Fachbereich 4: Informatik

Simulation großer Netzwerke in der VNUML-Umgebung

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von
Andreas Garbe

201210337

Erstgutachter:	Prof. Dr. Christoph Steigner Institut für Informatik
Zweitgutachter:	Frank Bohdanowicz Institut für Informatik

Koblenz, im September 2010

Erklärung

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

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

.....

(Ort, Datum) (Unterschrift)

Kurzfassung

Bislang wurde VNUML (Virtual Network User Mode Linux) innerhalb der AG Rechnernetze der Uni Koblenz dazu verwendet, um die eigene Protokollerweiterung zu RIP, RIP/MTI (RIP with minimal topology information), auf Stärken und Schwächen zu testen. Hauptsächlich wurden dafür spezielle Testszenarien verwendet, um zu untersuchen, ob ein Count-to-Infinity-Problem (CTI) erfolgreich verhindert wird und wie schnell das Netz nach Ausfall einer Route konvergiert.
Diese Arbeit wird untersuchen, ob die MTI-Erweiterung auch Performance-Vorteile in größeren Netzwerken bietet, ob sich der Einsatz des Script-Tools EDIV (spanisch: Escenarios DIstribuidos con VNUML, englisch: Distributed Scenarios using VNUML) aufgrund der besseren Skalierbarkeit lohnt und ob sich durch die Verteilung eines XML-Szenarios auf mehrere Rechner signifikant auf die Konvergenzzeit auswirkt. Dazu werden neben Simulationen auch Testszenarien entworfen und umfangreichen Tests unterzogen, um Erkenntnisse zur Effizienz und Skalierbarkeit des Distance Vector Routing Protokolls RIP/MTI zu ziehen.

Abstract

So far VNUML (Virtual Network User Mode Linux) has been used by the group for Computer Networks at the University of Koblenz in such a way as to test its own protocol enhancement for RIP (Routing Information Protocol) on strengths and weaknesses. The modified version of RIP is called RMTI (RIP with minimal topology information).
In particular, special test scenarios have been used to investigate wether a Count-to-Infinity (CTI) problem can be completely avoided and how quickly the network converges after the failure or breakdown of a router. Tius thesis investigates whether the MTI enhancement also provides for better performance in larger networks. Furthermore, it will be investigated if it is worth using the script tool EDIV ((spanish: Escenarios DIstribuidos con VNUML, english: Distributed Scenarios using VNUML) due to its enhanced scalability and whether the distribution of an XML scenario on several computers has a significant impact on the convergence time. Apart from simulations, test scenarios will be developed and tested in order to generate results about the efficiency and scalability of the Distance Vector Routing Protocol.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Kapitel 1

Einleitung

Die Forderungen an ein Routing Protokoll innerhalb des Internets sind hoch und die Fehlertolleranz gering, RIP (Routing Information Protocol) existiert schon seit den Anfängen des Internets und steuert dezentral die Verbindungen in Autonomen Systemen. Mit der wachsenden Größe des Internets gilt es immer mehr dieser Systeme zu vernetzen, die Kommunikation dazwischen geschieht über Router, auf denen RIP implementiert ist. Die Aufgaben sind das Weiterleiten von Datenpaketen in die richtige Richtung und die Verhinderung von Schleifen, d.h. die Datenpakete sollten vom Absender bis zum Empfänger keinen Router mehrmals passieren, schließlich kostet unnötiger Datenverkehr Zeit und Geld, im schlimmsten Fall gehen Daten verloren. Die Entwickler von RIP müssen sich ebenso wie die Entwickler alternativer Routing Protokolle (OSPF[1], IS-IS[2]) um Lösungen bemühen.
Die Weiterentwicklung von RIP, RIP/MTI (RIP with Minimal Topology Information), zielt mittels neuer Algorithmen auf die absolute Vermeidung von Schleifen, ohne dabei auf die Vorteile eines Distance Vector Protocols zu verzichten. Vergangene Untersuchungen haben gezeigt, dass die Schleifenauflösung in klassischen Problemfällen eine Verbesserung zu RIP darstellt, es wurde allerdings noch nicht untersucht in wie weit die Skalierbarkeit funktional ist. RIP besitzt einen maximalen Hopcount von 16, der als künstliche obere Grenze des Count-to-Infinity-Problems, also einer unendlichen Schleife, festgelegt wurde. Sobald eine Route mehr als 15 Router passiert, deklarieren Algorithmen in RIP diese Route als ungültig. RIP/MTI erhöht diesen Wert auf 64, was sich positiv auf die maximale Größe der Topologie auswirkt. Zusammen mit der Schleifenvermeidung stellt diese Lösung eine sinnvolle Alternative zu den aktuell eingesetzten Routing Protocols dar. Diese Arbeit wird Unterschiede in den Konvergenzzeiten untersuchen, die zwischen RIP und RIP/MTI bestehen könnten, dazu werden größer skalierte Netzwerkszenarien herangezogen und Testfälle

[1] OSPF Protocol
RFC: http://tools.ietf.org/search/rfc2328
[2] OSI IS-IS Intra Domain Routing Protocol
RFC: http://tools.ietf.org/html/rfc1142

durchgeführt, die Ausfälle von Netzen oder Routern simulieren. Es soll festgestellt werden ob die neuen Algorithmen auch in reelen Situationen ausfallsicher und performant funktionieren.
Es wird empfohlen diese Arbeit als elektronisches Dokument (PDF) zu lesen, da viele Querverweise klickbar auf andere Stellen dieser Arbeit verweisen. Die Inhalte der DVD werden online auch unter http://www.shakuras.net/dipl2010/ zugänglich sein (für bis zu 2 Jahre nach Abgabe)

1.1 Werdegang von RIP/MTI

Die Arbeit der AG-Rechnernetze setzt sich zum Ziel das nur begrenzt nutzbare Distanz-Vektor-Protokoll RIP derart zu verändern, dass man es auch für größere Netzwerktopologien verwenden kann. RIP hatte in der Vergangenheit das Problem, dass sich zu oft Schleifen innerhalb der Netzwerktopologie bildeten, die zu einer schlechten Performanz oder gar Datenverlust führten. Nichtsdestotrotz war die Informationsweitergabe effizient, da Router nur mit direkten Nachbarroutern kommunizierten, anstatt das Netzwerk mit Routing-Informationen zu fluten (flooding). Da eine Loop-Erkennung nicht existent war, sollten in der Theorie Routing-Schleifen, auch Count-to-Infinity-Probleme (CTI)[3] genannt, dadurch identifiziert werden, dass Routen mit mehr als 15 Hops als ungültige Route deklariert werden. Man nahm an, dass Routen mit einer so hohen Metrik zwangsläufig eine Schleife beinhalten, somit war die Skalierbarkeit von RIP eingeschränkt und Netztopologien mit einem höheren Diameter als 16 waren nicht möglich.
Die AG-Rechnernetze erweitert RIP, es wurde bereits eine Schleifenerkennung implementiert, die theoretisch keine Count-to-Infinity-Probleme mehr zulässt, die Modifikation nennt sich RIP/MTI (RIP with minimal topology information). Der Hop-Count, die für das RIP-Protokoll höchst zulässige Anzahl von Routern, um ein Zielnetz zu erreichen, wurde erhöht, um auch größere Netzwerke zuzulassen, denn ohne die Gefahr eines CTIs könnten Distanz-Vektor-Protokolle wieder Anwendung in Netzwerken finden. Klassische Topologien, in denen regelmäßig CTI-Probleme auftraten, wurden ausgiebig getestet und evaluiert, das RIP/MTI Protokoll konnte bereits einen Performanzvorsprung gegenüber RIP aufweisen.

1.2 Ziele dieser Arbeit

Diese Arbeit soll sich mit der Performanz von großen Netzwerken beschäftigen. Es ist nun möglich Topologien mit einem weitaus höherem Diameter zu erstellen und zu messen, wie

[3] näher beschrieben in Kapitel 2.2.1

schnell die modifizierte Version von RIP zum ursprünglichen Protokoll funktioniert. Neben den üblichen Coldstart-Tests (Initiierung eines Netzwerks, in dem Router überhaupt erst die Topologie *lernen* müssen) werden auch Szenarien geprüft,

- die nur RIP/MTI Router beinhalten,
- in denen RIP/MTI Router nur an Schlüsselpositionen eingesetzt werden (stark frequentierte Netzwerkknotenpunkte),
- in denen an bestimmten Positionen Netze oder ganze Router ausfallen,
- in denen CTI-Situationen provoziert werden.

Die Varianten der Szenarien werden sowohl mit RIP als auch RIP/MTI durchgeführt um einen adequaten Vergleich zu erzielen. Das Ziel einer jeden wissenschaftlichen Arbeit ist die Wiederholbarkeit, entsprechend werden diese Tests auch oft wiederholt, um möglichst genau an ein Ergebnis heran zu kommen. Die Ergebnisse dieser Tests werden in Tabellen festgehalten und ausgewertet. Das Ergebnis soll zeigen, ob der Routing-Algorithmus in größeren Netzwerken im Vergleich zu RIP besser und/oder schneller arbeitet.

1.3 Vorgehensweise

Die Analyse geschieht über simulierte bzw. virtualisierte Netzwerke, bereitgestellt über die VNUML-Software. Mit jedem Netzwerk bzw. Netzausfallszenario werden die Router zunächst initiiert, die Netztopologie über das aktive Protokoll (RIP oder RIP/MTI) propagiert und nach einer Zeitmessung Tests durchgeführt. Router- oder Netzausfall zwingen die Router über das aktive Protokoll einen konvergenten Zustand wiederherzustellen, d.h. die ausgefallenen Routen über das unterbrochene Netz müssen eleminiert und alternativen eruiert werden. Die Zeit, die das Netz für die Konvergenzwiederherstellung benötigt, soll gemessen werden.
Darüber hinaus ist es hilfreich sich bei großen Netzen nicht nur auf einen Host-Rechner zu beschränken. Dafür wurde EDIV entwickelt, eine Scriptsammlung, mit der VNUML-Szenarien auf mehrere Hosts verteilt gestartet werden können. Die Informationen werden über die Hosts innerhalb von VLANs versendet, sodass die Simulation und die dafür geschriebene Szenario-Datei im XML-Format keinerlei Anpassung benötigt. Die AG Rechnernetze hat zur Analyse und Steuerung der VNUML-Szenarien das Programm XTPeer entwickelt, welches aber nicht mit EDIV zusammenarbeitet. Dieses Problem soll während der Arbeit gelöst werden. (siehe Kapitel 3). Aufgrund von Speicherproblemen im Zusammenhang mit großen simulierten Netzwerken werden Konvergenzzeiten nicht mit dem XTPeer, sondern mit kommandozeilenbasierte Programm Zimulator (Kapitel 2.5) gemessen.

