Universität Koblenz-Landau

Fachbereich 4 (Informatik)
Institut für Informatik
Arbeitsgruppe Rechnernetze

Generierung von Testfällen für den RIP-MTI Algorithmus

Diplomarbeit
zur Erlangung des akademischen Grades
„Diplom-Informatiker“

Bearbeiter:	Tim Keupen
	Matrikelnummer: 202110012
Betreuer:	Prof. Dr. Ch. Steigner
	Dipl. Inform. H. Dickel
Bearbeitungszeit:	01.03.2007 - 31.08.2007

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als den von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Koblenz, den 26. August 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Abstract

The „Routing Information Protocol“ (RIP) is the internet standard of distance vector routing algorithms. This protocol is widely supported and easy to configure. But it suffers from a problem which occurs when adapting to changes in topology. This is the „counting to infinity“ issue. Due to timing incidences between regular (periodically) and triggered updates, updates with non up-to-date information may propagate through the net. If this happens in a cylce the update could be forwarded forever and metrics count to infinity. The RIP-MTI algorithm extends the local information based on the already provided informations by the RIP-updates. It checks for every known subnetwork if there does exist a cycle and computes the length of this cycle. When topology changes, MTI decides whether the new route can be accepted or not. If this approach appears to be feasible the advantages are fundamental: the upper bound of 16 hops can be increased to much higher values and network load is reduced and so distance vector routing could be used in complex networking environments as well.
This thesis analyses the assumptions in a simulation environment. The analysis is done by a RIP extension, which offers to control rip updates from the outside. So a suitable way to enforce update orders, which might occur in a real network because of latency and other timing issues, is given. A test environment, that enables the user to test the MTI algorithm, is implemented using the features of the simulator VNUML. If counting to infinity can still occur in MTI, this should be discovered by this implementation.

KEYWORDS: RIP, Counting-to-Infinity, VNUML, RIP-XT

1 Einleitung

Ziel dieser Arbeit war es, den RIP-MTI Algorithmus (s. Kapitel 1.3.1), durch zu entwickelnde Verfahren, strukturiert und in einer repräsentativen Auswahl von Topologien zu testen. Zu diesem Zweck wurde die Testumgebung „XTPeer“ entwickelt, mit der für eine vorgegebene Topologie Updatereihenfolgen generiert und ausgeführt werden können.

Für eine Analyse des daraus resultierenden Verhaltens, waren jedoch weitere umfangreiche Schritte notwendig. Besonderer Dank geht deshalb an Stefan Lange, der in seiner Arbeit „*Zentrale Betrachtung von Routing-Informationen zur Analyse des Konvergenzverhaltens verschiedener RIP-Algorithmen und Unterstützung des Generierens von Testfällen*“ [Lan07] weitere wichtige Funktionalitäten erarbeitet und dem Programm hinzugefügt hat. Durch die zentrale Überwachung aller Ereignisse im Netz, kann das durch den Update-Generator erzeugte Verhalten direkt beobachtet und analysiert werden. Die Implementierung der Testumgebung erfolgte dabei in Zusammenarbeit über ein SVN-Repository.

1.1 Motivation

Um die korrekte Funktionsweise von Software sicherzustellen, muss diese ausgiebig getestet werden. Insbesondere bei Routing-Protokollen gestalten sich Tests jedoch schwierig, da die Zielumgebung dafür nicht verwendet werden kann. Aufgrund der hohen Anschaffungspreise existiert die nötige Hardware nur für den laufenden Betrieb und kann daher nicht für Tests und Analyse neuer Software verwendet werden. Eine Alternative bietet eine Simulationsumgebung, in der jede mögliche Zielumgebung abgebildet werden kann. Ein Beweis der Korrektheit über formale Methoden scheint bei Routing-Protokollen kaum anwendbar, da diese als verteilte und oftmals nicht synchronisierte Anwendung laufen und deshalb ein Beweis, aufgrund der Komplexität, nicht handhabbar wäre.

Ein Routing-Protokoll muss im alltäglichen Betrieb in zwei Dimensionen korrekt funktionieren: Zum Einen in jeder erdenklichen Topologie und zum Anderen muss jede mögliche zeitliche Abfolge von Ereignissen sicher behandelt werden. Um diese zeitliche Abfolge kontrollierbar, und damit für Tests zugänglich zu machen, wurde eine Erweiterung entwickelt, die es erlaubt Reihenfolgen der versendeten Updates zentral vorgeben und ausführen zu können.

Konvergenzprobleme bei Routing-Protokollen werden u.a. durch ungünstige Konstellationen von Updatereihenfolgen hervorgerufen. Ein besserer Algorithmus muss also genau mit diesen ungünstigen Konstellationen umgehen können, darf aber dabei in allen anderen Fällen keine Verschlechterung sein. Genau dieser Gedanke liegt dieser Arbeit zu Grunde: wenn eine Abfolge von -von außen gesteuerten- Updates bei Verwendung des Routing-Protokolls RIP ein

Konvergenzproblem (z.B. CTI) erzeugt, so sollte exakt dieselbe Abfolge bei Verwendung eines verbesserten Routing-Protokolls (RIP-MTI) kein Problem verursachen.

1.2 Terminologie

Testfall
Ein Testfall entspricht der Ausführung einer Config.

Config
Eine Config beschreibt einen Testfall. Dazu müssen alle involvierten RIP-Router angegeben werden. Weiterhin ist in einer Config ein Forecast abgebildet. Configs werden in Dateien gespeichert (.config).

Forecast
Ein Forecast (Vorhersage) beschreibt für eine Menge von RIP-Routern den Ablauf des Nachrichtenaustauschs, also die Reihenfolge der versendeten Updates. Er enthält für jeden RIP-Router und für jede Periode Werte, wann ein Update versendet werden soll. Die Reihenfolge wird dabei durch zeitliche Verzögerung erzwungen: soll Update1 nach Update2 eintreffen, muss Update1 auch später versendet werden.

Trigger
Zu jedem, im Forecast definierten Zeitpunkt wird ein Update durch den Generator getriggert, also versendet. Ein Trigger beschreibt dabei entweder den Wert der Verzögerung in Zehntelsekunden oder den Updateverlust (entspricht dem Wert „x“).

Generator
Der Generator führt die, durch einen Forecast beschriebene Updatereihenfolge aus oder generiert selbstständig Updatereihenfolgen.

RIP-Router
Ein RIP-Router ist eine Netzwerkomponente, dessen Software es ermöglicht den RIP-Algorithmus auszuführen und mit anderen RIP-Routern zu kommunizieren. Während dieser Kommunikation werden Informationen über die Erreichbarkeit von Zielnetzen ausgetauscht, so dass im Anschluss Datenverkehr über den richtigen Weg durch das Netz fließen kann. In dieser Arbeit wird der Begriff als Synonym für eine virtuelle Maschine innerhalb der VNUML-Simulationsumgebung verwendet.

1.3 Grundlagen

In diesem Abschnitt wird ein Überblick über die, dieser Arbeit zugrunde liegenden, Routing-Protokolle, die verwendete Simulationsumgebung VNUML und die RIP-XT Erweiterung gegeben.

1.3.1 Distance-Vector-Routing

Distance-Vector-Routing bezeichnet eine Protokollfamilie, die mit dem Distance-Vector-Algorithmus arbeitet. Dabei handelt es sich um einen dynamischen Routing-Algorithmus, der nach dem Prinzip „Teile deinen Nachbarn mit, wie du die Welt siehst“ funktioniert und intern auf dem Bellman-Ford-Algorithmus basiert. Jeder Teilnehmer sendet dabei an alle seine direkten Nachbarn die ihm bekannten Ziele in Form eines Tupel (Ziel, Kosten). Distanzvektorprotokolle sind selbstorganisierend, vergleichsweise einfach zu implementieren und funktionieren nahezu ohne jede Wartung. Zu den Nachteilen gegenüber Link-State-Protokollen (z.B. OSPF), zählen die schlechtere Konvergenzeigenschaft und die mangelhafte Skalierbarkeit. [DV07] Für eine detaillierte Beschreibung des Algorithmus sei auf [PD04] verwiesen.

RIP

Despite RIPs age and the emergence of more sophisticated routing protocols, it is far from obsolete. RIP is mature, stable, widely supported, and easy to configure. Its simplicity is well suited for use in stub networks and in small autonomous systems that do not have enough redundant paths to warrant the overheads of a more sophisticated protocol. [Cis06]

Das Routing-Information-Protocol (im Folgenden kurz RIP genannt) ist ein Routing-Protokoll aus der Distance-Vector-Familie. Diesem Algorithmus entsprechend wird die Route zum Ziel durch das Minimum der Kosten über alle Nachbarn bestimmt. Jeder Router führt dazu eine Tabelle mit Routingziel, Distanz zum Ziel (Metrik) und erstem Nachbarrouter auf dem Weg zum Ziel. Alternativen werden nicht gespeichert. Ändert sich die Topologie, muss eine Neuberechnung der entsprechenden Routen stattfinden.

Um überhaupt Informationen austauschen zu können, versendet RIP in regelmäßigen Abständen (Periode von 30 Sekunden) sein Wissen an alle Nachbarn. Jede versandte Nachricht enthält Informationen über das lokale Wissen. Um im Fehlerfall ein Ausbleiben einer Nachricht erkennen zu können, müssen Timer eingeführt werden. Trifft ein Update für einen Zeitraum von 180 Sekunden (Timeout-Timer) nicht ein, geht RIP von einem Ausfall aus und markiert die

Route als unerreichbar und ein weiterer Timer von 120 Sekunden (Garbage-Collect) wird gestartet. Innerhalb dieser Zeit teilt RIP seinen Nachbarn die Unerreichbarkeit mit und nimmt neue Routen zum Ziel an, sofern diese angeboten werden. Ist der Garbage-Collect-Timer abgelaufen und wurde keine neue Route gelernt, wird der Eintrag aus der Tabelle entfernt.

Die im vorigen Absatz erwähnte Unerreichbarkeit wird durch die Metrik 16 repräsentiert. Dies bewirkt zum einen die Begrenzung der Netzgröße auf 15 Hops zwischen den entferntesten Routern, zum anderen begrenzt dies jedoch auch das -nicht gewünschte- Hochzählen der Metrik im Fehlerfall in folgender Beispieltopologie:

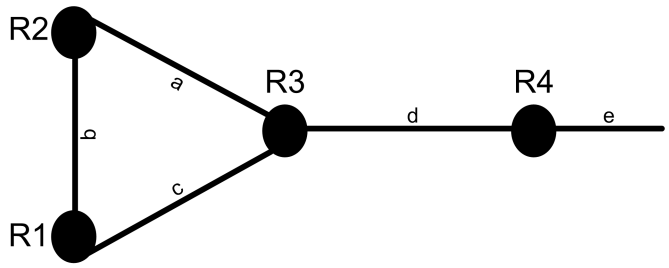


Abbildung 1: Entstehung eines CTI

Ausgangspunkt ist ein bereits konvergiertes Netz, so dass die Router die in Tab. 1 gezeigten Routing-Tabellen vorhalten.

Router R1		
Netz	Metrik	via
a	2	R2
b	1	-
c	1	-
d	2	R3
e	3	R3

Router R2		
Netz	Metrik	via
a	1	-
b	1	-
c	2	R1
d	2	R3
e	3	R3

Router R3		
Netz	Metrik	via
a	1	-
b	2	R2
c	1	-
d	1	-
e	2	R4

Router R4		
Netz	Metrik	via
a	2	R3
b	3	R3
c	2	R3
d	1	-
e	1	-

Tabelle 1: Routing-Tabellen

Tritt folgende Ereignisabfolge auf, kann ein gravierendes Konvergenzproblem (CTI) entstehen:

- Der Router R4 fällt aus und sendet somit keine Updates mehr an R3.
- R3 bemerkt dies nach 180 Sekunden, markiert das Netz e als unerreichbar (e, 16, -) und sendet die Information an seine Nachbarn R1 und R2.
- R2 aktualisiert seinen Eintrag für Netz e zu (e, 16, -).

- R1 sendet jedoch noch ein Update an R2, bevor er von R3 über den Ausfall informiert wird. Aus diesem Update entnimmt R2, dass über R1 noch ein Weg zu Netz e führt. Er aktualisiert seinen Eintrag zu (e, 4 , R1). Ab diesem Zeitpunkt besitzt R2 eine inkorrekte Routing-Tabelle.
- R1 erhält das Update von R3 und aktualisiert seinen Eintrag für Netz e zu (e, 16, -).
- R2 teilt seine neue Information allen Nachbarn mit. Da Netz e für R3 momentan unerreichbar ist, nimmt R3 den neuen Weg über R2 an und aktualisiert seinen Eintrag zu (e, 5, R2). Er nimmt also fälschlicherweise an, dass R2 ihm einen gültigen Weg anbietet.
- Ab diesem Zeitpunkt hat sich im Zyklus zwischen den Routern R1, R2 und R3 eine Abhängigkeit entwickelt, in der R3 einen Weg akzeptiert, der ihn zweimal durchläuft. R3 informiert nun R1 über die neue Erreichbarkeit von Netz e und R1 informiert wiederum R2, usw. Da sich bei jedem Schritt die Metrik um 1 erhöht, würde der Wert bis ins Unendliche hochgezählt werden. In der Praxis endet dies jedoch beim Wert 16.

Dieses Verhalten wird mit Counting-To-Infinity (CTI) bezeichnet. Begründet liegt dies in der Art der Kommunikation. Informationen werden nur zwischen Nachbarroutern ausgetauscht und sind somit für weiter entfernte Ziele *Informationen aus zweiter Hand*, da sie nicht vom Ziel selbst gelernt wurden.

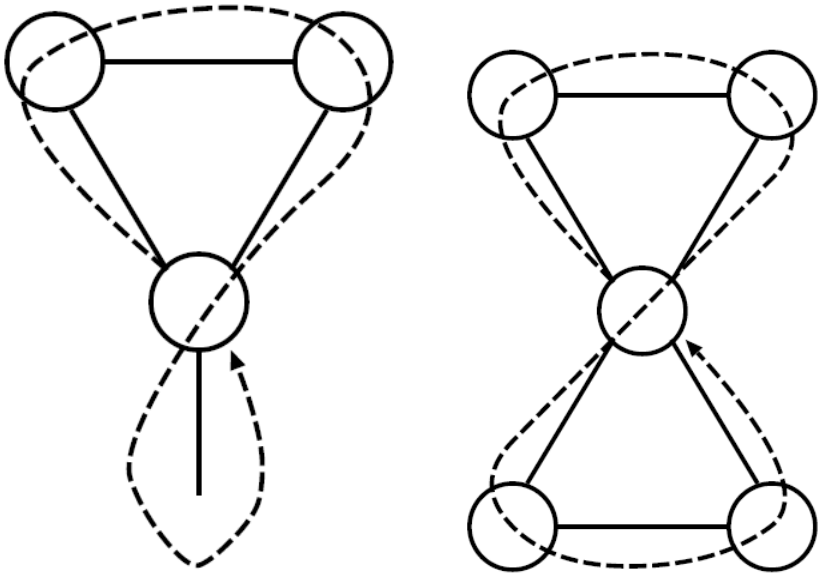
RIP-MTI

RIP-MTI wurde erstmals in [Sch99] als Erweiterung des RIP-Algorithmus und als Lösung des Counting-To-Infinity Problems vorgestellt. Zu diesem Zweck sammelt der MTI Algorithmus Informationen über die Netztopologie. Die Informationen können zu einem groben Abbild des Netzes zusammengefügt werden, so dass jeder Router lokal eine Schleife zwischen 2 seiner Interfaces erkennt. Dieses Verfahren spiegelt sich in der Namensgebung wieder: **RIP with minimal topology information** (kurz RIP-MTI) [Koc05].

Die Arbeitsweise des MTI soll nun kurz vorgestellt werden. Für eine ausführliche Beschreibung des Algorithmus sei auf [Sch99] verwiesen. Bei jedem neu angebotenen Pfad führt der MTI-Algorithmus einen Test auf vorhandene Schleifen durch und lehnt eine Nachricht, die aus einer Schleife stammt (also von ihm selbst abgeschickt wurde), ab. Da so jeder Zyklus unterbochen wird, kann das Counting-To-Infinity Problem nicht mehr auftreten. Der erkennende Router wird mit Source-Router, die durchlaufene Schleife mit Source-Loop bezeichnet. Mit Beweis durch Widerspruch konnte gezeigt werden: Wenn Source-Loops lokal auf jedem Router vermieden werden, können CTIs nicht auftreten. [Sch99]

Da Split Horizon (Informationen nicht an den Nachbarn zurücksenden, von dem sie gelernt

wurden) verwendet wird, konnte weiterhin gezeigt werden, dass nur 2 Typen von Source-Loops ausgeschlossen werden müssen um CTIs zu verhindern: Y-Kombinationen (Schleife mit beliebiger Länge) und X-Kombinationen (2 verbundene Schleifen)[1].



(a) Y-Kombination (b) X-Kombination

Abbildung 2: mögliche Source-Loops

Anhand der Informationen aus eingehenden Updates, legt RIP-MTI auf jedem Router R zwei Tabellen an. Die $mincyc_{i,j}$-Tabelle enthält an der Position i, j die Länge der kleinsten Schleife zwischen den Interfaces i und j. Trifft sowohl über Interface i, als auch über Interface j ein Update für Netz n ein, so gilt: $mincyc_{i,j} = m_i^n + m_j^n - 1$. Die kleinste ermittelte Schleife wird gespeichert. Die $minm_i$-Tabelle enthält die kleinste Schleife zwischen Interface i und einem beliebigen anderen Interface: $minm_i = min(mincyc_{i,j})\ \forall j \in Interface(R),\ j \neq i$.

Wird nach Ausfall einer Route ein Update mit einer Alternativroute empfangen, muss auf Y- und X-Kombination[2] geprüft werden. Annahme: die alte Route zu Netz n führte über das Interface i von Router R und die Alternativroute soll über das Interface j führen.

Y-Kombination: Es handelt sich um eine Y-Kombination, wenn die folgende Bedingung nicht erfüllt ist:

$$minm_j > m_j^n - m_i^n \quad mit\ m_j^n > m_i^n$$

Falls die neue Metrik abzüglich der alten Metrik immernoch gleich groß oder größer als

[1] Die Grafiken in Abb. 3 sind aus [Koc05] entnommen

[2] Die Beschreibung der Y- und X-Kombination ist aus [Wol06] übernommen.

der minimale Schleifenumfang am Interface j ist, muss der Router selbst schon einmal durchlaufen worden sein. Es handelt sich somit um eine Y-Kombination und die Route muss abgelehnt werden.

X-Kombination: Es handelt sich um eine X-Kombination, wenn die folgende Bedingung nicht erfüllt ist:

$$minm_i + minm_j > m_{i,j}^{R,n,R} \; mit \; m_{i,j}^{R,n,R} = m_i^n + m_j^n - 1$$

Ist die Kombination aus alter und neuer Metrik gleich groß oder größer als die Summe der beiden kleinsten Schleifenumfänge, handelt es sich um eine X-Kombination. Der neue Weg hat dann jeden Router schon mindestens einmal durchlaufen und die Route muss abgelehnt werden.

RIP-MTI ist eine Änderung der Implementierung, aber nicht der Kommunikation (Protokoll). Die zusätzlichen Informationen über Schleifen werden ausschließlich aus den, bereits durch das RIP-Protokoll zur Verfügung stehenden, Daten gewonnen. Der RIP-MTI Daemon ist somit kompatibel zu jedem RIP-Router der Quagga-Suite[3].

1.3.2 VNUML

[4] „Virtual-Network User-Mode Linux“ (VNUML) ist ein Tool zur Simulation von Computernetzen. Durch die Verwendung von User-Mode Linux (UML) können virtuelle Rechner in einem Szenario erzeugt werden, die nahezu alle Eigenschaften eines realen Linux-Rechners haben. Dabei wird jeder virtuelle Rechner als ein Prozess (mit eigenem Filesystem etc.) auf einem Hostrechner ausgeführt. Das ermöglicht, entsprechend der Performanz des Hostrechners, den Aufbau und Test fast beliebiger Netzwerktopologien, ohne die dazu erforderliche Hardware selbst zu besitzen. Die virtuellen Maschinen können beliebig konfiguriert werden und in einer Simulation verschiedene Rollen einnehmen (z.B. Router, Webserver, Client, etc).

Welche Simulation ausgeführt wird (Beispiel: jede VM nimmt die Rolle eines RIP-Routers ein), wird definiert durch eine Folge von Kommandos, die auf jedem virtuellen Rechner beim Aufruf einer Startsequenz ausgeführt werden. Somit eignet sich VNUML z.B. zum Testen von verteilter Software oder zur Untersuchung von Netzwerkprotokollen. VNUML besteht aus zwei Komponenten:

VNUML-Sprache Die XML-Sprache von VNUML dient dem Beschreiben von Szenarien.

[3] Quagga Routing Software Suite, www.quagga.net.

[4] Die nachfolgenden Ausführungen wurden vom Verfasser bereits in ähnlicher Weise unter dem Aspekt „verteilte Simulationen mit VNUML“ behandelt, siehe [Keu06].

Eine XML-Datei beschreibt ein Szenario.

VNUML-Parser Der VNUML-Parser baut die Netzwerktopologie entsprechend der XML-Beschreibung mit Hilfe von User-Mode Linux auf. Die komplexen Details von UML werden vor dem Nutzer versteckt.

Der VNUML Simulator dient als Grundlage für die in dieser Arbeit vorgestellten Testumgebung. Ein Netz von RIP-Routern muss schon vor Start der Testumgebung in einer XML-Datei beschrieben, konfiguriert und gestartet sein. Die erforderlichen Voraussetzungen werden in Kapitel 7.1.1 beschrieben. Für weiterführende und detaillierte Ausführungen bezüglich der Verwendung des Simulators sei auf [Vnu07] und [Mue06] verwiesen.

1.3.3 RIP-XT

Die Erweiterung RIP-XT (RIP-e**X**ternally **T**riggered) wurde in der Diplomarbeit „Extern steuerbare Routing-Updates im RIP-Daemon der Quagga-Programmsuite“ [Pae06] entwickelt. Der Daemon des RIP-Prozesses kann mit dieser Erweiterung „von außen“, basierend auf einer Client-Server Struktur, gesteuert werden. „Von außen“ bedeutet in diesem Fall über eine Netzwerkverbindung, die zwingend zwischen Hostrechner und virtueller Maschine existieren muss. RIP-XT erlaubt also, von zentraler Stelle aus Updatereihenfolgen kontrollieren zu können.

Um dies zu ermöglichen wurde der C-Quellcode so abgeändert, dass sich jedes Interface des RIP-Routers entweder im Modus `AUTO` oder `MANUAL` (Defaultwert) befindet. `AUTO` entspricht dabei dem normalen Verhalten des RIP-Protokolls. `MANUAL` erzwingt ein Verhalten, bei dem der RIP-Daemon eigenständig keine Updates aussenden kann, nur der Client kann in diesem Modus den Update-Versand veranlassen.

Die Kommunikation erfolgt über ein Klartext-Protokoll auf Port 5000 nach dem Schema: Client sendet Befehl, Server sendet Antwort. Befehle können z.B. sein: `IF_ALL MODE_MANUAL`, `IF_ETH1 SHOW_IP`, `IF_ALL SHOW_STATUS` oder `IF_ETH0 TRIGGER_UPDATE`. Antworten können z.B. sein: `DONE`, `ERROR` oder `MODE_AUTO` (für eine detaillierte Syntaxbeschreibung siehe [Pae06], S.15ff). RIP-XT ist sowohl als Patch für den RIP-, als auch den RIP-MTI-Daemon aus der Quagga-Suite verfügbar und kann somit zum direkten Vergleich der beiden Protokolle verwendet werden.

Evaluation des RIP-XT

Der XT-Server wurde nach Ende der Entwicklung leicht modifiziert, damit auch die zu Beginn versandten Update-Requests unterdrückt werden. Die neue Version ist auf der Homepage des Entwicklers (*http://www.uni-koblenz.de/~tulkas/diplomarbeit/*) verfügbar.

Voraussetzung für die Verwendung des RIP-XT ist, dass innerhalb der VNUML Simulation die Management-Interfaces aktiviert wurden, damit eine Netzwerkkommunikation vom Host zu jeder virtuellen Maschine möglich ist. Darauf muss beim Erstellen eines Szenarios explizit geachtet werden, besonders in zukünftigen VNUML Versionen, da dort eine andere Art der Host-zu-VM-Kommunikation eingeführt wird, die nicht mehr netzwerkbasiert ist.

Nachteil dieser netzwerkbasierten Verbindung ist der Overhead in der Kommunikation, da die Steuerbefehle in Paketen verpackt, mithilfe virtueller Interfaces, über ein virtuelles Netz geschickt werden. Die dadurch auftretenden Latenzen waren jedoch nicht messbar und können deshalb vernachlässigt werden. Lediglich bei einigen dutzend Befehlen pro Sekunde scheint der Lesepuffer des Servers an seine Grenzen zu stoßen. Es folgt jedoch auch dann eine `ERROR`-Antwort.

Aufgrund der Multicast-Eigenschaft des RIP-Protokoll sind jedoch einige Updatekonstellationen nicht modelliebar. In einem Netz wie rechts abgebildet (3 Router sind über ein Ethernet verbunden), könnte folgendes Verhalten auftreten:
- R3 sendet ein Update auf dem oberen Interface.
- R1 erhält dieses Update, R2 aufgrund von Übertragungsfehlern auf der Leitung jedoch nicht.
- Wenn R2 nun ein Falsch-Update an einen Nachbarn sendet, könnte sich diese Falsch-Nachricht im Netz ausbreiten.

Dieses Verhalten lässt sich jedoch nicht mit der XT-Erweiterung darstellen, da Updates durch den RIP-Daemon an die Multicast-Adresse 224.0.0.9 gesendet werden. Damit erhält -sofern das simulierte Netz einwandfrei funktioniert- jeder, am entsprechenden Ethernet des Interfaces, angeschlossene Router dieses Update. Über die Konfiguration lassen sich zwar bestimmte Nachbarn ausschließen, jedoch nicht dynamisch zur Laufzeit. Hierfür müsste also ein anderer Ansatz, der den Updateversand nicht -oder nicht nur- am ausgehenden Interface steuert, gewählt werden.

