(x,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(x,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(x,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(x,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
R0(x,0); R1(0,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,0);
CTI-Phase
R0(x,0); R1(x,0,0,0); R2(0,0); R3(0,0); R4(0,0); R5(0,0); R6(0,x);
Ausbreitungsphase
R0(x,0); R1(x,0,0,0)a; R2(0,1)a; R3(0,0)a; R4(0,0)a; R5(0,0)a; R6(0,x)a;

Anhang I

„X-Mod“

Anhang

// autogenerated config file
Rip-Router r3
Rip-Router r2
Rip-Router r1
Rip-Router r7
Rip-Router r6
Rip-Router r5
Rip-Router r4

```
// update_time in ms
periodic_update 3000

Update-Forecast
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(0,0); r5(0,0); r4(1,0);
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(0,0); r5(0,0); r4(1,0);
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(0,0); r5(0,0); r4(1,0);

r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(x,0); r5(0,0); r4(1,0);
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(x,0); r5(0,0); r4(1,0);
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(x,0); r5(0,0); r4(0,0);
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(x,0); r5(0,0); r4(0,0);
r3(0,0); r2(0,0); r1(0,0,0); r7(0,0); r6(x,0); r5(0,0); r4(0,0);

r3(0,0); r2(0,x); r1(0.5,0.5,0); r7(0,0); r6(x,0); r5(0,0); r4(x,0);
r3(0,0)a; r2(0,x)a; r1(0,0,0)a; r7(1,1)a; r6(x,0); r5(0,0)a; r4(x,0)a;
```

Anhang J

„Big-Net1“

```
// Konfiguration Big-Net1

// Hostname
Rip-Router R0
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5
Rip-Router R6
Rip-Router R7
Rip-Router R8
Rip-Router R9
Rip-Router R10
Rip-Router R11
Rip-Router R12
Rip-Router R13
```

Rip-Router R14
Rip-Router R15
Rip-Router R16

// Parameter in ms
periodic_update 30000

Update-Forecast
// Konvergenzphase
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);

// Ausfallphase
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);

// Ausbreitungsphase
R0(0,0,0); R1(0,x,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);

// CTI-Phase
R0(0,0,0)a; R1(0,x,0,0)a; R2(0,0,0)a; R3(1,0,0)a; R4(0,0)a; R5(0,0)a; R6(0,0,0)a; R7(x,0);

R8(x,0); R9(0)a; R10(0)a; R11(0)a; R12(0,0,0)a; R13(0,0)a; R14(0,0,0)a; R15(0,0)a; R16(0,0)a;

„Big-Net2“

// Konfiguration Big-Net2

// Hostname
Rip-Router R0
Rip-Router R1
Rip-Router R2
Rip-Router R3
Rip-Router R4
Rip-Router R5
Rip-Router R6
Rip-Router R7
Rip-Router R8
Rip-Router R9
Rip-Router R10
Rip-Router R11
Rip-Router R12
Rip-Router R13
Rip-Router R14
Rip-Router R15
Rip-Router R16

// Parameter in ms
periodic_update 3000

Update-Forecast
// Konvergenzphase
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0); R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0); R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0); R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0); R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0); R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);

R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(0,0); R8(0,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(0,0);

// Ausfallphase
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(x,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(x,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(x,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(x,0);
R0(0,0,0); R1(0,0,0,0); R2(0,0,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(0,0); R14(0,0,0); R15(0,0); R16(x,0);

// Ausbreitungsphase
R0(0,0,0); R1(0,x,0,0); R2(0,x,0); R3(0,0,0); R4(0,0); R5(0,0); R6(0,0,0); R7(x,0); R8(x,0); R9(0);
R10(0); R11(0); R12(0,0,0); R13(x,0); R14(x,0,0); R15(x,0); R16(x,0);

// CTI-Phase
R0(0,0,0)a; R1(0,2,0,0)a; R2(0,x,0)a; R3(1,x,0)a; R4(0,0)a; R5(0,0)a; R6(0,0,0)a; R7(x,0);
R8(x,0); R9(0)a; R10(0)a; R11(0)a; R12(1,1,x)a; R13(x,0)a; R14(2,0,0)a; R15(x,0)a; R16(x,0);

Literaturverzeichnis

[BEL57] R. E. Bellman, Dynamic Programming; Princeton University; 1957

[BOH08] Frank Bohdanowicz; Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons; Diplomarbeit; Universität Koblenz; 2008

[ITW09] IT Wissen: Distance vector Algorithm; http://www.itwissen.info/definition/lexikon/distance-vector-algorithm-DVA-Distance-Vektor-Algorithmus; Stand: 18.08.2009; 2009

[KEU07] Tim Keupen; Generierung von Testfällen für den RIP-MTI Algorithmus; Diplomarbeit; Universität Koblenz; 2007

[KLE01] Thomas Kleemann, RIPeval: Evaluierung und Weiterentwicklung des RIP-MTI-Algorithmus; Diplomarbeit; Universität Koblenz; 2001

[KOC05] Tobias Koch, Implementation und Simulation von RIP-MTI; Diplomarbeit; Universität Koblenz; 2005

[PET00] Larry L. Peterson, Bruce S. Davie: Computernetze; Dpunkt Verlag; 2003

[QUA07] Quagga Projekthomepage; http://wiki.quagga.net/index.php/Main/HomePage; Stand: 18.08.2009; 2007

[RFC95] RFC1812: Requirements for IP Version 4 Routers; http://www.faqs.org/rfcs/rfc1812.html; Stand: 18.08.2009; 1995

[RIO09] Routing Information Protocol; http://www.datenschutz-praxis.de/lexikon/r/routing_information_protocol.html Stand; 18.08.2009; 2009

[SCH99] Andreas Schmid; RIP-MTI: Minimum-effort loop-free distance vector routing algorithm; Diplomarbeit; Universität Koblenz; 1999

[StDiKe08] Ch. Steigner, H. Dickel, T. Keupen, RIP-MTI: A New Way to Cope with Routing Loops; Universität Koblenz; 2008

[TAN03] Andrew S. Tanenbaum: Computernetzwerke; Pearson Studium; Auflage 4; 2003

[UML09] The User-mode Linux Kernel Home Page; http://user-mode-linux.sourceforge.net/index.html; Stand: 18.08.2009; 2009

[VNU07] Virtual Network User Mode Linux (VNUML); http://www.dit.upm.es/vnumlwiki/index.php/Main_Page; Stand: 18.08.2009; 2007

[VNU09] VNUML – Universität Koblenz Projekthomepage; http://www.uni-koblenz.de/~vnuml/index.de.php; Stand: 18.08.2009; 2009

[WEI08] Rainer Weißenfels: Routing with metric-based topology investigation; Seminararbeit; Universität Koblenz; 2008

[WIK09] Wikipedia: Distanzvektoralgorithmus; http://de.wikipedia.org/wiki/Distanzvektoralgorithmus; Stand: 18.08.2009; 2009

[XML98] XML – A Technical Instruction to XML; http://www.xml.com/pub/a/98/10/guide0.html; Stand: 18.08.2009; 1998

[ZIM09] Zimon: VNUML – Ein Netzwerksimulator mit UML; http://zinformatik.de/tipps-tricks/interessante-programme/vnuml-ein-netzwerksimulator-mit-user-mode-linux/; Stand: 18.08.2009; 2009