Anpassung der XTPeer auf VNUML 1.8

Studienarbeit
AG Rechnernetze
SS 2008

Hamza Armagan
armagan@uni-koblenz.de
Matrikelnummer: *205210663*

Universität Koblenz-Landau

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

31. Oktober 2008

Erklärung

Ich, HAMZA ARMAGAN (Student der Informatik an der Universität Koblenz-Landau, Matrikelnummer 205210663), versichere dass ich die vorliegende Studienarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Studienarbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

HAMZA ARMAGAN

Veröffentlichung

Ich erkläre mich damit einverstanden, dass diese Studienarbeit in digitaler oder ausgedruckter Form von der Universität Koblenz öffentlich zugänglich gemacht wird. Dies schließt die Auslage von Exemplaren in der Universitätsbibliothek ein.

Inhaltsverzeichnis

1 Einleitung

Die Einleitung beinhaltet als erstes Unterthema die Motivation und Zielsetzung. Dort wird das breite Anwendungsgebiet von Rechnernetzen vorgestellt und den Zusammenhang der sich daraus resultierenden Motivation wiedergeben. Im nächsten Unterthema "Aufbau der Studienarbeit" wird auf eine kurze Art und Weise auf die Kapiteln mit kurzen Tehmatikbeschreibungen eingegangen. Im Abschnitt Entwicklungsumgebung wird die in dieser Arbeit verwendete Hard- und Software beschrieben. Als Schlusskapitel der Einleitung werden grundlegende Begriffserklärungen gemacht, die in dem Verlauf der Studienarbeit vorkommen und zu verstehen sind.

1.1 Motivation und Zielsetzung

Rechnernetze gehört zu den Fächern, das ein breites Spektrum an Anwendungsgebieten abdeckt. Die vielen unterschiedlichen Protokolle sind Belege dafür. Jedes Protokoll hat sozusagen ein eigenes Anwendungsbereich und doch sind deren Aufgaben unter dem Nenner gleich. Miteinander zu Kommunizieren, Informationen auszutauschen und das möglichst auf eine sichere und schnelle Art und Weise. Als Beispiel denke man an Autos, Handys, Lokale Netze in einem kleinen Betrieb, Intranet in einer größeren Organisation oder in Netze die über Länder hinweggehen. Ein modernes Auto an sich hat schon eine Vielzahl an Bussystemen in sich integriert. CAN, LIN, FlexRay oder MOST Bus sind ein paar Stichworte dazu. Wenn man jetzt von der Autoindustrie weggeht und an die Autonomen Systeme kommt, gibt es dort ein Routing Information Protokoll dass innerhalb von Autonomen Systemen verwendet wird. Routing Information Protokoll ist eines der meist verwendeten Protokollen innerhalb von Autonomen Systemen. Das Routing Information Protokoll ist einer zentralen Bestandteile des Forschungsbereiches der AGSteigner. Die Arbeit widmet sich unter Anderem dieser Forschungsunterstützung.

Diese Studienarbeit hat das Ziel die Testumgebung XTPeer auf die VNUML Version 1.8 zu integrieren. Die Arbeit soll es den zukünftigen Studenten wie Diplomanden, Praktikanten oder Studienarbeit Interessenten, der in diesem Gebiet tätig werden will, seine Forschungen auf der Basis von aktuellster Entwicklungsumgebung durchführen zu können. Ferner bringt die Arbeit den Vorteil mit sich, das die Einarbeitungszeit in die theoretische und praktische Thematik reduziert wird. Des weiteren beinhaltet diese Arbeit Fehler und Lösungsvorschläge, die bei der praktischen Umsetzung des Projektes vorgekommen sind. Die Festhaltung von Erfahrungsberichten bringen immer Zeitvorteile mit sich, die man anderweitig wie in die Forschung und Entwicklung investieren kann.

1.2 Aufbau dieser Studienarbeit

Der Abschnitt 2 geht auf die Grundlagen in Bezug auf Theorie und der verwendeten Software ein. Die Theorie beinhaltet das Routing Information Protokoll. Unter ungünstigen Informationsaustauschzeiten zwischen Router können Routingschleifen entstehen. Ziel dieses Kapitel ist das Verständnis dieses Problems. Ferner gehe ich in dem Abschnitt 2 auf die Tools VNUML und XTPeer ein. Was

sind deren Aufgaben? Wie ist deren GUI aufgebaut? Was für Funktionalitäten lassen sich mit den Tools realisieren? Desweiteren wird auf die Softwaresuite Quagga eingegangen. Wie man aus dem Namen Softwaresuite ableiten kann handelt es sich hierbei um eine Ansammlung von Routingprotokolle die größtenforms als Standardkonform zertifiziert wurden. Fragen wie welche Protokolle sind in der Quagga implementiert, wie ist die Softwarearchitektur aufgebaut und was für Gedanken stecken hinter der Quagga Architektur werden im Abschnitt 2.4 behandelt.

Der Abschnitt 3 widmet sich dem Root Filesystem. Zuerst wird eine Einführung in das Root Filesystem gemacht und erklärt was ein Root Filesystem ist. Ferner werden unterschiedliche Root Filesystem kurz angeschnitten und deren Unterschiede vermerkt. Desweiteren wird auf die Softwareinstallationsmöglichkeiten (auch Vor- und Nachteile) eines Root Filesystems eingegangen. Im Abschnitt 3.1 werden Autools vorgestellt die bei der Projektumsetzung von Wichtigkeit sind. Denn eventuelle Versionsabweichungen der Autotools bringen dann Folgeprobleme mit sich. Der letzte Abschnitt 3.2 dieses Kapitels widmet sich dem Mini Image. Alternativ zum „normalen“ Root Filesystem gibt es das Mini Image. Nach der kurzen Vorstellung wird dann das Vorgehen zur Softwareinstallation beim Mini Image beschrieben. Die im Abschnit 3 vorgestellten Hinweise, die bei der praktischen Anpassungs- und Umsetzungsbeschreibungen mitunter aufgetreten sind, mögen zwar in erster Linie „klar“ sein, aber dennoch bringen gerade solche kleine Ungenauigkeiten bei der praktischen Umsetzung viel Zeit- und Kostenaufwand mit sich.

Der Abschnitt 4 geht auf die Umsetzung und Verwendung von den bis dahin vorgestellten Tools, Software, verändertem Root Filesystem und den theoretischen Grundlagen ein. Zu Beginn von Abschnitt 4 wird die Beispielstopologie vorgestellt. Im Anschluss darauf wird eine Auflistung der Komponenten gemacht, die für das Simulation notwendig sind und die auch in der begleitenden CD vorhanden sind. Danach wird gezeigt wie das Szenario gestartet, gezielt ausgeführt bzw. Count-To-Infinity provoziert wird. Anhand von den resultierenden Diagrammen in XTPeer kann das Protokoll analysiert und interpretiert werden. Das ordnungsmäßige herunter fahren des Systems darf natürlich auch nicht vergessen werden. Der Abschnitt 4 kann gewissermaßen als eine kleines Einstiegs-, Bedienungs- und Lerntutorial betrachtet werden, indem alle Komponenten (VNUML, XTPeer, Quagga, angepasstes Rootfilesystem, Wissen) zum Einsatz kommen.

Der Abschnitt 6 widmet sich einer kleinen Abschlussbetrachtung mit persönlichen Nutzen.

1.3 Entwicklungsumgebung

Ausgeführt wurde diese Studienarbeit auf einem Notebook der Marke Acer Extensa 2900. Arbeitsspeicher mit 1,2 GB RAM haben zum genügen ausgereicht. An dieser Stelle sei noch erwähnt das der Arbeitsspeicher nicht unter 512 MB sein sollte. Denn die VNUML Simulation ist sehr rechenintensiv. Beim CPU handelt es sich um einen Intel® Pentium® M Prozessor 1.50 GHz. Als Be-

triebssystem habe ich zum einen mal die Ubuntu Version 7.10 und zum anderen mal die Ubuntu Version 8.04 benutzt. Mit der Ubuntu Version 7.10 war ich nicht so ganz zu frieden wie im Vergleich zur 8.04 Version. Denn in der Version 7.10 war mein Notebook meistens stark beansprucht indem er immer erwärmt wurde. Bei der Ubuntu Version 8.04 war die Durchführung meiner Arbeit viel angenehmer. Der Rechner wurde vom äußeren Gefühl her nicht so sehr beansprucht.

1.4 Begriffe

1.4.1 Router:

Ein Router ist eine Hardware, welcher auf der Schicht 3 des OSI[1] - Modells arbeitet. Er verbindet verschiedene Rechnernetze miteinander. Ankommende Datenpakte werden analysiert. Es wird geschaut ob die Zieladresse in der Routingtabelle vorhanden ist. Ist die Zieladresse aufgeführt, so wird das Paket zur Zieladresse über das entsprechende Interface weitergeleitet. Man sollte an diesem Punkt den Unterschied zwischen Routing und Forwarding erklären. Häufig wird fälschlicherweise mit Routing eigentlich Forwarding gemeint. Forwarding bedeutet das Weiterleiten der Pakete anhand der vorhanden Routingtabellen. Das Routing hingegen ergänzt die Routintabellen in Abhängigkeit von Routingprotokollen (BGP[2], RIP[3], OSPF[4]) in Router.

1.4.2 Autonomes System: AS

In einem AS sind Router und Netzwerke einer einzigen administrativen Instanz untergeordnet. Unsere Universität Koblenz wäre zum Beispiel ein AS. Jede Einrichtung (Labor) hat dementsprechend sein eigens Netz. Der Verbund solcher Netze stellt quasi ein AS dar. Jedes Autonome System besitz eine einzigartige Identifikationsnummer die weltweit durch die RIRs (Regional Internet Registries) Vergeben wird. Somit können unterschiedliche Autonome System miteinander kommunizieren. Der Verbund von Autonomen Systemen stellt das Internet dar.

1.4.3 Graphical User Interface: GUI

Die GUI ist eine Benutzerschnittstelle für den Benutzer. Sie erlaubt eine einfachere Handhabung der elektrischen Geräte (Computer, MP3, Navigation,...). Eine Vereinfachte Handhabung bei der Benutzung von Computer sind zum Beispiel die Betriebssysteme (wie Microsoft Windows, Ubuntu) die über eine GUI Schnittstelle verfügen. Somit kann der Anwender leicht und übersichtlich sein Computer bedienen. Zur leichten Bedienung tragen mitunter die Maus, Buttons, Menüleisten, Eingabefelder, Radio Buttons usw. bei. Die Grafische Benutzerschnittstelle ist aber nicht unbedingt schneller als das Command Line Interface (CLI) zu Bedienen. Wenn der Anwender/Entwickler die Befehle und Verzeichnisse seines Computers kennt, so ist er auf textbasierten Kommandozeile (CLI) doch schneller. Alles in Allem ist die GUI eine erhebliche Vereinfachung für

[1] OSI-Referenzmodell; engl. Open Systems Interconnection Reference Model
[2] BGP - Border Gateway Protocol
[3] RIP - Routing Information Protocol
[4] OSPF - Open Shortest Path First

Anwender in allen Schichten. Die Einarbeitungszeit in dem Bedienen von elektrischen Geräten ist vergleichsweise nicht so hoch. Das Verständnis über das System fällt nicht so abstrakt.

1.4.4 Extensible Markup Language: XML

XML steht für Extensible Markup Language und ist ein Verbindungsglied zwischen HTML und einer Objektsprache. Mittlerweile wird XML als genormtes Datenaustauschformat zwischen verteilten Systemen gesehen. Man bezeichnet XML als Datenmodell für semi - strukturierte Daten. Darunter versteht man Daten, die zum großen Teile eine fest vorgegeben Struktur besitzen, gleichzeitig aber auch Elemente beinhalten, die diesem statischen Schema nicht unterliegen. Die XML Sprache ist sehr einfach aufgebaut. Das Dokument besteht hauptsächlich aus drei Abschnitten:

- Einer optionalen Präambel (u.a. sollte die zugrunde liegende XML-Version angegeben sein).
- Einem optionalen Schema (der so genannten Document Type Defintion, DTD, oder dem neueren XML Schema).
- Einem einzigen Wurzelelement, das dann beliebig viele und beliebig tief geschachtelte Unterelemente beinhalten kann. Ein Element wird immer von einem Start- und Ende-Tag eingeschlossen.

1.4.5 Secure Shell: SSH

SSH ist ein sicheres Netzwerkprotokoll mittels dem man ein Zugriff auf ein entfernten Rechner machen kann. Der Grund für die Entwicklung des SSH Protokolls war, das die unsicheren Verbindungsprotokolle wie Telnet ersetzt werden sollte. Bei Telnet wird bekanntlich sogar das Passwort unverschlüsselt versendet. Demgegenüber bietet die SSH eine sichere verschlüsselte und authentifizierte Verbindung an. Anwendungsgebiete finden sich hauptsächlich in linux/Unix basierten Systemen. Eine typisches Anwendungsgebiet von SSH ist das man sich von einem Hostrechner aus in einem entfernten Rechner einwählt und dort entsprechende Befehle ausführt. SSH leitet die Befehl vom Hostrechner aus weiter zum Remoterechner. Entsprechende Terminalausgaben kommen dann vom Remoterechener zurück und werden im eigenen Terminal ausgegeben.

2 Grundlagen

Dieses Kapitel geht auf die Grundlagen ein, die zum Verständnis des theoretischen und praktischen Hintergrundes, die in Verbindung mit dieser Studienarbeit stehen. Zunächst einmal wird auf das RIP Protokoll eingegangen. RIP steht für Routing Information Protocol und ist eines der meist verbreiteten Protokollen innerhalb von Autonomen Systemen. In Netzen wo das RIP Protokoll zum Einsatz kommt kann anhand von Routingschleifen der Count-to-Infinty entstehen. Dieses Count-to-Infinty Problem wird dann im Anschlusskapitel zu Routing Information Protocol erklärt. Im Anschluss zu den beiden Kapiteln wird dann auf die Softwaregrundlagen eingegangen die ebenfalls in Verbindung mit dieser Arbeit stehen . Als erstes wird das Softwaretool VNUML vorgestellt. VNUML ist ein Tool zur Simulation von einem virtuellen Netzwerk auf einem Hostrechner, welches an der Technischen Universität Madrid entwickelt wurde. Folgend zu VNUML wird dann eine Routing Software Suite vorgestellt. Bei dieser Routing Software Suite handelt es sich um Quagga. Quagga implementiert verschiedene Routingprotokolle für das Internet in der auch das Routing Information Protocol mit enthalten ist. Nach der Einführung in Quagga wird letztendlich das Tool XTPeer vorgestellt. XTPeer wurde an der Universität Koblenz entwickelt und ist ein Steuerungs- und Analysetool. Als Abschluss dieses Kapitels wird dann eine Zusammenfassung gegeben.