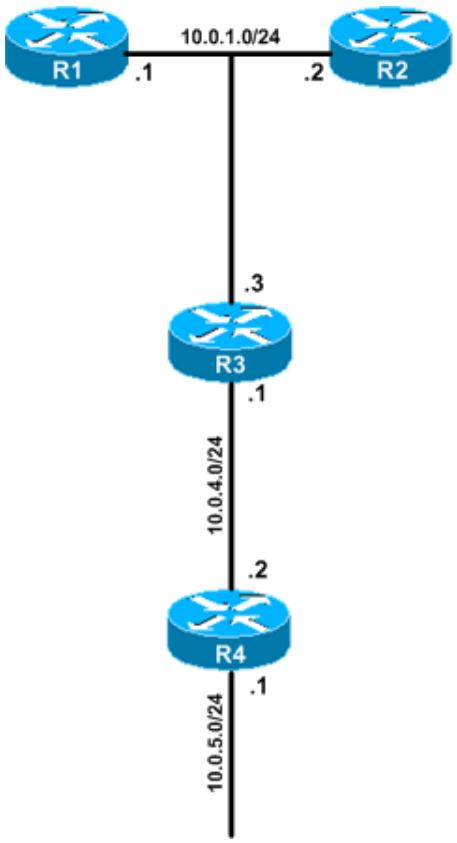


Abbildung 3: NonP2P-Szenario

Trotz der kaum messbaren Latenzen bei virtuellen Netzen gilt für den gewählten Ansatz: Versandzeitpunkt auf dem ausgehenden Interface ist nicht gleich Empfangszeitpunkt auf dem eingehenden Interface. Da das System abhängig von der Prozessumschaltung des Schedulers

ist, kann es in seltenen Fällen vorkommen, dass ein Update vor einem kurz vorher versendeten eintrifft. Dies führt bei besonders kurzen Zeitabständen dazu, dass nicht immer die gewünschten Updatereihenfolgen auftreten.

Ein gewisser Grad an Indeterminiertheit bleibt also vorhanden, trotzdem sind bei passenden Config-Dateien sehr konstante Ergebnisse zu erwarten (ein Test im Y-Szenario (s. Kapitel 7.2) ergab z.B. bei 100 Durchläufen eine Erfolgsquote von 98% für das Auftreten eines CTI bei Verwendung einer entsprechend angepassten Config-Datei). Insgesamt liegt die Erfolgsrate bei bei 90 - 95%.

Der in Java geschriebene XT-Client ist gut implementiert und dokumentiert. Die grafische Oberfläche bietet alle nötigen Grundfunktionalitäten. Deshalb kann die, in dieser Arbeit und der Arbeit von Stefan Lange, entwickelte Testumgebung direkt auf diesem Rahmen aufsetzen.

2 Das Counting-To-Infinity Problem

Dieses Kapitel behandelt im ersten Abschnitt die Ereignisse, die einen CTI auslösen können. Im zweiten Abschnitt werden Ansätze zur Lösung des CTI-Problems, bzw. zur Unterbrechung von Zyklen vorgestellt. Im letzten Abschnitt wird eine obere zeitliche Grenze für die Dauer eines CTI ermittelt.

2.1 Phasen eines CTI

In diesem Abschnitt wird eine Klassifizierung der Ereignisse vor und während eines CTI vorgestellt. Die so enstandene Gruppierung wird im Kapitel 5.2 zur Umsetzung in eine Updatereihenfolge verwendet. Die Phasen beziehen sich auf eine Netztopologie, die einen Zyklus beinhaltet. Eine Route, die von ausserhalb des Zyklus gelernt wurde, fällt aus.

Konvergenzphase Um von einem konsistenten Zustand ausgehen zu können, muss jeder Router jede Route kennen. Insbesondere muss jeder Router die ausfallende(n) Route(n) kennen. Der längste mögliche Weg im Netz bestimmt die Länge dieser Phase. Am Ende ist das Netz in einem konvergenten Zustand.

Ausfallphase Damit eine oder mehrere Routen unerreichbar werden, muss ein Router oder die Verbindungsleitung zu diesem Router ausfallen. Sofern sich das Verhältnis der RIP-Timer nicht ändert, werden, nach dem sechsten nicht erhaltenen Update, die entsprechenden Routen als unerreichbar (Metrik 16) markiert. Der Timer, der dies überwacht, ist standardmäßig auf 180 Sekunden eingestellt.

Ausbreitungsphase Nachdem ein Router den Ausfall bemerkt hat, informiert er, durch Triggered Updates, umgehend seine Nachbarn über den Ausfall. Diese informieren wiederrum ihre Nachbarn. Ein Router erhält diese Information jedoch nicht oder sehr verspätet (Latenzen oder Paketverlust auf der Verbindungsleitung) und geht somit weiterhin von der Korrektheit seiner Information aus.

CTI-Phase In dieser Phase versendet der Router, der zuvor nicht über den Ausfall der Route informiert wurde, Updates an seine Nachbarn. Der Nachbar, der auch Teil des Zyklus ist und die Route schon als unerreichbar übernommen hat, nimmt diesen vermeintlich besseren Weg an und informiert seine Nachbarn. Die Falsch-Information kann nun mehrfach durch den Zyklus laufen.

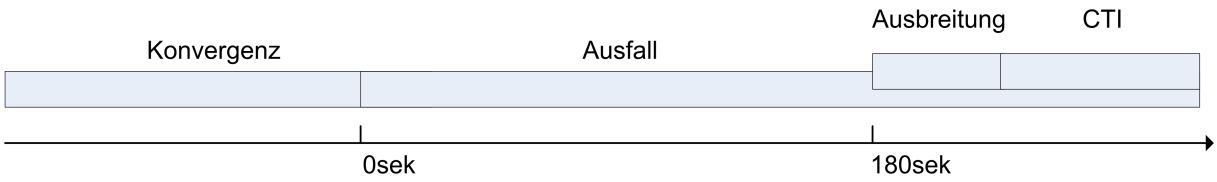


Abbildung 4: Phasen eines CTI

2.2 Lösungen

Schon im RFC 1058 [RFC88] aus dem Jahr 1988, in dem RIP zum Standard wurde, wurde auf das Counting-To-Infinity Problem hingewiesen. Als Lösungen wurden RIP-INFINITY = 16, Triggered Updates und Split Horizon vorgestellt. Ersteres begrenzt dabei das Hochzählen auf den Wert 16, so dass spätestens nach 15 Iterationen Konvergenz auftritt, insbesondere dann, wenn jedes andere Verfahren gescheitert ist. Dies begrenzt aber gleichzeitig auch den längsten Pfad im Netz auf 15 Hops. Triggered Updates beschleunigen die Konvergenz im Fehlerfall und Split Horizon verhindert den CTI zwischen 2 Knoten. Es folgte jedoch im gleichen RFC der Hinweis: *„[..] If triggered updates happen quickly enough, this is very unlikely. However, counting to infinity is still possible.“* Bei ungünstigem Timing zwischen periodischem und getriggertem Update kann sich also weiterhin eine Falsch-Nachricht im Netz ausbreiten.
In diesem Abschnitt werden 4 Lösungen des CTI-Problems kurz vorgestellt und bewertet.

2.2.1 Triggered Extensions

Der RFC 2091 [RFC97] erweitert das RIP-Protokoll unter anderem um Acknowledgements und Retransmission. In Verbindung mit einem Hold-Down-Timer, der startet sobald eine Route unerreichbar wird, soll die Annahme von falschen, nicht mehr aktuellen Informationen komplett unterbunden werden.

Dieser Ansatz wird in [Lin04] als Lösung für das Counting-To-Infinity-Problem vorgeschlagen, ein Beweis wird dort jedoch nicht geführt. Da der Ansatz aufgrund weiterer Funktionalitäten vom ursprünglichen RIP-Protokoll stark abweicht, geht die Kompatibilität zu diesem verloren.

2.2.2 Ad hoc On-Demand Distance Vector (AODV) Routing

Der RFC 3561 [RFC03] aus der Kategorie Experimental beschreib eine Variante des Distance-Vector-Protokolls, bei der jeder Nachricht eine Sequenznummer hinzugefügt wird. Anhand dieser kann ein Router entscheiden, ob er eine Update-Nachricht annehmen oder ablehnen muss.

Jeder Eintrag in der Routing-Tabelle enthält zusätzlich die Sequenznummer des Paketes von dem der Eintrag stammt. Bei Eintreffen einer Update-Nachricht für diesen Eintrag werden die Sequenznummern verglichen. Ist die Sequenznummer kleiner als die in der Routing-Tabelle gespeicherte, so wird die Nachricht verworfen. Ist die Sequenznummer größer, wird der Eintrag in der Routing-Tabelle ersetzt. Das Verfahren wurde für Ad-hoc-Netzwerke entwickelt, die keine fixe Infrastruktur besitzen (z.B. durch Mobile Clients) und muss daher Zyklenfreiheit zu jedem Zeitpunkt garantieren. Ein Netz mit fixer Infrastruktur kann als Spezialfall dieser Ad-hoc-Netzwerke angesehen werden. Das Verfahren ist beweisbar frei von Zyklen und verhindert deshalb das Counting-To-Infinity Problem. [Per01] Techniken wie die Begrenzung der Metrik oder Split-Horizon sind hier nicht mehr nötig. [Lin04]

Dieser Ansatz verhindert auf wirksame Weise jeden Routing-Zyklus und damit auch das CTI-Problem, ist jedoch aufgrund seiner Protokollerweiterung nicht kompatibel zum ursprünglichen RIP.

2.2.3 Enhanced Interior Gateway Routing Protocol (EIGRP)

Das proprietäre EIGRP-Protokoll gehört Cisco. Es ist der heute weit verbreitete Nachfolger von IGRP und wurde von Cisco in den frühen 90-ern vorgestellt. Es eignet sich sowohl für kleine als auch große Netzwerke und ist einfach zu konfigurieren. EIGRP verwendet ein weiterentwickeltes Distance-Vector-Protokoll. Die Verfahren Split Horizon, Poison Reverse und variable Subnetzmasken (VLSM) finden auch hier Anwendung. [Net06] Die Größe des Netzes ist nicht auf 15 Hops beschränkt, sondern kann eine Länge von bis zu 224 Hops erreichen. Folgende Pakettypen werden für die Kommunikation zwischen benachbarten Knoten verwendet: Hello (Multicast), ACK (Unicast), Aktualisierung (Multicast oder Unicast), Query (Multicast), Reply (Unicast) und Request (Multicast oder Unicast). EIGRP verdankt seine Zuverlässigkeit dem Reliable Transport Protocol (RTP), welches für die garantierte und geordnete Zustellung der versendeten Pakete zuständig ist. Cisco bestätigt seinem Protokoll sogar absolute Gleichwertigkeit zu jedem anderen Protokoll: *„The convergence time with DUAL rivals that of any other existing routing protocol.“* [Cis07]
Die für das CTI-Problem interessante Erweiterung ist der in diesem Protokoll verwendete DUAL-Algorithmus: *“The convergence technology [..] employs an algorithm referred to as the Diffusing Update Algorithm (DUAL). This algorithm guarantees loop-free operation at every instant throughout a route computation and allows all devices involved in a topology change to synchronize at the same time.“* [Cis07] Dieser Algorithmus wertet dabei alle von anderen Routern empfangenen Daten aus und berechnet primäre und redundante Netzwerkpfade (sog. plausible Nachfolger). Der Primäre Pfad ist i.Allg. der Pfad mit den niedrigsten Kosten um das Ziel zu erreichen, der redundante Pfad der mit den zweitniedrigsten Kosten. Alle Pfade werden vorge-

halten, aber nur einer dieser Pfade wird auch aktiv genutzt. Somit werden Schleifen automatisch vermieden. [Dua07]

Durch die garantierte Schleifenfreiheit können in Netzen, in denen EIGRP verwendet wird, keine Routing-Zyklen auftreten. Das CTI-Problem kann deshalb nicht auftreten. Das Protokoll kann jedoch nur auf Cisco Geräten verwendet werden. Es ist nicht kompatibel zum ursprünglichen RIP und kann deshalb nicht in gemischten Netzen eingesetzt werden.

2.2.4 RIP-MTI - RIP with Minimal Topology Information

Dieser, an der Uni-Koblenz entwickelte, Algorithmus basiert auf einem Verfahren, welches die Informationen der schon vorhandenen Pakete auswertet. Grundgedanke des Algorithmus ist die Schleifenerkennung zwischen zwei Interfaces. Diese Topologie-Informationen sammelt RIP-MTI und wertet sie immer dann aus, wenn sich das Netz verändert und es zu einer ausfallenden Route einen neuen Weg mit neuer Erreichbarkeit gibt. Diese neuen Routen untersucht der Algorithmus auf Schleifen und entscheidet aufgrund der gesammelten Informationen, ob eine alternative Route akzeptiert werden darf oder ob es sich um eine falsche Route handelt, die bei Akzeptanz und Weitergabe unter Umständen zu einem CTI führen könnte. [Wol06] Das Protokoll und die übertragene Nachricht wird dabei nicht verändert.

RIP-MTI ist somit kompatibel zur ursprünglichen Version des RIP und kann deshalb in gemischten Netzen mit dem ursprünglichen RIP-Protokoll eingesetzt werden. Es muss mindestens auf ausgewählten Knoten (Source-Router) installiert werden. Der Aufwand dafür ist jedoch gering, da keine Konfigurationsänderungen am bestehenden Netz vorgenommen werden müssen. Der Algorithmus wurde jedoch bis jetzt nur in Simulationsumgebungen getestet.

2.2.5 Zusammenfassung

Zusammenfassend gilt die Regel: bewiesen sichere Lösungen des CTI-Problems sind nicht kompatibel zum ursprünglichen RIP und kompatible Lösungen sind (noch) nicht bewiesen sicher. Das AODV- und das EIGRP-Protokoll lösen das CTI-Problem, da in Netzen, die diese Protokolle verwenden -auf logischer Ebene- keine Zyklen existieren. Beide sind jedoch nicht kompatibel zum ursprünglichen RIP. Das RIP-MTI Protokoll ist kompatibel, ein Beweis über dessen Korrektheit und Vollständigkeit existiert jedoch nicht.

2.3 Worst-Case-Time

Dieser Abschnitt behandelt die Frage nach zeitlichen Grenzen für einen CTI und versucht diese Grenzen, durch Abschätzung nach oben, mit konkreten Werten zu belegen.
Aus dem RFC 1812 [RFC95]:

```
[..]
A router MUST send a triggered update when routes are deleted or
their metrics are increased.
[..]
(1) When a router sends a triggered update, it sets a timer to a
    random time between one and five seconds in the future.  The
    router must not generate additional triggered updates before
    this timer expires.
```

Annahme:
Latenzen auf Knoten und im Netz werden vernachlässigt
Definition:
Durchlauf = falsche Nachricht ist einmal im Zyklus über alle Knoten gelaufen
Abstand a = Anzahl Hops, die das ausfallende Netz vom Zyklus entfernt ist
R_z = ein Router innerhalb des Zyklus, R_f = Router der das Falsch-Update aussendet
Gegeben:
Netz mit einem Zyklus von $n \geq 2$ Knoten. Die ausfallende Route hat vom Router R_f aus eine Metrik von k. Es können nur Zyklen mit einer Länge $n \leq (15 - Abstand\ a)$ betrachtet werden, da in größeren Zyklen kein CTI entstehen kann, da die Nachricht sonst schon RIP-INFINITY erreicht, bevor sie zum ersten Mal an einem bereits durchlaufenen Knoten ankommt (dieser Fall kann auch von RIP-MTI nicht behoben werden).

Die Betrachtung beginnt zum Zeitpunkt $T = 0$, an dem der Router R_f das Falsch-Update an einen Nachbarn weiter gibt. Bei n Knoten im Zyklus dauert ein Durchlauf D_i (Beginn dieses Durchlaufs bis zum Beginn des nächsten) von getriggerten Updates (s.o.) also mindestens 1 Sekunde und maximal $(5 \cdot n)$ Sekunden: $D_i \in [1,\ 5 \cdot n]$. Bei jedem Durchlauf i erhöht sich die Metrik insgesamt um n, d.h. auf jedem Router R_z (mit Metrik $m_0 = k$) gilt nach dem i-ten Durchlauf: Metrik $m_i = k + (i \cdot n)$. Die Anzahl der möglichen Durchläufe ist also direkt abhängig vom Abstand zum ausfallenden Netz und der Zykluslänge.

Im ersten Schritt wird die Anzahl der Durchläufe i betrachtet, die nötig sind um mindestens den Router R_f bis auf eine Metrik von 16 hochzuzählen. Dafür muss gelten:
$k + (i \cdot n) \geq 16 \ \Rightarrow\ i \geq \frac{(16-k)}{n} \ \Rightarrow\ i = \left\lceil \frac{(16-k)}{n} \right\rceil$.
Es können demnach $i = \left\lceil \frac{(16-k)}{n} \right\rceil$ Durchläufe stattfinden, bevor RIP-INFINITY (16) auf R_f

erreicht ist.

Im Zyklus kennen nun alle Router hinter dem, der als erster die Metrik 16 erhält, die richtige Metrik von 16. Es fehlen aber noch die, die im letzten Durchlauf auf Werte ≤ 15 hochgezählt wurden. Das sind noch $r = (15 - k)\ mod\ n$ fehlende Router. Da jetzt nach obiger Berechnung für i mindestens ein Router, und zwar R_f, die Metrik 16 kennt, liegt das Ergebnis für die Anzahl der noch fehlenden Router -wie zu erwarten- innerhalb von $[0\ \ldots\ (n-1)]$.
r hat jedoch einen kleineren Wert für den Spezialfall, dass die richtige Nachricht der falschen hinterherläuft (mit Abstand e). In diesem Fall lernen „hinter" dem Falsch-Update sukzessive alle Router die richtige Metrik. Der Rest nach dem letzten Durchlauf entspricht dann nur noch e, wobei $e \leq r - 1$ gilt. Da für e keine konkreten Werte vorliegen, lässt sich damit nicht allgemein rechnen, für die Worst-Case Abschätzung nach oben ist das jedoch unerheblich, da r immer größer ist.

Unter der Annahme, dass die Wartezeit auf jedem Router den Maximalwert von 5sek beträgt, ergibt sich für die Gesamtzeit bis zur Konvergenz: ((#Durchläufe · #Knoten)$\cdot 5sek$)+(#restliche Knoten $\cdot 5sek) = (i \cdot n) \cdot 5 + r \cdot 5$. Also kann ein durch TriggeredUpdates erzeugter CTI in einem Netz mit einem n-Knoten Zyklus nur eine Maximaldauer (i und r eingesetzt)

$$D_{wcet} = 5sek \cdot \left(n \cdot \left\lceil \frac{(16-k)}{n} \right\rceil + (15-k)\ mod\ n \right)$$

$$\Leftrightarrow\ D_{wcet} = \left(5 \cdot n \cdot \left\lceil \frac{(16-k)}{n} \right\rceil + 5 \cdot ((15-k)\ mod\ n) \right) sek \tag{1}$$

haben. Im besten Fall werden alle Updates sofort getriggert und jeder Durchlauf dauert nur eine Sekunde = minimale Wartezeit nach einem getriggerten Update (D_i = 1, s.o.):

$$D_{bcet} = \left\lceil \frac{(16-k)}{n} \right\rceil = i\ sek \tag{2}$$

Als Nebenprodukt lässt sich die Anzahl A der Updates berechnen, die nach dem Versenden eines Falsch-Updates bis zur vollständigen Konvergenz notwendig sind:

$$A = n \cdot \left\lceil \frac{(16-k)}{n} \right\rceil + ((15-k)\ mod\ n) = i \cdot n + r \tag{3}$$

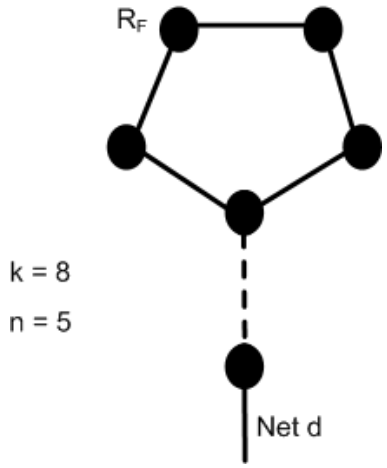
Die Dauer eines CTI liegt also immer innerhalb des Intervalls $D_{CTI} \in [D_{bcet}, D_{wcet}]$, jedoch keinesfalls gleichverteilt. Um überhaupt wesentlich schlechter als D_{bcet} zu sein, müssen dauerhauft Updates bezüglich anderer Routen kurz vor dem CTI Update eintreffen, damit der Timer neu startet. Weiterhin beträgt dann die Wahrscheinlichkeit, dass jedesmal 5 Sekunden gewartet

werden muss nur $(\frac{1}{5})^A$. Für das obige Beispiel ergibt sich unter den genannten Voraussetzungen und das überhaupt ein CTI auftritt, folgende Wahrscheinlichkeit W für die Worst-Case-Execution-Time: $W = (\frac{1}{5})^{12} = \frac{1}{244140625}$.

Folgende Werte ergeben sich für die bekannten Y und X Szenarien:
Y-Szenario: (3 Knoten Zyklus, ausfallendes Netz mit Metrik k=2):
CTI Zeitspanne D_{CTI} = 5 - 80 Sekunden, Anzahl Updates bis Konvergenz: A = 16
X-Szenario: (3 Knoten Zyklus, ausfallendes Netz mit Metrik k=3):
CTI Zeitspanne: D_{CTI} = 5 - 75 Sekunden, Anzahl Updates bis Konvergenz: A = 15

2.3.1 Beispiel

R_f sendet das Falsch-Update zum Zeitpunkt $T = 0$ an den rechten Nachbarn (vorher wurde die Nicht-Erreichbarkeit von Net d schon propagiert). Bis Net d auf R_f die Metrik 16 hat, ist das Falsch-Update $i = \left\lceil \frac{(16-8)}{5} \right\rceil = 2$ mal durch den Zyklus gelaufen. Es fehlen danach noch $r = (15 - 8)\ mod\ 5 = 2$ Router die noch nicht die Metrik 16 haben. Insgesamt sind also $5 \cdot 2 + 2 = 12$ Updates notwendig, bis alle Router die (richtige) Metrik 16 kennen.
Im schlimmsten Fall würde das 5sek · 12 = 60 Sekunden dauern, im besten Fall jedoch nur 2 Sekunden.



2.3.2 Allgemeine Betrachtung

Die folgende Grafik zeigt eine vollständige Darstellung aller möglichen Zyklen bei einem minimalen Abstand $a = 2$. Es gibt also nur Zyklen mit der Länge $n \in [2, 3, \ldots, 13]$. Vergrößert sich der Abstand a, sind nur noch entsprechend geringere Zyklenlängen möglich. Die gepunkteten Linien deuten die möglichen Spezialfälle von Zyklen an: Zyklus zwischen 2 Routern (s. Kapitel 7.8) und Zyklus läuft nicht über den Source-Router (s. Kapitel 7.7).

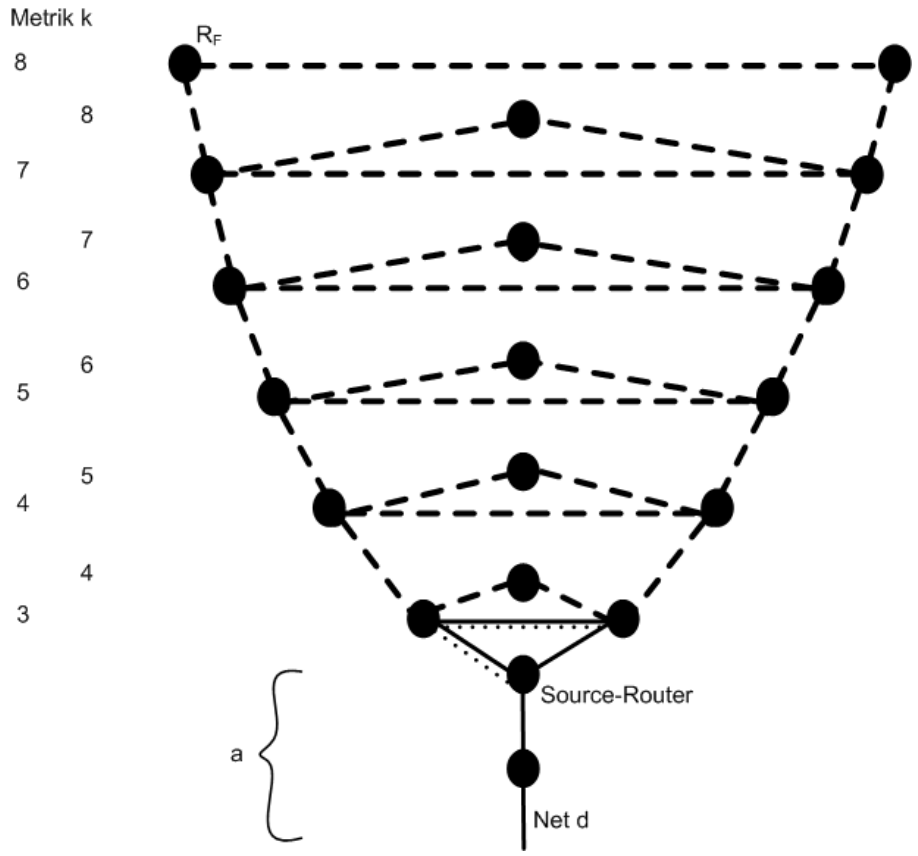


Abbildung 5: mögliche Zyklen

Die folgende Tabelle listet einige ausgewählte Berechnungen auf. R_f ist dabei jeweils immer der Router links außen (aufgrund der Symmetrie keine Beschränkung):

i	**k**	**n**	**r**	**A**	D_{wcet} (sek)	D_{bcet} (sek)
1	8	13	7	20	100	1
1	7	12	8	20	100	1
1	7	11	8	19	95	1
1	6	10	9	19	95	1
2	6	9	0	18	90	2
2	5	8	2	18	90	2
2	5	7	3	17	85	2
2	4	6	5	17	85	2
3	4	5	1	16	80	3
4	3	4	0	16	80	4
5	3	3	0	15	75	5
7	3	2	0	14	70	7

Tabelle 2: ausgewählte Berechnungen

3 Anforderungsanalyse

Die hier aufgelisteten Anforderungen sind auf den im Rahmen dieser Arbeit entstandenen Teil der Testumgebung „XTPeer“ bezogen. Der Kern der Arbeit ist mit der Bedingung „muss“, weitere Anforderungen mit der Bedingung „soll“ und Anforderungen, die als Vision gesehen werden können, mit „kann“ ergänzt.
Die Aufteilung in funktionale Anforderungen, technische Anforderungen, den Anforderungen an die Benutzerschnittstelle sowie den qualitativen Anforderungen ist aus [Rup04] entnommen. Der Glossar findet sich unter Terminologie im Kapitel 1.2.