1.4 Gang der Untersuchung

Die theoretischen Grundlagen des Themas RIP/MTI finden sich in voran gegangenen Diplomarbeiten der AG Rechnernetze der Universität Koblenz ([Boh08], [Kob09], [Sch99]) und bieten zusammen mit den Vorlesungen des Informatik-Hauptstudiums eine solide Grundlage für die Thematik und die Arbeit mit dem Analyse- und Steuerungstool XTPeer (Kapitel 2.4). Die Diplomarbeit von Marcel Jacobs [Jac10] geht näher auf das Tool „Zimulator“ ein und wie es funktioniert. Im Rahmen dieser Arbeit werden die Grundlagen kurzgefasst aufgeschrieben, um die Folgeschritte selbst nachvollziehen zu können. Danach werden Lösungsideen vorgestellt, wie eine VNUML-Simulation auch über EDIV gestartet von XTPeer ausgelesen werden kann.
Um diese Workarounds nicht für jede Simulation wiederholen zu müssen, beschäftigt sich diese Arbeit auch mit einer script-artigen Softwarelösung, um diese automatisiert für beliebige XML-Szenarien durchzuführen. Die Implementierung der Scripts wird in einem eigenen Kapitel beschrieben (Kapitel 4.3).
Ursprünglich war es das Ziel mithilfe dieses Workarounds tatsächliche Untersuchungen über den XTPeer durchzuführen, allerdings haben sich effizientere Alternativen ergeben. Zeitgleich mit dieser Arbeit wurde das bereits oben genannte Diagnose-Tool, der Zimulator, entwickelt, mit dem ebenfalls Untersuchungen durchgeführt werden können. Der eigentliche Unterschied ist die nicht-vorhandene GUI[4], es wird nur über die Kommandozeile gesteuert. Zimulator ist auch für große Netzwerke geeignet, da es nicht in Echtzeit die Daten der einzelnen Daemons[5] ausließt, sondern die Netzwerke über tcpdump[6] ausließt und nach Abschluss des Versuchs die Daten über einen Parser auswertet. Die statistische Auswertung geschieht nahezu automatisch, was den Teil der Konvergenzuntersuchung deutlich vereinfachen sollte. Das Tool wird im Kapitel 2.5 mit den wichtigsten Parametern vorgestellt und im Laufe der Arbeit verwendet, um Konvergenzzeiten zu bestimmen.
Die Untersuchung des Konvergenzverhaltens von RIP/MTI im Vergleich zu RIP in großen Netzwerken wird über ausgesuchte Testfälle geschehen, die in Varianten und hinreichend wiederholten Versuchen Aussage über die Performanz geben soll. Die Ergebnisse werden in approximierter Form analysiert und der Unterschied wird betrachtet. Geplant sind 4 unterschiedlich mächtige Netze, in denen diverse Ausfallszenarien getestet und ausgewertet werden sollen.
Um die Tests mittels XML-Szenarien nachzuvollziehen wird diese Arbeit die genutzten

[4] Graphical User Interface, eine graphische Benutzeroberfläche
[5] unter Linux ein im Hintergrund laufendes Programm, das bestimmte Dienste zur Verfügung stellt und vom Benutzer nur indirekt beeinflusst werden kann
Nähere Informationen @ http://en.wikipedia.org/wiki/Daemon_%28computer_software%29
[6] ein Linux-Programm, welches in der Lage ist den Datenverkehr einer bestimmten Netzwerkschnittstelle auszulesen und in eine Datei zu schreiben
Offizielle Webseite: http://www.tcpdump.org/

Werkzeuge sowie die Protokolle kurz vorstellen und im Kapitel 2 „Grundlagen“ die wichtigsten Funktionen und Fachbegriffe beschreiben. Im Kapitel 3 „Evaluation“ werden die Testfälle vorgestellt, sowie die benötigte Konfiguration der Diagnosetools. Die Untersuchungsergebnisse sollen tabelarisch aufgelistet werden. Anhand von Mittelwerten durch die Wiederholung von Ausfallszenarien werden zu jeder Testreihe Schlussfolgerungen gezogen, welches Protokoll aus welchem Grund ein besseres Ergebnis erbracht hat. Die Ergebnisse in Rohform sind auf einer DVD gespeichert, die der ausgedruckten Form dieser Arbeit beiliegt. In den Anhängen (Kapitel 6) der Arbeit stehen zudem Tutorials zur Installation von VNUML und EDIV auf einem Linux-System, mit vielen Hinweisen, die bei Problemen während der Installation hilfreich sein sollten. Das Tool „P.E.T“, welches EDIV-Simulationen in XTPeer zugänglich macht, erhält hier eine kurze Bedienungsanleitung, wie es zu benutzen ist. Wie es intern funktioniert, kann man in Kapitel 4 nachlesen.

Kapitel 2

Grundlagen

Zwar wurden Grundkenntnisse zur Thematik Netzwerk-Routing in nahezu jeder Qualifikationsarbeit der AG Rechnernetze vorgestellt, um aber nicht zwischen diesen Ausarbeitungen hin- und herzuspringen werden in diesem Kapitel noch einmal die wichtigsten Zusammenhänge und Begriffe vorgestellt.
Im Anhang dieser Arbeit befinden sich diverse Installationsanleitungen und Hilfen zu allen Programmen, die während dieser Arbeit benutzt wurden (siehe Kapitel 8).

2.1 Glossar

Dieses Kapitel beschreibt Fachbegriffe, die in dieser Arbeit öfter verwendet werden. Oft ist die Bedeutung auch nicht aus dem Kontext ersichtlich, es ist also empfehlenswert, dass falls man noch wenig Erfahrungen mit Netzwerkprotokollen, Rechnernetzen, virtuellen Maschinen (VNUML) oder Linux gemacht wurden, sich zumindest diese Seite vorher einzuverleiben. Im Fachjargon haben sich auch englische Wörter im Sprachgebrauch etabliert, die allerdings in dieser Arbeit nur verwendet werden wenn es sich nicht vermeiden lässt.

2.1.1 Netz

Ein **Netz** beschreibt einen IP-Adressbereich, der durch eine Subnetzmaske begrenzt ist (CIDR[1]). Es besteht aus verschiedenen Teilnehmern, denen jeweils eine eindeutige IP-Adresse zugewiesen ist. Ein **Netzwerk** ist die Menge aller Netze, die über verbindende Knotenpunkte die gesamte Topologie aufspannen.

[1] CIDR = Classless Inter-Domain Routing, einem Netz wird der IP-Adressbereich über die verkürzte Subnetzmaske zugewiesen, beispielsweise 192.168.0.0/16 entspricht der Subnetzmaske von 255.255.0.0. 16 entspricht 2x 8bit, also 2x 255. Dadurch wäre dieses Netz begrenzt auf die Adressen 192.168.0.0 bis 192.168.255.255 (genaueres kann man in der Fachliteratur nachlesen [Tan00])

2.1.2 Router

Router sind die Knotenpunkte zwischen den Netzen, zusätzlich zur Verbindung speichern sie Informationen die Lokalität einzelner Netze im Netzwerk in einer Tabelle, sogenannte **Routing Table Entries** (RTEs, engl.: Einträge der Routing-Tabelle). In diesen Einträgen steht, zu welchem Router ein Datenpaket gesendet werden muss, um dessen Ziel zu erreichen.

2.1.3 Routing Protokoll

Das **Routing-Protokoll**, Teil der Software, die auf dem Router implementiert ist, führt Algorithmen aus, die entscheiden, welche dieser Einträge präferiert werden. Es wird dabei unterschieden zwischen Distance Vector Protocols und Link-State Protocols. Distance Vector verfolgt die Strategie, dass die Anzahl der Router, die ein Datenpaket passieren muss, um das Zielnetz zu erreichen, entscheidend für die Dauer des Transfers ist. Bei RIP wird diese Router-Anzahl, die Distanz zum Zielnetz, wird als **Hopcount-Metrik** bezeichnet, Routen (gespeicherte Informationen in den RTEs) mit der geringsten Metrik werden bevorzugt. Link-State Protokolle fluten die gesammte Topologie mit Routing-Paketen, um Informationen über diese zu erlangen. Jeder Router hat dadurch eine Datenbank mit Aufbauinformationen über Netzwerk, während bei Distance Vector Protokollen nur die direkten Nachbarn bekannt sind. Manche Link-State-Protokolle bewerten die Datenlast der Leitung zwischen den Routern und präferieren Datenwege mit der geringsten Netzlast.

2.1.4 Interface

Jeder Router besitzt ein oder mehrere Interfaces (engl.: Schnittstelle), die als Verbindung zwischen Netz und Router dienen. In der Regel handelt es sich dabei um einen physikalischen Netzwerkanschluss, unter Linux eth0, eth1 etc. genannt. Ein Interface sendet und empfänget Daten über ein bestimmtes Netz.

2.1.5 Konvergenz

Unter Konvergenz innerhalb eines Netzwerks versteht man die Fähigkeit eines Routing-Verfahrens nach einer Netzänderung möglichst schnell wieder einen stabilen Betriebszustand herzustellen. Das bedeutet, dass jeder Router eine funktionierende Route zu allen erreichbaren Netzen im Routing Table gespeichert haben muss. Sinn und Zweck von Routing Protokollen ist die Sicherstellung, dass ein Netzwerk stabil oder auch konvergent bleibt bzw. im Falle einer Fehlfunktion (z.b. eines Netzes oder Routers) die betroffenen Geräte als unerreichbar deklariert, sowie die Findung der kürzesten Pfade zwischen den Routern. Die Zeit zwischen einem Netz- oder Routerausfall und einem neuem stabilen Zustand wird

Konvergenzzeit genannt. Die Zeit zwischen der erstmaligen Netzinitialisierung und dem ersten stabilen Zustand wird **Coldstart-Konvergenz** genannt.

2.1.6 Diameter

Innerhalb einer Netztoplogie, dem Graphen eines Netzwerks (bestehend aus Routern (Knoten), Netzen (Kanten)), wird der längste kürzeste Pfad zwischen zwei Routern Diameter genannt.

2.1.7 Testfall

Eine vorgegebene Abfolge von Ereignissen, die während der Messung der Konvergenzzeit einer Topologie durchgeführt werden. Ein Testfall sollte wiederholbar sein, um die Varianz der Messergebnisse zu approximieren.

2.1.8 Szenario

Die Simulation einer Netzwerk-Topologie mittels VNUML (siehe 2.3) wird als Szenario bezeichnet. Definiert über eine XML-Datei werden Router als UML-Rechner (UML = User Mode Linux) gestartet, die über virtuelle Netzwerk-Interfaces kommunizieren. Man nennt die XML-Datei deshalb auch häufig selbst Szenario, da es die zu simulierenden Eigenschaften zusammenfasst.