2.1 Routing Information Protocol

Das RIP (Routing Information Protocol) ist ein dynamisches Routingprotokoll. RIP baut auf dem Bellman - Ford Algorithmus (Distanzvektoralgorithmus) auf und wird in den Protkollen wie IP und IPX benutzt. Das Einsatzgebiet vom RIP Protokoll sind überschaubare kleine bis mittlere Netze innerhalb von Autonomen Systemen von maximal bis zu 15 Router. Routingprotokolle die innerhalb von Autonomen Systemen arbeiten werden als IGP (Interior Gateway Protocol) bezeichnet. RIP gehört auch zu den IGP Protokollen und ist eines der meist verbreiteten IGP Protokollen. Es gibt 3 Versionen von RIP (RIP, RIPv2 und RIPng). Das RIP Protokoll in der Version 1 wurde erstmals 1988 im RFC 1058 spezifiziert. In den darauf folgen Jahren unterlag RIP mehreren Erweiterungen bis letztendlich RIP in der Version 2 (RFC 2453) im Jahre 1993 spezifiziert wurde. In RIPv2 sind Erweiterungen wie Authentifizierungsmöglichkeiten, Unterstützung von Subnetzen und CIDR eingeflossen. Beide Version von RIP (RIP und RIPv2) kommen gegenwärtig zum Einsatz. Letztendlich gibt es auch die Version RIPng vom RIP Protokoll. Der RIPng wurde auf den IPv6 angepasst und im Jahre 1997 (RFC 2080) spezifiziert.

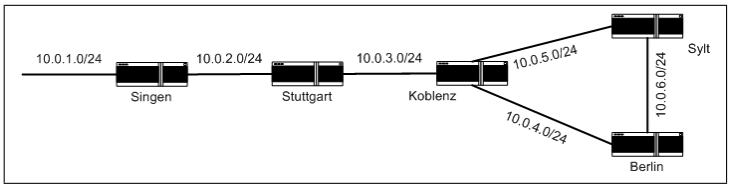


Abbildung 1: Beispielszenario YLoop

RIP arbeitet wie folgt: Beim Start kennen alle RIP Router nur ihre anliegenden Netze. Alle Router senden in bestimmten Zeitintervallen ihre Routingtabellen an ihre Nachbarrouter. Dieses Zeitintervall ist beim RIP Protokoll mit 30 Sekunden spezifiziert und wird als update oder Aktualisierungsfrist bezeichnet. Mit den ankommenden Routingtabellen ergänzen alle Router ihre Routingtabelle mit den fehlenden Informationen. Jeder Router hat mindestens eine Routingtabelle die mit fehlende Informationen und Änderungen im Netz zu aktualisieren ist. In einer Routingtabelle wird gemerkt welches Netz mit welcher Metrik über welches Interface zu erreichen ist. Wenn man unten in der Abbildung in die Routingtabelle von Koblenz hineinschaut so sieht man Beispielsweise das das Netz 10.0.1.0/24 über das Interface 10.0.3.2 mit der Metrik 3 zu erreichen ist (siehe Abb. 1). Wenn es beim Erreichen einer neuen Routingtabelle eine alternative Route von einem Start Router A zu einem Ziel Router B gibt, so entscheidet der RIP Router sich für die kürzere Route (Distanzvektoralgorithmus). Die Zeit indem alle Router ihre Routingtabellen untereinander austauschen bis sie eine einheitliche Sicht auf die gesamte Netztopologie haben wird als Konvergenzzeit bezeichnet. RIP hat den Vorteil das Änderungen im Netz update bedingt schnell bemerkt und dementsprechend in die eigene Routingtabelle eingetragen werden. Solche Änderungen können durch ein Routerdefekt, Verbindungsausfall oder durch das Hinzukommen neuer Router im Netz verursacht werden. So ist der Router bezüglich dem Netz immer auf den aktuellsten Stand. Bekommt ein Router innerhalb von 180 Sekunden kein update von seinem Nachbarrouter, so wird dessen Metrik auf 16[5] gesetzt. Im Rückschluss bedeutet das wenn 180 Sekunden kein update eingetroffen ist, so ist entweder die Netzverbindung unterbrochen oder der Router ist defekt[6] .

	Network	Next Hop	Metric	From	Tag	Time
R(n)	10.0.1.0/24	10.0.3.2	3	10.0.3.2	0	00:40
R(n)	10.0.2.0/24	10.0.3.2	2	10.0.3.2	0	00:40
C(i)	10.0.3.0/24	0.0.0.0	1	self	0	
C(i)	10.0.4.0/24	0.0.0.0	1	self	0	
C(i)	10.0.5.0/24	0.0.0.0	1	self	0	
R(n)	10.0.6.0/24	10.0.4.2	2	10.0.4.2	0	00:40
C(i)	192.168.0.100/30	0.0.0.0	1	self	0	
R(n)	192.168.0.104/30	10.0.5.2	2	10.0.5.2	0	00:39
R(n)	192.168.0.108/30	10.0.4.2	2	10.0.4.2	0	00:40
R(n)	192.168.0.112/30	10.0.3.2	2	10.0.3.2	0	00:40
R(n)	192.168.0.116/30	10.0.3.2	3	10.0.3.2	0	00:40

Abbildung 2: Routingtabelle von Router Koblenz

Alle Router die zwischen dem Start- und Zielrouter sind zählen den Hop und somit die Metrik um eins hoch. Wie schon erwähnt ist beim RIP für die Unerreichbarkeit eines Routers mit der Metrik 16 definiert. Somit wird die Ausdehnung von Netzen mit RIP Routern auf maximal 15 begrenzt. Wenn man eine genauere Betrachtung der Metrik macht dann stellt man fest das die Entfernungsrepräsentation mit der Anzahl an Hops nicht so ganz der Wirklichkeit entsprechen. Denn die Verbindungen zwischen den Routern können unterschiedlich lang sein. Zudem kommt noch die Verweildauer der Paketen in den einzelnen

[5]Die Metrik 16 steht beim RIP für die Unerreichbarkeit eines Netzes. Aus diesem Grund ist auch die Maximale Anzahl der Router im Netz auf 16 begrenzt.

[6]Die Fehlerquelle nicht genau feststellbar, weil die Advertisements über UDP (user datagram protocol) versendet werden.

Router hinzu wo die durchaus unterschiedlich sein können. Die Verweildauer im Router ist wiederum abhängig vom Router selbst. Die kommerziellen Router „Gießen“ den Algorithmus in die Hardware, welches dann zu einer besseren Performanz und somit zu einer wenigeren Verweildauer der Pakete in einem Router führt. Ferner ist die Übertragungszeit in Abhängigkeit von den Verbindungsmedien (Luft, Kupferkabel, Glasfaserkabel) unterschiedlich lang.

Obwohl das RIP ein altes Protokoll ist und obwohl es bessere Routingprotokolle gibt, ist es noch weit vom Aussterben entfernt. Ausgereiftheit, Stabilität, Flexibilität und leichte Konfiguration sprechen für Zukunftssicherheit. In Netzen wo kein RIP Protokoll zum Einsatz kommt hat dann der Netzwerkadministrator die Aufgabe die Router im Netz ständig anzupassen. Hier und da schleichen sich beim Administrator immer wieder Fehler ein und legen das Netz lahm. Die Fehlersuche im Netz kostet dann viel Zeit und Geld. So ist mittels RIP ein zuverlässig stabiles Netz gegeben. Es gibt aber auch ein paar Nachteile vom RIP die an dieser Stelle erwähnt werden sollen. Lange Konvergenzzeiten (update bedingt), Einsatzgebiet von bis mittleren Netzen (max. 15 Router), zusätzliche Netzbelastung (Routingtabellen) und Routingschleifen gehören zu den Nachteilen. Routingschleifen die in RIP Netzen auftreten führen dann zum CTI (Count-to-Infinity). Der CTI wird allerdings im folgenden Kapitel behandelt und wird somit darauf verwiesen.

2.2 Count to Infinity

Routingschleifen bedingt kann es in einem RIP Netz zu einem Count - to - Infinity führen. Routingschleifen treten dann auf, wenn eine aktuelle Routing - Information von einer alten „falschen“ Routing - Information des Nachbarrouters überschrieben wird. Die Aufnahme der falschen Routing - Information ist anhand den periodischen Routing updates durchaus möglich. Diese aufgenommene falsche Routing - Information wird dann ebenfalls zum Nachbarrouter weitergereicht. So wird bei jedem Router die Metrik um eins erhöht bis letztendlich die Metrik 16 (infinity) erreicht ist. Dieses ungewollte erhöhen der Metrik bis 16 nennt man CTI (Count to Infinity). Im dem folgenden Netz, bestehend aus den Router A, B, C und D, wird der CTI erklärt .

Wie in der Abbildung zu sehen ist der Router A mit Router B, der Router B mit Router C und der Router C mit dem Router D direkt miteinander verbunden. Der Router D ist zudem noch mit dem Netz d in Kontakt. In der Tabelle unterhalb der Abbildung ist die Metrik (Distanz) der Router zum Netz d auszulesen. Wenn man Beispielsweise den Router B in der Tabelle betrachtet so ist dessen Metrik zum Netz d = 3. In der ersten Spalte werden die Zeitpunkte als diskrete Werte aufgelistet.

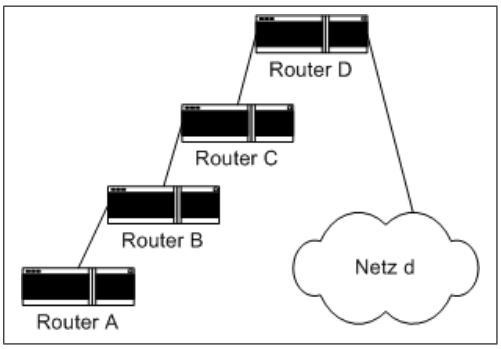


Abbildung 3: Netz vor dem CTI Eintreffen

Zeitpunkt	A	B	C	D
0	4	3	2	1

In der Abb. 4 wird anhand dem Ausfall vom Netz d der CTI demonstriert werden. Die rot gestrichelte Verbindung zwischen dem Router D und Netz D zeigt den Ausfall der Verbindung. Der Router D Bemerkt den Ausfall vom Netz d, da er seit 180 Sekunden keinen update über Netz d bekommen hat. Demzufolge setzt der Router in seine Routingtabelle den Wert auf 16 (infinity) das aus dem Zeitpunkt 1 der Tabelle zu ersehen ist. Es ist desweiteren möglich das der Router D ein periodisches update von Router C bekommt bevor er sein eigenes periodisches update an die Nachbarrouter, in dem Fall Router C, versendet. So denkt der Router D anhand dem periodischen update das ein Weg zum Netz d über den Router C vorhanden ist. Dabei handelt sich hierbei noch um eine alte bzw. nicht mehr aktuelle Metrik, welches Router D nicht weiß. Unwissend mit der Annahme auf eine Verbindungsexistenz zum Netz d übernimmt der Router D die Metrik von Router C, inkrementiert einen Hop drauf und speichert diesen falschen Wert 3 in die eigene Routingtabelle ein. Ersichtlich ist das aus dem Zeitpunkt 2 der beistehenden Tabelle.

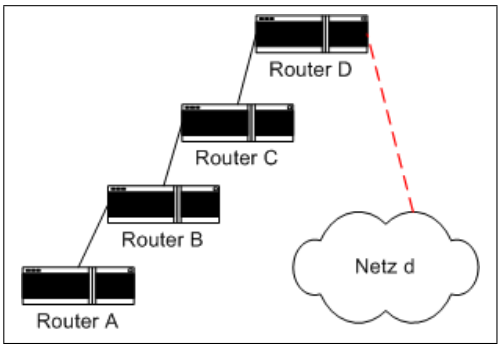


Abbildung 4: Netz während CTI

Zeitpunkt	A	B	C	D	Bemerkung
1	4	3	2	16	Ausfall wird bemerkt
2	4	3	2	3	Falsche Metrikübernahme von Router d
3	4	3	4	3	
4	6	5	4	5	
..	..	..	..	..	

Zum Zeitpunkt 3 bekommt Router C ein periodisches update von Router D. So übernimmt der Router C nun die falsche Metrik 3 von Router D, inkrementiert die Metrik auf 4 und speichert sie in die eigene Routingtabelle. Zum Zeitpunkt 4 verbreitet der Router C wiederum ein periodisches update mit der falschen Metrik 4 an seine Nachbarrouter aus. So schaukelt das immer hin und her bis der Wert 16 erreicht ist. Diesen ungewollten Vorgang der Routingschleifen nennt man dann Count-to-Infinity.

Um den CTI zu vermeiden gibt es verschieden Mechanismen. Dazu gehören Triggered Updates, Split Horizon, Split Horizon with Poisoned Reverse und RIP MTI. Beim Triggered update liegt der Gedanke das die Änderung im Netz möglichst schnell verbreitet werden sollen. Das heißt sobald beispielsweise ein Router ausfällt wird das update sofort gefeuert ohne die update time 30 Sekunden abzuwarten. Mit diesem Verfahren können nach wie vor Routingschleifen und somit auch der CTI auftreten. Jedoch mit einer viel kleineren Wahrscheinlichkeit wie zuvor. Abgesehen von der kleineren Wahrscheinlichkeit für Routingschleifen bringt dieses Verfahren noch den Vorteil mit sich das die Konvergenzzeit auf Kosten der Netzauslastung reduziert wird.