3.1 Funktionalität

Abbilder

- Ein Abbild muss alle Updateereignisse in einem Netz für einen bestimmten zeitlichen Rahmen festlegen.
- Ein Abbild muss reproduzierbar sein.
- Ein Abbild muss in einer .config Datei gespeichert werden und heißt „Config“.
- Eine Config besteht aus Plain-Text im ISO-Format.
- Die Syntax der Config sollte durch eine allgemeine Beschreibung festgelegt sein.
- Eine Config muss automatisch erzeugbar sein.
- Eine Config muss den Namen aller Router, die Periode und den Forecast enthalten.

Generator

- Der Generator muss Abbilder einlesen, verarbeiten und ausführen können.
- Der Generator muss in 2 Modi arbeiten können: Ausführung eines fest vorgebenen Forecasts und Ausführung eines komplett zufälligen Forecasts.
- Der Generator sollte in einem weiteren Modus arbeiten können: Ausführung eines vorgebenen Forecasts, jedoch zufällige generierung der kritischen Zeilen.
- Der Generator sollte in einem intelligenten Modus arbeiten können: Nach Analyse des vorhandenen Netzes sollten automatisch eine oder mehrere Forecasts generiert und ausgeführt werden.
- Dem Generator müssen über Parameter -die Ausführung betreffende- Werte übergeben werden können.

- Die Anzahl der Wiederholungen muss über einen Parameter einstellbar sein.
- Die Periode muss automatisch ermittelt werden oder -falls nicht bekannt- über einen Parameter einstellbar sein.
- Die Zeit, die das Netz nach einem Durchlauf im Modus AUTO verweilt, muss über einen Parameter einstellbar sein.
- Für die möglichen Werte, die bei einer zufälligen Ausführung auftreten können, sollten vor der Ausführung Wahrscheinlichkeiten festgelegt werden können.
- Der Generator muss nach jedem Durchlauf per Hand gestoppt werden können, um eine Ausführung vorzeitig zu beenden.
- Der Generator kann idealerweise zu jedem Zeitpunkt -auch während eines Durchlaufsgestoppt werden.
- Zur Ausführung sollte der Generator für jeden vorhandenen Router im Netz einen Thread generieren, der das zeitlich korrekte Versenden der XT-Befehle übernimmt.
- Bei teilweise zufällig, komplett zufällig und intelligent generierten Forecasts muss, sofern ein CTI aufgetreten ist, der entsprechende Forecast automatisch in einer Datei (mit der Syntax einer Config-Datei) abgespeichert werden.
- Der Ort der Speicherung dieser Dateien sollte frei wählbar sein.

sonstige Funktionen

- Der Generator muss in die Testumgebung „XTPeer“ integriert sein.
- Die Testumgebung muss CTIs erkennen und sichtbar machen.
- Ein geladenes Netz muss jederzeit per Hand zwischen den Modi `AUTO` und `MANUAL` umgestellt werden können.

3.2 Technik

Programmiersprache

- Die Testumgebung muss in Java imlementiert werden, da auf dem schon vorhandenen Rahmen des XT-Client (siehe Kapitel 1.3.3) aufgesetzt wird und dieser in Java implementiert ist. Die Plattformunabhängigkeit von Java ist nicht entscheidend, da das Programm nur auf Linux-Rechnern gestartet wird.

3.3 Benutzerinteraktion

Benutzergruppe

- Die Testumgebung muss von den Mitgliedern der Arbeitsgruppe Rechnernetze verwendet werden können. Weitere Benutzer sind nicht vorgesehen, da es sich um ein internes Projekt handelt.

Benutzerschnittstelle

- Die Benutzerschnittstelle sollte graphisch umgesetzt werden.
- Die Steuerung des Generators sollte in die GUI integriert werden.
- Debug-Ausgaben sollten möglich sein.

3.4 Qualität

Zuverlässigkeit

- Die Testumgebung muss im Rahmen der gewünschten Genauigkeit korrekt funktionieren.
- Die Testumgebung kann ausfallsicher und robust gegenüber Fehlern erstellt werden.

Wartbarkeit

- Die Simulationsumgebung muss mittels Tests auf ihre Funktion getestet werden.
- Die graphische Benutzerschnittstelle, die Dokumentation des Codes und die Bezeichner im Code müssen in Englisch gehalten sein.

Portabilität

- Die Testumgebung muss sich auf einen Einsatz auf Linux-Rechnern beschränken. Durch die Verwendung von Java ist die Umgebung zwar auch auf anderen Platformen lauffähig, ist dort jedoch nutzlos, da das darunterliegende VNUML System fehlt.

Effizienz

- Die Testumgebung sollte 100 Test-Durchläufe für ein beliebiges Netz durchführen und auswerten können.
- Der Generator sollte im Rahmen der Genauigkeit (100ms) zuverlässig und vorhersagbar Ereignisse auslösen.

Benutzerfreundlichkeit

- Ein mit dem Thema vertrauter Benutzer sollte innerhalb einer Stunde in der Lage sein, die Funktionsweise zu verstehen und einen eigenen Test durchführen zu können.

4 Implementierung

In diesem Kapitel werden ausgewählte Code-Abschnitte der Implementierung, die den Kern dieser Arbeit bilden, vorgestellt und erläutert. Eine vollständige Dokumentation befindet sich im doc-Verzeichnis auf der beiliegenden CD. Es empfiehlt sich die Kapitel 5 und 6 zuvor zu lesen, da darin die hier implementieren Verfahren erläutert werden.

4.1 Klassendiagramm

Das vereinfachte Klassendiagramm in Abb. 6 beschreibt die Zusammenhänge zwischen den Klassen, die den Kern dieser Arbeit bilden.

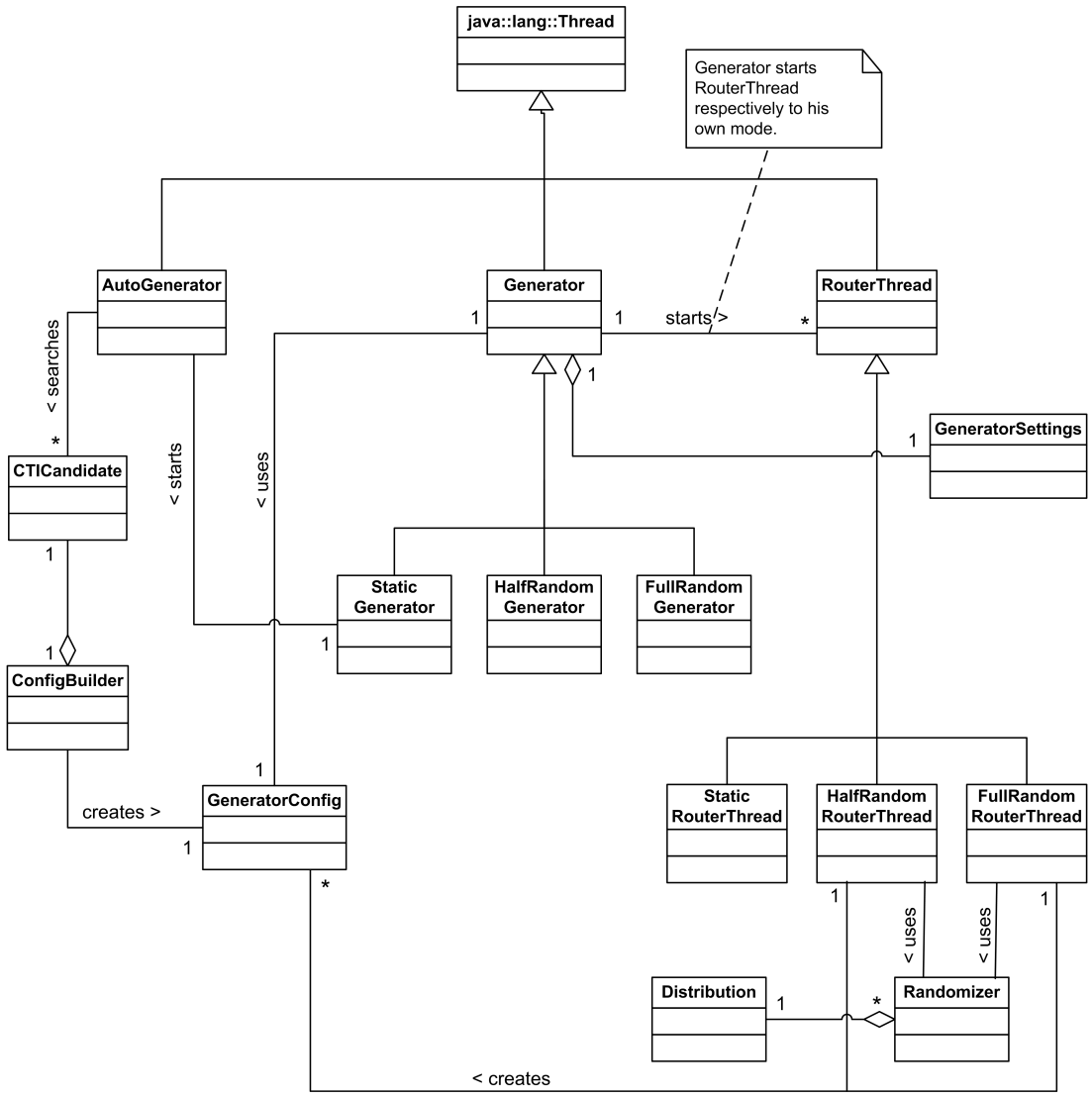


Abbildung 6: Klassendiagramm

4.2 Die Klasse Generator

Die Klasse Generator erweitert die Klasse Thread und stellt Hilfsmethoden für alle abgeleiteten Klassen (StaticGenerator, HalfRandomGenerator, FullRandomGenerator) zur Verfügung. Jede abgeleitete Klasse startet und verwaltet Instanzen der Klasse RouterThread. Stellvertretend wird hier die Methode startRouterThreads(..) der Klasse StaticGenerator vorgestellt.

Der übergebene Parameter `generatorconfig` enthält alle notwendigen Informationen (Routernamen, Periode, Forecast). Für jeden XTServer (ein XTServer entspricht einem RIP-Router) wird ein Objekt der Klasse RouterThread erzeugt. Diesem wird der XTServer selbst, der Forecast und weitere Einstellungen übergeben. Die Threads werden im globalen Vektor `srt` verwaltet.

```
1  /**
2   * Starts the a new thread for each router.
3   * @param generatorconfig
4   */
5  public void startRouterThreads(GeneratorConfig generatorconfig){
6   Vector<XTServer> serverlist = Utilities.
7     getDistinctXTServer(gui.getXtServerTable());
8   srt.clear(); //cleans the list of currently started RouterThreads
9
10  for(Iterator iter = serverlist.iterator(); iter.hasNext();){
11    XTServer server = (XTServer)iter.next();
12
13    //create a new routerthread with params defined in config
14    RouterThread routerthread =
15      new StaticRouterThread(server,
16        (Vector<String[]>) generatorconfig.getConfigTable().
17          get(server.getName().toLowerCase()), settings);
18    routerthread.setName(server.getName());
19    routerthread.start();
20    srt.add(routerthread);
21  }
22 }
```

4.3 Die Klasse RouterThread

Auch die Klasse RouterThread erweitert die Klasse Thread. Die abgeleiteten Klassen implementieren je eine Variante der run()-Methode.

Die Methode getInterfacesToTrigger(..) ermittelt für einen bestimmten Zeitpunkt die Interfaces, über die ein Update versendet werden muss. Dazu wird die Liste aller -im Forecast eines Routers vorkommenden- Interfaces durchlaufen. Ein Interface wird in die Ergebnismenge übernommen, falls der dort gefundene Wert dem aktuellen Zeitpunkt entspricht (z.B. wenn zum Zeitpunkt 1.1 Sek., der Wert „1.1“ im Forecast gefunden wird). Die Abfrage erfolgt dabei in Schritten von Zehntelsekunden und wird daher durch die aufrufende Methode alle 100ms erneut abgefragt. Die Variable `pointoftime` wird dabei jeweils um den Wert 1 erhöht.

```
1   /**
2    * Gets the interfaces that have to be triggered at this
3    * point of time (e.g. 0 or 1 or 2 ...).
4    * As pointoftime can never be greater than periodic_update_time
5    * all values greater than periodic_update_time are automatically
6    * ignored. Correct values have to be ensured by the config-file.
7    * @param param
8    * @param pointoftime
9    * @return a Vector of the names of all interfaces that have to
10   * be triggered at this point of time.
11   */
12  public synchronized Vector<String> getInterfacesToTrigger(
13  String[] param, int pointoftime){
14   Vector<String> interfacestotrigger = new Vector<String>();
15   double t = ((double) pointoftime/10);
16   String time = new String(String.valueOf(t));
17
18   for(int i=0; i<param.length; i++){
19     if(param[i].equals(time)) {
20       interfacestotrigger.add("eth"+(i+1));
21     }
22   }
23   return interfacestotrigger;
24  }
```

Im letzten Durchlauf (letzte Zeile des Forecast) wird mithilfe der Methode getMaxDelay(..) der höchste numerische Wert dieser Forecastzeile ermittelt. Die Werte „x“ und „a“ werden ausgeschlossen und die längste Verzögerung wird zurückgegeben. Wird kein numerischer Wert gefunden wird 0 zurückgegeben, da das Umschalten in den Modus `AUTO`, sofort ausgeführt werden kann.

```
25  /**
26   * This method gets the greatest value that param contains.
```

```
27     * @param param contains the triggers
28     */
29    public static synchronized double getMaxDelay(String[] param){
30     double compare = 0;
31     for (int i = 0; i < param.length; i++)
32       if (!param[i].equalsIgnoreCase("x") && !param[i].
33           equalsIgnoreCase("a"))
34         if (Double.valueOf(param[i]) > compare) compare = Double.
35             valueOf(param[i]);
36     return (compare * 10.0);
37    }
```

Die Methode checkForAutoAndSetMode(..) wird nach der längsten Verzögerung der letzten Periode aufgerufen. Kommt das Suffix „a“ vor, werden 3 Schritte vollzogen. Alle Interfaces dieses Routers (ausser das Management-Interface eth0) werden in den Modus `AUTO` versetzt. Dazu wird über alle Interfaces iteriert und deren Methode setMode(..) aufgerufen. Der Thread schläft nun für die durch den Benutzer definierte Zeit `auto_time`. In dieser Zeit versendet der Router ohne Beeinflussung von außen seine Updates. Nach Ablauf dieser Zeit wird der Modus aller Interfaces erneut gewechselt, um die weitere Steuerung der Updates durch das Programm zu ermöglichen.

```
38    /**
39     * Checks if the suffix "a" in the config-file occured.
40     * If yes it switches all interfaces to MODE_AUTO, then waits
41     * for auto_time seconds and then switches all interfaces back
42     * to MODE_MANUAL.
43     * @param param contains the triggers
44     */
45    public synchronized void checkForAutoAndSetMode(String[] param){
46     if(param[param.length-1].trim().equals("a")){
47
48      /** set all interfaces to MODE_AUTO */
49      for(Iterator iter = xtserver.getInterfaceIterator();
50      iter.hasNext();) {
51       NetworkInterface nif = (NetworkInterface) iter.next();
52       if (! nif.getName().equals("eth0")) {
53        try {
54             nif.setMode(NetworkInterface.MODE_AUTO);
55       } catch (XTServerException e) {e.printStackTrace();}
56       }
```

```
57      }
58
59
60      /** wait auto_time milliseconds */
61      try{
62      Thread.sleep((auto_time));
63      } catch (InterruptedException ie) {ie.printStackTrace();}
64
65
66      /** set all interfaces to MODE_MANUAL */
67      for(Iterator iter = xtserver.getInterfaceIterator();
68      iter.hasNext();) {
69       NetworkInterface nif = (NetworkInterface) iter.next();
70       if (! nif.getName().equals("eth0")) {
71        try {
72              nif.setMode(NetworkInterface.MODE_MANUAL);
73        } catch (XTServerException e) {e.printStackTrace();}
74       }
75      }
76
77     }
78    }
```

4.4 Die Klasse Randomizer

Die Klasse Randomizer stellt Methoden zur zufallsbasierten Erzeugung von String-Arrays zur Verfügung. Wahrscheinlichkeiten können über eine Wahrscheinlichkeitsverteilung definiert werden. Die Methode normalize(..) normalisiert diese Verteilung auf Werte zwischen 0 und 1. Dazu wird der größte Wert der Verteilung ermittelt und anschließend alle Elemente durch diesen Wert geteilt.

```
1    /**
2     * Normalizes the given array of double values to
3     * values between 0.0 and 1.0.
4     * @param distribution
5     * @return normalized double array
6     */
7    private double[] normalize(double[] distribution){
8      double[] result = new double[distribution.length];
```

```
9       double max = 0.0;
10      for(int i=0; i<distribution.length; i++){
11       if(distribution[i] >= max) max = distribution[i];
12      }
13      for(int i=0; i<distribution.length; i++){
14       result[i] = (distribution[i]/max);
15      }
16      return result;
17    }
```

Die Methode randomizerWithDistribution() unterscheided zwischen Wahrscheinlichkeitsverteilungen mit gleichen und verschiedenen Werten. Sind alle Werte gleich (checkIfAllSame() gibt `true` zurück) kann jeder Zufallswert -unabhängig- durch die Funktion nextInt() der Klasse Random erzeugt werden (`else`-Fall). Liegt jedoch eine spezifische Verteilung vor, muss jeder Wert mit seiner entsprechenden Wahrscheinlichkeit erzeugt werden. Dazu wird innerhalb einer do-while-Schleife zufällig ein Element ausgewählt (`x`) und eine zufällig erzeugte Vergleichswahrscheinlichkeit (`compare`) bestimmt. Die Schleife wiederholt sich, bis die Vergleichswahrscheinlichkeit zwischen 0 und der durch die Verteilung vorgegebenen Wahrscheinlichkeit (`dist[x]`) liegt.
Damit kann jeder Wert vorkommen, jedoch nur genau mit der gewünschten Wahrscheinlichkeit. Entspricht der ermittelte Wert dem höchstmöglichen, wird der String „x“ zurückgegeben (der höchste Wert entspricht der Periode selbst und wäre somit gleichzeitig der Anfang der nächsten Periode), in allen anderen Fällen wird der Wert selbst, in Form eines String, zurückgegeben.

```
18    /**
19     * Randomizes between '0.0' and 'x'.
20     * But every selected number has a special probability
21     * which is given in the distribution array.
22     */
23    private String randomizerWithDistribution(){
24     Integer x = 0;
25     double compare;
26     double[] dist = normalize(d.toArray());
27
28     //only if values have not all the same value,
29     //because then it would be a standard distribution
30     if(!checkIfAllSame()){
31
32      //choose an element as long as a random weight is less or
33      //equal to the the weight according to this element
```

```
34      do{
35          x = (new Random().nextInt(d.getRows()));
36
37          //gets a random value between 0.0 and 1.0 in 0.1 steps
38          compare = (Math.floor(Math.random()*10)) / 10;
39      } while (compare >= dist[x]);
40
41      if (x == (d.getRows()-1))return new String("x");
42       else return
43        (new String(new Double((double) x/10).toString()));
44     } else {
45             x = (new Random().nextInt(d.getRows()));
46             if (x == d.getRows()-1) return new String("x");
47              else return
48               (new String(new Double((double)x/10).toString()));
49       }
50   }
```

Die Methode randomizeArray(..) benutzt randomizerWithDistribution() um ein übergebenes String-Array mit Zufallswerten zu füllen. Enthält das übergebene Array an letzter Position den Wert „a“, muss dieser unverändert zurückgegeben werden.

```
51   /**
52    * Randomizes the given array
53    */
54   public String[] randomizeArray(String[] x){
55    String[] result = new String[x.length];
56     for (int i = 0; i<x.length; i++) {
57       result[i] = randomizerWithDistribution();
58     }
59    //if the postfix a for MODE_AUTO is found this will not be changed
60    if (x[x.length-1].equals("a")) result[result.length-1] = "a";
61    return result;
62   }
```

4.5 Die Klasse ConfigBuilder

Die Klasse ConfigBuilder wird durch den AutoGenerator aufgerufen und erzeugt für jeden CTI Kandidaten (s. Kapitel 6.4.2) eine Config. Der darin enthaltene Forecast wird, den Phasen ei-

nes CTI entsprechend (s. Kapitel 2.1), angepasst. Zuvor wird eine, mit dem Wert „0“ gefüllte, Standard-Config erzeugt, in der nacheinander die Elemente an den entsprechenden Positionen geändert werden: Ausfall eines Links, Verlust eines Updates, Versand eines Falsch-Updates und Ausbreitung der richtigen Metrik.

```
1  /**
2   * Alters the config based on the remotepath
3   * informations, so that it could lead to a CTI.
4   */
5  public void createCtiConfig(){
6   createLinkFail();
7   createUpdateLost();
8   createFalseUpdateSend();
9   createPropagateCorrectMetric();
10 }
```

Die Methode createLinkFail() ändert die Werte aller Interfaces des Routers, der ausfällt, in jeder Zeile nach der Konvergenzphase, auf den Wert „x“. Nach der Konvergenzphase sendet dieser Router also keine Updates mehr.

```
11 /**
12  * Changes elements in the configtable, so that the
13  * lost router sends no more updates.
14  */
15 public void createLinkFail(){
16  XTServer xts = c.getLostRouter();
17
18  for(int j = CONVERGENCE_LINES; j<lines ; j++){
19   for(int i = 0; i<xts.getInterfaceCount()-1; i++){
20     changeElement(xts,j,i,"x");
21   }
22  }
23 }
```

Die Methode createUpdateLost() bewirkt den Verlust eines Updates auf dem Loose-Router. Das Interface, welches zum False-Router führt, erhält für die letzten zwei Zeilen des Forecasts den Wert „x“. An dieser Stelle könnte auch anstatt eines Verlustes eine Verzögerung modelliert werden. Da ein Verlust immer einen CTI auslösen kann, wurde hier dieser Weg gewählt.

```
24 /**
25  * Changes one element in the configtable, so that the correct
26  * update which the false router should learn is not send.
```

```
27      */
28     public void createUpdateLost(){
29      XTServer xts = c.getLooseRouter();
30      int line1 = lines -1 ; //this is the cti line
31      int line2 = lines -2 ;
32      int interfacenumber = c.getInterfaceNumberToNext(xts)-1;
33
34      changeElement(xts,line1,interfacenumber,"x");
35      changeElement(xts,line2,interfacenumber,"x");
36     }
```

Um das Falsch-Update im Netz zu verbreiten wird auf dem False-Router, mithilfe der Methode createFalseUpdateSend(), ein Update mit einer Verzögerung von 1 Sekunde versendet. Diese Verzögerung garantiert, dass sich die benachbarten Router bereits im Modus `AUTO` befinden und somit eine möglichst realitätsnahe Bearbeitung des Falsch-Updates erfolgt. Dazu wird das Interface gewählt, welches im Pfad zum nächsten Router führt.

```
37     /**
38      * Changes one element in the configtable, so that the wrong
39      * update is send out by the false router.
40      * Heart of the CTI.
41      */
42     public void createFalseUpdateSend(){
43      XTServer xts = c.getFalseRouter();
44      int line = lines -1 ; //this is the cti line
45      int interfacenumber = c.getInterfaceNumberToNext(xts)-1;
46      String delay = new String("1");
47
48      changeElement(xts,line,interfacenumber,delay);
49     }
```

Die Methode createPropagateCorrectMetric() sorgt für die richtige Ausbreitung der Metrik 16 in der Ausbreitungsphase (vorletzte Zeile). Alle Router die im Pfad nach dem False-Router vorkommen, müssen die Metrik 16 lernen, damit sie überhaupt das Falsch-Update annehmen. Die Methode kehrt sofort zurück, wenn keine weiteren Router existieren (kleinster Zyklus). Jeder Router erhält auf dem Interface, welches zum nächsten Router in Richtung False-Router führt, den Delay-Wert `delay`. Dieser Wert erhöht sich pro Router um die Konstante `STEP`, die innerhalb der Klasse mit 0.5f initialisiert wird.

```
50     /**
51      * Changes elements in the configtable, so that the correct
```

```
52      * updates are learned by all routers in the loop except
53      * the false router.
54      */
55     public void createPropagateCorrectMetric(){
56      Vector<XTServer> servers = c.getRoutersBehindFalseRouter();
57      if(servers.isEmpty()) return;
58      int line = lines - 2 ;
59      float delay = 0.0f;
60      for(int i = 0; i<servers.size()-1; i++){
61        XTServer server = servers.get(i);
62        XTServer next = servers.get(i+1);
63        int interfacenumber = Utilities.getInterfaceNumber(
64          server.getInterfaceToRouter(next)) - 1;
65        delay = ((i+1) * STEP)
66       changeElement(server,line,interfacenumber,Float.toString(delay));
67      }
68     }
```

Die mehrfach verwendete Methode changeElement(..) ändert den übergebenen Wert (`value`) an der übergebenen Stelle im Forecast. Der Forecast wird innerhalb der Klasse als Hash-Tabelle, die Vektoren von String-Arrays enthält, gehalten. Die Methode changeElement(..) nutzt den Namen des XTServers als Schlüssel und wählt über die Parameter `line` und `iface` die Zeile und Nummer des Interfaces dieses XTServers.

```
69     /**
70      * This method sets one element in the configtable
71      * to the given value.
72      * [..]
73      */
74     public void changeElement(XTServer xts, int line, int iface,
75     String value){
76      try{
77       this.configtable.get(xts.getName().toLowerCase()).
78        get(line)[iface] = value;
79      } catch(NullPointerException n) {n.printStackTrace();}
80     }
```

5 Abbild der Abläufe in einem Netz

In diesem Kapitel wird ein Verfahren entwickelt, welches eine persistente Speicherung aller, in einem Netz von RIP-Routern, vorkommenden zeitlichen und örtlichen Ereignisse zulässt.