2.1.9 Virtuelle Maschine

Simulationen einer Netzwerktopologie werden über VNUML realisiert, die Router innerhalb dieser Simulation werden auch Virtuelle Maschinen (**VM**) genannt, da sie einen echten Router innerhalb der der Simulation virtuell darstellen.

2.2 Das Distanzvektorprotokoll RIP

Das Routing Information Protocol (RIP) ist ein dynamisches Routing Protocol, welches in Local (LAN) und Wide Area Networks (WAN) verwendet wird. Es basiert auf dem Bellman-Ford-Algorithmus von 1957. RIP ist als ein Interior Gateway Protocol (IGP) klassifiziert, es wird also innerhalb von autonomen Systemen[2] für den Austausch von Routen-Informationen verwendet und ist auf dem User Datagram Protocol (UDP) implementiert. Der für RIP reservierte Port ist 520. Das Protokoll nutzt einen Distanzvektoralgorithmus,

[2]Ansammlung von IP-Netzen, die über ein internes Routing Protocol (IGP) verbunden sind
Wikipedia-Link: http://en.wikipedia.org/wiki/Autonomous_system_%28Internet%29

welcher in der RFC 1058[3] (1988) [Hed88] definiert wurde. Die Dynamik entsteht über der von der Topologie abhängigen Wahl der kürzesten Wege, die in die Weiterleitungstabelle der Router eingetragen werden. Ändert sich die Topologie während des laufenden Betriebs, werden diese Einträge vom Routing Protokoll modifiziert, um neue Routen einzutragen (vorausgesetzt ist ein alternativer Pfad zu jedem Router, also Redundanz). Das Protokoll wurde einige male erweitert, letztendlich bildeten die Neuerungen die Protokollversion 2 (RIPv2, [Mal98]). Beide Versionen sind heute in Benutzung, allerdings gelten sie als technisch veraltet, da neue Techniken wie Open Shortest Path First (OSPF) oder das OSI-Protokoll IS-IS bessere Leistungen erbingen. RIP wurde auch für IPv6-Netzwerke adaptiert. Dieser Standard nennt sich RIPng (RIP next generation, RFC 2080 (1997)).
Als Distanzvektor Protokoll sieht RIP den *Hop Count* als Routing Metrik. Jeder Router, der zum Erreichen eines Zielnetzes von einem Paket passiert wird, gilt als „Hop“. Die Standard-Einstellungen des Protokolls sind:

- update timer = 30sek (Periodische Updates zu allen Nachbarn)
- invalid timer = 180sek (Route ungültig deklarieren, Metrik = 16)
- flushdown timer = 120sek (Route verwerfen, auch Timeout genannt)

Ursprünglich schickte jeder RIP-Router alle 30 Sekunden vollständige Updates an seine direkten Nachbarn, Ausnahme sind Einträge der Router, von dem der sendende Router Informationen „erlernt“ hat (-> Split Horizon, [Hed88, Punkt 2.2.1]), kleine Netze konnten diese regelmäßien Updates stämmen, bei großen Netzen wurde diese intervallartige Updateflut zu einem Problem, da es massiven Datenverkehr produzierte. Moderne modifizierte RIP-Versionen haben eine geringfügige Varianz in den Updatezeiten, sodass unterschiedliche Router die Updates zu unterschiedlichen Zeitpunkten versenden.
Um zu verhindern, dass falsche Routing-Informationen propagiert werden, implementiert RIP spezielle Funktionen wie Split Horizon und Poison Reverse. Poison Reverse tritt ein, wenn ein Nachbar eines ausgefallenen Routers feststellt, dass der Weg dorthin nicht mehr verfügbar ist. Diese Feststellung geschieht durch den Ablauf des entsprechenden Timers, in der Routing-Tabelle werden Routen über das unverfügbare Netz mit der Metrik 16 versehen, welches bei RIP die Bedeutung „infinite“ (engl. unendlich) hat. Das bedeutet allerdings auch, dass Netzwerke mit einem größerem Diameter als 16, also der längste kürzeste Weg zwischen zwei Routern, zumindest mit klassischem RIP unmöglich werden. EIGRP (distance vector protocol), OSPF (linkstate protocol) oder auch IS-IS (linkstate protocol) skalieren besser mit größeren Netzwerken und konvergieren schneller. Der Vorteil in RIP ist die einfache Konfiguration, denn das Protokoll benötigt im Gegensatz zu anderen

[3] „Request for Comments“, veröffentlicht von der Internet Engineering Task Force (IETF). Bei RFCs handelt es sich um Texte, die Methoden, Verhaltensweisen, Forschung und Innovationen beschreiben, die für das Internet und internetverbundene Systeme Anwendung finden.

Protokollen keine routerspezifischen Parameter, es ist selbstorganisierend. Zusätzlich dazu ist der Kommunikationsoverhead, die Hardware-Belastung, sowie die Komplexität des Routingvorgangs niedrig und die Verbreitung richtiger Routing-Informationen geschieht zügig. Nachteile von RIP sind die Verbreitung von Routing-Informationen ausgefallener oder temporär nicht erreichbarer Netze, also Falschnachrichten, und die Bewertung von Routen, über die Hop-Count Metrik, kennzeichnet zwar die kürzeste Route, jedoch nicht zwangsläufig die schnellste.

2.2.1 Count-to-Infinity

Die langsame Verbreitung von Ausfallnachrichten führt unter Umständen auch zum sogenannten CTI-Problem (Count-to-Infinity). Es bezeichnet Fälle, in denen routing loops entstehen. Routing Informationen, die asynchron zwischen den Nachbarn weitergereicht werden, können dazu führen, dass Router innerhalb der Schleife Einträge zu ihren Nachbarn schicken, die nicht korrekt sind. So kann es sein, dass eine Route jeweils immer über den nächsten „Hop“ geht, den die Router fälschlicherweise ermitteln, und die Falschinformation bis zum Infinity-Wert weitergibt, bevor die Route verworfen wird. Die Konvergenz des Netzwerks dauert in diesem Fall sehr lange. Hier ein Beispielszenario:

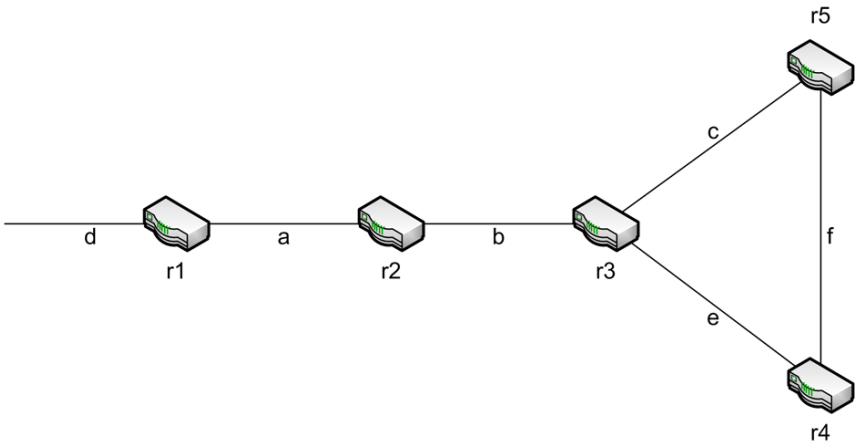


Abbildung 2.1: Topologie des Y-Szenario

Die Y-Topologie ist anfällig für CTI-Situationen, wenn RIP eingesetzt wird. Nach einem Coldstart und der resultierenden Konvergenz sehen die Routing-Tabellen für alle Router ist in Tabelle 2.1 zu sehen. Jeder Tabelleneintrag ist zu lesen als: Netzname - Metrik zu diesem Netz - gelernt von Router XX, z.B. (blau markiert): Netz f ist 4 hops entfernt, nächster Knoten zu diesem Netz ist R2

Fällt in diesem konvergenten Zustand der Router **R1** aus, ist das Netz **d** für keinen anderen

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 2 | r1	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 3 | r2	a | 3 | r3	d | 4 | r3
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 4 | r4	e | 2 | r4

Tabelle 2.1: CTI: Routing-Tabellen des Y-Szenarios nach einem Coldstart

Router mehr zu erreichen[4]. Der Nachbar-Router R2 registriert nach einer festgelegten Zeit, nämlich 180 Sekunden, dass R1 nicht mehr erreichbar ist, weil keine Periodischen Updates eintreffen, und setzt die Metrik von 1 auf 16, die RIP-Metrik für „Infinity“. (Tabelle 2.2).

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 16 | r1	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 3 | r2	a | 3 | r3	d | 4 | r3
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 4 | r4	e | 2 | r4

Tabelle 2.2: CTI: unmittelbar nach dem Ausfall von R1

Es bestünde die Möglichkeit eines Two-Hop-Loops zwischen R2 und R3, da R2 eine Route zu Netz d mit Metrik 16 kennt, R3 aber den Weg noch mit einer Metrik von 3 gespeichert hat. Beide Router würden sich jeweils das bessere Update zuschicken. Dies kommt jedoch nicht vor, da durch Split Horizon keine Routeninformationen an Router weitergegeben werden, von denen diese auch empfangen wurden. RIP sieht bei der Registrierung eines Ausfalls ein **Triggered Update** vor, das die neue Metrik des Netzes d, nämlich Infinity, an alle Router weitergibt, damit wieder ein konvergenter Zustand hergestellt wird. (Tabelle 2.3).

Die Probleme entstehen, sobald eines dieser Triggered Updates stark verzögert oder überhaupt nicht empfangen werden kann. Paketverlust in Netzen ist nie ausgeschlossen, eine Schwäche von RIP ist es jedoch Updates nicht zu bestätigen[5]. Erreicht also in diesem Szenario das Triggered Update für Netz d einen Router nicht, beispielsweise Router 4, entsteht

[4] in der Graphentheorie nennt man Router wie R1, also Router ohne redundante Verbindung, auch Blatt

[5] dies wird gleichzeitig auch als Stärke aufgeführt, da die Updates deutlich schneller verarbeitet werden können

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 16 | r1	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 16 | r2	a | 3 | r3	d | 16 | r3
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 16 | r4	e | 2 | r4

Tabelle 2.3: CTI: neuer (optimaler) konvergenter Zustand nach einem Ausfall

nach Abschluss des Updates eine Anomalie im Netzwerk -> Tabelle 2.4).