Ein weiterer Mechanismus zur Vermeidung der CTI ist der Split Horizon. Die Idee hierbei ist: *Belehre nicht deinen Lehrer*. Das heißt das die Pakete nicht rückwärts propagiert werden dürfen sondern nur vorwärts. In dem oberen Beispiel bedeutet das, das der Router C den Router D keine Pakete sendet (nicht belehren) und ihn somit nicht die Metrik 2 über das erlernte Interface zurück propagiert. Der Split Horizon kann Routingschleifen in Netzen ohne redundante Routingpfaden unterdrücken. Aber in Netzen mit redundanten Routingpfaden können nach wie vor Routingschleifen (somit auch CTI) auftreten.

Erweitert auf dem Split Horizon Verfahren gibt es das Split-Horizon with poisoned reverse. Der Unterschied zum Split Horzon ist das der Router über das gelernte Interface die Metrik 16 an den „Lehrer“ zurück propagiert. Das heißt das der Router C die Metrik 16 (bezüglich zum Netz d) an den Lehrer Router D zurück propagiert weil der Router C von Router D lernt. Somit sagt Router C dem Router D das er keine Verbindung zum Netz d hat. Split Horizon with poisoned reverse bringt eine Verbesserung in Netzen mit mehreren Routingpfaden. Jedoch bringt er gegenüber dem Splitz Horitzon in Netzen mit einem Routingpfad keine Verbesserung. Durch diese Zurückpropagierung an den Lehrer entsteht eine höhere Netzauslastung.

Der RIP-MTI (RIP with Minimal Topology Information) ist ein Verfahren das an der Universität Koblenz entwickelt wurde um Rouitngschleifen und somit den CTI vollständig zu verhindern. Alle Entwicklungen (Forschung und Imple-

mentierung) bezüglich diesem Thema basieren auf der Idee von [Sch99]. Bei der Idee geht es darum die Routing - Informationen, die am Router anliegen auszuwerten, Routingschleifen zu erkennen und diese zu unterbinden. Der große Vorteil denn man dadurch gewinnt ist das die ursprüngliche Struktur von RIP nicht verloren geht(Abwärtskompatibilität). Somit kann das RIP-MTI Protokoll in jedem RIP Netzwerk eingesetzt werden. Im Laufe der RIP-MTI Forschung wurden unterschiedliche RIP-MTI Modi wie Normal Mode, Strict Mode und Careful Mode entwickelt. Der Normal Mode erkennt CTI nur in einfachen Netzwerktopologien verhindert den CTI aber nicht in verschachtelten Netzwerkpfaden. Des weiteren gibt es den Strict Mode. Wie man es aus dem Namen ableiten kann ist dieses Verfahren zu streng. CTI werden beim Strict Mode immer erkannt und verhindert. Auch in verschachtelten Netzwerkpfaden, welches beim Normal Mode nicht der Fall ist. Allerdings lehnt der Strict Mode unter Umständen auch gültige alternative Routen ab welches auch nicht erwünscht ist. Dann gibt es noch den Careful Mode. Der Careful Mode ist eine Erweiterung vom Strict Mode (siehe [Boh08]).

Zusammenfassend ist zu sagen das das Verfahren Triggerd Update die Wahrscheinlichkeit für Routingschleifen auf Kosten der Netzauslastung reduziert aber nicht vollständig eliminiert. Der Split Horizon *Belehre nicht deinen Lehrer* verhindert in Netzwerken ohne redundante Pfade die Routingschleifen. Bei Netzwerken mit redundanten Pfaden kann hier nach wie vor Routingschleifen und somit auch der CTI auftreten [Rfc58]. Der Split-Horizon with poisoned reverse bringt in Netzen ohne Redundante keine wirkliche Verbesserung gegenüber dem Split Horizon Verfahren. Der RIP-MTI wurde an der Universität Koblenz auf der Basis von RIP Protokoll weiterentwickelt. Dieses Verfahren wurde allerdings nicht in industriellen Systemen eingesetzt. Demzufolge kann man auch keine größeren Aussagen bezüglich diesem Verfahren machen.

2.3 VNUML

VNUML (Virtual Network User Mode Linux) ist eine Open Source (GNU Public Licence) Simulationssoftware für Rechnernetze unter Linux. Das Tool basiert auf dem Virtualisierungssoftware User Mode Linux [UML08]. User Mode Linux ermöglicht das Erzeugen von mehreren virtuelle Rechner auf einem physischen Hostrechner. VNUML wurde von der Technischen Universität Madrid entwickelt und im Jahre 2002 veröffentlicht. Die Simulationssoftware ermöglicht das schnelle aufstellen und testen von komplexen Netzwerken. Zu den Anwendungsbereichen von VNUML gehören unter Anderem Bereiche wie Forschung, Entwicklung, persönliches „Ausprobieren“ oder auch die Lehre von Grundlagen in Rechnernetze. Wenn man beispielsweise unsere Hochschule betrachtet wird VNUML begleitend in der Lehre von Rechnernetzen und in der Forschung eingesetzt.

Wenn man mit VNUML arbeiten will gibt es unterschiedliche Phasen die zu überwinden sind. Zuerst muss man sich überlegen was man Simulieren will. Es werden sozusagen grundlegende Entscheidungen bezüglich dem zu simulierenden Netzwerk getroffen. Solche Entscheidungen können zum Beispiel über die Anzahl der virtuellen Rechner oder über den Netzwerkaufbau gemacht werden.

Diese Phase wird als Designphase genannt. Nach der Designphase kommt die Implementierungsphase. In dieser Phase wird dann das zu simulierende Wunschszenario in einer XML Datei erstellt. Dazu gibt es die VNUML Sprache. Bei der VNUML Sprache gibt es kleinere Abweichungen je nachdem mit welcher Version VNUML man arbeitet. Die VNUML Sprache dient letztendlich dazu das Szenario in eine XML Datei zu schreiben. Zum Schluss kommt dann die Ausführungsphase. In der Ausführungsphase geht dann der VNUML Parser die XML - Datei durch und stellt das Netzwerk in einem Hostrechner her. Für jede virtuelle Maschine unter VNUML wird dann ein eigener Prozess erzeugt.

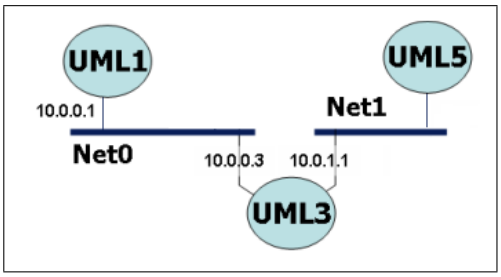


Abbildung 5: Beispielszenario VNUML

Das Netzwerk von Abb. 5 wurde der VNUML Projektseite [Vnu08] entnommen und entsprechend gekürzt. Solche Überlegungen, wie oben in der Abbildung oben dargestellt, macht man sich in der Designphase. Wie soll das zu simulierende Netzwerk aussehen? Wie viele virtuelle Rechner braucht man für das zu analysierende Problem? Welche IP Adressen bekommen die virtuelle Rechner zugeteilt? usw. Die dazugehörige XML Datei sieht dann wie in der Abb. 6 aus. Die XML Datei wird dann in der Implementierungsphase, in diesem Fall nach der VNUML Version 1.8, hergestellt. Der VNUML Parser parst in der Ausführungsphase die XML Datei durch und stellt das Szenario in dem Hostrechner her.

```
<vnuml>
  <global>
    <version>1.8</version>
    <simulation_name>.....</simulation_name>
    .....
  </global>

 <net name="Net0" mode="uml_switch" />
 <net name="Net1" mode="uml_switch" />

  <vm name="uml1">
    <if id="1" net="Net0">
      <ipv4>10.0.0.1</ipv4>
    </if>
    <route type="ipv4" gw="10.0.0.3">default</route>
  </vm>

  <vm name="uml3">
  ......
  </vm>

  <vm name="uml5">
  ......
  </vm>
</vnuml>
```

Abbildung 6: XML Datei zu Beispielszenario VNUML

In dieser Studienarbeit hat man sich für die VNUML Version 1.8 entschieden. Der Hintergrund ist das man die Szenarien in VNUML 1.8 mit unterschiedlichen Nutzerrechten starten lassen kann. Mit den VNUML Versionen bis 1.5 ist das nicht möglich, welches auch nicht der Benutzerrechtphilosophie von Linux entspricht. Die Vorteile mit der Einführung von unterschiedlichen Nutzerrechte sind selbst erklärend. Wenn man beispielsweise VNUML an den Hochschulen als Lernzweck einsetzt, so will man dem Studenten nicht die volle Kontrolle (Rootrechte) über das Betriebssystem geben. Denn eine (un-) bewusste falsche Handhabung kann schlimme Konsequenzen mit sich bringen. Demzufolge gibt man dem Studenten Beschränkte Userrechte. Damit kann der Student zwar nicht alle Features von VNUML ausnutzen, aber es reicht ihm dennoch für Lernzwecke aus. VNUML bietet abgesehen von den beschränkten Userrechten noch 2 weitere Nutzerrechte an. Zum einen handelt es sich um die Userrechte alleine und zum anderen die Rootrechte. Bei den Beschränkten Userrechten und Userrechten wird der VNUML Parser ohne Rootrechte gestartet. Bei der Anmeldung als Root wird der VNUML Parser als Root gestartet und somit stehen den User volle Kontrollen über alle Features zur Verfügung.

Zusammenfassend kann man sagen das der reale Aufbau von Rechnernetzen oft mühselig und sehr zeitaufwendig ist. Mittels VNUML werden diese beiden Faktoren reduziert. Der Aufbau lässt sich mittels XML Datei leicht realisieren. Veränderungen in der Topologie können einfach in der XML Datei abgeändert werden. Anhand den Möglichkeiten wie das Starten und Stoppen der Netzwerksimulation oder die einfache Abänderung von der Netzwerksimulation ist dem Anwender eine größere Flexibilität gegeben. Zudem braucht man bei dem Einsatz von VNUML keine zusätzliche Hardware kaufen (Kosteneinsparung). Für weiteren Fragen bezüglich VNUML wie Installation, Syntax der VNUML Sprache, Anwendungsbeispiele und Befehle zur Anwendung von VNUML siehe die offizielle Projektseite [Vnu08].

2.4 Quagga

Quagga ist eine Weiterentwicklung der GNU Zebra des Entwicklers Kunihro Ishiguro. Sie ist eine Routing Software Suite in Open Source (GPL[7]) das alle wichtige Routingdaemons für den IPv4 und IPv6 (Internet) implementiert. Zu den Routingdaemons die in Quagga enthalten sind gehören zum Beispiel RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, BGP-4 und BGP-4+. Der Vorteil von Quagga ist das man die Routingdaemons auf eine kostengünstige Hardware (Server) aufzusetzen und starten lassen kann. Damit fungiert der Server als ein kostengünstiger Router Abb. 7. Die Routingdaemons die auf den Server laufen werten die ankommenden Routingtabellen aus, modifizieren die eigene Routingtabelle und leiten schlussendlich die eigene Routingtabelle weiter. Im Gegensatz zu der kostengünstigeren Variante gibt es kostenpflichtige Router von kommerziellen Herstellern. Router von kommerziellen Herstellen haben eine bessere Performanz und unterstützen einen besseren Support. Die bessere Performanz kommt zum größten Teil durch das „Gießen“ der Routingalgorithmen in Hardware zustande.

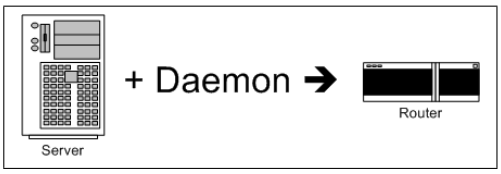


Abbildung 7: Idee von Quagga

Die Softwarearchitektur der Quagga Routing Software Suite ist modular aufgebaut (siehe Abb. 8). Sie beinhaltet, wie schon oben erwähnt, alle wichtigen Routingdaemons für das Internet. Zusätzlich hat Quagga noch das Verwaltungsdaemon Zebra. Das Zebra Daemon ist eine Abstraktionsschicht zwischen dem Unix Kernel und den Routingdaemons. Es übernimmt die Kommunikation mit dem Unix Kernel. Weitere Aufgaben von Zebra sind die Verwaltung der Routingdaemons (Aufgabenzuteilung), Gewährleistung der Kommunikation von Routingdaemons untereinander (Austausch der Routing Informationen) und die Erstellung bzw. Vervollständigung der Routingtabelle. Die Kommunikation untereinander (Daemons, Zebra, Unix Kernel) geschieht mit den Unix - Domain - Sockets. Anhand der modular aufgebauten Softwarearchitektur und der Kommunikation mit den Sockets fällt die unabhängige Weiterentwicklung der einzelnen Routingdaemons (z.B. RIP-MTI) oder der einfachen Implementierung neuer Routingprotokolle nicht schwer. Ferner ist eine höhere Stabilität gegeben. Wenn beispielsweise ein Routingdaemon bzw. ein Prozess abstürzt so ist wird die Stabilität der anderen Routingdaemons nicht negativ beeinträchtig.

[7]General Public License: `http://www.gnu.org/licenses/gpl.html`

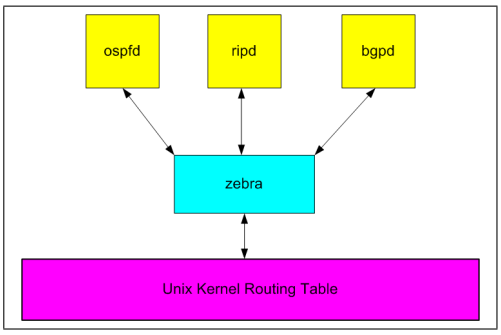


Abbildung 8: Quagga System Architektur

2.5 XTPeer

Um die RIP-MTI Forschungen zu unterstützen wurde das Steuerungs- und Analysewerkzeug XTPeer (eXternally Triggered Peer) an der Universität Koblenz entwickelt. Der XTPeer stellt zusammen mit dem VNUML - Netzwerksimulation eine Netzwerktestumgebung dar, die lokal auf einem Hostrechner ausgeführt wird. Die erste Version von XTPeer wurde von Daniel Pähler [Päh08] in der Programmiersprache JAVA implementiert. Im Laufe der Zeit Unterlag das Tool XTPeer ständigen Erweiterungen wie Funktionserweiterungen, Benutzerfreundlichkeit und Robustheit. Dazu Beigetragen haben die Diplomarbeiten von Stefan Lange [Lan07], Tim Keupen [Keu07] und Frank Bohdanowicz [Boh08].