Betrachtet man ein solches Netz als Beobachter von außen, erkennt man schnell, wie sich lokale Ereignisse global auswirken und welches Muster die Ereignisse aufweisen müssen um einen bestimmten Zustand auszulösen. Diese globale Sicht fehlt jedem einzelnen RIP-Router, dieser kann lediglich Entscheidungen treffen, die die Elemente seines noch erkennenbaren Teilbereiches des Netzes betreffen. Der RIP-Router teilt dabei allen direkten Nachbarn sein lokales Wissen mit, dies geschieht durch Aussenden einer RIP-Response Nachricht auf jedem -für RIP aktivierten- Interface. Da jeder Router autonom arbeitet und keine Synchronisierung stattfindet, kann jede erdenkliche Konstellation von Updatereihenfolgen auftreten. Da zusätzlich die Verbindung im Sinne der Erreichbarkeit zwischen zwei Rip-Routern über ein beliebiges Netz und mit dem verbindungslosen Transportprotokoll UDP gegeben ist, muss auch mit Latenzen auf der Verbindungsleitung und sogar komplettem Paketverlust gerechnet werden.

Die Gründe warum Konstellationen von Updatereihenfolgen auftreten bleiben, jedoch hier unbeachtet. Wichtig ist lediglich die auftretende Konstellation selbst, da durch diese bestimmte Vorgänge im Netz ausgelöst werden können, wie z.B. den CTI, falls eine Nachricht mit alten Informationen kurz vor dem Eintreffen einer aktuellen Information abgesendet wird. Um als Benutzer in einer globalen Sicht die Abläufe konsistent steuern zu können, ist ein Verfahren notwendig, dass es erlaubt für jeden Router und für jeden Zeitpunkt die Aktionen des Routers bestimmen zu können. Dabei muss zum Ersten die Fähigkeit erhalten bleiben, in periodischen Abständen Updates zu versenden und zum Zweiten, dass innerhalb dieser Periode zu bestimmten Zeitpunkten auf bestimmten Interfaces gesondert Updates versendet werden können.

5.1 Die config Datei

Um die Anforderung der Reproduzierbarkeit gewährleisten zu können, ist es sinnvoll Dateien (ASCII) zu verwenden, die diese Informationen speichern können, um zu einem späteren Zeitpunkt erneut ausgeführt werden zu können. Die Syntax wird durch die folgende Beschreibung in Erweiterter Backus-Naur-Form (EBNF) definiert (Stringelemente durch Hochkomma gekennzeichnet):

```
Configfile = Routerdefinition Timedefinition Forecastdefinition ;
Routerdefinition = "Rip-Router" Routername "/n;"
                   {"Rip-Router" Routername "/n;"} ;
Routername = Char{Char} ;
```

```
Timedefinition = "periodic\_update" Time ;
Time = Integer n (100 <= n <= 30000, n mod 100 = 0);
Forecastdefinition = "Update-Forecast /n" {Line "/n"} LastLine "/n" ;
Line = Routername"("Trigger{","Trigger}");"
       {Routername"("Trigger{","Trigger}");"} ;
Trigger = Number | "x";
Number = n (0.0 <= n <= (Time/100));
LastLine = Routername"("Trigger{","Trigger}(");" | ")a;")
           {Routername"("Trigger{","Trigger} ("");" | ")a;")} ;
```

Eine Config-Datei besteht also aus drei Teilen (s. Anforderungsanalyse):

Routerdefiniton Hier werden alle RIP-Router des VNUML-Netzes mit dem Schlüsselwort gefolgt vom Hostnamen der VM bekannt gemacht, z.B. „Rip-Router R1“. Jeder Eintrag muss in einer neuen Zeile stehen.

Timedefinition Hier wird die Zeit definiert, die der Periode entspricht in der RIP die periodischen Updates versendet. Sobald ein Interface im Modus `MANUAL` ist, werden die schon vorhandenen regelmäßigen Updates unterdückt, deshalb muss dies vom Generator übernommen werden. Das Schlüsselwort „periodic_update“, gefolgt von einem Ganzzahlwert t definiert diese Periode. Die Zeitangabe erfolgt in Millisekunden und muss in Schritten von Hundert erfolgen (t mod 100 = 0), da dies der feinsten Auflösung des Generators entspricht. Die Periode muss passend zu den, in der Konfigurationsdatei des RIP-Deamon, definierten Timern sein (s. Kapitel 7.1.2).

Forecastdefinition Dieser Definitionsteil spiegelt alle Abläufe im gesamten Netz in zeitlicher Reihenfolge wieder. Nach dem Schlüsselwort „Update-Forecast“ können beliebig viele neue Zeilen vorkommen, wobei jede Zeile eine Periode repräsentiert. Innerhalb jeder Zeile kann für jeden -in Routerdefinition bekannt gemachten- Router und für jedes Interface auf diesem Router eine Information abgelegt werden, ob und wann dieses Interface ein Update triggert.

Notiert werden muss dies in Form von Routername gefolgt von durch Komma getrennten Triggern, die durch Klammern eingefasst werden. Abgeschlossen wird eine solche Definition durch ein Semikolon. Die Anzahl der Trigger ist abhängig von der Anzahl der Interfaces des Routers. Es können nur RIP-aktive Interfaces verwendet werden, in der RIP Konfiguration als „passiv“ markierte Interfaces können keine Updates versenden. Durch die Verwendung von VNUML und dessen Kontrollmechanismus der Management-Interfaces, wird auf jeder virtuellen Maschine das erste Interface (eth0) für die Kontrollverbindung verwendet. Dieses Interface wird in der Implementierung nicht beachtet, so dass der erste Eintrag eines Triggers immer für das erste echte RIP-aktive In-

terface bestimmt ist. Die Zuordnung der Interfaces erfolgt nun fortlaufend von links nach rechts. Der erste Eintrag entspricht dem ersten in der XML-Szenariobeschreibung deklarierten Interface, der zweite Eintrag dem zweiten usw. Folgt man der Standard-Namesgebung ergibt sich z.B. diese Zuordnung für vier RIP-aktive Interfaces auf dem Router R1: R1(eth1,eth2,eth3,eth4);.

Ist die Liste der Trigger nicht vollständig, erfolgt für die fehlenden Interfaces kein Update. Die Notation R1(0,0,x,x); kann daher durch R1(0,0); ersetzt werden. Aus Gründen der Nachvollziehbarkeit empfiehlt es sich jedoch alle Interfaces mit Triggern zu belegen. Kommt ein Router nicht in einer Zeile vor, entspricht dies dem Eintrag Router(x,...,x). Der Router sendet in dieser Periode kein einziges Update, ein Ausfall des kompletten Routers kann also durch einfaches entfernen aus den entsprechenden Perioden erreicht werden. Er darf in späteren Zeilen jedoch nicht mehr vorkommen.

Ein Trigger ist entweder eine auf Zehntelsekunden genaue Angabe des Zeitpunktes, an dem der Trigger erfolgen soll, oder der Character „x“. Zweites entspricht einem kompletten Verlust des Updates auf der Verbindungsleitung. Da eine Zeile einer Periode entspricht, kann der numerische Wert eines Trigger nur im Intervall 0.0 bis (periodic_update - 0.1) liegen. Der folgende Zeitpunkt entspricht der Null der nächsten Zeile/Periode. Ganze Zahlen können ohne Punkt eingetragen werden. Das folgende Beispiel lässt also Interface eth1 nach 1 Sekunde und Interface eth2 nach 1.8 Sekunden triggern: R1(1,1.8);

In der letzten Zeile ist zusätzlich der Parameter „a“ erlaubt. Er darf nur beim letzten Vorkommen eines Routers und nach der schließenden Klammer auftauchen. Dieser Parameter gibt an, dass alle Interfaces des entsprechenden Routers (auch die nicht explizit erwähnten) in den Modus `AUTO` geschaltet werden. Dieser Mechanismus erlaubt es dem Benutzer im Netz einen gewünschten Zustand zu erzeugen und nach anschließender Umschaltung auf den Modus `AUTO` die weitere, autonome Verhaltensweise des Netzes beobachten zu können. Der Parameter auto_time des Generators bestimmt die Zeit, nach der der Modus `AUTO` wieder beendet wird.

Kommentare innerhalb der config-Datei sind möglich und bleiben beim Einlesen unbeachtet. Sie müssen jedoch in einer eigenen Zeile stehen und mit einem Doppelslash beginnen.

5.2 Die config Datei - Ein Abbild eines CTI

Entsprechend der Syntax gibt es zwei Wege einen CTI abzubilden. Zum Einen ist dies der Weg alle Trigger von der Startphase bis zum Ende des CTI komplett vorzugeben. Der Zweite Weg lässt mehr Selbstorganisation des Netztes zu. Hier wird ein bestimmer Zustand des Netzes provoziert und nach dem Umschalten aller Router in den Modus `AUTO`, müssen diese auf den

bestehenden Zustand reagieren.

Die erste Möglichkeit ist jedoch keinesfalls weniger realistisch. Wenn über diesen Weg eine Updatekonstellation bestimmt wird, so ist diese in der Realität genauso möglich, wenn auch möglicherweise sehr unwahrscheinlich. Das Verhalten, dass die richtige Nachricht der falschen Nachricht hinterherläuft (s. Kapitel 7.2) kann so modelliert werden. Hier trifft die Nachricht mit der aktuellen Metrik auf jedem Router erst kurz nach der Nachricht mit der falschen Metrik ein. Dieses Verhalten tritt von alleine sehr selten über eine längere Dauer auf, ist jedoch trotzdem nicht auszuschließen. Der zweite Weg stellte sich als sehr erfolgsversprechend und informationsgebend, bezüglich des Verhaltens des Netzes während und nach einem CTI, heraus.

Um die in Kapitel 2.1 vorgestellten Phasen in einen Forecast umzusetzen, sind folgende Schritte notwendig:

Konvergenzphase Die Länge dieser Phase (Anzahl der Zeilen) ist bestimmt durch den längsten direkten Weg im Netz. Um von einem konsistenten Zustand ausgehen zu können, muss jeder Router jede Route kennen. Insbesondere muss jeder Router die ausfallende(n) Route(n) kennen. Als recht genaue Abschätzung eignet sich die Anzahl der nötigen Hops vom ausfallenden Netz bis zum entferntesten Router, als Maß für die Anzahl der zur Konvergenz nötigen Zeilen.

Ausfallphase In dieser Phase fällt ein Router aus. Dies kann entweder durch entfernen des Routers aus diesen und den nachfolgenden Zeilen geschehen, oder durch einfügen eines Trigger mit dem Wert „x“ an das entsprechende Interface. Die Länge dieser Phase ist mit 5 Zeilen festgelegt, sofern sich das Verhältnis der RIP-Timer nicht ändert.

Ausbreitungsphase Diese Phase wird durch eine Zeile repräsentiert. In dieser Phase muss dafür gesorgt werden, dass die richtige Metrik an alle Router im Netz verbreitet wird, jedoch nicht an den Router (False-Router), der in der nächsten Phase das Falsch-Update versenden soll. Dies kann z.B. durch eine leichte inkrementelle Verzögerung im Bereich 0.2 bis 0.5 Sekunden beginnend ab dem Source-Router erreicht werden. Kleinere Werte haben in den Tests dazu geführt, dass sich die neue Metrik nicht vollständig ausbreiten konnte. Bei größeren Werten ist zu beachten, dass die Dauer einer Periode nicht überschritten werden kann. Damit der False-Router das Update nicht erhält, sollte der Nachbar auf seinem entsprechenden Interface ein Trigger mit dem Wert „x“ erhalten. Alternativ kann anstatt einem Paketverlust auch eine Verzögerung simuliert werden. Dazu muss die nächste Phase aber in diese integriert werden, damit der False-Router sein falsches Update versendet bevor ihn das richtige erreicht (s. nächster Absatz).

CTI-Phase Diese Phase wird durch eine Zeile repräsentiert. Der Router, dessen Update in der vorigen Zeile verloren ging, sollte auch hier ein Update verlieren oder es erst absenden,

nachdem der False-Router das Falsch-Update im Netz verbreitet hat. Wenn das Falsch-Update nach ca. 1 Sekunde abgesendet wird, ergibt sich die größte Wahrscheinlichkeit für das Auftreten eines CTI. Diese Abschätzung ergibt sich nach der Auswertung von 100 Testdurchläufen im Y-Szenario.
Hier integriert ist eine anschließende Phase in der das Netz autonom agiert und alle weiteren Ereignisse durch den RIP-Prozess selbst ausgelöst werden.

Die beiden letzten Phasen können auch zusammen in einer Zeile abgebildet werden. Dies bringt jedoch den Nachteil mit sich, dass die Wahrscheinlichkeit steigt, dem Ablauf des Timeout-Timers zuvor zu kommen. Daher muss eine kurze Wartezeit eingeplant werden, damit auch sicher die neue RIP-Infinity Metrik verbreitet wird. In den Testdurchläufen zeigten sich die gleichen Ergebnisse wie bei der 2-zeiligen Abbildung, jedoch traten diese mit geringerer Wahrscheinlichkeit auf.

5.2.1 Beispiel einer config Datei für einen CTI

Der nachfolgend aufgeführte Forecast beschreibt die Provokation eines CTI in einem Y-Szenario bestehend aus 4 Routern:

```
1  Update-Forecast
2  // Konvergenzphase
3  R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
4  R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
5  // Ausfallphase
6  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
7  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
8  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
9  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
10 R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
11 // Ausbreitungsphase
12 R1(0,0); R2(0,0); R3(0,0.5,x); R4(x,0);
13 // CTI-Phase
14 R1(1,0)a; R2(0,0)a; R3(0,0,x)a; R4(x,0);
```

Der dazu äquivalente minimale Forecast:

```
1  Update-Forecast
2  R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
3  R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
4  //R4 trennen
5  R1(0,0); R2(0,0); R3(0,0,0);
```

```
6  R1(0,0); R2(0,0); R3(0,0,0);
7  R1(0,0); R2(0,0); R3(0,0,0);
8  R1(0,0); R2(0,0); R3(0,0,0);
9  R1(0,0); R2(0,0); R3(0,0,0);
10 //180s Timer ist abgelaufen
11 R1(0,0); R2(0,0); R3(0,0.5);
12 R1(1,0)a; R2(0,0)a; R3(0,0)a;
```

Die beiden ersten Zeilen erreichen, dass jeder Router jeden anderen und insbesondere auch R4 kennt. In allen folgenden Perioden wird R4 vom Netz getrennt, indem auf Interface eth1 kein Update versendet wird. In der vorletzten Periode ist der Timer, für die von R4 gelernten Routen, auf R3 abgelaufen. R3 sendet nun die Metrik an seine Nachbarn, an R2 (nach 0.5 Sekunden) und an R1. Das Update für R1 geht jedoch verloren. R1 und R2 verhalten sich normal. In der letzten Periode bekommt R1 wieder nichts vom Ausfall der Route mit und sendet, kurz nachdem die anderen Router ihre Informationen ausgetauscht haben, ein Update, das noch die alte Metrik beinhaltet, an R2. Direkt im Anschluss werden die Router im Zyklus auf den Modus `AUTO` umgestellt. R2 wird das Falsch-Update von R1 annehmen, da es eine bessere Metrik beinhaltet. Damit wurde der CTI provoziert und das Falsch-Update wird im Zyklus umherlaufen.

6 Generator

In diesem Kapitel wird der entwickelte Update-Generator vorgestellt und dessen Funktionsweise erläutert.

Der durch die RIP-XT Erweiterung vorhandene Mechanismus der Updatekontrolle kann auf direktem Weg, z.B. via Telnet, benutzt werden. Eine händische und einzelne Kontrolle jedes Updates ist jedoch, aufgrund der großen Anzahl notwendiger Benutzerinteraktionen, nicht geeignet um langwierige Tests durchzuführen. Ein automatisierter Ansatz erscheint hier zielführender. Dabei soll sich das System (im Folgenden Generator genannt) selbstständig mit dem XTServer verbinden und zufällige bzw. vom Benutzer vorgegebene Updatereihenfolgen ausführen.

Dies führte zu den beiden Spezialisierungen des Generators: Static-Generator und FullRandom-Generator. Da sich im Laufe der Tests jedoch herausstellte, dass mit reinen Zufallswerten keine Ergebnisse zu erwarten sind, wurde der HalfRandom-Generator entwickelt, der eine Brücke zwischen festen und zufälligen Updatereihenfolgen schlägt.

Der im letzten Abschnitt vorgestellte Auto-Generator generiert selbstständig Config-Dateien anhand festgelegter Regeln, die auf jedes geladene Szenario angewendet werden können.

6.1 Static-Generator

Diese Form des Generators führt eine vorher vom Benutzer erstellte Config aus. Dazu muss über den Menüpunkt „load Config-File“ eine Config geladen werden. Die darin definierte „update_time“ wird verwendet und kann nicht geändert werden. Der Forecast wird Zeile für Zeile ausgeführt. Über den Menupunkt „Generator → normal Generator → StaticGenerator“ wird der Static-Generator gestartet. Im folgenden Dialog (Abb. 7) können die Standardparameter angepasst werden: Der Parameter „runs“ beschreibt, wie oft die geladene Config hintereinander

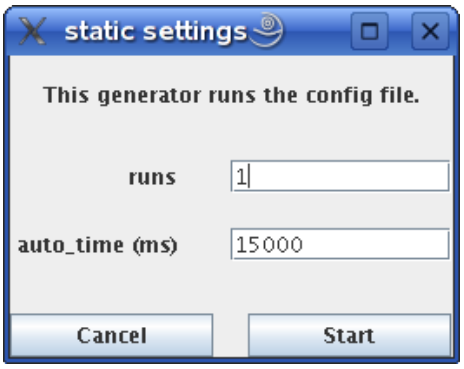


Abbildung 7: Static-Generator

ausgeführt werden soll. Der Parameter „auto_time“ beschreibt die Zeit, die nach dem Wechsel in den Modus `AUTO` gewartet wird (Angabe in Millisekunden).

6.2 HalfRandom-Generator

Der HalfRandom-Generator verdankt seinen Namen der Art der Ausführung einer Config. Wie in Kapitel 5.2 beschrieben, wird die nach Ausfall einer Route kritische Phase durch die letzten zwei Zeilen der Config beschrieben. Der Beginn dieser Config ist jedoch immer gleich: Konvergenzphase und Ausfallphase. Diese beiden Phasen werden vom HalfRandom-Generator aus der Config übernommen, die CTI-Phase wird jedoch aus Zufallswerten generiert. Die Voraussetzungen für einen CTI werden also mit fest vorgegebenen Werten geschaffen, die Updatereihenfolgen in den kritischen Zeilen unterliegen jedoch dem Zufall.

Analog zum Static-Generator muss über den Menüpunkt „load Config-File“ eine Config geladen werden. Die darin definierte „update_time“ wird verwendet und kann nicht geändert werden. Über den Menupunkt „Generator → normal Generator → HalfRandomGenerator“ wird der HalfRandom-Generator gestartet. Im folgenden Dialog (Abb. 8) können die Standardparameter angepasst werden: Der Parameter „runs“ beschreibt die Anzahl der Hintereinanderausführun-

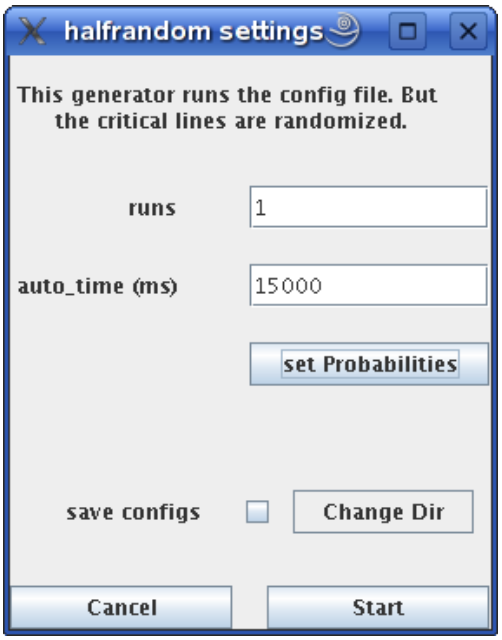


Abbildung 8: HalfRandom-Generator

gen, „auto_time“ beschreibt die Zeit, die nach dem Wechsel in den Modus `AUTO` gewartet wird (Angabe in Millisekunden). Über die Checkbox „save configs“ kann angegeben werden, ob bei

Auftreten eines CTI die aktuell generierte Config gespeichert werden soll. Das Verzeichnis kann über den Button „Change Dir“ angepasst werden.

Zusätzlich können für alle möglichen Werte (also Werte in Schritten von 0.1 zwischen 0 und (update_time - 0.1)) Wahrscheinlichkeiten angegeben werden. Dafür kann über den Button „set Probabilities“ eine Tabelle aufgerufen werden, die jedem möglichen Wert eine Wahrscheinlichkeit zuordnet. Die Wahrscheinlichkeit kann als double-Wert angegeben werden. Standardmäßig haben nur die Ganzzahlwerte und der Wert „x“ je eine Wahrscheinlichkeit von 1.0. Eine Verteilung von 76% für den Wert „1“ und 24% Prozent für den Wert „x“ kann mit (1 , 76), (x , 24) oder z.B. auch mit (1 , 0.76), (x , 0.24) angegeben werden.

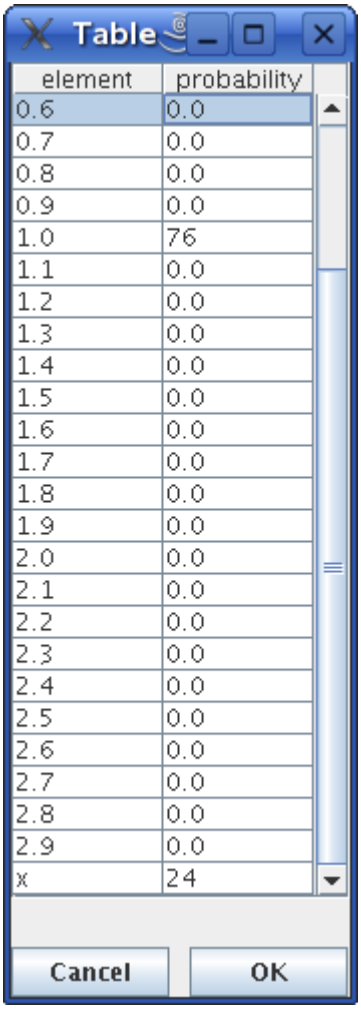


Abbildung 9: Beispiel einer Probability-Table

Die folgende Config wurde durch den HalfRandom-Generator erzeugt und gespeichert:

```
1 Update-Forecast
2 R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
```

```
3  R1(0,0); R2(0,0); R3(0,0,0); R4(0,0);
4  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
5  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
6  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
7  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
8  R1(0,0); R2(0,0); R3(0,0,0); R4(x,0);
9  R1(1,2); R2(0,1); R3(2,x,0); R4(x,x);
10 R1(2,x)a; R2(x,1)a; R3(1,x,1)a; R4(x,1);
```

Es werden nur die Werte innerhalb der Klammern (in Zeile 9 und 10) durch Zufallswerte ersetzt. Der Parameter „a“ bleibt -sofern vorhanden- erhalten. Der ausfallende Router muss bei Anwendung des HalfRandom-Generators immer in jeder Zeile vorkommen.

6.3 FullRandom-Generator

Dieser Generator verwendet ausschließlich zufällig generierte Werte. Eine Config muss deshalb nicht geladen werden. Die Anzahl der Router und der Interfaces pro Router sind, nach Laden der XML-Szenariobeschreibung, bekannt. Aus diesem Wissen kann eine Zeile des Forecasts erstellt werden (Bsp: `R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0);`). Jedes Element wird mit einem zufällig generierten Wert belegt. Dieses Verfahren wird zu Beginn jeder Periode wiederholt und garantiert so vollkommen zufällige Updatereihenfolgen.

Über den Menupunkt „Generator → normal Generator → FullRandomGenerator“ wird der FullRandom-Generator gestartet. Im folgenden Dialog (Abb. 10) können weitere Einstellungen angepasst werden: Der Parameter „update_time“ muss bei diesem Generator angegeben wer-

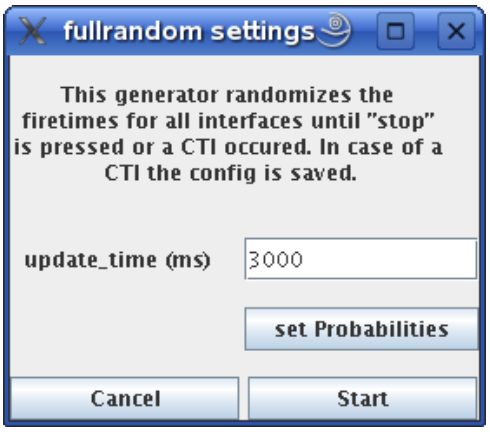


Abbildung 10: FullRandom-Generator

den, da er nicht aus der Config ausgelesen werden kann. Er gibt die Periode an, mit der RIP die periodischen Updates aussenden würde. Zusätzlich können für alle möglichen Werte (also Werte in Schritten von 0.1 zwischen 0 und (update_time - 0.1)) Wahrscheinlichkeiten angegeben werden. Dafür kann über den Button „set Probabilities“ eine Tabelle aufgerufen werden, die jedem möglichen Wert eine Wahrscheinlichkeit zuordnet. Die Wahrscheinlichkeit kann als double-Wert angegeben werden. Standardmäßig haben nur die Ganzzahlwerte und der Wert „x“ je eine Wahrscheinlichkeit von 1.0.