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 16 | r1	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 16 | r2	a | 3 | r3	d | 16 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 4 | r3	e | 2 | r4

Tabelle 2.4: CTI: Zustand nach einem Ausfall, Triggered Update zu Router R4 geht verloren

Aufgrund nicht vorhandener Bestätigungsnachrichten verbreitet Router R4 nach der vordefinierten Update-Zeitspanne nun Falschnachrichten. Zwar sendet Router R5 die Metrik 16 ebenfalls zu R4, da R4 aber aufgrund des verlorenen Datenpakets eine bessere Route gespeichert hat, nämlich eine Metrik von 4 über den Router R3 (orange markiert in Tabelle 2.4), wird die richtige Information verworfen, denn sie ist ja „schlechter“ als die gespeicherte Route. Die richtige Metrik kann nur von R3 aktualisiert werden. Router R4 verbreitet nach Ablauf des Update-Timers neue Informationen, dass Routen zum eigentlich ausgefallenen Netz mit einer geringeren Metrik als 16 bekannt sind, alle Router werden diese Information auch fälschlicherweise annehmen. In Tabelle 2.5 bis Tabelle 2.7 wird dies nun sichtbar, falsche Informationen sind auch hier orange markiert:

Erklärung: Der nicht korrigierte Routing-Eintrag im Router R4 für das nicht erreichbare Netz d erzeugt im der gesamten Netzwerk für einen CTI. R4 kennt dank des verlorenen Updates weiterhin eine Route zum Netz d über den Router R3. R5 sendet zwar die richtige Metrik, nämlich 16, diese wird aber verworfen, da R4 noch eine bessere Route in der Tabelle stehen hat.
Die Einträge des Routers R5 in Tabelle 2.7 sind der Beginn des CTI-Effekts. Routen, die

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 16 | r1	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 16 | r2	a | 3 | r3	d | 5 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 4 | r3	e | 2 | r4

Tabelle 2.5: CTI: Ausbreitung der Falschnachrichten (1)

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 16 | r1	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 6 | r5	a | 3 | r3	d | 5 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 4 | r3	e | 2 | r4

Tabelle 2.6: CTI: Ausbreitung der Falschnachrichten (2)

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 7 | r3	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 6 | r5	a | 3 | r3	d | 5 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 7 | r3	e | 2 | r4

Tabelle 2.7: CTI: Ausbreitung der Falschnachrichten (3)

von den direkten Nachbarn empfangen werden, werden auch zeitgleich von diesen Nachbarn aktualisiert. Die orange markierten Einträge werden also nun von 7 bis 16 hochzählen, also bis „Infinity“, gezählt.

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 7 | r3	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 6 | r5	a | 3 | r3	d | 8 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 7 | r3	e | 2 | r4

Tabelle 2.8: CTI: Ausbreitung der Falschnachrichten (4)

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 7 | r3	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 9 | r5	a | 3 | r3	d | 8 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 7 | r3	e | 2 | r4

Tabelle 2.9: CTI: Ausbreitung der Falschnachrichten (5)

R1	R2	R3	R4	R5
d | 1 | self	a | 1 | self	b | 1 | self	e | 1 | self	c | 1 | self
a | 1 | self	b | 1 | self	c | 1 | self	f | 1 | self	f | 1 | self
b | 2 | r2	d | 10 | r3	e | 1 | self	b | 2 | r3	a | 3 | r3
f | 4 | r2	f | 3 | r3	a | 2 | r2	c | 2 | r3	b | 2 | r3
c | 3 | r2	c | 2 | r3	d | 9 | r5	a | 3 | r3	d | 8 | r4
e | 3 | r2	e | 2 | r3	f | 2 | r5	d | 10 | r3	e | 2 | r4

Tabelle 2.10: CTI: Ausbreitung der Falschnachrichten (6)

Treten solche Situationen auf, dauert die Konvergenz sehr lange. Alle Router, die Teil dieser Schleife sind, zählen allmählich auf den eingestellten Infinity-Wert, denn die Routenfindung „dreht sich im Kreise“. RIP hat jedoch mit RMTI nun eine effektive Schleifenerkennung, das Ziel besteht darin CTI-Situationen komplett zu verhindern (-> Kapitel 2.2.5).

2.2.2 RIPv1

Das ursprüngliche RIP [Hed88] nutzt Classful Routing[6], periodische Routing Updates führen keine Subnet-Informationen mit sich und auch eine variable Länge der Subnet Masks wird nicht berücksichtigt. Dadurch ist es nicht möglich unterschiedlich große Subnets in einem Netzwerk zu haben. Jedes Netz muss dieselbe Subnet Mask haben, was die Freiheit im Aufbau des Netzwerks stark limitiert. RIPv1 unterstützt keine Router-Autentifizierung, somit ist es anfällig für absichtlich eingespeißte Falsch-Updates oder andere Angriffe.

2.2.3 RIPv2

RIP Version 2 [Mal98] wurde 1993 entwickelt und 1998 standardisiert, um einige der Mängel in RIPv1 zu beheben. RIPv2 unterstützt Classless Routing[7]. Um abwärtskompatibel zu RIPv1 zu bleiben wird der maximale Hop-Count von 15 beibehalten. Um unnötige Last auf Rechner im Netzwerk, die nicht routen, zu reduzieren, wird bei RIPv2 kein Broadcast zum versenden von Updates genutzt, wie es bei RIPv1 der Fall war. Stattdessen wird die Routing Tabelle eines Routers über Multicast an alle angrenzenden Nachbarrouter geschickt.[8] Die MD5-Autentifizierung für RIPv2 wurde 1997 eingeführt, ebenfalls neu in Version 2 sind die „Routing Tags“, welche es Routern ermöglichst die Informationen netzwerkinterner Routen von externen zu unterscheiden.

2.2.4 RIPng

RIPng (RIP next generation) [Mal97] ist eine Erweiterung von RIPv2, welche das IPv6 unterstützt, das Internetprotokoll, welches IPv4 irgendwann einmal ersetzen soll. RIPng unterstützt keine RIPv1 Update Autentifizierung, wie es noch bei RIPv2 der Fall war. Da RIPng über IPv6 funktioniert, soll die Integrität von Routerinformationen über den

[6] Routing Protokolle, die *classful* arbeiten, beinhalten keine Subnet Mask Informationen in ihren Routing Updates. Das macht sie unbrauchbar für hierarchieartige Adressierung, welche eine VLSM (Variable Length Subnet Mask, engl. variable Länge der Subnetz-Maske) und ein nicht zusammenhängendes Netz benötigt.

[7] Routing Protokolle, die *classless* arbeiten, beinhalten Subnet Mask Informationen in ihren Routing Updates. Auf diese Weise erweitern sie das Standard-Klassen-Schema von Klasse-A, -B oder -C Netzen, denn mit Hilfe einer Subnetmask können kleinere Netze innerhalb des physikalischen Netzes definiert werden. Auch identifizieren sie die Klasse des Netzes, die Subnet Mask kann zusammengefasst in der CIDR-Notation (Classless Inter-Domain Routing) auf bestimmte Teile des Netzes weisen, diese flexible Vergabe von Masken an IP-Bereiche macht es auch möglich mehrere Netzwerke in einem einzigen Eintrag in der Routing Tabelle zu gruppieren, was den Overhead des Protokolls reduziert.

[8] Versenden von Informationen ist auf drei Arten definiert:
Broadcast = Versenden von Paketen an alle Teilnehmer des Netzwerks
Multicast = Versenden von Paketen an mehrere Teilnehmer des Netzwerks
Unicast = Versenden von Paketen an einen Teilnehmer des Netzwerks

IP Authentification Header und die IP Encapsulating Security Payload (ESP) gesichert werden. In RIPv2 ist der Next-Hop, also der Router, der in einer Route von A nach B in der Routing Tabelle eingetragen ist, teil des Routing Table Entry (RTE). Anhand der Spezifikation von RIPng würde diese Technik die Länge jedes Eintrags in der Tabelle stark vergrößern, deshalb werden die next-Hops in einem eigenen Eintrag festgehalten, der mit Einschränkungen für alle RTEs gilt.

2.2.5 RMTI

Ehemals RIP/MTI (RIP with minimal topology information), wird es seit Kurzem der Einfachheit halber nochmals abgekürzt zu RMTI. Die Änderungen zielen hauptsächlich auf die Skalierbarkeit, denn die Hauptschwachpunkte von RIP waren der maximale Hop-Count von 15 und die oberflächliche Schleifenerkennung. Diese sowie die Implementierung von Split Horizon oder Poison Reverse reduzieren nur die Wahrscheinlichkeit einer Count-To-Infinity-Situation, mit RMTI wurden Algorithmen eingeführt, die Schleifen erkennen und verhindern sollen. RMTI erkennt Simple- und Source Loops, Informationen darüber werden in den mrpm- und msilm[9]-Tabellen des Routing-Daemons gespeichert, um neu erkannte Schleifen von Router zum Subnetz effizient mit bekannten Schleifen vergleichen zu können. Auf diese Weise erkennt RMTI Simple Loops und kann mithilfe der erhobenen Daten Source Loops vermeiden.

Simple Loops

Ein Simple Loop (Abbildung 2.3) ist ein Pfad, der an einem bestimmten Interface eines Routers beginnt und an einem anderen Interface desselben Routers endet, ohne dass ein Router des Pfads öfter als einmal passiert wurde. RMTI implementiert einen Algorithmus, der eine topologische Routing-Schleife erkennt, indem dieser die Metrik mit dem Schleifenumfang vergleicht. Betrachtet man sich also erneut das Beispiel des Y-Szenarios, so ist erkennbar, dass es einen topologischen Loop von drei Routern, nämlich R3, R4 und R5, gibt, in Kapitel 2.2.1 wird auch das Ausmaß einer CTI-Situation erklärt. Netzwerkschleifen, die aufgrund der Topologie unvermeidbar sind, weil sie eine logische (Alternativ-)Route darstellen, werden Simple Loops genannt.