Mit dem Tool XTPeer kann man kontrolliert einen CTI provozieren. Den CTI stellt man dann graphisch dar. Anhand dem graphischen kann man das selbst implementierte bzw. erweiterte Protokoll analysieren. So sieht man ob das implementierte Protokoll sich so verhält wie es vom Entwickler erdacht wurde. In dem RIP-MTI Verfahren müssten die Routingschleifen erkannt und unterbunden werden. Falls dann das implementierte Protokoll nicht den Anforderungen genügt so kann man Verbesserungen am Protokoll vornehmen und daran weiterentwickeln. Um das beschriebene Vorgehen mit dem XTPeer zu erreichen wird im Folgenden auf die wichtigsten Merkmale der XTPeer GUI eingegangen, die in Verbindung mit dieser Studienarbeit in Berührung kommen.

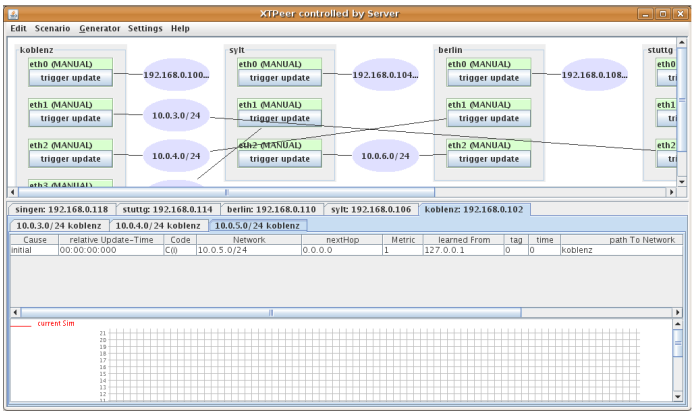


Abbildung 9: XTPeer Oberfläche

Erst einmal eine Beschreibung der XTPeer Oberfläche. Nach der Diplomarbeit von [Boh08] besteht die GUI der XTPeer aus 2 Abschnitten. Einem Oberen und einen unteren Abschnitt. Im oberen Abschnitt ist eine grafische Darstellung der Topologie zu sehen. Das rechteckige Symbol repräsentiert einen RIP - Router wohingegen das ovale Symbol ein Subnetz des Netzwerks darstellt. Aus dem Beispielszenario Y-Loop im Kapitel 4 sind die Router Koblenz, Sylt, Berlin, Stuttgart und Singen als Rechtecke dargestellt (siehe Abb. 10). Wenn man in der XTPeer GUI mit der Maus auf ein Router Rechteck fährt und die rechte Maustaste betätigt so hat man die Möglichkeit den RIP Router zu konfigurieren. Mann kann beispielsweise verschiedene RIP MTI Algorithmen (normal, strict, careful) laufen lassen oder man kann den MTI ganz ausschalten. In den RIP Router Rechtecken sind weitere rechteckige Symbole drin wodurch die Interfaces der Router bedient werden. Dadurch lassen sich die Interfaces direkt ansteuern. Sie lassen sich zwischen den Modi AUTO, Triggered update und Manual Trigger umschalten. Im AUTO Modus sendet ein RIP - Router nach [Rfc58] alle 30 Sekunden sein Routing update an den Nachbarrouter. In dem Beispielszenario im Kapitel 4 ist der Routing update allerdings auf 10 Sekunden abgesetzt worden um die Konvergenzzeit zu reduzieren. Ein weiterer Modus den man an Routerinterfaces mittels der XTPeer GUI aktivieren kann ist der Triggered Update Modus. Der Triggered Updates Modus dient dazu das neu erworbene Wissen möglichst sofort zu verbreiten und somit die Wahrscheinlichkeit für einen CTI zu reduzieren. Triggered Update wurde in dem Kapitel 2.2 bereits erklärt und demzufolge sei hier darauf zurück verwiesen. Ferner gibt noch den Modus Manual Trigger, der sich ebenfalls mittels XTPeer am Interface aktivieren lässt. Im Manual Trigger Modus hat der Benutzer die Möglichkeit mit einem „Klick“ selbst zu bestimmen wann er den Routing update von einem Router zum Nachbarrouter versenden will. Dieser Modus ist Sinnvoll wenn man den Ausfall einer Verbindung zwischen 2 Router erzwingen will.

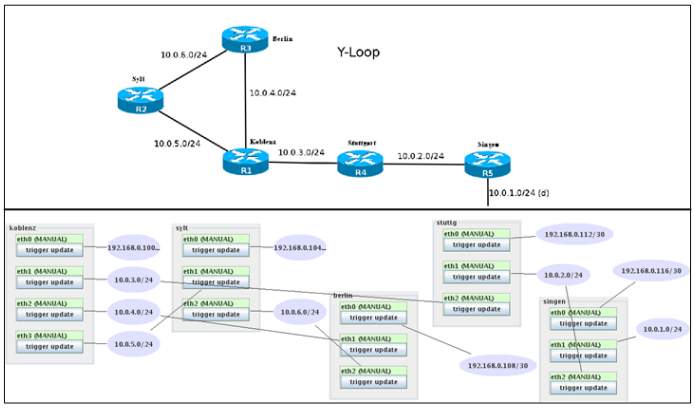


Abbildung 10: Graphische Repräsentation YLoop

Im unteren Abschnitt der XTPeer GUI sind die RIP Router als Registerreiter dargestellt. Je nachdem welchen RIP Router man genauer Anschauen will so wählt man den zugehörigen Registerreiter aus. Innerhalb eines Registerreiters kann man weitere Registerreiter auswählen, die die Subnetze der RIP Router protokollieren und diese dann tabellarisch auflisten. In der tabellarischen Darstellung ist der Verlauf der Veränderung der Router zu finden. Wird im Verlauf der Veränderung, wie in dem folgenden Fall zu sehen ist, ein CTI erkannt so werden die entsprechenden tabellarischen Zeilen mit der gelben Farbe gekennzeichnet (siehe Abb. 11).

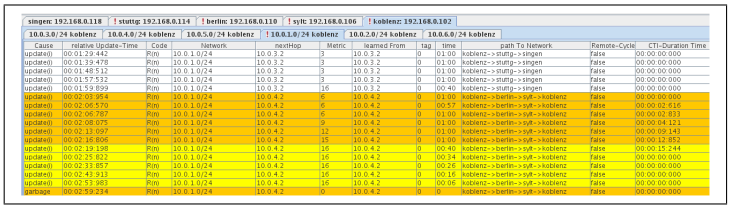


Abbildung 11: unterer Abschnitt der XTPeer GUI

Ein weiteres schönes Feature vom XTPeer, welches in dieser Arbeit verwendet wird, ist der Metrikgraph. Der Metrikgraph ist eine graphische Distanzrepräsentation zwischen einem RIP Router und dem entsprechendem Subnetz. Dazu fährt man im oberen Abschnitt mit der Maus auf das Subnetz (ovales Symbol) und „klickt“ dann die rechte Maustaste. Zum Erscheinen kommt dann der in Abb. 12 dargestellte Metrikgraph. Der Metrikgraph wird im Kapitel Simulation und Anwendungsbeispiel benutzt um das CTI Problem elegant graphisch darzustellen. Für detaillierte Beschreibungen der Graphische Benutzeroberfläche XTPeer und deren zugehörigen Funktionserweiterungen findet man in [Päh08], [Keu07] und [Boh08].

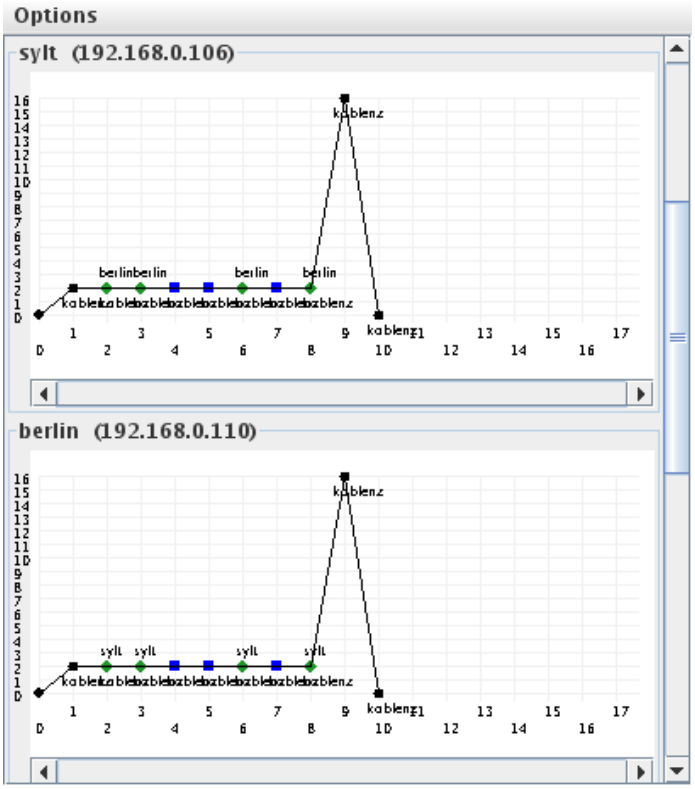


Abbildung 12: Metrikgraph

2.6 Zusammenfassung

Das Routing Information Protocol ist ein dynamisches Routingprotokoll das auf der Basis vom Distanzvektoralgorithmus arbeitet. Unter ungünstigen Verhältnissen kann beim Routing Information Protocol Routingschleifen und somit der Count-to-Infinity auftreten. Der Count-to-Infinity ist das Verbreiten und Inkrementieren von falschen Metriken in den Routing Updates. Die Metrik wird dabei bis zum diskreten Wert 16 inkrementiert, welches beim RIP Protokoll mit der infinity „unendlichkeit“ spezifiziert ist. Zur Vermeidung von Count-to-Infinity gibt es verschiedene Verfahren wie Triggered Updates, Split Horizon und Split Horizon with Poisoned Reverse die allerdings den Count-to-Infinity nicht ganz verhindern können. Ein weiters Verfahren das sich mit der Verhinderung von Routingschleifen beschäftigt ist der RIP-MTI, welches an der Universität Koblenz entwickelt wurde. Um diese Forschung zu unterstützen macht man sich den Gebrauch von VNUML, Quagga und XTPeer.

VNUML stellt ein kostenloses Tool zur Verfügung indem ein virtuelles Netzwerk auf einem Hostrechner simuliert werden kann. Jede virtuelle Maschine in VNUML enthält dabei ein Filesystem das man der offiziellen VNUML Projektseite [Vnu08] runter laden kann. In das Filesystem wird dann die freie Routing Software Suite Quagga installiert, indem verschiedene Internet Protokolle (BGP, RIP, OSPF) implementiert sind. Nachdem das Filesystem angepasst wurde kann

dann das VNUML Szenario hoch gefahren werden. Anschließend müssen die RIP Protokolle in den virtuellen Maschinen der VNUML gestartet werden. Nachdem die RIP Protokolle gestartet wurden kann dann das Steuerungs- und Analysetool XTPeer gestartet werden. Anhand von XTPeer kann dann gezielte Eingriffe auf die Interfaces der virtuellen Maschinen gemacht werden. Dadurch kann der CTI kontrolliert provoziert werden. Anhand den Betrachtungen in XTPeer, sowohl graphisch als auch tabellarisch, können dann Verbesserungen im implementierten Protokoll vorgenommen werden.

3 Einrichtung des Root Filesystems

Jede virtuelle Maschine braucht ein Root Filesystem und einen Kernel[8] beim Booten. Die Filesysteme der virtuellen Maschinen müssen den persönlichen Bedürfnissen angepasst werden, indem man beispielsweise die Quagga Routing Software Suite installiert. Man hat zum Einen die Möglichkeit im laufenden Betrieb sich mit einer der virtuellen Maschinen zu verbinden(SSH) und deren Filesystem zu verändern. Allerdings gehen in einem COW[9] Filesystem beim Beenden der VNUML Simulation sämtliche Änderungen verloren. Deswegen sollte man diese Installationsweise vermeiden und sich für die folgende mögliche Variante entscheiden. Die andere Variante besteht darin das Root Filesystem auf dem Hostrechner zu mounten und es dann anzupassen. Das Root Filesystem kann wiederum selbst erstellt oder ein fertiges runter geladen werden. Sowohl die Beschreibung zur selbstständigen Filesystemerstellung als auch das fertige Filesystem mit dem dazugehörigen Kernel gibt es auf der offiziellen VNUML Projektseite [Vnu08].

Auf der offiziellen VNUML Homepageseite gibt es 2 „fertige“ Root Filesysteme (root_fs_tutorial-0.5.2 und n3vlr-0.11-vnuml-v0.1.img) die man runter laden kann. Dabei wird ausdrücklich empfohlen die beiden Root Filesysteme nur in Verbindung mit dem UML Kernel “linux-2.6.18.1-bb2-xt-4m“ zu verwenden. Bei den 2 Root Filesystemen handelt es sich zum Einen um das Root Filesystem (root_fs_tutorial-0.5.2) und zum Anderen um das Mini Image (n3vlr-0.11-vnuml-v0.1.img). Das root_fs_tutorial-0.5.2 ist mit viel Software ausgestattet und genügt somit den meisten Anforderung. Der Nachteil bei diesem Root Filesystem ist das es lange zum Booten braucht, da eine viel größere Mengen an Daten in die virtuellen Maschinen kopiert werden müssen. Das Booten vom n3vlr-0.11-vnuml-v0.1.img (Mini Image) geht im Gegensatz dazu um einiges schneller. Das Mini Image beschränkt sich auf die wichtigsten Komponenten. Allerdings ist hier die Installation der fehlenden Komponenten etwas aufwändiger. Es ist letztendlich dem Benutzer überlassen für welches Filesystem er sich entscheidet. Wenn ein schnelles Booten erwünscht ist, so ist die Entscheidung auf das Mini Image zu treffen.