Nach dem Start läuft der FullRandom-Generator solange, bis der erste CTI aufgetreten ist oder er über den Menüpunkt „stop Generator“ beendet wird. Im Falle eines CTI werden die generierten Forecast-Zeilen automatisch in Form einer Config im Arbeitsverzeichnis des Programms gespeichert.

6.4 Auto-Generator

Der Auto-Generator stellt eine Sonderform des Generators dar. Er ist keine Spezialisierung des Generators, sondern erweitert selbst direkt die Klasse Thread und ruft nach jeder Berechnung einen Static-Generator auf.

6.4.1 Pfadbestimmung

Grundlage der Berechnung ist die Klasse GraphObserver[5], in der das Netz mit einer Tiefensuche traversiert wird. Für die Suche nach Pfaden gilt folgende Regel: Starte auf jedem Router und besuche jeden von dort erreichbaren Router solange, bis ein Pfadende[6] erreicht, oder ein Router ein zweites Mal besucht wird[7]. Ein Pfad wird dabei in Form eines Vektors, dessen Elemente Instanzen der Klasse AdjacTupel sind, gehalten. Ein AdjacTupel beschreibt einen Router und ein Interface, so dass ein Pfad eine genaue Beschreibung eines Weges, in Form von Routername und Name des Interface, das zum nächsten Router führt, darstellt.

[5] Die Klasse GraphObserver implementiert eine Tiefensuche auf der Netztopologie, beginnend auf jedem Knoten (s. [Lan07]).

[6] Pfadende ist gleichzusetzen mit: Der besuchte Router hat keine weiteren Interfaces ausser dem, über das die Pfadsuche zu diesem gelangt ist.

[7] Das zweite Vorkommen eines Routers im Pfad entspricht einem Zyklus.

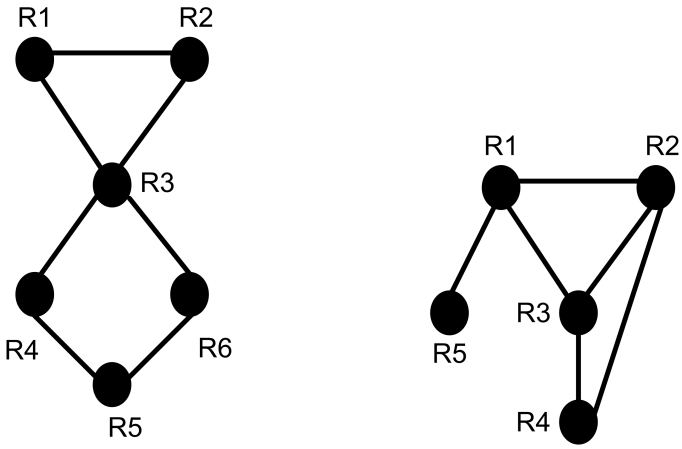
Beispiele[8]:

```
R4(1) eth1 -> R3(2) eth1 -> R1(1) eth1 -> R2(1) eth2 -> R3(2)
R5(1) eth2 -> R4(1) eth1 -> R3(2) eth1 -> R1(1) eth1 -> R2(1) eth2 -> R3(2)
R3(1) eth2 -> R4(1) eth2 -> R5(1)
R3(2) eth1 -> R1(1) eth1 -> R2(1) eth2 -> R3(2)
```

Aus allen gefundenen Pfaden können nun solche bestimmt werden, die einen Zyklus enthalten. Das sind aus obigem Beispiel der erste, zweite und vierte Pfad, da in diesen ein Router (R3) zweimal vorkommt.

In Kapitel 2 wird ein Zyklus als Voraussetzung für einen CTI genannt. Zusätzlich muss ein Teilnetz, das ausserhalb des Zyklus liegt und mit diesem verbunden ist, ausfallen, damit sich eine Falsch-Information im Netz ausbreiten kann. Diese Voraussetzungen erfüllen alle Pfade die einen Zyklus und ein weiteres Teilnetz ausserhalb des Zyklus enthalten. Diese Pfade (im Folgenden genannt Remote-Pfade) werden selektiert. Das sind im Beispiel die ersten beiden Pfade, da sie an einem Router ausserhalb des Zyklus beginnen und dann einmal den Zyklus durchlaufen.

Diese Selektierung ist jedoch immer noch zu grob, wie die folgenden Beispiele verdeutlichen:



(a) Teilnetz ist Zyklus

(b) Weitere Anbindung an Zyklus

Abbildung 11: Pfadbeispiele

In (a) ist der Pfad von R6 über R5 zum oberen Zyklus keine Alternative für den direkten Pfad von R6 zum Zyklus, da Erreichbarkeiten immer über den kürzeren Pfad gelernt werden.

[8]Die Pfade sind ein Auszug der Print-Funktion der Klasse GraphObserver, die auf einem Y-ähnlichen Szenario aufgerufen wurde. Die Zahlen in Klammern geben die Anzahl der Vorkommen des Routers im Pfad an.

Existieren von einem Router mehrere Remote-Pfade muss der kürzeste ausgewählt werden. In (a) werden also für R4 nur die Pfade `R4-R3-R2-R1-R3` und `R4-R3-R1-R2-R3` und für R6 nur die Pfade `R6-R3-R2-R1-R3` und `R6-R3-R1-R2-R3` selektiert.

In (b) kann der Remote-Pfad `R4-R3-R2-R1-R3` -sofern die Verbindung von R4 zu R3 ausfällt- keinen CTI auslösen, da die zusätzliche Verbindung von R4 zu R2 eine weitere Erreichbarkeit garantiert. Es dürfen also nur solche Pfade selektiert werden, in denen Router ausserhalb des Zyklus keine direkten Verbindungen zu Routern innerhalb des Zyklus besitzen.

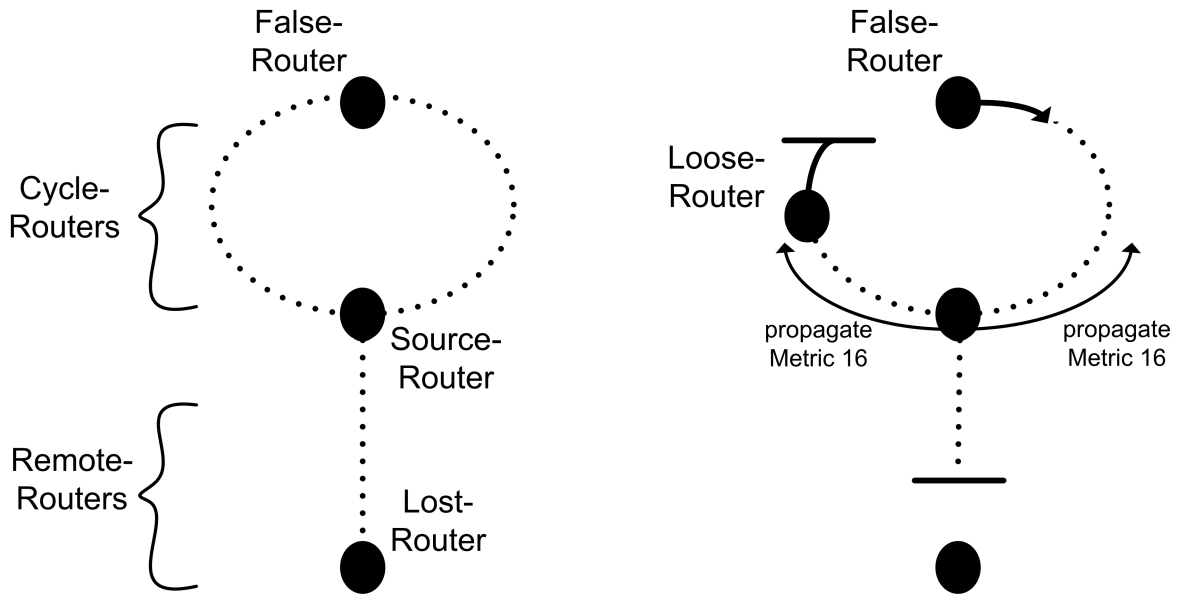
Nach den genannten Einschränkungen ist jeder selektierte Remote-Pfad potentieller Kandidat um einen CTI auszulösen. Es können jedoch noch weitere ungeeignete Pfade vorkommen, wenn Router ausserhalb des Zyklus über einen oder mehrere Pfade (bestehend aus einem oder mehreren Routern) mit Routern innerhalb des Zyklus verbunden sind. Mögliche Lösungen werden hier jedoch nicht behandelt, da die Häufigkeit solcher Konstellationen gering ist.

6.4.2 CTI Kandidaten

Der Auto-Generator behandelt jeden selektierten Remote-Pfad als CTI Kandidat. Ein CTI Kandidat besitzt die nachfolgend genannten Eigenschaften (vgl. Abb. 12):

- Der Pfad besteht aus einem Teilpfad (Remote-Routers), der mit einem Zyklus (Cycle-Routers) verbunden ist.
- Der erste Router im Pfad fällt aus und wird als Lost-Router bezeichnet.
- Der Router, der die Remote-Routers mit den Cycle-Routers verbindet, wird als Source-Router[9] bezeichnet.
- Innerhalb des Zyklus exisitert ein Router (Loose-Router), der ein Update zu spät oder nicht an seinen Nachbarn sendet. In einem Zyklus aus drei Routern ist der Source-Router auch der Loose-Router.
- Der Nachbar des Loose-Router versendet das Falsch-Update und wird als False-Router bezeichnet.

[9] Gemäß der von A. Schmid gewählten Definition: „Der Source-Router bezeichnet den Router des Zyklus, der dem ausfallenden Netz am nächsten ist.“



(a) Remote-Pfad als CTI Kandidat

(b) CTI Entstehung

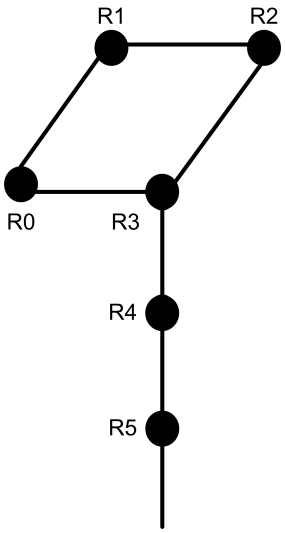
Abbildung 12: CTI Kandidaten

6.4.3 automatische Generierung einer Config

Mithilfe der Eigenschaften eines CTI Kandidaten können Regeln, die eine automatische Generierung einer Config ermöglichen, definiert werden (vgl. Abb. 12):

1. Die Länge (Anzahl Zeilen) der Konvergenzphase beträgt *(Anzahl Remote-Routers + (Anzahl Cycle-Routers / 2))*.
2. Der Lost-Router fällt aus, d.h. alle Interfaces erhalten für die restlichen Phasen den Wert „x“.
3. Die Metrik 16 wird in der Ausbreitungsphase für die ausfallenden Routen, vom Source-Router aus, durch den Zyklus propagiert. Dies geschieht mit kurzen zeitlichen Abständen (0.5sek), um die korrekte Ausbreitung sicherzustellen. Die Metrik darf sich jedoch nicht bis zum False-Router ausbreiten.
4. Der Loose-Router erhält in den letzten beiden Phasen auf dem Interface, welches zum False-Router führt, den Wert „x“. Damit ist sichergestellt, dass die richtige Metrik den False-Router nicht erreicht.
5. Der False-Router sendet in der letzten Phase auf dem Interface, welches zu dem Nachbarn führt, der nicht der Loose-Router ist, ein Update mit einem Wert von „1“.

6.4.4 Beispiel



Nach Anwendung aller Selektionsregeln auf das rechts abgebildete Beispiel[10] bleiben 4 Pfade übrig, die als CTI Kandidaten behandelt werden. In allen 4 Pfaden ist R3 der Source-Router und R1 der False-Router.

Die ersten beiden und die letzten beiden Pfade beschreiben je dieselbe Topologie, jedoch einmal im und einmal entgegen dem Uhrzeigersinn. In Zyklen, die aus einer ungeraden Anzahl von Routern bestehen, macht dies keinen Unterschied. Bei gerader Anzahl ist es sinnvoll beide Pfade zu verwenden, da nicht bekannt ist von welcher Seite der False-Router die Route zum Zielnetz annimmt. So ist sichergestellt, dass der Router, von dem die Route gelernt wurde, einmal zum Loose-Router wird.

Pfade:

```
R5(1) eth1 -> R4(1) eth1 -> R3(2) eth2 -> R2(1) eth1 -> R1(1) eth2
-> R0(1) eth2 -> R3(2)
R5(1) eth1 -> R4(1) eth1 -> R3(2) eth1 -> R0(1) eth1 -> R1(1) eth1
-> R2(1) eth2 -> R3(2)
R4(1) eth1 -> R3(2) eth1 -> R0(1) eth1 -> R1(1) eth1 -> R2(1) eth2
-> R3(2)
R4(1) eth1 -> R3(2) eth2 -> R2(1) eth1 -> R1(1) eth2 -> R0(1) eth2
-> R3(2)
```

Für jeden Pfad wird nun nach dem gleichen Schema ein Forecast generiert. Exemplarisch wird dies hier für den ersten Pfad gezeigt (s. Forecast S.47).
R5 ist der Lost-Router und fällt aus, d.h. jedes seiner Interfaces erhält -beginnend ab der Ausfallphase- den Wert „x“. Die vorletzte Zeile des Forecast entspricht der Ausbreitungsphase. Hier sendet R3 die Metrik 16 an seinen Nachbarn R0. Die künstliche Verzögerung von 0.5sek wird verwendet, um eine Zustellung der richtigen Metrik zu garantieren, da zum Zeitpunkt 0.0 auch gerade der Timer der Route abläuft und somit nicht garantiert ist, dass bereits die Metrik 16 verbreitet wird. Da der Nachbar R2 (Loose-Router) diese Metrik nicht weitergeben wird, kann R3 auf der Verbindung zu R2 das Update sofort versenden.
Damit R1 die neue Metrik nicht von R2 lernt, erhält R2 in den letzten beiden Zeilen auf dem

[10] Das vollständige Szenario ist auf der beiligenden CD im Ordner „ylong“ zu finden.

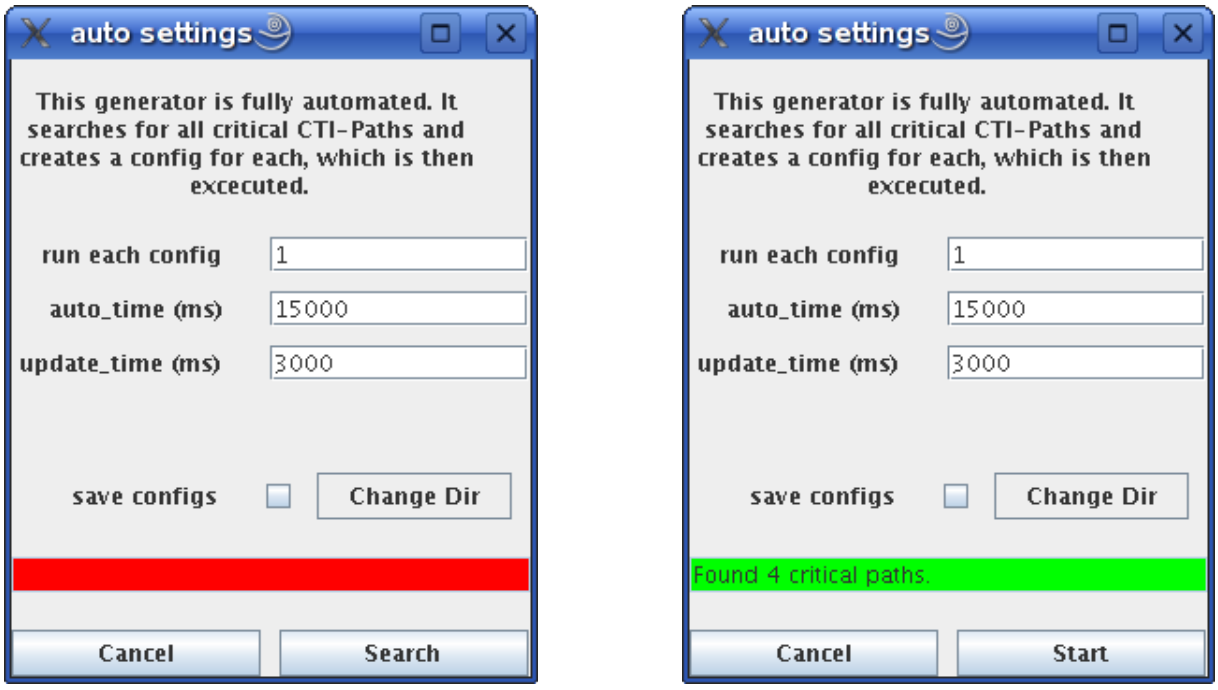
Interface, welches zu R1 führt, den Wert „x“. In der letzten Zeile des Forecast versendet R1 ein Update (mit Falsch-Informationen) an den Nachbarn R0 mit 1 Sekunde Verzögerung.

Durch die Erzwingung dieser Update-Konstellation wurde die Voraussetung für einen CTI geschaffen. Im letzten Schritt werden alle Interfaces auf den Routern R0, R1, R2, R3 und R5 in den Modus `AUTO` versetzt. Der Lost-Router (R4) erhält diesen Parameter nicht, da sich sonst seine Erreichbarkeit vor Eintreten des CTI wieder im Netz ausbreiten würde. Der CTI tritt nun, ohne weitere Einwirkung von außen, ein.

```
1  Update-Forecast
2  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
3  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
4  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(0,0); R5(0,0);
5  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(x,x); R5(0,0);
6  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(x,x); R5(0,0);
7  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(x,x); R5(0,0);
8  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(x,x); R5(0,0);
9  R0(0,0); R1(0,0); R2(0,0); R3(0,0,0); R4(x,x); R5(0,0);
10 R0(0,0); R1(0,0); R2(x,0); R3(0.5,0,0); R4(x,x); R5(0,0);
11 R0(0,0)a; R1(0,1)a; R2(x,0)a; R3(0,0,0)a; R4(x,x); R5(0,0)a;
```

Der generierte Forecast wird an einen Static-Generator übergeben und durch diesen ausgeführt.

Über den Menupunkt „Generator → intelligent Generator → AutoGenerator“ wird der Auto-Generator gestartet. Im folgenden Dialog (Abb. 13 (a)) können weitere Einstellungen angepasst werden:



(a) Suche nach CTI Kandidaten

(b) gefundene CTI Kandidaten

Abbildung 13: Auto-Generator

Der Parameter „run each config“ bestimmt, wie oft jede generierte Config ausgeführt wird, „auto_time“ entspricht der Zeit, in der sich nach Ausführung der Config alle Interfaces im Modus `AUTO` befinden und „update_time“ entspricht der Periode. Die Angabe erfolgt in Millisekunden. Die entsprechende Config wird nach Auftreten eines CTI gespeichert, wenn „save configs“ gewählt wurde.
Der Button „Search“ startet die Suche nach CTI Kandidaten. Wurden solche gefunden (Abb. 13 (b)), kann über den Button „Start“ die Ausführung der generierten Configs begonnen werden.

Jeder Generator kann während der Ausführung über „Generator → stop Generator“ beendet werden. Die aktuelle Periode wird jedoch auch dann noch ausgeführt.

7 Anwendungsbeispiele

In diesem Kapitel werden die Testumgebung „XTPeer“ und eine Auswahl von Testszenarien, die für die Untersuchung des RIP-MTI hilfreich waren, vorgestellt. Zu jedem Szenario werden die Beobachtungen und Ergebnisse beschrieben. Alle vorgestellten Szenarien sind auf der beiliegenden CD im Ordner Beispiele enthalten.

7.1 Die Testumgebung XTPeer

Die in Abb. 14 veranschaulichte Funktionsweise spiegelt den Grundgedanken des XTPeer wieder. Auf dem Hostrechner läuft eine VNUML-Simulation bestehend aus RIP-Routern. Der Generator (s. Kapitel 6) erzeugt über den XT-Client bestimmte Updatereihenfolgen. Der SL-Server[11] erhält gleichzeitig alle Änderungen der Routing-Tabellen der RIP-Router. Diese Informationen werden vom XTPeer zentral gesammelt, ausgewertet und visualisiert.

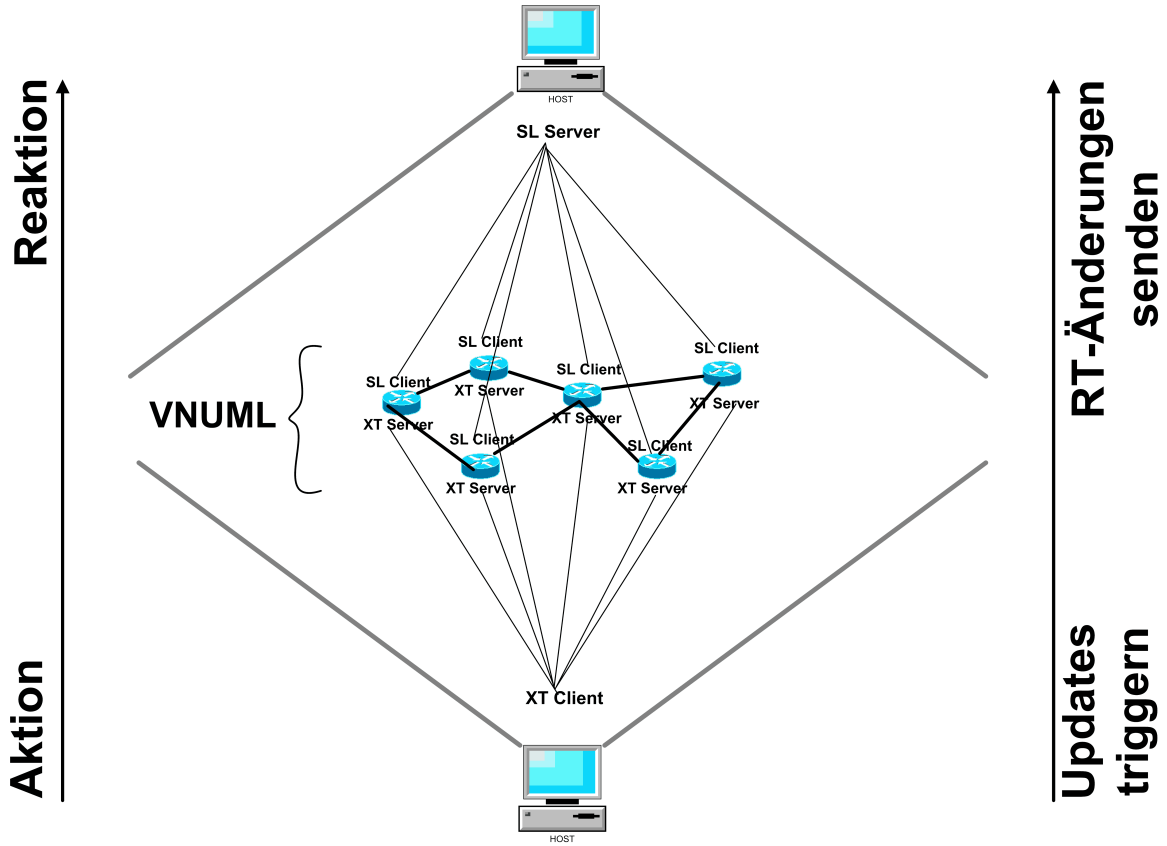


Abbildung 14: Funktionsweise des XT-Peer

[11] In [Lan07] wurde der RIP-Daemon dahingehend erweitert, dass bei jeder Änderung in den lokalen Routing-Tabellen der auf dem Hostrechner laufende SL-Server informiert wird.

Diese Arbeit befasste sich im wesentlichen mit dem Bereich „Aktion“, der durch die Config-Dateien und den Generator realisiert wird.

Nachdem ein Testszenario gestartet wurde, müssen dem Programm alle RIP-Router bekannt gemacht werden. Dies kann zum Einen durch manuelles Hinzufügen jedes einzelnen Routers erfolgen, zum Anderen können über den Menüpunkt „Edit → open → add XTServers from XML-File“, alle Router durch Laden der XML-Szenariobeschreibung automatisch hinzugefügt werden. Im Anschluss kann über den Menüpunkt „Edit → open → load Config-File“ eine Config-Datei geladen werden.

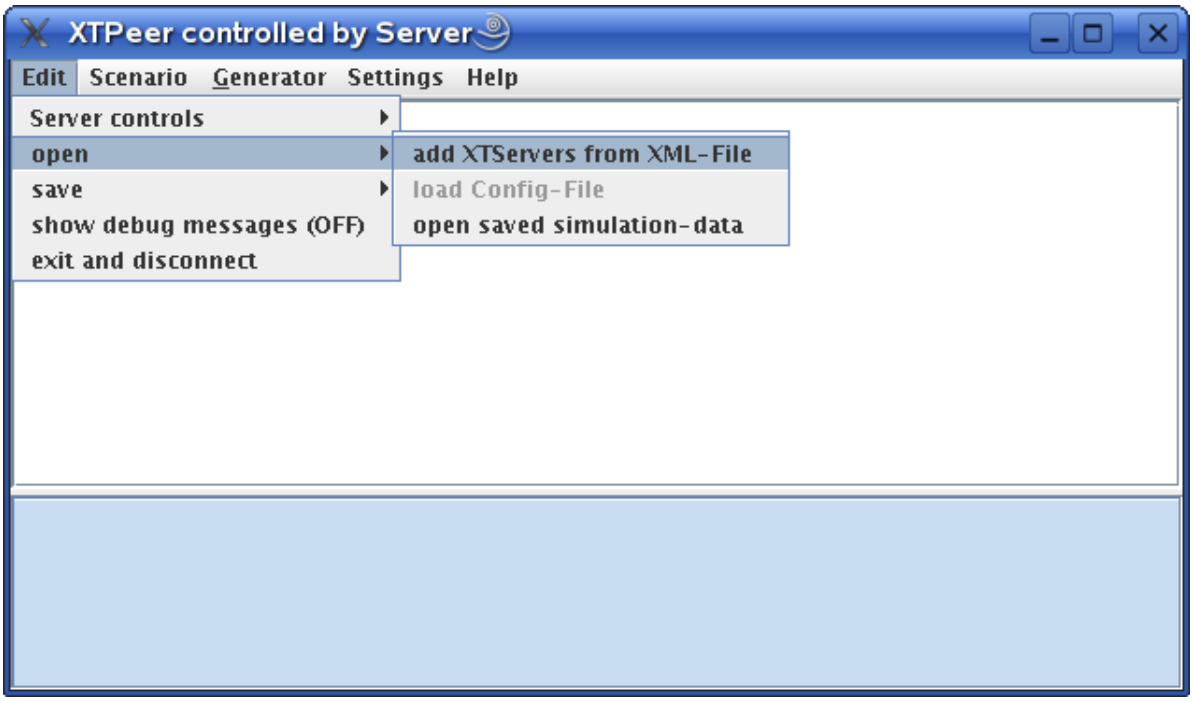


Abbildung 15: XTPeer: Menü

Wird keine Config-Datei geladen, können der FullRandom- und Auto-Generator trotzdem gestartet werden. Zudem kann die Update-Triggerung auch per Maus auf jedem Interface bedient werden, oder im Modus `AUTO` können die standardmäßig ausgetauschten RIP-Nachrichten beobachtet werden (vgl. Abb. 16).