In Abbildung 2.3 gibt es zum Netz f zwei unterschiedliche Wege, nämlich über die Router R4 und R5. Sollte eine Route ausfallen, wird vom RIP-Algorithmus die Alternative in die Routingtabellen eingetragen. Diese Alternativen werden, wenn benötigt, erst nach dem Ausfall festgestellt, da die Routingtabelleneinträge keine Alternativ-Einträge vorsehen. Wird ein Netz- oder Router-Ausfall durch den entsprechenden RIP-Timer festgestellt,

[9] msilm = minimal Simple Loop metric
mrpm = minimal return path metric

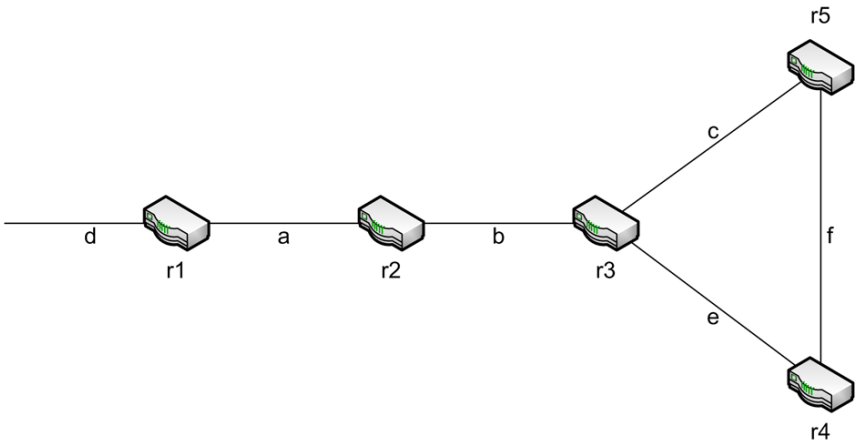


Abbildung 2.2: Topologie des Y-Szenario

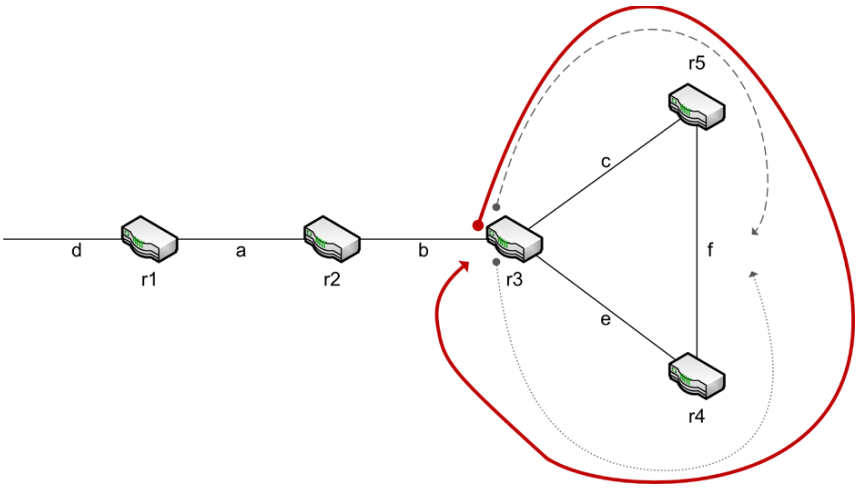


Abbildung 2.3: Topologie des Y-Szenario inkl. Simple Loops

wird ein Triggered Update propagiert, dass die Route auf Infinity (im Falle von RIP = 16) setzt, sollte das ausgefallene Element in den Routing-Einträgen zu finden sein. Die bessere Alternativ-Route wird schließlich über periodische Updates verteilt und eingetragen. In größeren Netzwerken können durchaus auch mehre Alternativrouten bestehen, die Metrik muss dabei nicht identisch sein[10].
In Netzwerken wird der Simple Loop für RMTI nutzbar gemacht, um den Umfang von Schleifen zu berechnen. Im Y-Szenario erreichen den Router R3 mehrere Updates, über Netz c und e. Anhand der propagierten Wege aus den zwei Richtungen wird der Pfad durch die Konkatenation beider Teilpfade gebildet. Sei Interface A mit Netz c und Inferface B mit Netz e verbunden, so wird die Metrik zwischen Interface A bzw B und Netz f von RMTI berechnet. In diesem Beispiel wären die Teilergebnisse, die **Pfade einer Netzwerkschleife** genannt, $P_A^{(R3,f)}$ und $P_B^{(R3,f)}$. Die Summe daraus ist $P_{(A,B)}^{(R3,f,R3)}$ - 1. Man rechnet -1, da der Router sich sonst selbst zur Schleife dazuzählen würde.

Allgemeine Formel für die Berechnung des Simple-Loop-Pfades[11]:
$P_{(A,B)}^{(i,n,i)} = P_A^{(i,n)} + P_N^{(i,n)}$ - 1

Anhand des Beispiels 2.3:[12]

Pfad 1: $P_A^{(R3,f)}$ -> $m_A^{(R3,f)} = 2$
Pfad 2: $P_B^{(R3,f)}$ -> $m_B^{(R3,f)} = 2$

Simple Loop: $P_{A,B}^{(R3,f,R3)}$ -> $m_{A,B}^{(R3,f,R3)} = 3$

Der Loop wird durch Addition der beiden Metriken zwischen Interface und Zielnetz berechnet. Im aktuellen Beispiel werden die Werte $m_A^{(R3,f)}$ und $m_B^{(R3,f)}$ zu $m_{(A,B)}^{(R3,f,R3)}$ - 1 zusammengefasst, -1 deshalb, da auch hier der Router sich selbst zur Schleife zählen würde.

Source Loops

Ein Source Loop ist ein Pfad, der an einem bestimmten Interface eines Routers beginnt und an einem anderen Interface desselben Routers endet, ňachdem ein Router des Pfads

[10] Ein Spezialfall ist ein Full-Mesh Netzwerk, bei dem jeder Router mit jedem anderen Router verbunden ist, es gibt also [Routeranzahl]-1 Alternativen. In einem Full-Mesh entstehen keine CTI-Situationen, denn die jeweils höchste Metrik ist 1.
[11] i=Router // n=entferntestes Netz der Schleife // A,B=Interfaces von Router i
[12] Loop gekennzeichnet durch obere grau gestrichelte Linie
Loop gekennzeichnet durch untere grau gepunktete Linie
Loop gekennzeichnet durch rote Linie

öfter als einmal passiert wurde. Simple Loops sind ein Indiz für topologische Schleifen, Schleifen die zu erkennen sind, man kann sie aber nicht vermeiden. Es gibt also mindestens eine alternative Route zu Subnetzen innerhalb des Loops. Source Loops sind hingegen ein Indiz für einen Routing Loop, den man in optimalen Netzen unbedingt vermeiden will. Ein Beispiel zu Source Loops ist in Abbildung 2.4 zu sehen:

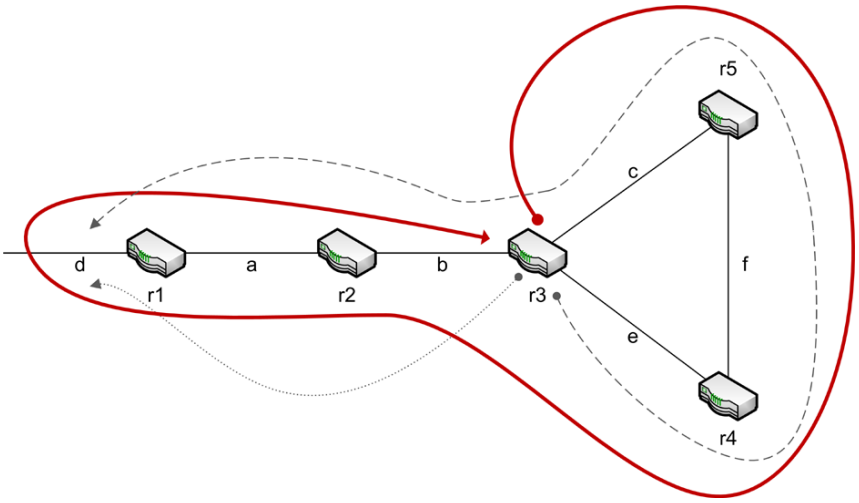


Abbildung 2.4: Topologie des Y-Szenario inkl. Source Loops

Berechnungen der Schleifenwerte:
Pfad 1: $P_A^{(R3,d)}$ -> $m_A^{(R3,d)} = 5$
Pfad 2: $P_C^{(R3,d)}$ -> $m_C^{(R3,d)} = 2$
Source Loop: $P_{A,C}^{(R3,d,R3)}$ -> $m_{A,C}^{(R3,d,R3)} = 6$

Es muss festgestellt werden, ob ein vom Routing-Algorithmus entdeckter Simple-Loop-Pfad $P_{A,B}^{(i,d,i)}$ kein Source Loop ist. Um Source Loops zu vermeiden, wird die Metrik aller bekannten Simple Loops verglichen und der Loop mit der niedrigsten Metrik, die MSILM (minimal simple loop metric), in eine interne Tabelle eingetragen.

$msilm_{A,B}^{i} = \min(silm_{A,B}^{i,d,i})$ für alle Subnetze d

Zusätzlich wird die mrpm (minimal return path metric) gespeichert, die man von der msilm-Tabelle ableiten kann.

$mrpm_{A,B}^{i} = \min(msilm_{A,B}^{i})$ für alle benachbarten Schnittstellen, mit A != B von Router i

Der Vergleich mit der Summe der kürzeste Metrik zu Subnetzen und der kürzesten Metrik von Subnetzen zum Router mit dem Pfad, den man gerade betrachtet, nennt man auch **Simple Loop Test**. Wenn:
$m_A^{i,d} > mrpm_A^i + m_B^{i,d}$

eine wahre Aussage ist, dann ist $P_{A,B}^{i,d,i}$ ein Simple Loop, aber kein Source Loop (weil einfach zu kurz). RMTI unterscheidet bei diesen Tests zwischen zwei exklusiv einsetzbaren Modi, dem Strict-Modus und dem Careful-Modus. Strict verwirft Routing Informationen, die diesen Test nicht bestehen. In wenigen Fällen kommt es jedoch zu Situationen, in denen auch richtige Einträge verworfen würden. Der Careful-Modus markiert der Algorithmus die ausgefallene Route, wenn die angebotene Alternativ-Route den Simple-Loop-Test nicht besteht. Zusätzlich löst der fehlgeschlagene Test eine Updatenachricht aus, in der der Infinity-Wert zum betroffenen Netz an die Nachbar-Router gesendet wird. „Eingeschlichene“ Source-Loops werden somit überschrieben. Ein Request-Timer wird gestartet, währenddessen werden Updates zum ausgefallenen Netz verworfen. Erst Nach Ablauf dieses Timers werden neue Routing-Informationen akzeptiert. Aufgrund der RIP-Eigenschaft, dass sich schlechte Nachrichten nur sehr langsam verteilen, ist diese Maßnahme notwendig um aufgetretene Source Loops möglichst schnell zu eliminieren. Dies hat vor allem dann Anwendungsmöglichkeiten, wenn RMTI- und RIP-Router gemeinsam in einem Netzwerk arbeiten sollen. Es gibt keinen eigenständigen „Source Loop Test“, durch den Simple Loop Test werden Source Loops allerdings ausgeschlossen. Die Situationen, die nicht durch den Simple Loop Test abgedeckt sind, eliminiert die Split Horizon Mechanik [FB10], [Kob09].