Hinweis: Bei der Installation vom root_fs_tutorial-0.5.2 Filesystem empfiehlt es sich das root-fs-installer Skript zu verwenden. Das root-fs-installer Skript kann ebenfalls auf der VNUML Projektseite runter geladen werden. Dabei das Skript in das Filesystem Verzeichnis (“/usr/share/vnuml/filesystem“) kopieren und dort ausführen. Zur Ausführung dann im Terminal in das Filesystem Verzeichnis wechseln und den Befehl “perl root-fs-installer“ als Root[10] ausführen. Es ladet dann das Filesystem von einem im Skript enthaltenen Mirror runter und ergänzt die fehlenden Pakete des Betriebssystems. Ist eine ältere Version vom Root Filesystem erwünscht (z.B: 0.5.0) so ist “perl root-fs-installer 0.5.0“ bei der Ausführung im Terminal einzugeben.

[8] Kernel ist der Betriebssystemkern, deren Aufgabe es ist die Prozesse des Systems zu Verwalten.

[9] COW = Copy On Write. Beim Booten des Filesystems hat jeder virtueller Rechner zunächst den selben kopierten Datenbestand des Filesystems. Veränderungen im laufenden Betrieb (SSH) werden von VNUML nicht zurück geschrieben und gehen somit verloren.

[10] Eine Anmeldung auf dem Linux Betriebssystem als super user (su).

Wie schon oben erwähnt sollte das Filesystem gemountet werden um das Root Filesystem den persönlichen Bedürfnissen anzupassen. Man kann sich dann zum Einen mittels „chroot“ in das Rootverzeichnis des Filesystems der virtuellen Maschine einhängen um dort die gewünschte Software zu installieren. In diesem Fall müssen sämtliche Pakete die zur Installation der Software notwendig sind auf dem Root Filesystem vorhanden sein. Zum Anderen besteht die Möglichkeit die erwünschte Software von dem Hostrechner aus zu installieren. Dabei wird die Software mit den entsprechenden prefixen kompiliert und anschließend dann die Binaries in das Filesystem installiert. Das heißt man hat hier den Vorteil der Nutzung von Distributionsfunktionen. Zur Gegenüberstellung der beiden Installationsmöglichkeiten siehe die folgenden Abbildungen, die der offiziellen VNUML Projektseite [Vnu08] entnommen wurden.

```
mount -o loop root_fs_tutorial /mnt/loop
mount -t proc none /mnt/loop/proc
chroot /mnt/loop
apt-get update
apt-get install something
...
exit
umount /mnt/loop/proc
umount /mnt/loop
```

(a) Installation mittels chroot

```
mount -o loop root_fs /mnt
cd /usr/local/src/zebra
./configure --prefix=/mnt
make
make install
make clean
make distclean
umount /mnt
```

(b) Installation von Host aus

Abbildung 13: Gegenüberstellung der beiden Installationsmöglichkeiten

3.1 Auto Tools Versionen

Eine komplexe GNU Software wie Quagga sollte bei Linux/Unix ähnlichen Betriebssystemen möglichst portabel untereinander sein. Im Laufe der Zeit sind diesbezüglich Autotools (Programmpakete) entstanden worden. Dabei handeln es sich unter Anderem um Pakete wie Libtool, Autoconf und Automake. Libtool ist eine Unterstützung für Bibliotheken auf allen Plattformen. Die Komplexität von gemeinsam benutzen Bibliotheken wird hinter einer portablen Schnittstelle gekapselt. Weitere wichtige Programmpakte sind Autoconf und Automake. Autoconf ist mittels configure.ac für die Konfigurationsdateierstellung (configure-Skript) zuständig. Automake bringt mittels makefile.am die Softwarepakete in eine übersetzbar und installierbar vereinfachte Form. Das erzeugte Configure Skript erzeugt dann endgültige Makefile Dateien. Von der GNU kann dann mittels make und make install kompiliert und installiert werden. Somit ist der Entwickler von Bastlerarbeiten (portablem Makefile schreiben) befreit.

In Bezug auf dieses Projekt muss der veränderte Quellcode in die Quagga Routing Software Suite gepatcht werden (siehe A.3). Allerdings läuft der Kompilierungsvorgang nicht immer reibungslos ab. Im A.3 wird beschrieben dass die Pakete Automake und Libtool installiert werden müssen. An dieser Stelle ist von Großer Bedeutung das die Pakete Automake und Aclocal in der Version 1.9 vorhanden sein müssen. Die Version 1.9 deshalb, weil währen der Kompilierung

mit der Version 1.10 kommt eine Warnmeldung. Sie lautet „missing run automake1.9“. Wenn man trotz dieser Warnmeldung die Software installiert so lassen sich die Daemons nicht ordnungsgemäß starten. Vermutlich wurden die originalen Makefiles von Quagga 0.99.4 mit der automake Version 1.9 erstellt. Nur mit den Versionen 1.9 von Automake und Aclocal lassen sich die endgültigen Makefiles korrekt erzeugen und einbinden. Das bestätigen auch einige Äußerungen im Internet das unterschiedliche Versionen nicht wirklich kompatibel zueinander waren und Probleme verursacht haben.

3.2 Mini Image

Das minimale Root Filesystem „n3vlr-0.11-vnuml-v0.1“ (18 MB) der virtuellen Maschinen unter VNUML kann man, wie schon oben erwähnt auf der VNUML Projektseite runter laden. Nach [Vnu08] basiert das „n3vlr-0.11-vnuml-v0.1“ auf „N3 Virtual Linux Router“ und wurde an das VNUML angepasst. Beim „N3 Virtual Linux Router“, auch N3VLR genannt, handelt es sich um ein kleines mini Linux Betriebssystem, welches viele Implementierungen von dynamischen Routing Protokollen basierend auf Open Source enthält. Desweiteren sind im N3VLR eine Vielzahl an gängigen Netzwerkanalysetools vorhanden.

Die Installation von Programme auf das „n3vlr-0.11-vnuml-v0.1“ ist im Vergleich zu gängigen Root Filesystem ein bisschen aufwändiger. Hierfür müsste die an das Projekt XTPeer angepasste Quagga Routing Software Suite unter uClibc kompiliert werden. Die Bibliothek uClibc ist eine freie C-Bibliothek die unter LGPL lizenziert ist. Konzipiert und erstellt wurde sie ist für Linux - Embedded - Systeme. UClibc erstellt im Vergleich zu glibc viel kleinere Programme. Glibc wurde entwickelt, um alle relevanten C-Standards auf einer Vielzahl von Plattformen voll zu unterstützen. Dazu gehören unter Anderem auch die Linux - Betriebssysteme. Es werden zwar nahezu alle Programme von der glibc auf die uClibc portiert, aber es gibt dennoch hin und wieder bei der Umsetzung Probleme. Deshalb sollte an dieser Stelle Zeit und Geduld mit gebracht werden und demzufolge auch im Vorraus die richtige Entscheidung getroffen werden.

4 Simulation und Anwendungsbeispiel

Als Beispielszenario in diesem Kapitel dient die in der Abbildung 15 Y-Loop aufgeführte Netztopologie. Dieses Szenario besteht aus den Router Singen, Stuttgart, Koblenz, Berlin und Sylt welches nach [Boh08] einer Y-Kombination entspricht. In diesem Kapitel wird das Szenario unter VNUML hochgefahren und mit anschließendem Starten der RIP Daemons. Aufbauend auf das VNUML wird dann ebenfalls der XTPeer auf dem selben Host gestartet. Nach einer bestimmten Konvergenzzeit der RIP - Router wird dann der CTI anhand dem XTPeer gezielt provoziert. Anschließend kann das Verhalten der einzelnen RIP Router und somit das ganze Verhalten vom Netz am XTPeer analysiert werden. Die zugehörige XML Datei, die das Netz für die VNUML Version 1.8 beschreibt, ist dem Anhang Beispielszenario Y-Loop zu entnehmen. Das Ziel in diesem Kapitel ist mit den Tools (VNUML, XTPeer), anhand eines kleinen Beispiels, vertraut zu werden.

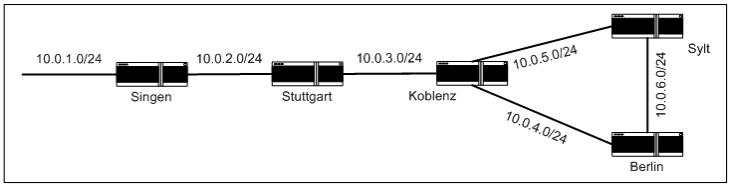


Abbildung 14: Y-Loop

4.1 Vorbereitung

Um dieses Beispielszenario auszuführen sind folgende Vorbereitungen zu treffen.

1. VNUML in der Version 1.8. Für die Installation und Anwendung von VNUML siehe [Vnu08].

2. xtpeer.jar (in der beiliegenden CD).

3. mini_vnuml-1.8.img. Das Mini Image ist in das Verzeichnis „/usr/share/vnuml/filesystems/“ zu kopieren. Ferner muss es mittels „ln -sf mini.img root_fs_18“ verlinkt werden. Falls ein anderes Verzeichnis als „/usr/share/vnuml/filesystems/“ erwünscht ist, dann ist das Filesystem Tag in der XML Datei (yloop.xml) anzupassen.

4. Beispielszenario yloop.xml welches das Netzwerk, bestehend aus den Router Singen, Stuttgart,Koblenz, Berlin und Sylt beschreibt.

5. UML Kernel linux-2.6.18.1-bb2-xt-4m (28/12/2007). Der UML Kernel ist Archiviert. Zuerst muss die Datei „linux-um_2.6.18.1-bb2-xt-4m.orig.tar.gz“ in das Verzeichnis „/usr/share/vnuml/kernels“ kopiert werden. Falls ein anderes Verzeichnis erwünscht ist so ist hier das kernel Tag analog zum Filesystemtag in der yloop.xml zu verändern. Danach ist die Datei mittels „tar -xvzf linux-um_2.6.18.1-bb2-xt-4m.orig.tar.gz“ zu entpacken und mittels „ln -sf linux-um-2.6.18.1-bb2-xt-4m/linux-2.6.18.1-bb2-xt-4m linux“ zu verlinken.

6. Java Runtime Environment (JRE). Mit dem Befehl „ java -version“ kann man die Version ersehen. Zur Installation siehe [Sun08]. In diesem Beispielszenario kam folgende Version zum Einsatz.

   - java version „1.6.0_03“
   - Java ^{TM} SE Runtime Environment (build 1.6.0_03-b05)
   - Java HotSpot ^{TM} Client VM (build 1.6.0_03-b05, mixed mode, sharing)

4.2 Beispielszenario Y-Loop hochfahren und XTPeer starten

Bevor man mit XTPeer arbeiten kann muss zuvor das Szenario unter VNUL hochgefahren und anschließend die RIP Daemons gestartet werden. Das Beispielszenario wird mit dem Befehl „vnumlparser.pl -t yloop.xml -vb -u -root“ hochgefahren. Der Befehl sollte als root ausgeführt werden. Die Option -vb veranlasst lediglich dafür das alle Schritte von VNUML Parser auf der Konsole ausgegeben werden um eventuell auftretende Probleme nachvollziehen zu können. Da es sich hierbei um das mini Image „mini_vnuml-1.8.img“ handelt, Booten die virtuellen Maschinen im Vergleich zum „normalen“ Filesystem schneller. Das „normale“ Filesystem, welches auf [Vnu08] ebenfalls zum Downloaden zur Verfügung steht, ist sehr rechenintensiv und kann je nach Hardwareausstattung bis zu mehreren Minuten in Anspruch nehmen.

Im nächsten Schritt sind die RIP Daemons zu starten. Mit dem Befehl „vnumlparser.pl -x start@yloop.xml -vb -u root“ werden die Execution Tags <exec seq=“start“> in dem yloop.xml ausgeführt und somit die RIP Daemons gestartet. Dabei wird zuerst das Verwaltungsdaemon Zebra und dann die RIP Daemons auf dem Port 5000 der virtuellen Maschinen gestartet. Anschließend kann man sich vergewissern, ob die RIP Daemons tatsächlich gestartet sind. Dazu verbindet man sich im Terminal per ssh auf eine der virtuellen Maschinen (RIP - Router). Dann kann man auf der Konsole den Befehl „ps -A“ ausführen und sich anzeigen lassen, welche Prozesse gestartet sind. In dieser Auflistung der aktiven Prozesse sollte dann auch der zebra und ripd Prozesse zu sehen bzw. gestartet sein. Dazu folgende Vorgehensweise:

1. Vom Host aus sich per ssh auf der virtuellen Maschine verbinden. Beispielsweise Router singen.
   root@hamza-laptop:/usr/share/vnuml/examples# **ssh root@singen**

2. Möglicher Dialog mit “yes“ bestätigen.
   The authenticity of host 'singen (192.168.0.118)' can't be established. RSA key fingerprint is 52:56:53:f6:78:a8:91:c4:07:28:44:17:26:4a:57:e4. Are you sure you want to continue connecting (yes/no)? **yes**
   Warning: Permanently added 'singen,192.168.0.118' (RSA) to the list of known hosts.

3. Auf dem Client Router den Befehl ps -A ausführen. Anhand den rot gekennzeichneten Zeilen sieht man das das zebra und rip Daemon gestartet ist.

root@singen:~# **ps -A**

PID	*Uid*	*VmSize*	*Stat*	*Command*
1	root	324	S	init
2	root		SWN	[ksoftirqd/0]
3	root		SW<	[events/0]
4	root		SW<	[khelper]
5	root		SW<	[kthread]
6	root		SW<	[kblockd/0]
38	root		SW	[pdflush]
39	root		SW	[pdflush]
40	root		SW<	[kswapd0]
41	root		SW<	[aio/0]
760	root	772	S	/usr/sbin/sshd
767	root	300	S	/usr/sbin/syslog-ng
770	root	336	S	/sbin/getty 38400 tty0
771	root	336	S	/sbin/getty 38400 tty1
772	root	336	S	/sbin/getty 38400 tty2
777	root	556	S <	zebra -f /etc/quagga/zebra.conf -d
780	root	720	S <	ripd -f /etc/quagga/ripd.conf -x 5000 -d
781	root	1192	R	/usr/sbin/sshd: root@pts/0
783	root	956	R	-bash
785	root	312	R	ps -A

Nach dem erfolgreichen Starten der RIP-Daemons der virtuellen Maschinen kann der XTPeer ebenfalls auf dem Host gestartet werden. Dazu geht man im Terminal in das Verzeichnis wo sich die xtpeer.jar Datei befindet und führt den Befehl „java -jar xtpeer.jar“ aus. Damit man die „.jar“ Datei starten kann ist eine Java Runtime Environment (JRE) notwendig. Bei erfolgreichem Starten ist die in der folgenden Abbildung gezeigte XTPeer GUI zu sehen.