Eine ausführliche Beschreibung der übrigen Funktionalitäten ist in [Lan07] zu finden.

7.1.1 Voraussetzungen

Um die Testumgebung zu starten, muss die jar-Datei ausgeführt werden (keine Parameter nötig). Dies kann auf der Linux-Konsole mit dem Befehl `java -jar xtpeer.jar` erfolgen.

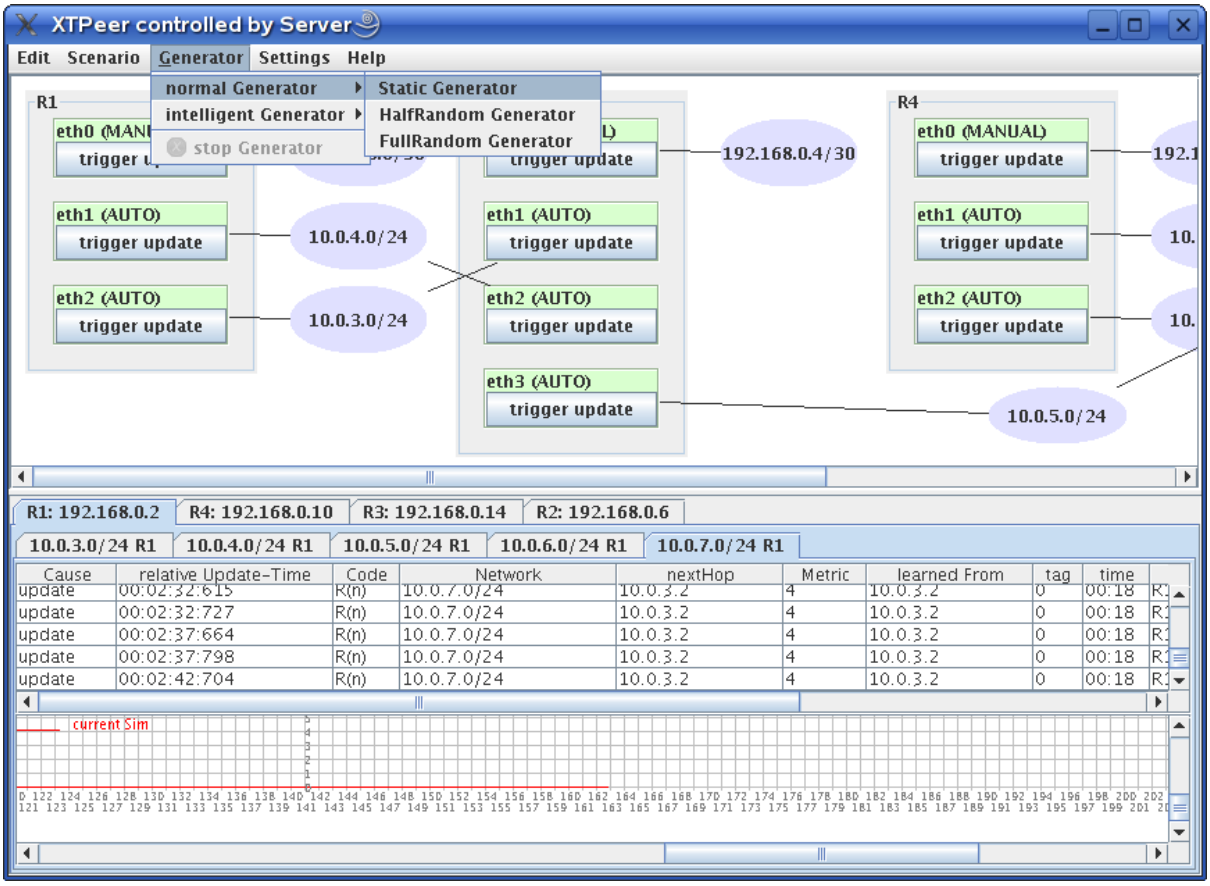


Abbildung 16: XTPeer: Szenario geladen

Zur Ausführung müssen die Bibliothek JGraph (jgraph.jar) und Schreibrechte im Arbeitsverzeichnis vorhanden sein.

Folgende Voraussetzungen müssen durch VNUML realisiert werden:

1. Es muss ein Filesystem verwendet werden, dass einen gepatchten RIP-Daemon[12] enthält. Der Daemon muss sowohl die XT-, die SL- und ggfs. die MTI-Erweiterung aufweisen.
2. Das Tag <host_mapping/> musst verwendet werden, um die Abbildung von Hostnamen auf IP-Adressen zu ermöglichen.
3. Es müssen netzwerkbasierte Management-Verbindungen zwischen dem Host und jeder virtuellen Maschine existieren.
4. Die Interfaces auf einer virtuellen Maschine müssen mit „ethN“ (N natürliche Zahl) benannt sein.

[12] entsprechende Pakete sind auf der beiliegenden CD im Ordner ripd zu finden

5. Das Management-Interface (eth0) jeder virtuellen Maschine darf nicht zum Austausch von RIP-Nachrichten verwendet werden.
6. Das Szenario darf nur aus RIP-Routern bestehen oder die RIP-Router müssen einzeln hinzugefügt werden.
7. Die RIP-Daemons auf jeder VM müssen bereits richtig konfiguriert und durch VNUML gestartet sein.

7.1.2 Änderung der RIP-Timer

Standardmäßig sind die von RIP (und RIP-MTI) verwendeten Timer auf folgende Werte eingestellt: Update: 30 Sekunden, Timeout: 180 Sekunden, Garbage-Collect: 120 Sekunden. Um den Ausfall einer Route schneller zu erkennen, bzw. den Ablauf insgesamt zu beschleunigen, können die Zeiten verringert werden. Der Befehl `timers basic update timeout garbage` kann dazu in die Konfigurationsdatei (ripd.conf) des RIP-Daemons eingefügt, oder direkt, über die Telnet-Verbindung zum Daemon, eingegeben werden.

Die Änderung muss dabei jedoch immer das Verhältnis der Werte erhalten, um die Vergleichbarkeit mit den Standardwerten zu gewährleisten. Mögliche Werte sind deshalb z.B. (6,36,24) oder (3,18,12). Das Tripel (3,18,12) muss jedoch in der Konfigurationsdatei mit (5,18,12) angegeben werden, da die Implementierung des RIP-Daemon keine kürze Periode als 5 Sekunden zulässt. Die tatsächlich verwendete Periode von 3 Sekunden wird durch die Konfiguration des Generators übernommen.

Kleinere Werte können zu unerwartetem Verhalten führen, da die Testumgebung keinen Echtzeit-Ansprüchen genügt und deshalb Triggerungen in zu kurzen Abständen andere Reihenfolgen als gewünscht erzeugen können. Das Tripel (3,18,12) stellte sich in den meisten Szenarien als zuverlässig und gleichzeitig schnell heraus. Ein Durchlauf dauert dann *Anzahl der Perioden * 3 Sekunden + Auto_time Sekunden*, was bei Verwendung der Standardwerte ca. 45 Sekunden entspricht. Pro Stunde können demnach ca. 80 Testdurchläufe durchgeführt werden. Lediglich im größeren Bignet-Szenario (s. Kapitel 7.6) müssen größere Werte für die Timer verwendet werden.

7.2 Y-Szenario

Szenariobeschreibung:
Dieses Szenario entspricht der in [Sch99] vorgestellten Y-Topologie. Das Y-Szenario bildet den einfachsten Fall, der zur Demonstration für das Auftreten und die Verhinderung eines CTI genutzt werden kann. Ein Zyklus bestehend aus 3 Routern ist mit einem weiteren Router verbunden, dessen Routen im Zyklus den CTI auslösen.

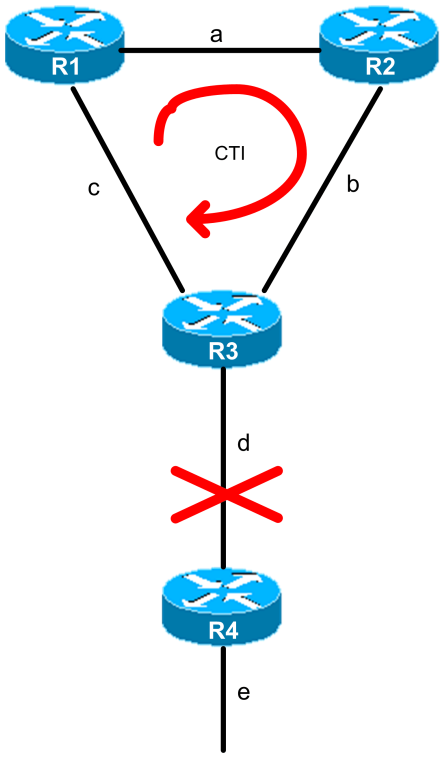


Abbildung 17: Y-Szenario

Beobachtungen:
Wie schon in [Fra05] gezeigt, kann in solchen Topologien ein CTI auftreten. Wird die Config aus Anhang A verwendet, tritt der CTI in ca. 95% der Fälle auf. Er wird in allen Fällen durch RIP-MTI verhindert. Bei Verwendung des FullRandom-Generators ist -auch nach ca. 12 Stunden- kein CTI aufgetreten.

Ergebnisse:
Nach 100 Durchläufen waren folgende Ergebnisse festzuhalten: Ein CTI dauert im Durchschnitt 12 Sekunden. Dieser Wert ändert sich bei Änderung der Periode (3-30sek) nicht, da die Triggered Updates unabhängig von der Periode bei Änderung der Metrik ausgelöst werden. Die Extremwerte lagen bei 5 und 30 Sekunden.

RIP benötigt, bedingt durch den CTI, mindestens 15 Updates, um die Unerreichbarkeit für Netz e im Zyklus bekannt zu machen. RIP-MTI benötigt 2 Updates, da nur auf R1 und R2 die Falsch-Information korrigiert werden muss.

Der RIP-MTI Algorithmus konvergiert im Durchschnitt nach 2 und spätestens nach 8 Sekunden. Die maximale Zeit gilt nur für den Router der das Falsch-Update erhält (hier R2), der Source-Router konvergiert sofort, da er das Falsch-Update ablehnt. Durch die Unterbrechnung des Zyklus durch den Source-Router (R3), können nur 2 Falsch-Updates durch das Netz laufen (R1 zu R2 und R2 zu R3) und in der Folge fließt der Datenverkehr nicht im Kreis durch das Netz.

Bei einer Periode von 30 Sekunden kann RIP-MTI die Zeit bis zur Konvergenz auf Router R2 verlängern. Nachdem R2 das Falsch-Update erhalten und versendet hat, unterbricht R3 den Zyklus. Die Falsch-Information bleibt jedoch auf R2 erhalten bis das nächste periodische Up-

date eintrifft. Dauert dies im schlechtesten Fall 30 Sekunden, hätte die Metrik 16 bei einem CTI in kürzerer Zeit den Router erreicht.

Einen Spezialfall stellt die Updatekonstellation, bei der die richtige Nachricht mit kurzer Verzögerung hinter der falschen Nachricht durch das Netz läuft, dar. Die Config aus Anhang F beschreibt den kompletten Ablauf eines CTI, der dieses Verhalten erzeugt. Abb. 18. zeigt die lokale Auswirkung auf Router R1 für das Netz e: Die Metrik alterniert zwischen den sich hochzählenden Werten und dem Wert 16.

update	00:00:19:567	R(n)	10.0.5.0/24	10.0.3.2	3	10.0.3.2
update	00:00:26:891	R(n)	10.0.5.0/24	10.0.3.2	16	10.0.3.2
update	00:00:30:036	R(n)	10.0.5.0/24	10.0.3.2	6	10.0.3.2
update	00:00:33:224	R(n)	10.0.5.0/24	10.0.3.2	16	10.0.3.2
update	00:00:36:349	R(n)	10.0.5.0/24	10.0.3.2	9	10.0.3.2
update	00:00:39:503	R(n)	10.0.5.0/24	10.0.3.2	16	10.0.3.2
update	00:00:42:658	R(n)	10.0.5.0/24	10.0.3.2	12	10.0.3.2
update	00:00:45:787	R(n)	10.0.5.0/24	10.0.3.2	16	10.0.3.2
update	00:00:48:971	R(n)	10.0.5.0/24	10.0.3.2	15	10.0.3.2
update	00:00:52:070	R(n)	10.0.5.0/24	10.0.3.2	16	10.0.3.2

Abbildung 18: Screenshot: richtige Nachricht folgt falscher Nachricht

In Tab. 3 ist die Anzahl der aufgetretenen CTIs im Y-Szenario, bei verschiedenen Wahrscheinlichkeitsverteilungen angegeben. Die Werte wurden mit dem HalfRandom-Generator in je 100 Durchläufen ermittelt. Die Werte „0“ und „1“ hatten die gleiche Wahrscheinlichkeit, der Wert „2“ sollte nicht vorkommen.

Verteilung				**CTIs**
0	*1*	*2*	*x*	
1	1	0	1	4
1	1	0	3	14
1	1	0	5	22
1	1	0	7	23

Tabelle 3: CTI Häufigkeiten

Hat ein Paketverlust (Wert „x“) die gleiche Wahrscheinlichkeit, wie die anderen Werte, ist in 4% der Fälle ein CTI aufgetreten. War ein Paketverlust 7-mal wahrscheinlicher, ist bereits in 23% der Fälle ein CTI aufgetreten. Die steigende Anzahl von CTIs zeigt die Anfälligkeit von RIP bei Übertragungsproblemen auf den Verbindungsleitungen.

7.3 YMany-Szenario

Szenariobeschreibung:
Dieses Szenario entspricht der Grundtopologie des Y-Szenario. Die zusätzlichen Elemente ausserhalb des Zyklus dienen zur Demonstration der Auswirkungen eines CTI, wenn mehr als eine Route betroffen ist.

Beobachtungen:
Die Config aus Anhang B erzeugt in 95% der Versuche einen CTI. Alle 12 Netze (e bis p), die aus Sicht des Zyklus von R4 kommen, sind vom CTI betroffen. Auf R1, R2 und R3 wurde die Metrik jeweils für alle 12 Routen hochgezählt.

Ergebnisse:
In komplexeren Topologien ist der Gewinn des RIP-MTI umso größer, da für jede CTI-Route der Datenverkehr einen ungültigen Weg durch das Netz nehmen würde. Der Source-Router R3 erkennt die Falsch-Information für alle 12 Routen und lehnt diese ab.

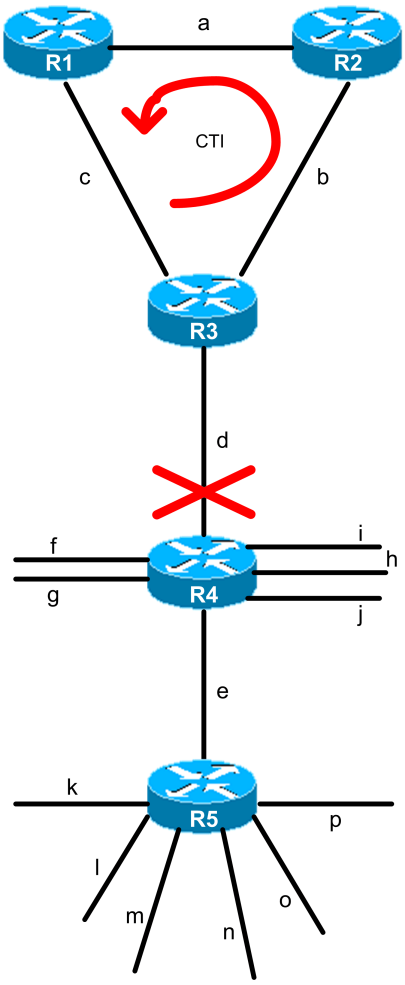


Abbildung 19: YMany-Szenario

R5: 192.168.0.18 | ! R2: 192.168.0.6 | ! R1: 192.168.0.2 | ! R3: 192.168.0.10 | R4: 192.168.0.14

! 10.0.17.0/24 R3 | ! 10.0.18.0/24 R3 | ! 10.0.19.0/24 R3 | ! 10.0.20.0/24 R3 | ! 10.0.21.0/24 R3

! 10.0.12.0/24 R3 | ! 10.0.13.0/24 R3 | ! 10.0.14.0/24 R3 | ! 10.0.15.0/24 R3 | ! 10.0.16.0/24 R3

10.0.2.0/24 R3 | 10.0.3.0/24 R3 | 10.0.4.0/24 R3 | 10.0.1.0/24 R3 | ! 10.0.5.0/24 R3 | ! 10.0.11.0/24 R3

Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From
update	00:00:38:608	R(n)	10.0.11.0/24	10.0.2.1	5	10.0.2.1
update	00:00:39:613	R(n)	10.0.11.0/24	10.0.2.1	8	10.0.2.1
update	00:00:39:826	R(n)	10.0.11.0/24	10.0.2.1	8	10.0.2.1
update	00:00:43:326	R(n)	10.0.11.0/24	10.0.2.1	11	10.0.2.1
update	00:00:44:302	R(n)	10.0.11.0/24	10.0.2.1	14	10.0.2.1
update	00:00:45:846	R(n)	10.0.11.0/24	10.0.2.1	14	10.0.2.1
update	00:00:46:614	R(n)	10.0.11.0/24	10.0.2.1	16	10.0.2.1

Abbildung 20: Screenshot: CTI für 12 Routen

7.4 Bigloop-Szenario

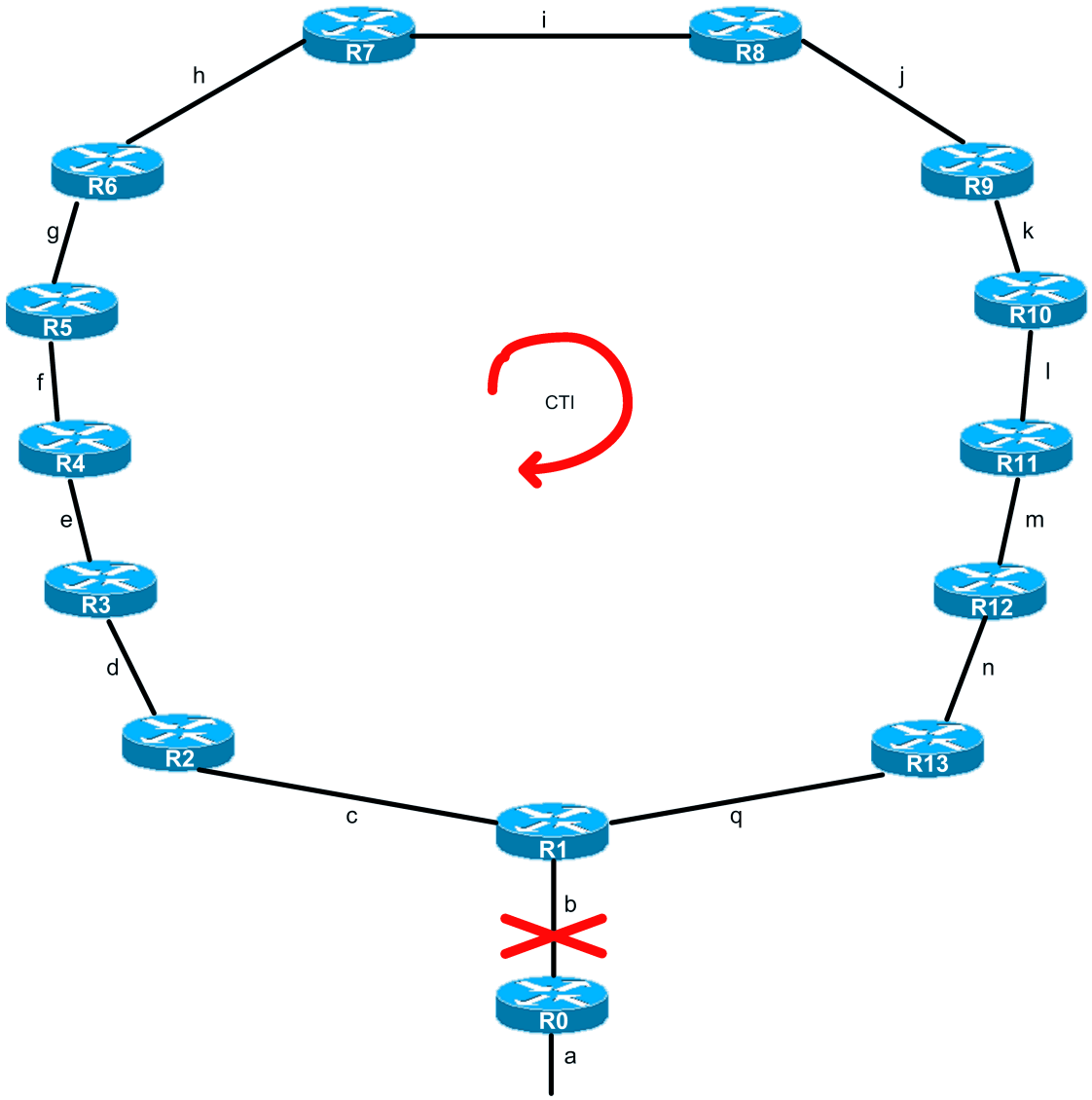


Abbildung 21: Bigloop-Szenario

Szenariobeschreibung:
Dieses Szenario entspricht der Grundtopologie des Y-Szenario, jedoch mit einem maximal großen Zyklus. Die Route zu Netz a erreicht den Router R1 nach einem Durchlauf durch den Zyklus mit der Metrik 15. R1 ist also der erste und letzte Router der eine Route annehmen könnte, die wieder zu ihm zurückführt. Für R2 ist Netz a nach einem Durchlauf schon unerreichbar, da die Metrik bereits RIP-INFINITY erreicht hat.

Beobachtungen:
Der CTI tritt, nach dem Versenden des Falsch-Update von R7 zu R8, auf. RIP-MTI unterbricht

auf R1 den Zyklus und verhindert damit den CTI. Die Falsch-Nachricht kann sich jedoch auch bei RIP-MTI auf den Routern R8, R9, R10, R11, R12 und R13 ausbreiten.

Ergebnisse:
Ein CTI dauert in diesem Szenario nur etwa 2-3 Sekunden. Dieser relativ kurze Wert ist bedingt durch die Tatsache, dass die Nachricht den Zyklus nur einmal durchlaufen muss (zum Vergleich: Ein Zyklus aus 3 Routern erfordert 5 Umläufe, s. Kapitel 2.3.2). Da nach Absenden eines Triggered Updates der Timer neugestartet wird, kann im ersten Umlauf das Update auf jedem Router sofort weitergereicht werden. Der Unterschied zwischen RIP und RIP-MTI fällt hier also geringer aus. RIP-MTI kann lediglich das letzte Falsch-Update auf R1 blockieren. Das Ausbreiten der Falsch-Nachricht auf den Routern R8 - R13 kann mit dem MTI-Ansatz nicht verhindert werden.

7.5 X-Szenario

Szenariobeschreibung:
Dieses Szenario entspricht der in [Sch99] vorgestellten X-Topologie. Es entsteht durch die Verbindung zweier Zyklen. In [Wol06] wurde gezeigt, dass ein CTI im oberen Zyklus auftreten kann, wenn eine Route aus dem unteren unerreichbar wird.

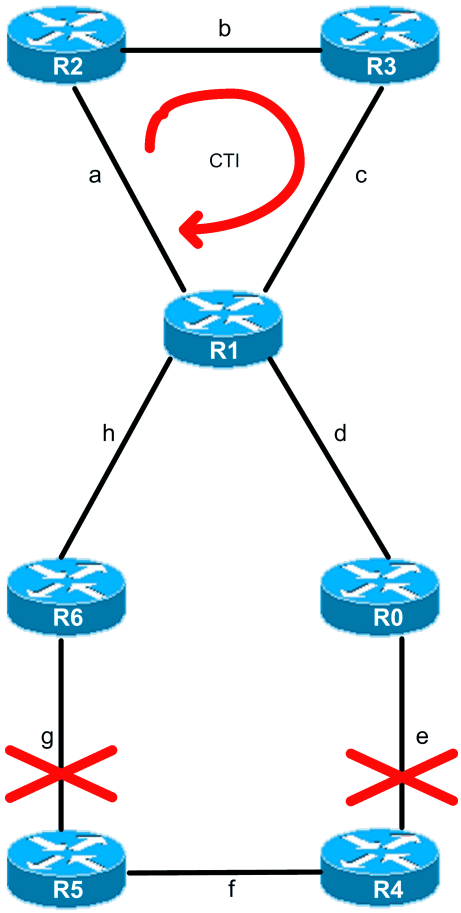


Abbildung 22: X-Szenario

Beobachtungen:
Mit der Config aus Anhang C (a) wird im oberen Zyklus ein CTI, nach Ausfall der Verbindung von R0 zu R1, erzeugt. Die Config aus Anhang C (b) erzeugt für Netz f einen CTI im oberen Zyklus. Der CTI tritt jeweils nach dem Versenden des Falsch-Update von R2 zu R3 auf. RIP-MTI unterbricht auf R1 den oberen Zyklus und verhindert in allen Testfällen den CTI. Auch bei zufällig generierten Latenzen durch den HalfRandom-Generator konnte der MTI alle CTIs verhindern.