2.3 VNUML

Bei VNUML (Virtual Network User Mode Linux) handelt es sich um eine Sammlung von Scripts (perlscript), mit denen effizient Netzwerksimulationen und -szenarien definiert und getestet werden können. Die Scripts sind nur unter einem Linux Betriebssystem lauffähig und benötigen diverse andere Tools, um zu funktionieren (siehe 8.1). Die Virtualisierung der Netzwerkelemente basiert auf User Mode Linux (UML) vgl. [Mad08]. Mit VNUML können Linux-Netzwerk-Szenarien simuliert werden, die Vernetzung zwischen den virtuellen Routern geschieht über TUN/TAP-Devices. TUN/TAP ist ein Kernelmodul, TUN simuliert Layer-3-Geräte (IP) bzw. generiert virtuelle Point-to-Point Interfaces, während TAP Layer-2-Geräte (Ethernet) bereitstellt, bzw. virtuelle Ethernet Interfaces generiert. Die Kommunikation über echte Netzwerke und Netzwerkschnittstellen übernehmen Virtual Bridges, die die virtuellen Schnittstellen an reale koppeln. Die Kommunikation zwischen virtueller Maschine und dem Host (dem Rechner, der die Ressourcen für die virtuellen Maschinen bereitstellt), geschieht über SSH.
Das Dateisystem und der Kernel sind für die virtuellen Maschinen (abgekürzt: VM) frei wählbar, entsprechend ist es auch einfach Anpassungen an der Konfiguration vorzunehmen. Die Dateisysteme basieren auf der Copy-on-Write-Technik (COW). Aus einem Dateisystem und einem Kernel werden über VNUML beliebig viele Instanzen einer virtuellen Maschine erzeugt, denen begrenzt durch die Rechenleistung Ressourcen zugewiesen werden können. Die Instanzen werden als einzelner Prozess im RAM gespeichert.
Über die Quagga Routing Suite, einer Routing Software Sammlung (bestehend aus dem Daemon 'zebra' und implementierten Protokollen für OSPF, RIP und anderen) werden Protokoll-Daemons bereitgestellt (siehe auch [OSP09b]), mit denen die virtuellen Maschinen mit diesen Protokollen über die virtuellen Netzwerke kommunizieren, die Maschinen immitieren in diesem Fall das Verhalten von Internetroutern. Welches Protokoll die Maschinen nutzen, ist Entscheidung des Benutzers. Es werden sowohl von der AG Rechnernetze als auch von der VNUML Webseite fertige Images mit der Routing Suite bereitgestellt, die Installation entfällt also. Diese Arbeit basiert auf folgender Software:

- VNUML v1.8.9
- linux-2.6.18.1-bb2-xt-4m (UML-Kernel)
- ripmti-64-vnuml18.img (Dateisystem mit implementierter XTPeer-Schnittstelle)
- ripmti-hello.img (Dateisystem ohne XTPeer-Schnittstelle)
- rmti-0.99.16.img (Dateisystem ohne XTPeer-Schnittstelle, bis dato aktuellste Version des RMTI-Protokolls)

Wie diese Software zu installieren ist, ist im Anhang (siehe Kapitel 8.1 ausführlich beschrieben.
Die Netztopologie der simulierten Router wird über ein VNUML-Szenario, welches im XML-Format [ORe98] geschrieben werden muss, definiert. In dieser Szenario-Datei wird festgelegt, welchen UML-Kernel, welches Dateisystem und welcher Netzwerkadressbereich verwendet werden soll, desweiteren werden für jeden simulierten Router jeweils Netzwerkschnittstellen, start- und stop-Scriptbefehle für die Steuerung der notwendigen Daemons und der Pfad für Konfigurationsdateien angegeben. Mit Dateisystem ist im Falle von VNUML ein Image gemeint, welches sich in ein Linux-System einbinden lässt (mount). Es beinhaltet die notwendigen Kernbefehle zur Nutzung des Betriebssystems sowie die für VNUML geeigneten Dienste. Es wurde von der AG Rechnernetze angepasst, so befindet sich für die Quagga Routing Suite ein zusätzlich implementiertes MTI-Protokoll (RIP/MTI), gesteuert über einen Daemon namens 'ripd'. Dieser unterstützt auch die herkömmlichen RIP-Funktionalitäten, was vorteilhaft ist, um über eine VNUML-Simulation beide Protokolle zu vergleichen - wie zwischen RIP und RMTI unterschieden wird steht in Kapitel 3.2.3. Das Verhalten, also das Lernen, Verwerfen und die Weitergabe von Routen, geschieht ganz im Sinne des jeweiligen Routing-Algorithmus, die Informationen lassen sich über das Diagnosetool XTPeer auslesen.
VNUML setzt die wohldefinierten XML-Dateien über das Script vnumlparser.pl in eine laufende Simulation um. Unterschiedliche Parameter steuern das Programm. In der XML-Datei ist stehts der Tag <!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">, mit Hilfe dieser DOCTYPE Definition prüft der Parser die XML-Datei auf syntaktische Korrektheit, bevor virtuelle Maschinen initiiert werden.

```
[root@workstation ~]\$ vnumlparser.pl -t vnuml-szenario.xml -vBZ
```

Code 2.1: Start eines VNUML-Szenarios

Der Parameter `-t` ist der Start-Parameter für VNUML, das darauf folgende Argument, in diesem Fall vnuml-szenario.xml, muss der Dateiname der Simulations-Datei sein. Die Parameter dahinter sind optional, sind aber hilfreich. Der Parameter `-v` aktiviert den Verbose-Modus von VNUML, es werden also mehr Informationen ausgegeben, was das Script während des Startvorgangs leistet, `-B` lässt den Parser den Blocking-Modus verwenden. Dieser Modus lässt alle Maschinen direkt nacheinander starten, ohne dass das Script auf jeweils eine Rückantwort des SSH-Servers der virtuellen Maschine wartet, bevor es die nächste startet. Dieser Parameter ist vor allem bei Simulationen mit mehr als 10 Routern empfehlenswert. Der Paramenter `-Z` veranlasst den vnumlparser dazu keine VUMLization des Dateisystems zu verwenden, es wird primär als Lösung angesehen VNUML auf 64-Bit-Rechnern ohne Absturz starten zu lassen. `-Z` funktioniert nur im Zusammenhang mit dem `-t`-Parameter. `-u root` veranlasst das Betriebssystem, dass die VNUML Prozesse

als root-user gestartet werden. Ubuntu-Linux-Systeme besitzen keinen richtigen root-user, man kann administrative Prozesse nur mit `sudo` starten, der `-u`-Parameter ist unter Ubuntu/Kubuntu/Xubuntu deshalb Pflicht.
Befehle, die in der Szenario-Datei über den EXEC-Tag vordefiniert werden, können vom Host aus mit `-x` aufgerufen werden.

```
[root@workstation ~]\$ vnumlparser.pl -x start@vnuml-szenario.xml -vB
```

Code 2.2: Aufruf eines Scenario-Kommandos

Es können auch mehrere Befehle durch das Beispiel-Kommando „start“ ausgeführt werden. In XML-Dateien, die von Zimulator generiert werden, startet der x-Parameter „start“ lediglich die Zebra-Daemons und „rip“ den RIP-Daemon. Wenn alle Untersuchungen oder Tests abgeschlossen sind, sollte man das Szenario mit `-P` wieder herunterfahren, um Ressourcen freizugeben oder um andere Szenarien zu starten. Auf einem Rechner sollte jeweils nur ein Szenario gleichzeitig laufen.

```
[root@workstation ~]\$ vnumlparser.pl -P vnuml-szenario.xml -vB
```

Code 2.3: Herunterfahren eines Szenarios

Ein Netzwerktopologie-definierendes XML-Szenario hat in einer einfachen Form folgende Elemente, die allerdings auch, abhängig vom Ziel des Szenarios, durch andere Optionen ersetzt werden können. Die XML-Tags werden im Kommentar innerhalb des Codes kurz erklärt:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">
3
4  <vnuml>
5   <!-- Beginn der Start Tags -->
6    <global>
7      <!-- Angabe der VNUML-Version -->
8      <version>1.8</version>
9      <!-- frei wählbarer Name für die Simulation -->
10     <simulation_name>test-szenario</simulation_name>
11     <!-- MAC-Adressen für die virtuellen Netzwerkschnittstellen werden automatisch
           generiert -->
12     <automac/>
13     <!-- Definiert die Einstellung zum Management-Netzwerk. "None" heisst, dass
           kein Management-Netz benutzt wird. Zugriff ist also nur über SSH möglich --
           >
14     <vm_mgmt type="none" />
15     <!-- Standard Einstellung für ausführbare Macros (Exec-Befehle) innerhalb der
           VM -->
16     <vm_defaults exec_mode="mconsole">
17        <!-- Pfad zum Dateisystem für die virtuellen Maschinen -->
```