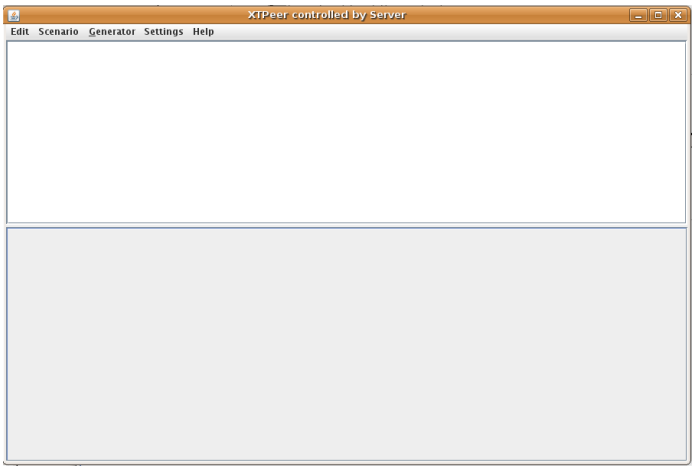


Abbildung 15: XTPeer GUI

4.3 XTPeer anwenden

Um sich einen besseren Einblick in das Tool XTPeer zu verschaffen wird jetzt an einem konkreten Netzbeispiel der CTI erzeugt. Davor ist allerdings das Beispielnetz in die XTPeer zu laden. Dazu muss die dazugehörige XML Datei vom Beispielszenario Y-Loop über die Menüleiste "Edit->Open->addXTServers from XML-File" ins XTPeer geladen werden. Bei erfolgreichem Laden des Netzes ist die untere Abbildung mit einem Bestätigungsdialog zu sehen.

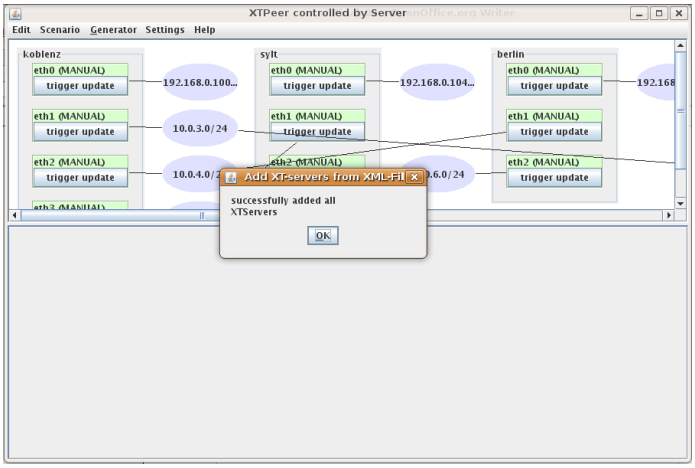


Abbildung 16: Erfolgreiches Laden von YLoop

Nun ist es an der Zeit den CTI zu erzeugen und somit das Steuerungs- und Analysetool anzuwenden. Man kann die verschiedene RIP-MTI Modi (Normal, Strict, Careful) zur Vermeidung von CTI auf den Router auswählen. Die verschiedenen RIP-MTI Modi wurden bereits im Abschnitt „Count-to-Infinity“ erklärt. Im Gegensatz zur CTI Vermeidung ist unser Ziel einen CTI zu erzeugen um mit der XTPeer Umgebung vertraut zu werden. Aus diesem Grund ist der Modus der RIP Router auf Listen oder Off zu setzen. In diesem Beispiel wird der Modus auf Listen gesetzt. Im Listen Modus wird der CTI zwar erkannt, das heißt der RIP-MTI Algorithmus ist zwar aktiv aber er hat dennoch keinen Einfluss auf das RIP Routing Verfahren. Es wird letztendlich nur der CTI erkannt und in der tabellarischen und graphischen Ansicht dargestellt. In der tabellarischen Ansicht, das heißt im unteren Abschnitt der GUI, wird der erkannte CTI gelb hinterlegt. Das beweist das der RIP - MTI Algorithmus im Listen Modus aktiv ist.

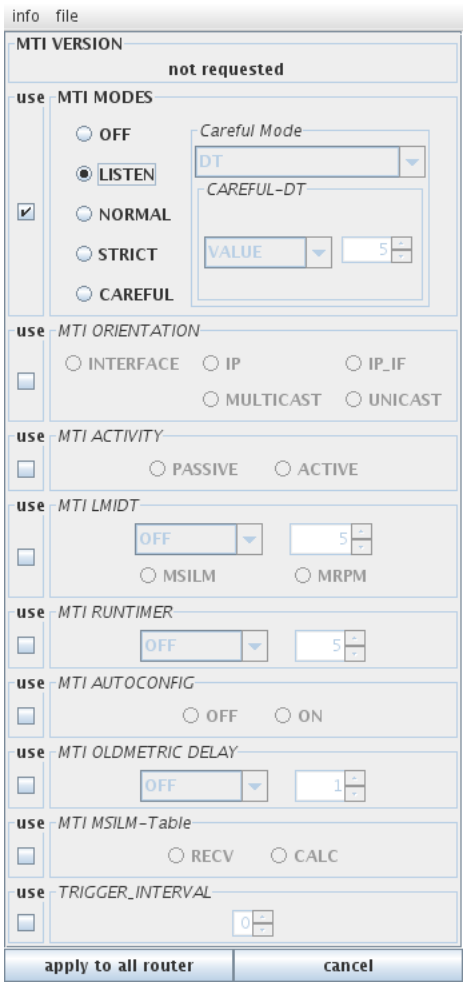


Abbildung 17: Modus Auswahl Dialog aller Router

Um letztendlich dann den Modus aller RIP Router auf Listen zu setzen ist der entsprechende Dialog über die Menüleiste „Scenario->Set MTI Options(All)“ aufzurufen. Nach dem Aufruf ist die obige Abbildung zu sehen. In diesem Dialog setzt man den Modus zuerst auf Listen setzen und dann mit dem Button „apply to all router“ die Veränderung in allen Router übernehmen. Man hat auch die Möglichkeit Veränderungen einzeln an den Router zu nehmen. Einzelne Modus Veränderungen in den RIP Router sind dann im oberen Abschnitt der GUI möglich. Dazu „klickt“ man dann einen Router an und anschließend die rechte Maustaste. In der Auflistung kann man dann den Modusauswahl Dialog aufrufen und den entsprechenden Modus für den speziellen Router auswählen.

Wie bereits schon bereits erwähnt können die Interfaces der Router zwischen MANUAL, AUTO und AUTOTRIGGER umgeschaltet werden. Zu Beginn sind die Interfaces im XTPeer standardmäßig auf MANUAL gesetzt. Damit die Router untereinander ihre Tabellen dennoch austauschen und ergänzen können sind die Interfaces der Router auf AUTO zu setzten. Das erreicht man durch den Menüeintrag „Scenario->Set Mode->Set mode to AUTO on all interfaces“. Nach [Rfc58] sind die routing updates mit 30 Sekunden spezifiziert. Da das bei einer größeren Ausdehnung der Topologie bis zu mehrere Minuten dauern kann ist das bei dieser Implementierung die routing updates auf 10 Sekunden reduziert worden. Das hat dann den Vorteil das die Konvergenzzeit reduziert wird.

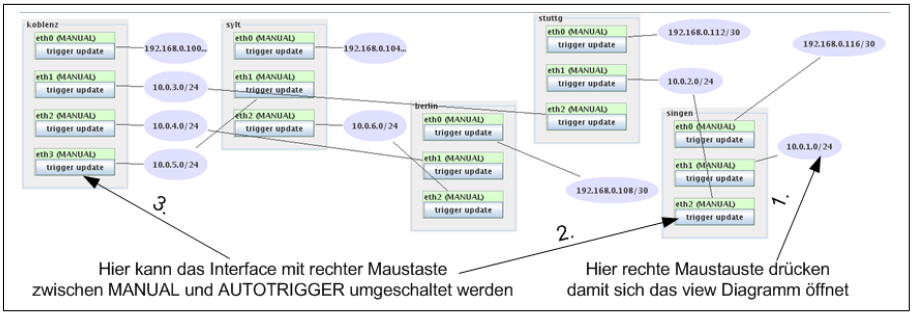


Abbildung 18: Hinweis zur XTPeer Benutzung

Nach der abgewarteten Konvergenzzeit kann letztendlich der CTI erzeugt werden. Mittlerweile haben die RIP Router ihre Routingtabellen ergänzt und vervollständigt. Um das ganze Geschehen vom CTI übersichtlich Verfolgen zu können ist der Metrikgraph vom Subnetz 10.0.1.0/24 hervorzuholen. Dazu mit der rechten Maustaustaste auf dem entsprechendem Subnetz klicken (Pfeil 1. obere Abbildung). Zum Vorschein kommt der Metrikgraph vom Subnetz 10.0.1.0/24. Danach wird der Ausfall vom Netz 10.0.1.0/24 erzwingt. Das erreicht man indem man das entsprechende Interface von Singen auf MANUAL schaltet (Peil 2. obere Abbildung). Demzufolge werden keine Routingtabellen vom Singen aus Richtung Stuttgart ausgegeben. Nach 180 Sekunden vom „Ausfall“ des Netzes 10.0.1.0/24 aus, würde dann Stuttgart die Route zum Subnetz 10.0.1.0/24 als unerreichbar (Metrik 16) kennzeichnen. Nachdem man das entsprechende Interface von Singen auf Manual gesetzt hat sind etwa 50 Sekunden abzuwarten bis man den Schritt macht. Im nächsten Schritt ist dann das Interface von Koblenz

auf Sylt ebenfalls auf MANUAL zu setzen (Pfeil 3. obere Abbildung). Somit wird die Unerreichbarkeitsinformation vom Router Koblenz nur in Richtung Berlin weitergeleitet. Im Metrikgraphen vom Subnetz 10.0.1.0/24 ist dann Koblenz zu beobachten. Als nächstes bekommt er dann eine Unerreichbarkeitsinformation von Stuttgart. Dabei geht dann die Metrikanzeige im Metrikgraphen auf 16 hoch. Diese Unerreichbarkeitsinformation wird dann von Koblenz aus in Richtung Berlin weitergeleitet weil das Interface von Koblenz zu Sylt auf MANUAL gesetzt wurde. Sobald aber Koblenz dann eine alte „falsche“ Metrik von einem anderen Router bekommt so ist das entsprechende Interface (Pfeil 3. obere Abbildung) auf AUTOTRIGGER zu setzten. Um sich ein Bild vom Metrikgraphen zu machen, wann das Interface von Koblenz auf AUTOTRIGGER umzuschalten ist, kann man in der unteren Abbildung am dem Zeitpunkt 4 ersehen. Nach dieser Aktion wird die falsche Metrikinformation sich in einer Routingschleife zwischen den Router Koblenz, Sylt und Berlin verbreiten. Bei jedem Router wird dann die Metrik zum Subnetz 10.0.1.0/24 um eins bis zum infinity erhöht.

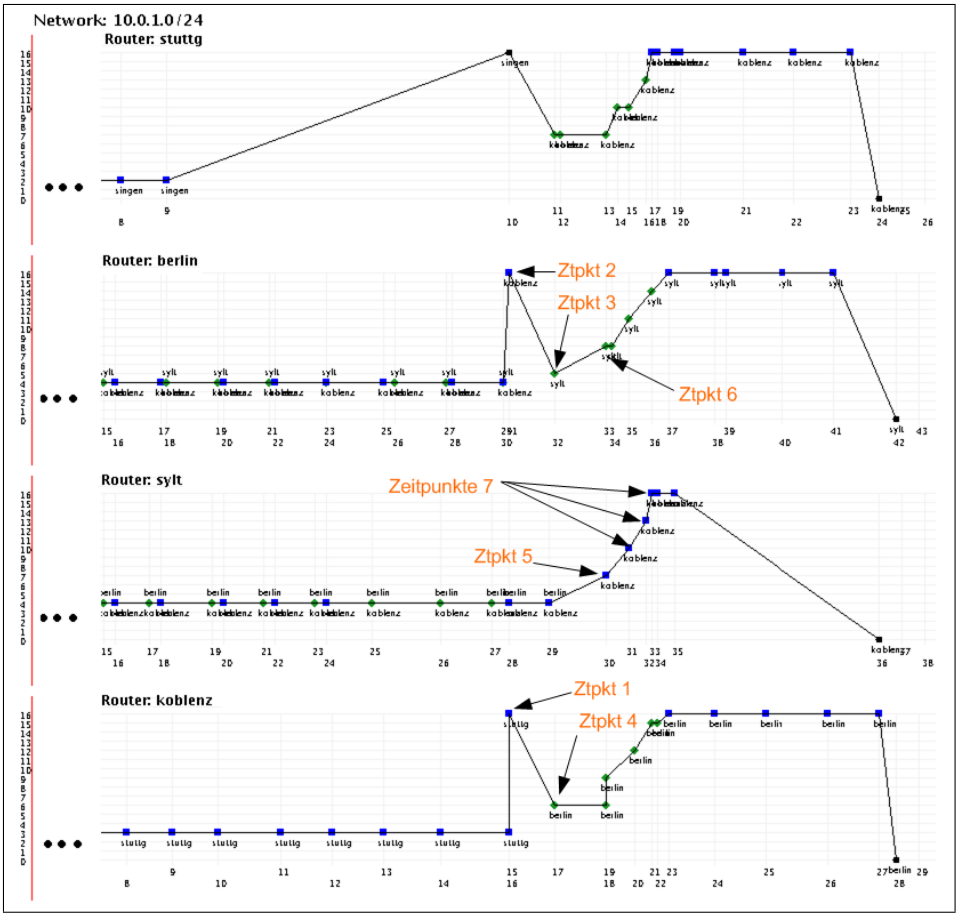


Abbildung 19: Count To Infinity

Als Resultat ist im Metrikgraph oben der CTI abzulesen. Das selbst erstellte Beispiel kann natürlich je nach eigenem von dieser Auswertung Abweichen. Zum Verständnis dieser Auswertung ist folgendes zu sagen. Das Ztpkt oben im Me-

trikgraphen bedeutet Zeitpunkt. Zum Zeitpunkt 1 sieht man das Router Koblenz die Unerreichbarkeitsinformation zum Subnetz 10.0.1.0/24 von Router Stuttgart bekommt. Daraufhin wird diese Unerreichbarkeitsinformation, nach dem Verlauf der update time von Koblenz, weiter an Berlin gesendet (Zeitpunkt 2). Da aber Sylt noch die alte ungültige erreichbare Metrik zum Subnetz 10.0.1.0/24 hat, wird diese nach dem Verlauf seiner update time dem Berlin weitergegeben. Berlin übernimmt die alte falsche Metrik zum Subnetz 10.0.1.0/24 von Sylt in die eigene Routingtabelle auf (Zeitpunkt 3) und inkrementiert die Metrik um eins. Berlin wiederum informiert Koblenz über eine angebliche Verbindung zum Subnetz 10.0.1.0/24 (Zeitpunkt 4). Der Zeitpunkt 5 zeigt das Sylt von Koblenz lernt. Im Zeitpunkt 6 sieht man das Berlin von Sylt lernt. Es gibt sozusagen eine Routingschleife zwischen den Router Koblenz->Sylt->Berlin->Koblenz. Jedesmal wird dabei in einem Router die Metrik zum Subnetz 10.0.1.0/24 um eins erhöht. Die Metriken werden dabei in dreier Schritten bis zum CTI inkrementiert und nach einer anschließenden bestimmten Zeit verworfen. Siehe Zeitpunkt 7 für Sylt.