Die Config aus Anhang C (a) (Ausfall der Verbindung von R0 zu R1) kann folgenden Spezialfall bewirken: Nachdem R1 die Route zu Netz e über R0 als unerreichbar markiert hat, trifft ein Update von R3 mit der Metrik 5 (über R2, R1 und R0) für Netz e ein. Der CTI beginnt im oberen Zyklus, jedoch nur solange, bis von R6 die bessere Metrik 6 zu Netz e gelernt wurde. Kommt das Update früh genug, kann der CTI unterbrochen werden. RIP-MTI nimmt die Alternativ-Route über R6 jedoch nicht sofort, sondern erst nach Ablauf des Garabage-Collect Timers, an. Dieses Konvergenzproblem des MTI wird in Kapitel 7.12 näher beschrieben.

Ergebnisse:
Nach 100 Durchläufen ist der CTI in 93% der Fälle bei Verwendung von RIP aufgetreten. Er dauerte im Durchschnitt 12 Sekunden. RIP benötigt, bedingt durch den CTI, mindestens 13 Updates, um die Unerreichbarkeit für Netz e bzw. f im oberen Zyklus bekannt zu machen. RIP-MTI benötigt 2 Updates, da nur auf R2 und R3 die Falsch-Information korrigiert werden muss. Da RIP-MTI die Alternativ-Route über R6 vor Ablauf des Garbage-Collect-Timers nicht annimmt, konvergiert RIP-MTI in diesem Fall langsamer.

7.6 Bignet-Szenario

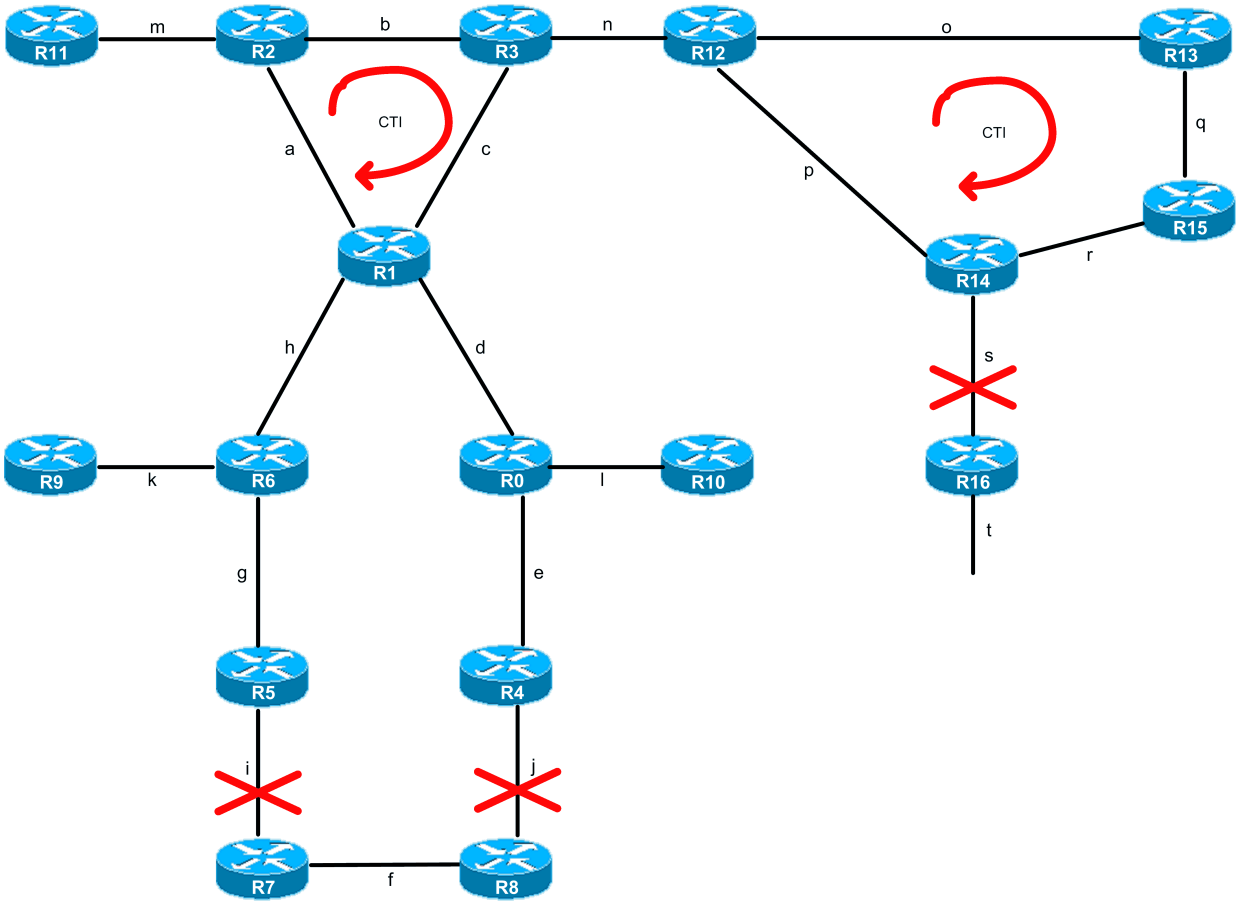


Abbildung 23: Bigloop-Szenario

Szenariobeschreibung:
Die Topologie entspricht der Zusammenführung eines erweiterten X- und Y-Szenarios. Wie zu erwarten, können deshalb auch CTI in beiden oberen Zyklen auftreten. Mit diesem Szenario kann gezeigt werden, dass ein CTI Auswirkungen auf alle mit dem Zyklus verbundenen Router hat.

Beobachtungen:
Tritt ein CTI im linken oberen Zyklus auf, wird die sich hochzählende Metrik an alle Nachbarn weitergegeben. Die in Abb. 24 orange markierten Router bekommen die falsche Metrik über die schon ausgefallene Route von Routern innerhalb des Zyklus mitgeteilt (Beispiel: Auf Router R11 wird die Metrik von 6 auf 9, 12, 15 und 16 hochgezählt). Diese Router sind nicht aktiv am CTI beteiligt, trotzdem weisen ihre Routing-Tabellen für die Dauer des CTI Inkonsistenzen auf.

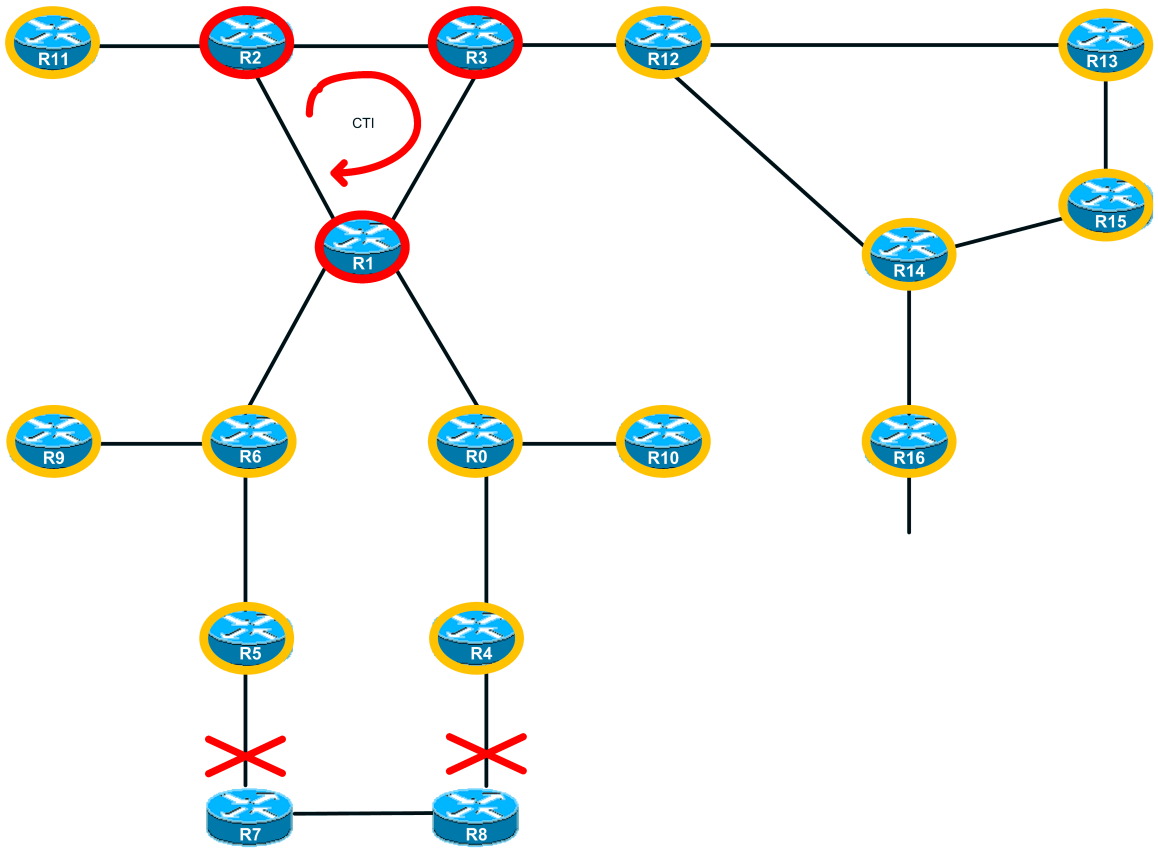


Abbildung 24: Auswirkungen des CTI

Werden in beiden oberen Zyklen gleichzeitig CTIs provoziert[13], können sich diese gegenseitig beeinflussen. Als Beispiel soll hier die Routingtabelle auf Router R14 dienen. Der Weg zu Netz f führt mit einer Metrik von 7 über R12. Ensteht ein CTI im linken Zyklus, wird auf R14 die Metrik auf 10, 13 und 16 hochgezählt. Wenn zur gleichen Zeit ein CTI im rechten Zyklus entsteht, kann kurzzeitig über R15 eine bessere Route zu Netz f gelernt werden: Da auf R12 - R15 durch den eigenen CTI die Timer für jedes Triggered Update neugestartet werden, kann ein Update von R12 verspätet über R13 und R15 den Router R14 erreichen. Von Router R15 erhält er fortlaufend die Metriken 9,12,15,16, von R12 entsprechend 7,10,13,16. Folgende Konstellation ist möglich und konnte auch beobachtet werden: 7,10,9,12,15,13,16,16.

Ergebnisse:
Ein CTI wirkt sich auf alle Router ausserhalb des Zyklus aus. Diese sind indirekt durch das Hochzählen der Metrik betroffen und weisen deshalb die gleichen Inkonsistenzen in ihren Routingtabellen auf, wie Router innerhalb des Zyklus. RIP-MTI konnte in allen Testfällen den CTI verhindern. Alle Router ausserhalb des Zyklus konvergieren deshalb ohne Inkonsistenzen.

[13]Die dazu verwendete Config befindet sich auf der beiligenden CD im Ordner des Szenarios.

7.7 Interloop-Szenario

Szenariobeschreibung:
Dieses Szenario wurde erstmals in [Wol06] als Problemfall des RIP-MTI vorgestellt. Es besteht aus einem Zyklus der einen weiteren Zyklus enthält.

Beobachtungen:
CTIs sind in diesem Szenario über 3 Zyklen möglich: R1-R2-R3-R4-R1, R1-R2-R4-R1 und R2-R3-R4-R2. In den ersten beiden Fällen kann RIP-MTI den CTI verhindern. Die Config aus Anhang D erzeugt einen CTI zwischen den Routern R2, R3 und R4, der auch von RIP-MTI nicht verhindert wird. Der Source-Router R1 verhindert zwar den Zyklus der über R1 läuft, durch die direkte Verbindung von R4 zu R2 kann der Source-Router jedoch übergangen werden.

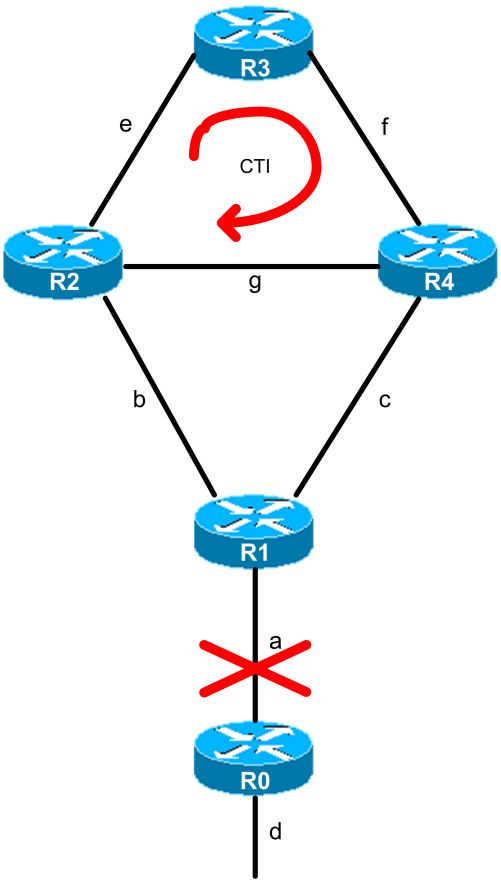


Abbildung 25: Interloop-Szenario

Ergebnisse:
RIP-MTI kann den CTI im großen Zyklus verhindern. Im oberen Zyklus sind jedoch auch bei Verwendung von RIP-MTI CTIs möglich. Häufigkeit und Dauer des Auftretens ist dabei bei Verwendung von RIP oder RIP-MTI identisch. Die Dauer beträgt im Durchschnitt jeweils 10 Sekunden.

Der Router R4 nimmt die Falsch-Nachricht von R3 an, obwohl er eine Y-Topologie erkennen müsste. Das Problem ist die zusätzliche Verbindung von R1 zu R4, ohne diese Verbindung würde die Topologie der Y-Topologie entsprechen. Durch diese Verbindung entsteht ein Zyklus, den RIP-MTI auf R4 in die Matrix, die alle Zyklenlängen beinhaltet, mit dem Wert 4 einträgt. Die errechnete Kombination (alte Metrik + neue Metrik - 1) aus altem Weg (Metrik 3) und neuem Weg (Metrik 5) hat den Wert 7. An der Stelle

```
1 if (combi > rme->metric) return 1;
```

wird deshalb der Wert 1 (entspricht Route annehmen) zurückgegeben, da $7 > 4$ gilt. Der Algorithmus muss an dieser Stelle angepasst werden, damit die Topologie korrekt erkannt und die Nachricht abgelehnt wird. Eine Lösung für dieses Problem wird in der (zum Zeitpunkt der Abgabe noch laufenden) Diplomarbeit von Frank Bohdanowicz [Boh07] entwickelt.

7.8 Doublecon-Szenario

Szenariobeschreibung:
Dieses Szenario erhielt seinen Namen aufgrund der Doppelverbindung zwischen den Routern R1 und R2.

Beobachtungen:
Ein CTI tritt sowohl bei Verwendung von RIP als auch RIP-MTI auf und dauert im Durchschnitt 15 Sekunden. Die Metrik von Netz g wird auf R1 und R2 in 2er-Schritten hochgezählt. Dabei wird die maximal mögliche Anzahl von 7 Umläufen benötigt, um beide Router auf die Metrik 16 hochzuzählen.

Ein CTI tritt jedoch nicht auf, wenn Router R3 ausfällt. Der Algorithmus greift dabei für die ausfallenden Netze f ung g. In der Log-Datei auf R2 ist zu sehen, dass der MTI-Algorithmus das Falsch-Update korrekt ablehnt.

Ergebnisse:
Da der MTI-Algorithmus ein Y-Szenario erkennen müsste und eine Berechnung des Algorithmus per Hand dies bestätigt, muss der Fehler in der Implementierung liegen.

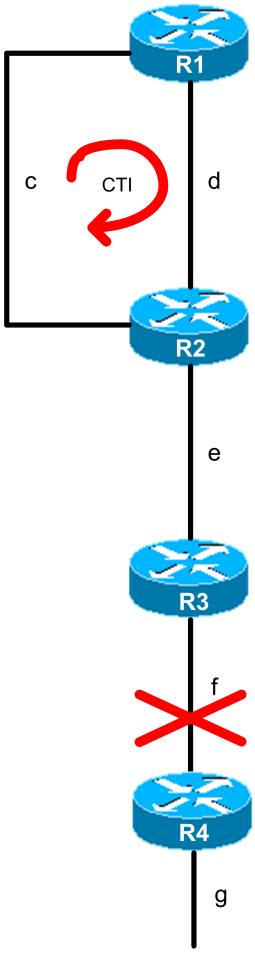


Abbildung 26: Doublecon-Szenario

Eine genauere Untersuchung zeigte, dass die Netze c und d nie in den Aufruf der MTI-Funktion gelangen. Da jeder Router an beide Netze direkt angeschlossen ist und daher jede gelernte Metrik länger ist, beendet der Aufruf der If-Abfrage

```
1 if (new_dist > old_dist || rte->metric ==
2  RIP_METRIC_INFINITY) {...; return;}
```

die Funktion rip_rte_process(), bevor darin die MTI-Funktion aufgerufen wird. Zusätzlich erkennt der Router R2 eine Schleife zu Netz f zwischen einem Interface der Doppelverbindung und dem Interface zu Netz e. Die Metriken der Interfaces zu Netz f betragen 4 bzw. 2, wodurch ein mincyc-Wert von 5 gespeichert wird. Das Netz g wird ebenfalls als Teil eines Zyklus zwischen diesen Interfaces erkannt, der größere mincyc-Wert dieses Zyklus überschreibt den

kleineren jedoch nicht. Ein Update weist nach einem Durchlauf durch die Schleife eine nur um 2 erhöhte Metrik auf und wird deshalb angenommen. Das Problem kann behoben werden, wenn in obiger Codezeile die MTI-Funktion explizit vor dem return aufgerufen wird. Diese Lösung wird in der Arbeit von Frank Bohdanowicz [Boh07] entwickelt und implementiert.

7.9 YDouble-Szenario

Szenariobeschreibung:
Dieses Szenario entspricht der Y-Topologie, jedoch mit einer Doppelverbindung zwischen Router R2 und R3.

Beobachtungen:
Ein CTI kann zwischen R2 und R3 und zwischen R1, R2 und R3 auftreten und kann von RIP-MTI nicht verhindert werden. Die Config aus Anhang E erzeugt in 95% aller Fälle einen CTI im äußeren Zyklus, die Dauer des CTI beträgt im Durchschnitt 10 Sekunden. Der CTI tritt jedoch nur für Netz f auf, für Netz e kann er verhindert werden.

Ergebnisse:
Eine Doppelverbindung zwischen zwei Routern eines Zyklus ermöglicht das Auftreten von CTIs, auch wenn RIP-MTI verwendet wird. Diese Feststellung ist zunächst verwunderlich, da der Router R3, über die Interfaces zu Netz b bzw. bb, jeweils eine normale Y-Topologie erkennen müsste. Erreicht ihn nun ein Update aus diesem Zyklus, sollte dieses vom MTI-Algorithmus überprüft und ggfs. abgelehnt werden (s. Y-Szenario). Die folgenden Zeilen sind ein Ausschnitt (kritische Zeilen) aus dem Forecast der in diesem Szenario den CTI auslöst.

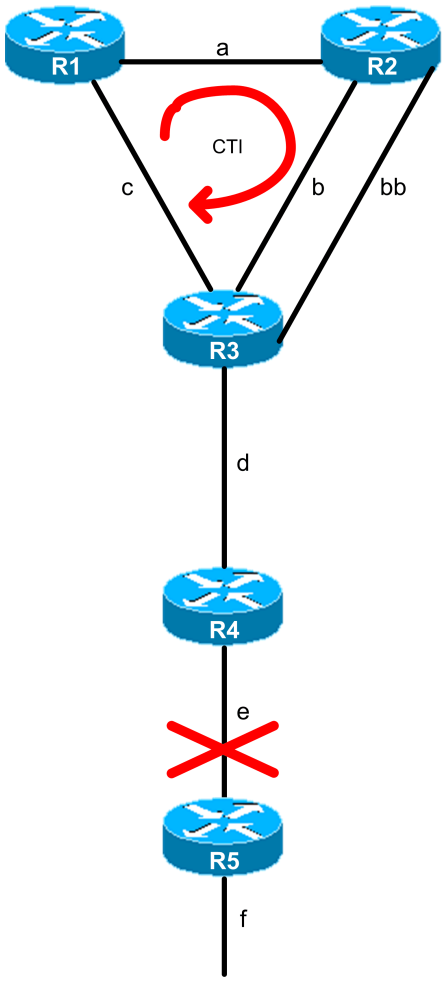


Abbildung 27: YDouble-Szenario

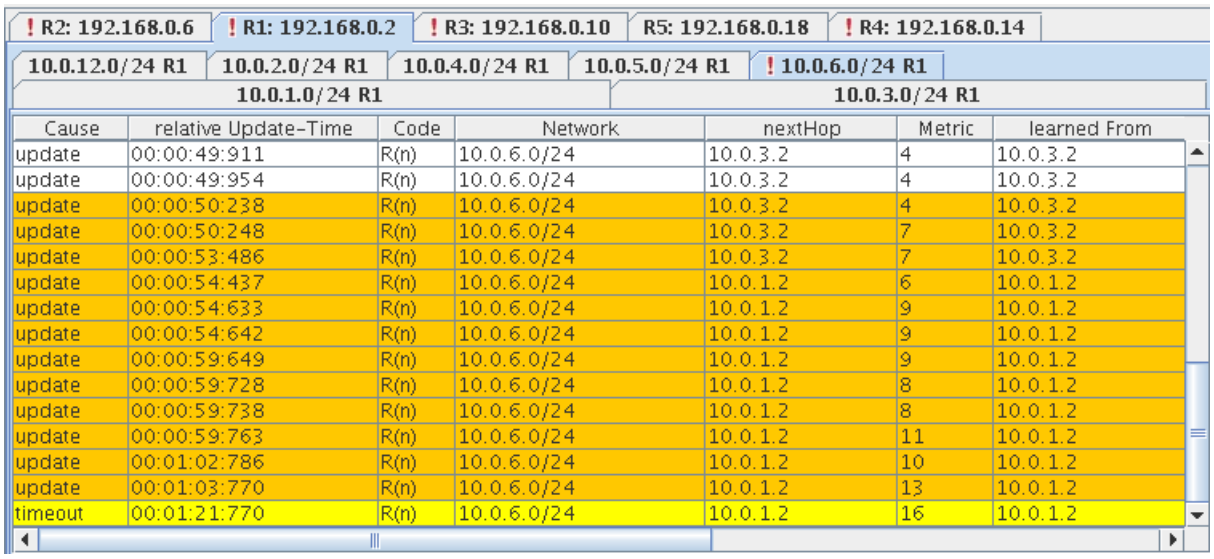
```
r3(x,1,1,0); r2(0,0,0); r1(0,0); r4(0,0); r5(x,0);
r3(2.4,1,0,0); r2(0,0,1.6)a; r1(0.8,0)a; r4(0,0)a; r5(x,0);
r3(2.4,1,0,0)a;
```

Durch Vorgabe bestimmter Latenzen lässt sich also ein Falsch-Update über alle 3 Router im Zyklus verteilen, ohne das RIP-MTI dies verhindern kann. Fällt anstatt R5 der Router R4 aus, kann beobachtet werden, dass der CTI zwar für Netz f, jedoch nicht für Netz e auftritt. Der Rückgabewert, der die Annahme des Updates bestimmt, erfolgt in beiden Fällen an der Stelle:

```
1 if (isold) return (combi == rme->metric);
```

Für Netz e hat der erste Parameter des Vergleichs den Wert 4 und der zweite den Wert 5, der Rückgabewert ist deshalb 0 (entspricht Ablehnen des Updates). Für Netz f haben beide Parameter den Wert 5, der Rückgabewert ist deshalb 1 (entspricht Annahme des Updates). Die Schleifenerkennung auf R1 wurde also aus dem gleichen Grund, der auch für das Fehlverhalten im Doublecon-Szenario sorgte, ausser Kraft gesetzt. Die Lösung des dort geschilderten Problems wird somit auch das Fehlverhalten des MTI-Algorithmus in diesem Szenario beseitigen.