4.4 XTPeer schließen und Y-Loop herunterfahren

Es ist ratsam das Beispielszenario ordnungsgemäß herunter zu fahren. Aber davor ist erst einmal das XTPeer zu schließen. Das kann gewöhnlich wie jedes andere Programm über die Menüleiste erfolgen. Und zwar unter dem Menüpunkt “edit“ ist “close“ zu finden. Danach sind Daemons in den virtuellen Maschinen zu stoppen. Dazu im Terminal in das Verzeichnis wechseln wo sich die XML Datei vom Beispielszenario befindet. Dann den Befehl „vnumlparser.pl -x stop@yloop.xml -vb -u root“ auszuführen. Demzufolge wird in der XML Datei alle befindlichen <exec seq=“stop“> Tags ausgeführt. Hier wird zuerst das Daemon Zebra und anschließend das Daemon “gekillt“.

...

< exec seq=“stop“ type=“verbatim“ > killall zebra < /exec >

< exec seq=“stop“ type=“verbatim“ > killall ripd < /exec >

...

Nachdem die Daemons “gekillt“ wurden muss letztendlich die Simulation sauber beendet werden. Das erreicht man mit dem Befehl „vnumlparser.pl -d yloop.xml -vb -u root“. Sind in Filesystem Änderungen gemacht wurden so muss schlussendlich der Befehl „vnumlparser.pl -P yloop.xml -vb -u root “ ausgeführt werden.

4.5 Zusammenfassung

Beginnend wurde das Beispielszenario Y-Loop vorgestellt. Folgend darauf wurden im Abschnitt 4.1 die Komponenten aufgelistet, die zur Ausführung dieses Beispiels notwendig sind. Bevor man allerdings XTPeer anwenden kann muss das Beispielszenario unter VNUML hochgefahren werden. Anschließend dazu

müssen die Daemons in den virtuellen Maschinen gestartet werden. Erst im Anschluss dazu kann der XTPeer gestartet und angewandt werden. Das XTPeer wird dann verwendet um das Beispielszenario anzusteuern und Beobachtungen zu machen. Hier in diesem Fall wird ein CTI erzeugt um mit der XTPeer Umgebung vertraut zu werden. Bei dir CTI Erzeugung wird dann Schritt für Schritt erklärt welche Konfigurationen man an den Router vornehmen muss. Ebenso wird der kontrollierte Eingriff an den entsprechen Router erklärt. Zum kontrollierten Eingriff gehört der Verbindungsausfall zwischen 2 Router oder das Versenden von Routingtabellen zu bestimmten Zeiten. Mit dem Ziel die Interpretation vom Metrikgraphen zu verstehen wurde folgend darauf der erzeugte CTI am Metrikgraphen erklärt. So kann der zukünftige Entwickler sich ein Gebrauch vom Metrikgraphen machen, indem das selbst Implementierte bzw. modifizierte Protokoll analysiert wird. Abgeschlossen wird dieses Kapitel mit dem ordnungsmäßigen herunterfahren des Beispielszenarios.

5 Probleme und Lösungen

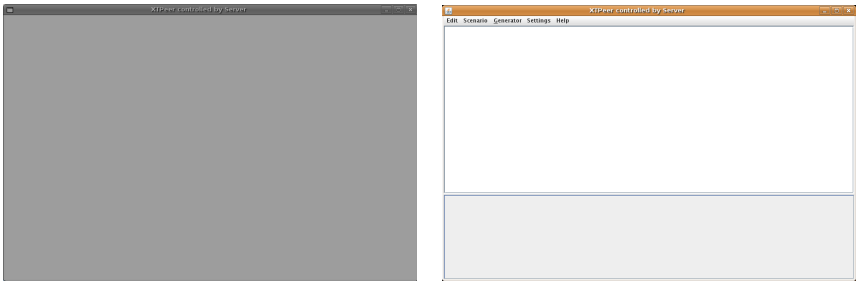
In diesem Kapitel werden Problem wiedergegeben und Lösungen vorgeschlagen, die während der praktischen Umsetzung der Studienarbeit aufgetreten sind bzw. auftreten könnten.

5.1 Problem: XTPeer Menüleisten fehlen

Nach dem Start von XTPeer fehlen sämtliche (Menü-) Leisten fehlt und es lässt sich nichts Bedienen. Sprich man keine Buttons betätigen. Zudem ist das XT-Peer dunkelgrau.

Mögliche Ursachen:

- Installation der Java Version von Drittanbietern. Zu empfehlen ist die Java Version von der Firma Sun [Sun08].
- Mit dem Befehl „java -version“ ist die Version zu ersehen. Hier in der Arbeit zum Vergleich:
  - java version „1.6.0_03“
  - Java ^{TM} SE Runtime Environment (build 1.6.0_03-b05)
  - Java HotSpot ^{TM} Client VM (build 1.6.0_03-b05, mixed mode, sharing)



(a) Java Version von Drittanbietern (b) Java Version von Sun

Abbildung 20: Gegenüberstellung der Java Versionen

5.2 Problem: Änderungen werden im Root Filesystem nicht übernommen

Sobald man Änderungen in dem Root Filesystem vorgenommen hat sollte man das alte Root Filesystem löschen bevor man mit dem neuen startet. Der Grund dafür ist das eine COW basierte Vorgehensweise einen Verweis auf das root Filesystem setzt von dem sie abgeleitet worden ist.

Vorgehendsweise:

- vnumlparser.pl -P szenario.xml -vb -u root // löscht das Root Filesystem
- vnumlparser.pl -t szenario.xml -vb -u root // startet die VNUML Simulation

5.3 Problem: die Simulation startet nicht richtig

Die Simulation startet nicht richtig. Es kommt von den virtuellen Maschinen möglicherweise die Fehlermeldung "xxx sshd is not ready ...". Laut der Projektseite von [Vnu08] kann es wie folgt mehrere Gründen dafür geben.

- Es kann sein das die Root Filesysteme der virtuellen Maschinen kaputt gegangen sind. Demzufolge starten die virtuellen Maschinen nicht mehr. Als Lösung wird hier vorgeschlagen, wie schon vorher beschrieben, das Root Filesystem mit der Option -P zu Löschen und anschließend wieder die VNUML Simulation mit der Option -t zu starten.
- Es sollte auf die Kompatibilität der Linux Kernel Version dem Root Filesystem geachtet werden. Bevor man allerdings sich an die komplexen Beispiele macht sollte zuvor das simple.xml Szenario von der Projektseite durchgespielt werden.
- Nachschauen ob das sshd Daemon in den virtuellen Maschinen läuft. Beispielsweise kann dann vom Host aus auf die virtuelle Maschine mit dem Telnet auf Port 22 verbunden werden.

5.4 Problem: SSH Verbindung zu einem virtuellen Rechner scheitert

Wenn die Verbindung zu einer virtuellen Maschine mittels SSH nicht zu Stande kommt so sollte man die Datei „/home/hamza[11]/.ssh/known_hosts" löschen. Danach versucht man sich wieder ganz normal mit einer virtuellen Maschine zu verbinden. Auf die Frage ob man die Verbindung zur virtuellen Maschine fortsetzen will ist mit *yes* zu bestätigen. Letztendlich kann man dann mit dem Root Passwort sich mit der virtuellen Maschine verbinden. Im Folgenden wird das ganze am Beispiel Router Singen demonstriert.

Fehlermeldung:

```
hamza@hamza-laptop:/usr/share/vnuml/examples$ ssh root@singen

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: POSSIBLE DNS SPOOFING DETECTED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

The RSA host key for singen has changed,
and the key for the according IP address 192.168.0.110
is unknown. This could either mean that
DNS SPOOFING is happening or the IP address for the host
```

[11] Hier steht der username drin

```
and its host key have changed at the same time.

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
02:ee:95:43:81:b7:6a:2c:1b:19:03:51:a8:b4:59:40.
Please contact your system administrator.
Add correct host key in /home/hamza/.ssh/known_hosts to get rid of this message.
Offending key in /home/hamza/.ssh/known_hosts:1
RSA host key for singen has changed and you have requested strict checking.
Host key verification failed.
```

Lösungsweg:

1. Datei /home/hamza/.ssh/**known_hosts** löschen
2. Verbindung zum router herstellen
   hamza@hamza-laptop:/usr/share/vnuml/examples$ ssh root@singen
   The authenticity of host 'singen (192.168.0.110)' can't be established.
   RSA key fingerprint is 02:ee:95:43:81:b7:6a:2c:1b:19:03:51:a8:b4:59:40.
3. Bestätigen
   Are you sure you want to continue connecting (yes/no)? **yes**
   Warning: Permanently added 'singen,192.168.0.110' (RSA) to the list of known hosts.
4. Das Filesystem Passwort eingeben um sich zu verbinden
   root@singen's password: **xxxx**
   Linux vm 2.6.18.1-bb2-xt-4m #1 Sun Dec 23 13:56:48 CET 2007 i686
   The programs included with the Debian GNU/Linux system are free software;
   the exact distribution terms for each program are described in the
   individual files in /usr/share/doc/*/copyright.
   Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
   permitted by applicable law.
   singen:#

5.5 Problem: XTPeer kann keine Verbindung herstellen

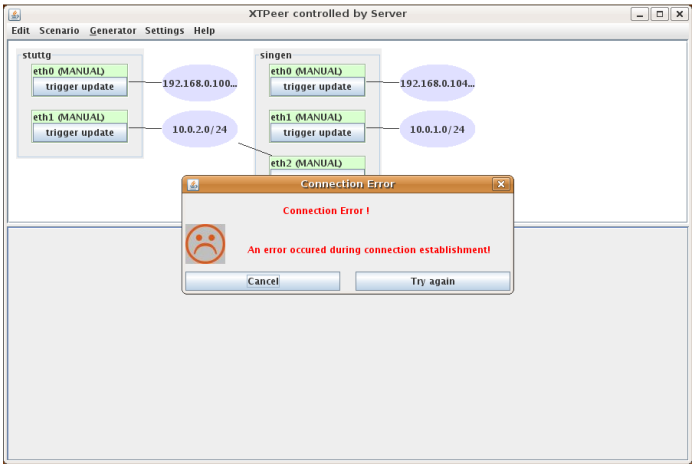


Abbildung 21: XTPeer connection error

Lösungsvorschlag:
Das VNUML Szenario runterfahren beziehungsweise auch die Option -P verwenden und dann nochmal starten (Abschnitt 5.2).

5.6 Problem: XTPeer Registerreiter werden nicht angezeigt

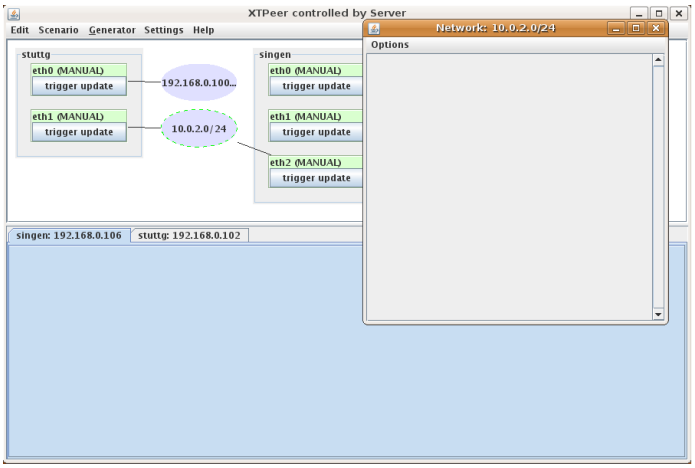


Abbildung 22: XTPeer Registerreiter werden nicht angezeigt

Lösungsvorschlag:
In diesem Fall sollten die Konfigurationsfiles von den Zebra und Ripd Daemons überprüft. Falls man das Root - Filesystem von der VNUML Projektseite runter ladet, dann könnten eventuelle Passwortaktivierungen Probleme verursachen.

5.7 Problem: Zebra lässt sich nicht starten

Fehlerbeschreibung:
Beim starten des Zebra Daomons kommt die Fehlermeldung:

„zebra: symbol lookup error: /usr/lib/libzebra.so.0: undefined symbol: cap_init“

Lösungsvorschlag:
Das Paket „libpcap-dev“ installieren und dann Quagga nochmal neu kompilieren und installieren. Ansonsten mal die Log-Dateien von Zebra und ripd Überprüfen. Das Logverzeichniss müsste sich unter „/var/log/zebra/“ oder „/var/log/quagga/“ befinden.

5.8 Problem: Linkerfehler beim kompilieren

Nach folgenden Schritten :

```
Quagga Extrahieren (tar ...)            --> ok
Patchen (patch -p0 ...)                 --> ok
Aclocal && libtoolize --copy --force    --> ok
./configure                             --> ok
Make                                    --> Linkerfehler
```

Fehlerbeschreibung: Wenn nach dem Patchen vom RIP-MTI bei der Kompilierung eine Linkerfehlerausgabe folgender Art kommt:

```
ripd.o: In function ‘rip_timeout’:

/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:203: undefined
 reference to ‘mti_state’
/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:222: undefined
 reference to ‘slc_state’
/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:224: undefined
 reference to ‘xt_sl_notify_peer’

ripd.o: In function ‘rip_garbage_collect’:

/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:166: undefined
 reference to ‘resetMTIModesAndTimers’
/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:168: undefined
 reference to ‘slc_state’
/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:171: undefined
 reference to ‘xt_sl_notify_peer’

ripd.o: In function ‘rip_timers’:

/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/ripd.c:3465: undefined
 reference to ‘slc_state’
ripd.o: In function ‘xtpeer_cleanRoutingTable’:
.......
rip_interface.o: In function ‘rip_interface_up’:
```

```
/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd/rip_interface.c:449: undefined
  reference to ‘rip_xt_interface_add’

collect2: ld returned 1 exit status

make[2]: *** [ripd] Fehler 1
make[2]: Verlasse Verzeichnis ’/home/hamza/Desktop/Quagga/quagga-0.99.4/ripd’
make[1]: *** [all-recursive] Fehler 1
make[1]: Verlasse Verzeichnis ’/home/hamza/Desktop/Quagga/quagga-0.99.4’
make: *** [all] Fehler 2
```

Lösungsvorschlag:
In der Datei „rip_mti.h“, ganz am Anfang, den Eintrag „#include <ripd.h>“ in „#include <ripd/ripd.h>“ umändern.

6 Rückblick und Fazit

Rückblickend ist zu sagen, das das gesamte Themengebiet doch sich als Umfangreich erwiesen hat. Man denke nur an Tools, Software, Betriebssystemkentnisse und theoretisches Wissen das zum Einklang gebracht werden muss. Es ist immer von Wichtigkeit das man in Gewissen Situationen von der eigentlichen Aufgabenstellung sich nicht abweichen lässt und immer zielstrebig darauf weiter arbeitet. Mittels dieser Arbeit hat der zukünftige Projektteilnehmer einen Leitfaden und Überblick über die einzelnen Gebiete bekommen. Je nach Aufgabenstellung sind die Gewichte dann in unterschiedlichen Bereichen zu setzen und genau daran weiter zu arbeiten. Bereiche sind zum Beispiel Weiterentwicklung an dem Routingschleifenvermeidungsprotokoll RIP-MTI oder Funktionalitätserweiterungen bzw. Anpassungen an dem Steuerungs- und Analysetool XTPeer.

Die Studienarbeit hat mir persönlich Nutzen in mehreren Bereichen gebracht. Zum Einen wurden meine theoretischen Kenntnisse verfestigt und erweitert. Zum Anderen hat mir die Arbeit Gelegenheit gegeben meine theoretischen Kenntnisse in praktischer Arbeit umsetzen zu können. Des weiteren wurde ich mit Problemen konfrontiert die am Anfang nicht wirklich ersichtlich waren aber dennoch zu meistern sind. Es ist doch was anderes wenn man versucht ein Projekt in einer gegeben Zeit mit mehreren Komponenten (XTPeer, VNUML, Wissen, Linux, Root Filesystem) in Einklang zu bringen.

Abbildungsverzeichnis

Literatur

[Abook] Autobook, `http://sources.redhat.com/autobook/autobook/autobook_toc.html`, Stand Oktober 2008

[Aconf] Autoconf Dokumentation, `http://www.delorie.com/gnu/docs/autoconf/autoconf_toc.html`, Stand Oktober 2008

[Amake] Automake Dokumentation, `http://www.delorie.com/gnu/docs/automake/automake_toc.html`, Stand Oktober 2008

[Boh08] Frank Bohdanowicz, *Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons*. Diplomarbeit, 2008, Universität Koblenz Landau

[Deb] Debian, `http://www.debian.org/`, Stand Oktober 2008

[Keu07] Tim Keupen, *Generierung von Testfällen für den RIP-MTI Algorithmus*. Diplomarbeit, 2007, Universität Koblenz Landau

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

[N3vlr] N3 Virtual Linux Router, `http://wiki.n3network.ch/index.php/N3_Virtual_Linux_Router`, Stand Oktober 2008

[Päh08] Daniel Pähler, *Extern steuerbare Routing-Updates im RIP-Daemon der Quagga-Programmsuite*. Diplomarbeit, 2006, Universität Koblenz Landau

[Qua08] Quagga Routing Suite, `http://www.quagga.net/`, Stand Juli 2008

[Rfc58] C. Hedrik, *Routing Information Protocol*. `http://tools.ietf.org/html/rfc1058`, RFC 1058, Juli 2008

[Scfg08] Sourceforge.net, `http://user-mode-linux.sourceforge.net/`, Open Source Software, Stand Juli 2008

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

[Sun08] Sun microsystems, `http://java.sun.com/`, Stand Juli 2008

[uCl08] Für Linux-Embedded-Systeme kleine C-Bibliothek, `http://www.uclibc.org/`, Stand Juli 2008

[Uml08] The User-mode Linux Kernel, `http://user-mode-linux.sourceforge.net/`, Stand Juli 2008

[Vnu08] Virtual Network User Mode Linux (VNUML), `http://www.dit.upm.es/vnumlwiki/index.php/Main_Page`, Stand August 2008

A Anhang

A.1 Beispielszenario Y-Loop

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">
<vnuml>
<global>
  <version>1.8</version>
  <simulation_name>yloop</simulation_name>
  <ssh_version>2</ssh_version>
  <ssh_key>/root/.ssh/id_rsa.pub</ssh_key>
  <automac/>
  <vm_mgmt type="private" network="192.168.0.0" mask="24"
       offset="100">
    <host_mapping/>
  </vm_mgmt>
  <vm_defaults exec_mode="mconsole">
    <filesystem type="cow">/usr/share/vnuml/filesystems/
        root_fs_18</filesystem>
    <kernel>/usr/share/vnuml/kernels/linux</kernel>
    <!--console id="0">xterm</console-->
    <!--xterm>gnome-terminal,-t,-x</xterm-->
    <forwarding/>
  </vm_defaults>
</global>

<net name="a" mode="uml_switch" type="lan"/>
<net name="b" mode="uml_switch" type="lan"/>
<net name="c" mode="uml_switch" type="lan"/>
<net name="d" mode="uml_switch" type="lan"/>
<net name="e" mode="uml_switch" type="lan"/>
<net name="f" mode="uml_switch" type="lan"/>

<vm name="koblenz">
  <if id="1" net="b">
    <ipv4>10.0.3.1</ipv4>
  </if>
  <if id="2" net="c">
    <ipv4>10.0.4.1</ipv4>
  </if>
  <if id="3" net="e">
    <ipv4>10.0.5.1</ipv4>
  </if>

  <exec seq="start" type="verbatim">zebra -f /etc/quagga/
      zebra.conf -d</exec>
  <exec seq="start" type="verbatim">ripd -f /etc/quagga/
      ripd.conf -x 5000 -d</exec>
```

```
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
  <exec seq="rp" type="verbatim">sysctl -w net.ipv4.conf.
      all.rp_filter=0</exec>
</vm>

<vm name="sylt">
  <if id="1" net="e">
    <ipv4>10.0.5.2</ipv4>
  </if>
  <if id="2" net="f">
    <ipv4>10.0.6.1</ipv4>
  </if>
  <exec seq="start" type="verbatim">zebra -f /etc/quagga/
      zebra.conf -d</exec>
  <exec seq="start" type="verbatim">ripd -f /etc/quagga/
      ripd.conf -x 5000 -d</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
  <exec seq="rp" type="verbatim">sysctl -w net.ipv4.conf.
      all.rp_filter=0</exec>
</vm>

<vm name="berlin">
  <if id="1" net="c">
    <ipv4>10.0.4.2</ipv4>
  </if>
  <if id="2" net="f">
    <ipv4>10.0.6.2</ipv4>
  </if>
  <exec seq="start" type="verbatim">zebra -f /etc/quagga/
      zebra.conf -d</exec>
  <exec seq="start" type="verbatim">ripd -f /etc/quagga/
      ripd.conf -x 5000 -d</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ripd</exec>
  <exec seq="rp" type="verbatim">sysctl -w net.ipv4.conf.
      all.rp_filter=0</exec>
</vm>

<vm name="stuttg">
  <if id="1" net="a">
    <ipv4>10.0.2.1</ipv4>
  </if>
  <if id="2" net="b">
    <ipv4>10.0.3.2</ipv4>
  </if>
  <exec seq="start" type="verbatim">zebra -f /etc/quagga/
      zebra.conf -d</exec>
```

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

<vm name="singen">
  <if id="1" net="d">
    <ipv4>10.0.1.1</ipv4>
  </if>
  <if id="2" net="a">
    <ipv4>10.0.2.2</ipv4>
  </if>
    <exec seq="start" type="verbatim">zebra -f /etc/
        quagga/zebra.conf -d</exec>
    <exec seq="start" type="verbatim">ripd -f /etc/quagga
        /ripd.conf -x 5000 -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
    <exec seq="rp" type="verbatim">sysctl -w net.ipv4.
        conf.all.rp_filter=0</exec>
</vm>
</vnuml>
```

A.2 Quagga Patchen

Die folgende Anleitung beschreibt das Compileren der Patches für die Quagga Version 0.99.4.Die gekürzte Vorgehensweise ist in [Lan07] zu finden.

```
You need the following to install RIP-XT:

- quagga-0.99.4.tar.gz
- quagga-0.99.4_RIP_MTI_XT_SL.patch

1.Unpack the tarball with

tar -xvzf quagga-0.99.4.tar.gz

2.Apply the patch with

patch -p0 < quagga-0.99.4_RIP_MTI_XT_SL.patch

(This will give you a ripd with MTI, XT and SL capabilities)


3.Enter the quagga source directory

cd quagga-0.99.4

4.Due to changes in the Makefiles, you will need to run

aclocal && libtoolize --copy --force

next. If either of these programs isn't available on your
system, install "automake" and "libtool" to get them.
If you run into problems, try to update both above mentioned
tools as well as "m4".

5.Finally, you can procede with configure and make as you
would in an unpatched quagga:

EASIEST:
./configure && make


CUSTOMIZED:
It is recommended to take a closer look at the output of
"./configure --help" and decide which features are needed and
which aren't.
An example might be:

./configure --enable-debug=full --disable-ipv6  --disable-ripngd
--disable-isisd --disable-ospf6d --disable-ospfd --disable-bgpd
```

```
--enable-user=root

In this configuration, debugging messages are enabled and most
unused daemons as well as IPv6 support are disabled.


YET MORE CUSTOMIZED:
When developing and testing XT, the following configure command
turned out to be quite useful:

TARGET=/home/tulkas/quagga ; ./configure --enable-debug=full
--disable-ipv6  --disable-ripngd  --disable-ospf6d --disable-ospfd
--disable-bgpd --enable-user=root --prefix=${TARGET}
--localstatedir=${TARGET}/var --libdir=${TARGET}/lib

This should install all of quagga into the directory given in TARGET
(you have to replace "/home/tulkas/quagga" with some path that exists
on your system).

Advantages:
- Compiling and installing can be done with user privileges.
- It leaves any quagga that might already be on your system
  untouched, allowing you to have several parallel versions.
- It does not confuse your distribution's package management
  system and can easily be uninstalled (i.e., deleted) if necessary.

Disadvantages:
- While this works perfectly well on the machine on which RIP-XT was
  developed, it lead to strange bugs after it was compiled in a
  UML instance using debian: zebra and ripd could still be run, but
  interprocess communication didn't work for some reason or other.
  If this happens to you, chose the "CUSTOMIZED" way.


6) A note on running ripd
To make the distinction after compiling easier, ripd's help message
was altered: calling "ripd -h" now also returns information about
which patch was applied, e.g.

Usage : ripd [OPTION...]
Daemon which manages RIP version 1 and 2.
This version includes the MTI and XT modifications.
```

A.3 Übersicht: Starten des Systems

Es ist darauf zu achten das die folgenden Operationen als Root ausgeführt werden. Bei Ubuntu ist vor den Befehlen „sudo“ voranzuschreiben.

- Filesystem mini_vnuml-1.8.img ins Filesystem Verzeichnis kopieren „/usr/share/vnuml/filesystems“
- Filesystem verlinken „ln -sf mini_vnuml-1.8.img root_fs_18“
- VNUML Kernel in das entsprechende Verzeichnis kopieren „/usr/share/vnuml/kernels“
- VNUML Kernel entpacken „tar -xvzf linux-um_2.6.18.1-bb2-xt-4m.orig.tar.gz“
- VNUML Kerlen verlinken „ln -sf linux-um-2.6.18.1-bb2-xt-4m/linux-2.6.18.1-bb2-xt-4m linux“
- In das Szenarioverzeichnis wechseln „cd /usr/share/vnuml/examples/“
- Simulation starten: „vnumlparser.pl -t yloop.xml -vb -u root“
- Daemons starten: „vnumlparser.pl -x start@yloop.xml -vb -u root“
- In das XTPeer Verzeichnis wechseln und das XTPeer starten „ java -jar xtpeer.jar“
- XTPeer anwenden

A.4 Checkliste

Falls bei der Erstellung bzw. Anwendung des Root Filesystems Probleme gibt kann die folgenden Checkliste als Gedächtnisstütze durchgegangen werden.

- Pakete die im Filesystem installiert und ausgewählt sein müssten
  - Libtool,
  - Automake1.9
  - Aclocal1.9
  - Openssh-server
  - Libpcap-dev
- Verzeichnisse
  - Konfigurationsdateien:
    - „/etc/quagga/zebra.conf“
  - Logfiles von zebra und ripd:
    - „/var/log/zebra“ oder
    - „/var/log/quagga“
  - Vnuml Verzeichnis (Exmaple, Kernel, Filesystem)
    - „/usr/share/vnuml/“
  - Root Filesystem
    - Login: „root“
    - Passwort: „xxxx“
    - Passwort bei zebra und ripd: „zebra“