Wenn das Falsch-Update zusätzlich auch zwischen den Routern R2 und R3 umherläuft, ändert sich die Metrik auf R1 nicht mehr mit dem Wert des Schleifenumfangs. Zwischen R2 und R3 wird die Metrik bei jedem Umlauf nur um den Wert 2 erhöht, zwischen R1, R2 und R3 jedoch um den Wert 3. Jeder Router wird dann seine Metrik immer auf den aktuell niedrigeren Wert anpassen. Der folgende Screenshot zeigt dieses Verhalten für den Router R1:



Cause	relative Update-Time	Code	Network	nextHop	Metric	learned From
update	00:00:49:911	R(n)	10.0.6.0/24	10.0.3.2	4	10.0.3.2
update	00:00:49:954	R(n)	10.0.6.0/24	10.0.3.2	4	10.0.3.2
update	00:00:50:238	R(n)	10.0.6.0/24	10.0.3.2	4	10.0.3.2
update	00:00:50:248	R(n)	10.0.6.0/24	10.0.3.2	7	10.0.3.2
update	00:00:53:486	R(n)	10.0.6.0/24	10.0.3.2	7	10.0.3.2
update	00:00:54:437	R(n)	10.0.6.0/24	10.0.1.2	6	10.0.1.2
update	00:00:54:633	R(n)	10.0.6.0/24	10.0.1.2	9	10.0.1.2
update	00:00:54:642	R(n)	10.0.6.0/24	10.0.1.2	9	10.0.1.2
update	00:00:59:649	R(n)	10.0.6.0/24	10.0.1.2	9	10.0.1.2
update	00:00:59:728	R(n)	10.0.6.0/24	10.0.1.2	8	10.0.1.2
update	00:00:59:738	R(n)	10.0.6.0/24	10.0.1.2	8	10.0.1.2
update	00:00:59:763	R(n)	10.0.6.0/24	10.0.1.2	11	10.0.1.2
update	00:01:02:786	R(n)	10.0.6.0/24	10.0.1.2	10	10.0.1.2
update	00:01:03:770	R(n)	10.0.6.0/24	10.0.1.2	13	10.0.1.2
timeout	00:01:21:770	R(n)	10.0.6.0/24	10.0.1.2	16	10.0.1.2

Abbildung 28: Auswirkungen des CTI

7.10 XMod-Szenario

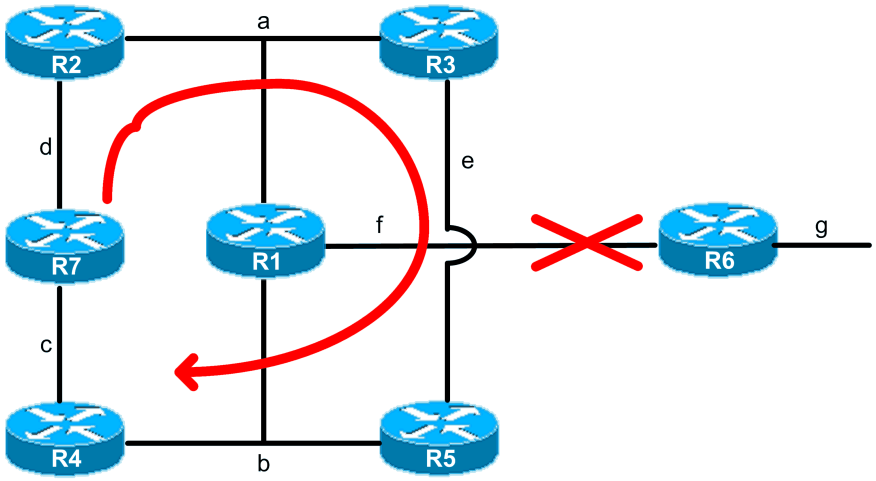


Abbildung 29: XMod-Szenario

Szenariobeschreibung:
Die Netze a und b sind „switched networks“, die je 3 Router miteinander verbinden. R1, R2, R3 und R1, R4, R5 bilden jeweils ein Teilnetz, in dem jeder Router mit den beiden anderen direkt verbunden ist (auf IP-Ebene). Da RIP Pakete per Multicast versendet, erreicht ein Update immer beide Nachbarrouter.

Beobachtungen:
Fällt der Router R6 aus, kann ein CTI im großen äußeren Zyklus auftreten. RIP-MTI kann diesen CTI nicht verhindern. R1 ist der Source-Router in dieser Topologie, er verhindert daher jeden CTI der in einem Zyklus über R1 auftritt. Der Zyklus über R2, R3, R5, R4, und R7 umgeht jedoch den Source-Router. Um den CTI in diesem Zyklus auszulösen, muss ein Update von R2 an R7 nicht, oder nur verspätet, bei R7 eintreffen. R7 sendet daraufhin ein Update mit alten Informationen an den Nachbarn R4. Diese Erreichbarkeit (mit alter Information) läuft im Folgenden durch den äußeren Zyklus.

Ergebnisse:
Dauer und Häufigkeit des Auftretens eines CTI unterscheiden sich bei RIP und RIP-MTI nicht. These: Da aus Sicht der Router R3 und R5 das Netz einer Topologie mit verschachtelten Schleifen entspricht, kann die Lösung des Interloop-Szenarios, hier angewendet werden.

7.11 YMod-Szenario

Szenariobeschreibung:
4 Router sind in einem Zyklus verbunden, wobei 2 gegenüberliegende Router eine Direktverbindung besitzen. Ein weiterer Router besitzt je eine Verbindung zu 2 Routern aus diesem Zyklus. Netz d wird nach Ausfall von R5 für R1, R2, R3 und R4 unerreichbar.

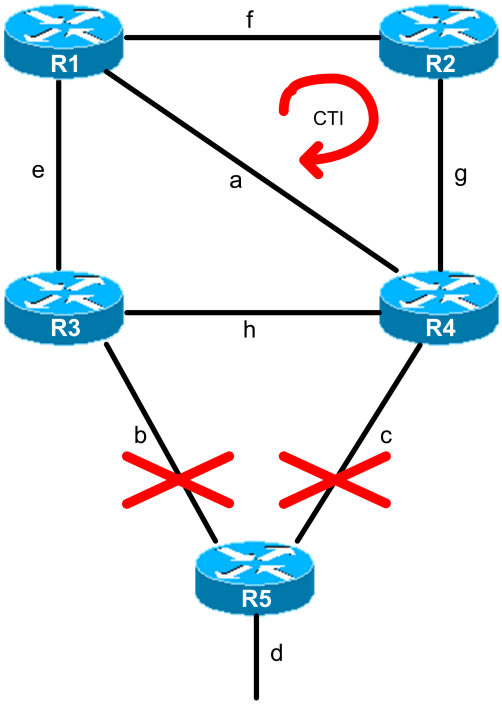


Abbildung 30: YMod-Szenario

Beobachtungen:
Fällt der Router R5 komplett aus, wird die Unerreichbarkeit, nach Ablauf des Timers auf R3 und R4, im Netz verbreitet. Geht das Update von R4 an R2 verloren und sendet R2 daraufhin ein Update mit alten Informationen an den Nachbarn R1, kann ein CTI innerhalb des Zyklus R1, R4, R2 entstehen. RIP-MTI kann diesen CTI nicht verhindern. Auf R3 arbeitet RIP-MTI korrekt und lehnt das Update mit falscher Metrik von R1 ab.

Ergebnisse:
Dauer und Häufigkeit des Auftretens eines CTI unterscheiden sich bei RIP und RIP-MTI für den Zyklus über R1, R4, R2 nicht. Ein CTI über alle 4 Router kann in diesem Szenario mit RIP-MTI nicht auftreten.
These: Da aus Sicht der Router R1 und R4 das Netz einer Topologie mit verschachtelten Schleifen entspricht, kann die Lösung des Interloop-Szenarios hier angewendet werden.

7.12 Cycle-Szenario

Szenariobeschreibung:
Dieses Szenario besteht nur aus einem Zyklus. Da keine weiteren Router ausserhalb des Zyklus existeren, kann kein CTI entstehen. Aus Sicht von Router R1 kann Netz b über R2 und R3 erreicht werden. Annahme: R1 lernt Netz b von R2.

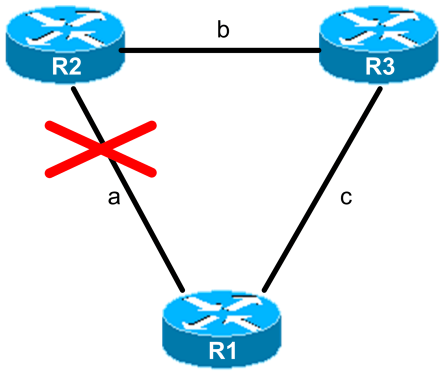


Abbildung 31: Cycle-Szenario

Beobachtungen:
Fällt die Verbindung zwischen R1 und R2 aus, wird auf R1 nach Ablauf des Timeout-Timers die Metrik für Netz b auf 16 gesetzt. Da RIP zuvor gelernte, aber nicht angewendete Routen vergisst, muss bis zum nächsten Update von R3 gewartet werden, bis die Alternativ-Route zu Netz b über R3 gelernt wird. Dies dauert, je nach eingestellter Periode, zwischen 3 und 30 Sekunden. RIP-MTI nimmt den neuen Weg über R3 jedoch nicht sofort an, sondern erst nach Ablauf des Garbage-Collect-Timers. Dies dauert, je nach eingestellter Periode, zwischen 12 und 120 Sekunden.
Im RFC 2453 [RFC98] ist beschrieben, dass eine Route, für die der Garbage-Collect-Timer läuft, durch eine neu gelernte ersetzt werden kann:

> `Should a new route to this network be established while the garbage-collection timer is running, the new route will replace the one that is about to be deleted. In this case the garbage-collection timer must be cleared.`

Das Verhalten wird von RIP-MTI hier nicht umgesetzt.

Ergebnisse:
Der RIP-MTI Algorithmus nimmt eine Alternativ-Route nicht an, wenn das Zielnetz Teil eines Zyklus ist. RIP nimmt diese Route spätestens nach 30 Sekunden an, RIP-MTI erst nach Streichung der alten Route aus der Routing-Tabelle (120 Sekunden). RIP benötigt also 1 Update zur Konvergenz, RIP-MTI benötigt 3 Updates. Die Einträge der Log-Datei bestätigen, dass die neue Route durch RIP-MTI abgelehnt wurde. Die `cycleok()`-Funktion wird mit den Metriken 2 und 0 aufgerufen. Der Rückgabewert ist 0 (entspricht Ablehnen des Updates), da der errechnete Wert für `combi` (1) kleiner als `rme->metric` (3) ist.

```
1 RIP: rip_mti_routeok: Old route from interface eth1 (1)
2      with metric 16
```

```
3 RIP: rip_mti_routeok: New route from interface eth2 (2)
4       with metric 2
5 RIP: oldroute->metric is 16, oldroute->oldmetric is 0
6 RIP: routemetric is 0
7 RIP: cycleok called for ifindexA 2, ifindexB 1,
8       metricA 2, metricB 0
9 RIP: is old, combi is 1, rme metric is 3
```

8 Fazit

Diese Arbeit stellt ein Werkzeug zur Verfügung, das strukturierte Tests des RIP-MTI Algorithmus vereinfachen, beschleunigen und automatisieren kann. Die vormals zwei Dimensionen Topologie und Updatekonstellation, auf die die MTI-Erweiterung getestet werden musste, konnten auf den variablen Anteil der Topologie vereinfacht werden. Die zeitliche Reihenfolge des Auftretens der Updates kann zentral gesteuert werden.

Bisher mussten Tests händisch und sehr aufwändig über Skripte auf der Konsole gesteuert werden. Die entwickelte Testumgebung „XTPeer“ ermöglicht es, die gleichen und viele weitere Tests mit kleinem Aufwand durchzuführen. Ein CTI kann dabei in ca. 45 Sekunden provoziert und beobachtet werden. Automatische Auswertungen, z.B. über Häufigkeit und Dauer von CTIs, erleichtern die Analyse der durchgeführten Tests.

Der Generator führt selbstständig vorgegebene Updatereihenfolgen aus oder kann zufällig Reihenfolgen generieren. Der Auto-Generator versucht innerhalb der Topologie kritische Pfade zu finden und generiert aus diesem Wissen Updatereihenfolgen, die CTIs verursachen können. Wurde ein Forecast, der einen CTI erzeugt, gefunden, kann dieser in einer Config-Datei abelegt und später erneut ausgeführt werden. Für alle Zufallswerte kann eine Wahrscheinlichkeitsverteilung angegeben werden. Insgesamt wurden alle Elemente der Anforderungsanalyse, die mit *muss* und *soll* bezeichnet wurden, realisiert.

Der Static-Generator erlaubt die Ausführung einer Config-Datei, die innerhalb des Forecasts die gewünschte Updatereihenfolge enthält. Weiterhin wurde festgestellt, dass die letzten beiden Zeilen eines Forecasts verantwortlich für das Auftreten eines CTI sind. Der HalfRandom-Generator erzeugt genau diese Zeilen durch Zufallswerte, so dass automatisiert eine große Anzahl verschiedener Testfälle durchgeführt werden kann. Die Erfolgsrate einen CTI über Zufallswerte zu entdecken, hängt stark von der verwendeten Wahrscheinlichkeitsverteilung ab. Je höher die Wahrscheinlichkeit für einen Paketverlust ist, umso höher ist auch die Wahrscheinlichkeit für einen CTI. Dieses -zu erwartende- Verhalten konnte durch eine Reihe von Testdurchläufen mit dem HalfRandom-Generator bestätigt werden.

Der Fullrandom-Generator erzeugt von Beginn an komplett zufällige Updatereihenfolgen. Dieser Brute-Force Ansatz führte jedoch zu keinem Erfolg[14]. Dies liegt an der geringen Wahrscheinlichkeit, dass ein Router über mehrere Perioden (mindestens 6) auf einem Interface kein Update mehr sendet. Hierzu muss an der gleichen Stelle des Forecasts, in jeder Periode, der Wert „x“ auftreten.

Der Auto-Generator findet in allen getesteten Szenarien mindestens einen kritischen Pfad

[14] Nach 12 Stunden Ausführung des FullRandom-Generators konnte in keinem Szenario ein CTI, bzw. ein verhinderter CTI, beobachtet werden.

der einen CTI verursacht. Insbesondere die, in den Szenarien Doublecon, YDouble und XMod, gefunden CTIs wurden durch den Auto-Generator entdeckt. Die vollständig automatisierte Ausführung ist in diesen Szenarien besonders hilfreich, da die Lücken in der MTI-Implementierung dazu führen, dass CTIs nur beim Ausfall bestimmter Router auftreten, beim Ausfall anderer Router jedoch verhindert werden.

8.1 Vergleich RIP und RIP-MTI

Die nachfolgende Tabelle betrachtet das Verhalten von RIP und RIP-MTI, mit Betonung der Konvergenzprobleme:

	RIP	RIP-MTI
normales Verhalten oder Startphase	0	0
CTI in X-/Y-ähnlichen Szenarien	-	++
CTI in verschachtelten Schleifen	-	-
CTI bei Doppelverbindungen	-	-
Nicht-Akzeptanz von richtigen Routen	+	-

Tabelle 4: Vergleich bezüglich CTI Verhalten (- $<$ 0 $<$ +)

Tritt ein CTI bei Verwendung des RIP-MTI auf, unterscheidet sich dieser nicht von einem CTI bei Verwendung des RIP-Protokolls. Dauer und Anzahl der Updates sind identisch. Dies ist mit der Implementierung des RIP-MTI zu begründen: Wird eine Route durch den MTI-Algorithmus akzeptiert, führt dies zur normalen weiteren Ausführung des RIP-Code.

Es konnte weiterhin beobachtet werden, dass ein CTI im Allgemeinen eine zeitliche Ausprägung im Bereich von 2 - 30 Sekunden besitzt. Das liegt an den, mit RIPv2[15] eingeführten, Triggered Updates, die die Konvergenz wesentlich beschleunigen. Zum Vergleich: Ein CTI ohne Triggered Updates konnte mit RIPv1 noch mehrere Minuten dauern.

Variationen innerhalb des oben genannten zeitlichen Rahmens werden hauptsächlich durch die Größe des Zyklus bestimmt. Dabei ist ein CTI in großen Zyklen potenziell kürzer als in kleineren. Dieses Verhalten ist auf die Anzahl der Umläufe und damit auch auf die Anzahl wie oft jeder einzelne Router durchlaufen wird, zurückzuführen (s. Kapitel 2.3). In einem Zyklus aus zwei Routern, muss nach jedem Hop eine Zufallszeit zwischen 1 und 5 Sekunden abgewartet werden, bevor das nächste Update getriggert wird. Die Wahrscheinlichkeit auf jedem Router warten zu müssen, ist hier höher und verzögert deshalb den CTI.

[15] RIPv2 wurde 1998 als Nachfolger von RIP eingeführt.

8.2 Problemfälle des MTI

Die in dieser Arbeit untersuchten und neu entdeckten Problemfälle des MTI-Algorithmus werden im Folgenden zusammenfassend beschrieben.

Bigloop-Szenario In diesem Szenario arbeitet der MTI-Algorithmus korrekt. Eine Eigenschaft des Algorithmus verringert hier jedoch den Nutzen des MTI: Durch den Umfang des Zyklus ist das abgelehnte Update am Source-Router auch gleichzeitig das letzte Update des CTI, d.h. auf allen Routern vor dem Source-Router existiert die Falsch-Information weiterhin. In der Praxis sollten Topologien in dieser Form jedoch nicht vorkommen.

Interloop-Szenario Im Interloop-Szenario kann der Source-Router den CTI über die äußere Schleife verhindern. In der oberen Schleife kann trotzdem ein CTI auftreten. RIP-MTI schneidet also nur minimal besser ab, da nicht alle CTIs verhindert werden.

Doublecon-Szenario In diesem Szenario verhindert RIP-MTI den CTI bei Ausfall von Router R3. Fällt R4 aus, kann er nicht verhindert werden. Ursache ist eine falsche Implementierung, der Algorithmus würde auch hier den CTI verhindern. RIP-MTI schneidet also nur minimal besser ab, da nicht alle CTIs verhindert werden.

YDouble-Szenario RIP-MTI kann die CTIs zwischen den Routern R2,R3 bzw. R1,R2,R3 nicht verhindern. Eine korrekte Implementierung sollte auch diesen Fall lösen, da der Algorithmus den CTI verhindert müsste. Zwischen RIP und RIP-MTI existiert in diesem Szenario kein Unterschied.

XMod-Szenario RIP-MTI kann einen CTI nur auf Schleifen, die über R1 laufen, verhindern. In der äußeren Schleife kann trotzdem ein CTI auftreten. Die Lösung des Problems im Interloop-Szenario sollte auch diesen CTI verhindern. Da nicht alle CTIs verhindert werden, schneidet RIP-MTI nur minimal besser als RIP ab.

YMod-Szenario In diesem Szenario kann RIP-MTI den CTI zwischen den Routern R1,R2,R4 nicht verhindern. Die Lösung des Problems im Interloop-Szenario sollte auch diesen CTI verhindern. Da nur der CTI in der äußeren Schleife verhindert wird, schneidet RIP-MTI nur minimal besser als RIP ab.

Cycle-Szenario In diesem Fall schneidet RIP-MTI schlechter als RIP ab, da mehr Zeit und mehr Updates bis zur Konvergenz benötigt werden. Eine Alternativ-Route, die aus einem Zyklus stammt, wird nicht angenommen. Der in Kapitel 7.5 (X-Szenario) vorgestellte Problemfall entspricht genau diesem Verhalten. Auch hier schafft die korrekte Implementierung Abhilfe, da sich der MTI-Algorithmus in diesem Fall wie ein normaler RIP-Router verhalten sollte.

8.3 Ausblick

Die vorgestellten Verfahren vereinfachen die Tests des MTI-Algorithmus in Bezug auf variable Updatereihenfolgen. Um auch die Vielzahl möglicher Topologien zu berücksichtigen, wurden diese Verfahren in einer repräsentativen Auswahl von Szenarien getestet. Jede Topologie muss dabei vom Benutzer selbst erstellt werden. Um eine vollautomatisierte Testumgebung zu erhalten, müssten auch die Szenarien automatisch generiert werden. Da die für das Starten und Beenden eines Testszenarios benötigte Zeit jedoch wesentlich größer als die zum Test selbst benötigte Zeit ist, scheint hier eine sorgfältige Vorauswahl durch den Benutzer sinnvoller.

Verbesserungpotenzial besteht weiterhin für den Auto-Generator, da in komplexen Topologien zuviele Pfade als CTI-Kandidaten behandelt werden. Insbesondere die Erweiterung auf eine Erkennung für mehrere Schleifen (X-Szenario) würde diesen Generator robuster machen.

Da die Schwachstellen des MTI nicht durch den Algorithmus selbst entstehen, sondern durch in der Implementierung nicht beachtete Fälle, können diese in nachfolgenden Arbeiten behoben werden. Im Laufe dieser Arbeit stellte sich immer wieder heraus, dass sich jeder Problemfall des MTI auf die hier genannten reduzieren lässt. Eine Lösung der Probleme:

- Verschachtelte Schleifen
- Doppelverbindungen
- Nicht-Akzeptanz von richtigen Routen

sollte demnach auch die Problemfälle in komplexeren Szenarien lösen. Sind die Schwachstellen behoben, sollten die gleichen Testfälle erneut durchgeführt werden, um die Korrektheit der neuen Implementierung für diese Fälle zu zeigen.

Literaturverzeichnis

[Boh07] Frank Bohdanowicz. Evaluierung der RIP-MTI Quagga Implementierung. Diplomarbeit, Uni-Koblenz, 2007.

[Cis06] Internetworking Technologies Handbook. URL: http://www.cisco.com/univercd/cc/td /doc/cisintwk/ito_doc/rip.pdf, Stand 17.07.2007. Cisco Documentation, 2006.

[Cis07] Configuring EIGRP. URL: http://www.cisco.com/en/US/docs/ios/12_0/np1/ configuration/guide/1ceigrp.pdf, Stand 17.07.2007. Cisco IOS Release 12.0 Network Protocols Configuration Guide, Part 1, 2007.

[Dua07] Diffusing Update Algorithm. http://de.wikipedia.org/wiki/Diffusing_Update_Algorithm, Stand 17.07.2007. Wikipedia - Die freie Enzyklopädie.

[DV07] Distanzvektoralgorithmus. URL: http://de.wikipedia.org/wiki/Distanzvektoralgorithmus, Stand 17.07.2007. Wikipedia - Die freie Enzyklopädie.

[Fra05] Leif Franker. Simulation des Routing Information Protocol und des Routing Information Protocol with Minimal Topology Information unter VNUML. Studienarbeit, Uni-Koblenz, 2005.

[Keu06] Tim Keupen. Verteilte Simulationen und externe Verbindungen mit Virtual-Network User-Mode Linux Studienarbeit, Uni-Koblenz, 2006.

[Koc05] Tobias Koch. Implementation und Simulation von RIP-MTI. Diplomarbeit, Uni-Koblenz, 2005.

[Lan07] Stefan Lange. Zentrale Betrachtung von Routing-Informationen zur Analyse des Konvergenzverhaltens verschiedener RIP-Algorithmen und Unterstützung des Generierens von Testfällen. Diplomarbeit, Uni-Koblenz, 2007.

[Lin04] Janne Lindqvist. Counting to Infinity. URL: http://www.tml.tkk.fi/Studies/T-110.551/2004/papers/Lindqvist.pdf, Stand 19.07.2007. Helsinki University of Technology, 2004.

[Mue06] Markus Müller. VNUML Einführung in die Simulation von Rechnernetzen. Studienarbeit, Uni-Koblenz, 2006.

[Net06] Routing-Protokolle, Grundlagen. URL: http://www.searchnetworking.de/themenkanaele/ einfuehrunginnetzwerke/routerbasics/articles/47695/, Stand 17.07.2007. 2006.

[Pae06] Daniel Pähler. Extern steuerbare Routing-Updates im RIP-Daemon der Quagga-Programmsuite. Diplomarbeit, Uni-Koblenz, 2006.

[Per01] Charles E. Perkins. Ad hoc networking. Addison-Wesley, Amsterdam 2001.

[PD04] Larry L. Peterson and Bruce S. Davie. Computernetze. Dpunkt Verlag, 3rd edition, San Francicso 2004.

[RFC88] RFC 1058 - Routing Information Protocol.
URL: http://www.faqs.org/rfcs/rfc1058.html, Stand 16.07.2007. 1988.

[RFC95] RFC 1812 - Requirements for IP Version 4 Routers.
URL: http://www.faqs.org/rfcs/rfc1812.html, Stand 16.07.2007. 1995.

[RFC97] RFC 2091 - Triggered Extensions to RIP to Support Demand Circuits.
URL: http://www.faqs.org/rfcs/rfc2091.html, Stand 16.07.2007. 1997.

[RFC98] RFC 2453 - RIP Version 2.
URL: http://tools.ietf.org/html/rfc2453, Stand 09.08.2007. 1998.

[RFC03] RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing.
URL: http://www.faqs.org/rfcs/rfc3561.html, Stand 16.07.2007. 2003.

[Rup04] Chris Rupp. Requirements-Engineering und -Management: professionelle, iterative Anforderungsanalyse für die Praxis. München, Hanser, 2004.

[Sch99] Andreas Schmid. RIP-MTI: Minimum-effort loop-free distance vector routing algorithm. Diplomarbeit, Uni-Koblenz, 1999.

[Vnu07] VNUML Projekthomepage. http://www.dit.upm.es/vnumlwiki/index.php/Main_Page, Stand 16.07.2007. Technical University of Madrid (UPM), 2007.

[Wol06] Bernhard Wolf. Untersuchung und Simulation des RIP-MTI-Algorithmus. Studienarbeit, Uni-Koblenz, 2006.

9 Anhang A

Forecast aus ycti.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 16 kommen nicht an
R1(0,0); R2(0,0); R3(x,0,0); R4(x,0);
// R1 versendet Falsch-Update
R1(1,0)a; R2(0,0)a; R3(x,0,0)a; R4(x,0);
```

10 Anhang B

Forecast aus ymanycti.config:

```
Update-Forecast
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);
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);

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

11 Anhang C

a) Forecast aus xcti2.config:

```
Update-Forecast
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);
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);

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

b) Forecast aus xcti.config:

```
Update-Forecast
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);
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);

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

12 Anhang D

Forecast aus interloopcti.config:

```
Update-Forecast
r3(0,0); r2(0,0,0); r1(0,0,0); r0(0,0); r4(0,x,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(0,0); r4(0,x,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(0,0); r4(0,0,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(x,x); r4(0,0,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(x,x); r4(0,0,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(x,x); r4(0,0,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(x,x); r4(0,0,0);
r3(0,0); r2(0,0,0); r1(0,0,0); r0(x,x); r4(0,0,0);

r3(0,0); r2(0,x,0); r1(0,0,0.5); r0(x,x); r4(0,0,0);
r3(1,1)a; r2(0,x,0)a; r1(0,0,0)a; r0(x,x); r4(0,0,0)a;
```

13 Anhang E

Forecast aus ydoublecti.config:

```
Update-Forecast
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);
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);

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

14 Anhang F

Forecast aus ypursuitcti.config:

```
Update-Forecast
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);
R1(0,0); R2(0,0); R3(0,0,x); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,x); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,x); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,x); R4(x,0);
R1(0,0); R2(0,0); R3(0,0,x); R4(x,0);
R1(0,0); R2(x,0); R3(x,0,x); R4(x,0);
// gute Nachricht folgt schlechter Nachricht
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
R1(1,x); R2(x,0); R3(1,x,x); R4(x,0);
```