```
18        <filesystem type="cow">/usr/share/vnuml/filesystems/root_fs_tutorial</
              filesystem>
19        <!-- Pfad zum Kernel für die virtuellen Maschinen -->
20        <kernel>/usr/share/vnuml/kernels/linux</kernel>
21        <!-- Angabe des Konsolen-Befehls, um nach dem Start der Simulation auf die
              VMs direkt zugreifen zu können -->
22        <console id="0">xterm</console>
23      </vm_defaults>
24    </global>
25    <!-- Ende der Start Tags -->
26
27    <!-- Definition der Netze -->
28    <!-- Der Modus "uml_switch" bedeutet, dass Host- und Guest-Systeme in einem
          gemeinsamen Netz sind -->
29    <net name="Net0" mode="uml_switch" />
30    <net name="Net1" mode="uml_switch" />
31    <net name="Net2" mode="uml_switch" />
32
33    <!-- Einstellungen für die 1. Virtuelle Maschine "uml1" -->
34    <vm name="uml1">
35      <!-- Konfiguration der Netzwerkschnittstellen, diese VM hat 1 Schnittstelle mit
            der IP 10.0.0.1 -->
36      <if id="1" net="Net0">
37        <ipv4>10.0.0.1</ipv4>
38      </if>
39      <!-- Definition eines Default-Gateways innerhalb der VM -->
40      <route type="ipv4" gw="10.0.0.3">default</route>
41      <!-- Macro-Befehle, die über vnumlparser.pl -x [startsequence]@[xml-datei]
            initiiert werden können -->
42      <!-- Beispiel: vnumlparser.pl -x start@vnuml-szenario.xml -->
43      <exec seq="start" type="verbatim">nohup /usr/bin/hello &lt;/dev/null &gt;/dev/
            null 2&gt;&amp;1 &amp;</exec>
44      <exec seq="stop" type="verbatim">killall hello</exec>
45    </vm>
46
47    <!-- Einstellungen für die 2. Virtuelle Maschine "uml2", Bedeutung ist analog zu
          uml1 -->
48    <vm name="uml2">
49      <if id="1" net="Net0">
50        <ipv4>10.0.0.2</ipv4>
51      </if>
52      <route type="ipv4" gw="10.0.0.3">default</route>
53    </vm>
54
55    <!-- Einstellungen für die 3. Virtuelle Maschine "uml3" -->
56    <vm name="uml3">
57      <if id="1" net="Net0">
58        <ipv4>10.0.0.3</ipv4>
59      </if>
```

Code 2.4: Beispiel eines VNUML-XML-Szenario

Das Code-Beispiel erstellt eine Topologie mit 5 Routern (sichtbar in 2.5, die über UML-Switches verbunden sind. Router 1 (uml1) besitzt zudem einige ausführbare Befeh-

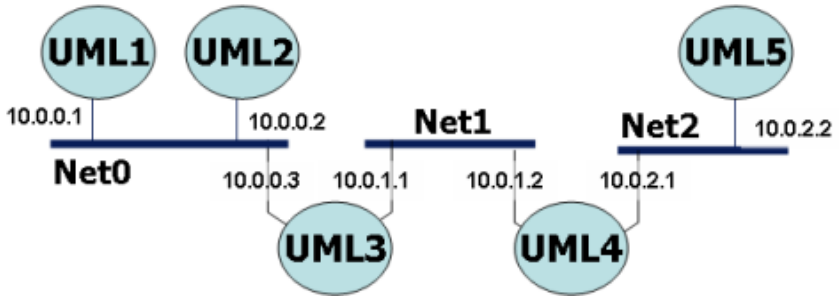


Abbildung 2.5: Diagramm des definierten Szenarios[Mad08]

le, definiert über *exec*-tags, die über den -x Parameter des vnumlparsers gestartet werden können. Sie verhalten sich wie Makros, sie werden über ein Kommando ausgeführt, das mit dem -x Parameter an alle Maschinen gesendet wird. Ist das passende Kommando nicht für den Router definiert, ignoriert er dieses. Die *exec*-tags sind interessant, um nach dem Start der Simulation Programme zu initialisieren, die im Dateisystem der Simulation implementiert wurden. In diesem Fall werden sie genutzt, um die zebra- und rip-daemons zu starten bzw. zu stoppen, die selbständig beginnen Informationen über die Topologie des Netzes zu propagieren und Routing Table Entries zu generieren. Die Typen von Netzen und Netzwerkschnittstellen können angepasst werden, um andere Verhaltensweisen zu generieren. Sobald neue Tags auftauchen, werden diese auch kurz erklärt, eine ausführliche und übersichtliche Beschreibung findet sich jedoch auch in der VNUML Referenz [Mad08]. Für jeden dieser Router werden 4 IP-Adressen reserviert:

- Management Network
- Virtual Host
- Subnetmask
- Broadcast

Virtual Host ist die IP-Adresse, die der Router innerhalb des simulierten VNUML-Netzwerks besetzt, Management Network ist die IP-Adresse, mit der der virtuelle Router von außen, also vom Host-Rechner, angesprochen werden kann, die Subnetmask- und Broadcast-Adressen sind offensichtlich. Das Wissen um diese IP-Adressen ist für die Nutzung von VNUML absolut optional, jedoch für den vorgestellten Lösungsansatz in Kapitel 4.2, EDIV für XTPeer kompatibel zu machen, hilfreich.

2.4 XTPeer

VNUML steuert die Verbindung von virtualisierten UML-Routern und deren Verbindung über TUN/TAP-Devices, während die Quagga Routing Suite die Aufgabe des Routings übernimmt. XTPeer (externally triggered peer) bietet die Möglichkeit den Routing-Daemon von Quagga zu steuern und auch auszulesen, ohne sich über die Kommandozeile auf dem Router einzuloggen. XTPeer muss auf dem Hostrechner, der auch für VNUML verwendet wird, gestartet werden. Die direkte Verbindung zu den virtuellen Routern ist dadurch garantiert, die DNS-Einträge für die Router nimmt VNUML temporär in der `/etc/hosts` vor. XTPeer wurde in Java implementiert und ist über diverse Diplomarbeiten an der Universität Koblenz entstanden ist. Die Steuerung des Routing-Daemons ist über den sogenannten XT-Server möglich, der im VNUML-Dateisystem und somit auf jedem virtuellen Router implementiert ist. Er ist an den RIP-Daemon gebunden, sobald RIP gestarted wird, ist auch der XT-Server erreichbar. Es ist möglich über die GUI von XTPeer die Router-Einstellungen ohne Neustart der Simulation zu verändern, Router oder Netze können mit Hilfe der Steuerungsbefehle heruntergefahren werden, wobei der Routerausfall dadurch realisiert wird, dass er einfach keine Updates mehr verschickt. Im Programm XTPeer ist dafür der XT-Client implementiert, der mit dem XT-Server kommuniziert. Vom Benutzer herbeigeführte Änderungen innerhalb der GUI werden vom XT-Client an jeden XT-Server gesendet, der wiederum die Routing Daemons entsprechend manipuliert. Die Kommunikation zwischen XT-Server und -Client ist in Bild 2.6 als rote Linie gekennzeichnet.

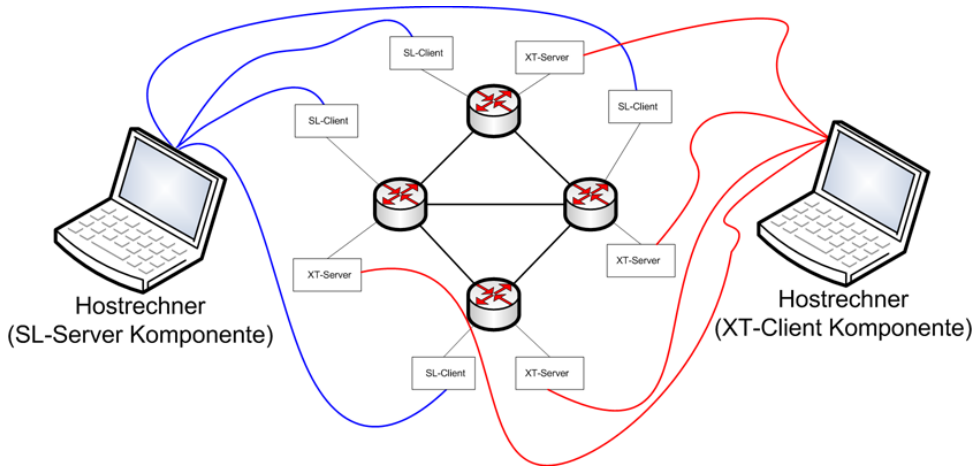


Abbildung 2.6: Kommunikation zwischen XTPeer und Hostrechner

Die blaue Linie in Bild 2.6 stellt den Datenfluss der SL-Client-Server-Beziehung dar. Der XTPeer besitzt in diesem Falle die Serverkomponente, die Daten des SL-Clients empfängt. Jede Aktion der virtuellen Router wird vom SL-Client, der wie der XT-Server auch auf dem RIP-Daemon implementiert ist, an den XTPeer gesendet, sollte eine Verbindung

bestehen. Dieser speichert die Daten und gibt diese tabellarisch wieder, um z.B. die gespeicherten Routen inklusive Metrik für jeden einzelnen Router sichtbar zu machen. Startprozedur ist größtenteils aus Kapitel 2.3 bekannt, das Exec-Kommando „start“ sei definiert als das Macro, dass auf allen VMs die Zebra- und RIP-Daemons startet.:

```
1 [root@workstation ~]\$ vnumlparser.pl -t szenario-datei.xml -vBZ
2 [root@workstation ~]\$ vnumlparser.pl -x start@szenario-datei.xml -vB
3 [root@workstation ~]\$ java -jar XTPeer.jar
```

Code 2.5: Standard-Startprozedur des XTPeers

Im XTPeer muss eben die Szenario-Datei geöffnet werden, die auch auf dem Host-Rechner aktiv ist. Nach dem öffnen der Szenario-Datei (Bild 2.7 oder einfach STRG-D drücken) erscheint die Topologie im Hauptfenster des XTPeers. Die obere Hälfte stellt das Netz graphisch dar, die untere zeigt alle Router und Netze, und da der RIP-Daemon aktiv ist, auch bereits gelernte Routen einzelner Router. Die Routeninformationen werden durch den SL-Client an den XTPeer geschickt.

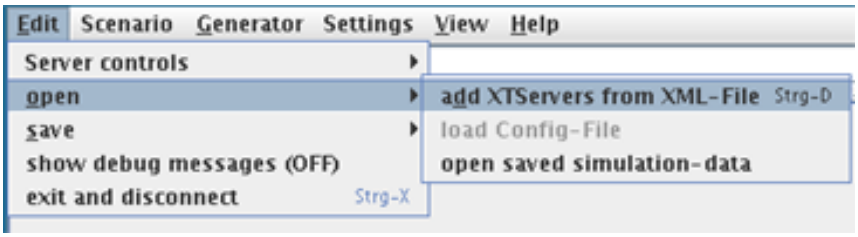


Abbildung 2.7: Öffnen des Szenarios in XTPeer

Die graphische Ansicht kann dazu verwendet werden, Router und Interfaces zu manipulieren. Hierfür nutzt man die „rechtsklickt“ man den Router, den man einstellen möchte und sucht sich die passende Option aus. Der XT-Client des XTPeers sendet darauf hin die passenden Kommandos an den XT-Server der entsprechenden virtuellen Maschine, der wiederum dem RIP-Daemon neu konfiguiert. So lässt sich möglichst einfach und intuitiv das Routing kontrollieren und in Echtzeit das Ergebnis beobachten. Sollte der XTPeer aufgrund von zu schnellen oder zu langsamen RIP-Timings[13] keine nachvollziehbaren Ergebnisse bringen, so kann man über *Scenario -> set Mode* den MANUAL-Modus auswählen, auf diese Weise senden Router nur dann periodische Updates, wenn der Benutzer dies selbst durch führt (manual trigger). Die Voreinstellung, AUTOTRIGGER, sendet Updates gemäß der Einstellungen, die man für das Szenario in der ripd.conf vorgenommen hat.

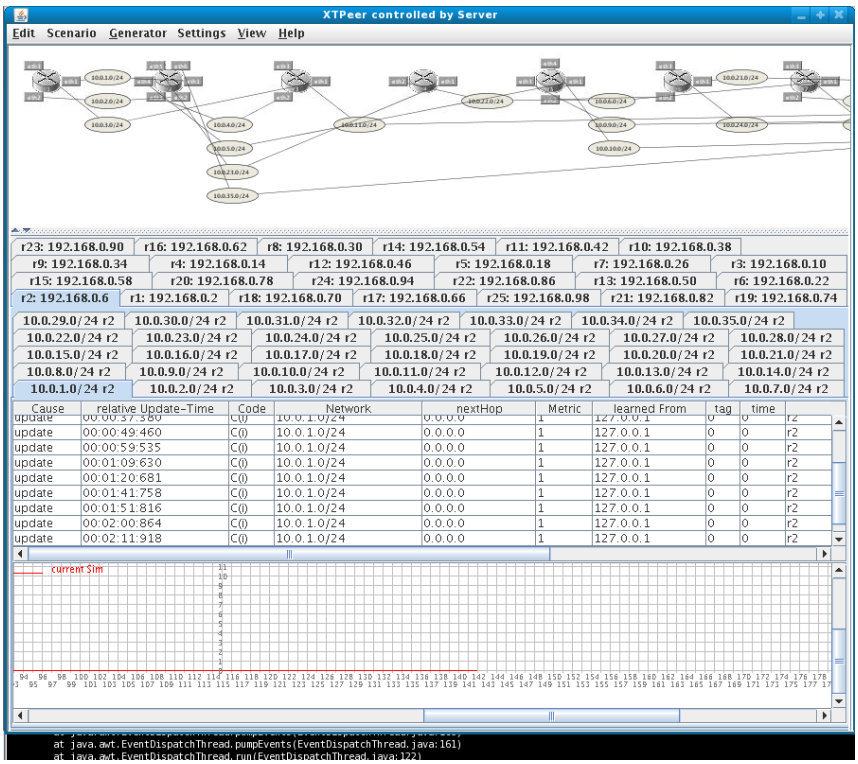


Abbildung 2.8: Screenshot der XTPeer-Oberfläche

2.4.1 Oberfläche

Die Oberfläche ist in zwei Teile aufgeteilt: Die Darstellung der Netzwerktopologie und die Tabelle der RTEs[14]. Der obere Teil zeigt die Topologie des Netzwerks mit Hilfe der JGraph-Bibliothek [Ltd09] an. Router, Netze und Interfaces sind sichtbar und lassen sich in ihrer visuelle Anordnung manipulieren ohne die Topologie zu verändern. Über diese Oberfläche können Router und Interfaces rekonfiguriert werden, ein Rechtsklick auf die einzelnen Elemente lässt Einstellungen zu. So können Interfaces oder ganze Router deaktiviert oder die Eigenschaften des RMTI verändert werden. Diese werden aber nicht in die ripd.conf übernommen, manuelle Einstellungen gelten nur für die aktuelle Instanz des Szenarios. Die periodisch gesendeten Update-Nachrichten können manuell oder automatisch (nach den RIP-Timings festgelegt) gesendet werden, abhängig davon ändert sich auch die Farbe des Interfaces im XT-Peer (engl. für auslösen: to trigger). Als Hintergrundaktion des XT-Peers wird nach einem ausgelösten Update ein Steuersignal vom XT-Client (in XTPeer) an den XT-Server (im RIP-Daemon) gesendet, in diesem Fall sorgt der XT-Server dafür, dass der virtuelle Router, auf dem der XT-Server implementiert ist, in diesem Moment eine Updatenachricht versendet. Auch die Unterdrückung von Update-Nachrichten, um z.B. einen

[13]eingestellt über die ripd.conf, die in der Szenario-Datei über den `filetree`-Tag festgelegt wird
[14]Routing Table Entries

Netzausfall zu simulieren, werden über dieses Prinzip realisiert.
Informationen des RIP-Daemons werden über die SL-Client-Server-Architektur an den XTPeer weitergegeben und im unteren Teil des Programmfensters in tabellarischer wiedergegeben. Für jeden Router im Netz wird ein eigener Reiter generiert, für jeden gerade aktiven Reiter werden die mit dem Router verbundenen Netze angezeigt. Die darunter liegenden Tabellen geben die „erlernten“ Informationen des Routing Daemons wieder. Farblich markierte Tabellen bzw. mit „!“ markierte Netze sind Indizien für Ausfälle, in denen entweder ein Count-to-Infinity-Problem vorliegt oder das Netz nicht mehr erreichbar ist. Eine optisch einfachere Darstellung für die Auswertung sind die Verlaufsdiagramme, die für jeweils ein Netz einen Graphen Metrik/Zeit aufspannen. Um den Graphen zu öffnen genügt ein Rechtsklick auf den passenden Netzreiter. Empfangene Updatenachrichten werden als Punkt gekennzeichnet, welcher anklickbar ist und detailierte Informationen über das Ergebnis der Schleifenprüfung sowie den Routingeintrag preis gibt.

2.4.2 Troubleshooting

Es gibt diverse kleine Fehler, die häufig auftreten und mit dem richtigen Hintergrundwissen einfach behoben werden können. Dazu gehören:
Problem: XTPeer kann nicht zu allen virtuellen Maschinen Verbindung aufnehmen
Mögliche Lösungen:

- Das Szenario ist nicht gestartet
- Der RIP-Daemon ist nicht gestartet (in der XML-Datei schaun, welches EXEC-Kommando `ripd` initiiert)
- Das Dateisystem, welches für die virtuellen Router verwendet wurde, hat keine SL-Server-Erweiterung
- Es gibt keine direkte Verbindung zu den virtuellen Routern (z.B. `ping r1`[15] oder `ssh r1` ohne Erfolg), Grund sind entweder fehlerhafte Tunnel (nur in Verbindung mit EDIV) oder die Einträge in der `/etc/hosts` fehlen
- Die direkte Verbindung zu den virtuellen Routern wurde abgelehnt mit „denied public key“ - siehe hierzu Kapitel 4.3.

Problem: XTPeer friert nach einer gewissen Zeit ohne Fehlermeldung ein
Mögliche Lösungen:

[15] r1 ist in diesem Beispiel der Name eines virtuellen Routers, definiert in der XML-Datei

- Ein interner Java-Fehler, tritt selten auf, jedoch wird empfohlen keine Third-Party-Distribution zu nutzen, da es bei diesen Java-Installationen häufiger auftritt. Eine unter Linux installierbare Datei gibt es auf der Offiziellen Webseite des Herstellers Oracle (ehemals Sun)[16]

- XTPeer benötigt viel Arbeitsspeicher, der Bedarf steigt mit der Anzahl der simulierten Router. Alle Daten, die der SL-Server im XTPeer sammelt, werden im Arbeitsspeicher zwischengelagert, aber dieser wird nie freigegeben. Falls das Betriebssystem den maximalen Speicherverbrauch pro Prozess begrenzt, kann man dies umgehen, indem man XTPeer mit folgendem Kommando startet:

```
1 [root@workstation ~]\$ java -jar -Xmx4096M XTPeer.jar
```

Code 2.6: Start des XTPeers mit mehr reserviertem Arbeitsspeicher

Die Option **-Xmx4096** bewirkt, dass dem Javaprogramm bis zu 4096MB Arbeitsspeicher zugewiesen werden. Eine weitere Option, **-Xms4096**, würde bewirken, dass dem Javaprogramm mindestens 4096MB Arbeitsspeicher zugewiesen werden, die Optionen können auch zusammen benutzt werden, sofern genug freier Speicher zur Verfügung steht.

Problem: Die Metriken von RIP und RMTI werden nicht weitergegeben oder stellen nur Verbindungen zu ihren direkten Nachbarn, aber zu keinem anderen Router her (das Metrik-1-Problem)
Mögliche Lösungen:

- Die Firewall des Hostsystems blockt Verbindungen zwischen den net-Devices der Simulation. Beim Zimulator wurde der Startvorgang deshalb geändert, um sicher zu stellen dass keine Verbindungen geblockt werden. Der manuelle Weg ist nach jedem Neustart des Linux-Systems folgendes einzugeben:

```
1 [root@workstation ~]\$ iptables --flush
```

Code 2.7: Eliminieren der Firewalleinträge

2.5 Zimulator

Der Zimulator ist das Ergebnis der Diplomarbeit von Marcel Jacobs [Jac10]. Es handelt sich um ein umfassendes Perl-Programm, mit dem Simulationen in VNUML gestartet,

[16] Offizielle Java Homepage: http://java.sun.com/javase/downloads/index.jsp

manipuliert und ausgewertet werden können. Dabei ließt das Programm keine Routing Table Entries über den SLserver, der auf dem VNUML-Image implementiert ist, sondern speichert den Netzwerkverkehr mit Hilfe von `tcpdump`. Nach einer vordefinierten Zeitspanne werden diese Dumps ausgewertet, ändern sich die dump-Einträge ab einer bestimmten Zeit nicht mehr, ist dies ein Indiz dafür, dass das Netz konvergent ist. Die Zeitmessung ist nicht „live“ wie beim XTPeer, verbraucht aber deutlich weniger Speicher und ist auch über eine Stapelverarbeitungsdatei steuerbar. Mehrere Versuche hintereinander durchzuführen bedarf keiner menschlichen Komponente, Start/Stop der VNUML-Simulationen sowie die Manipulation ist komplett automatisiert.
Zimulator skaliert bis auf 255 Netze / Router, vom Start bis Ende eine Simulation, die über spezifizierte Konfigurationsdateien gesteuert werden, wird jedes virtuelle Netz über `tcpdump` mitgeschnitten. Nach der Simulation werden die Dumps über einen Parser, der die RIP-Funkionalität immitiert, ausgewertet und das Ergebnis in eine Text-Datei geschrieben. Im Ergebnis stehen Informationen über Diameter, Konvergenzzeit, Anzahl der Router/Netze/Loops, Traffic etc [Jac10]. Die Informationen kommen also nicht direkt aus den RIP-Daemons, sondern sie werden durch die Paketinformationen und einen Parser interpretiert.
Als Grundlage für korrektes Arbeiten nimmt das Programm an, dass ein Netzwerk erst dann konvergent ist, wenn alle aktiven Router von allen anderen Routern erreicht werden können. Wird ein Ausfall bekannt, der dessen Auswirkungen dafür sorgen, dass ein Router von anderen Routern isoliert wird, errechnet das Programm keine richtigen Konvergenz-Zeiten mehr. Es sollte also vermieden werden Blatt-Router oder Artikulations-Netze ausfallen zu lassen, denn so entstehen seltsame Messergebnisse. Fälle wie sie in Kapitel 2.2.1 beschrieben werden, kann der Zimulator nicht errechnen. Router 1 als Ausfallrouter ist im abgezeichneten Graph ein „Blatt“ und kann zu Falschmessungen führen (Beispiel für eine Artikulation: 3.14). Eine kleines Tutorial wie das Programm zu konfigurieren ist gibt es in Kapitel 8.2. Zimulator wird über Konfigurationsdateien und eine Stapelverarbeitungsdatei gesteuert, eine Konfiguration sieht in der Regel wie folgt aus:

```
[root@workstation ~]\$ cat routerausfall_r14_60_300.cfg
start(scenario)
sleep(3)
start(tcpdump)
start(protocol)
sleep(60)
disable(r14)
gettime()
sleep(200)
stop(tcpdump)
stop(scenario)
```

Code 2.8: Beispiel einer Zimulator Szenariokonfiguration

start(scenario) startet das VNUML-Szenario über einen vordefinierten Startup-String, der in der Zimulatorkonfiguration (Beschrieben in Kapitel 8.2) definiert wird. **sleep(XX)** ist ein Systemcall, der das Programm für XX Sekunden unterbricht und wird verwendet, um dem Protokoll innerhalb der Simulation Zeit zu geben einen konvergenten Zustand herzustellen. Hierbei sollte stehts deutlich mehr Zeit gegeben werden, als das Protokoll in der Regel benötigt. **start(tcpdump)** und **start(protocol)** sind feste Bestandteile jeder Simulation, sie starten das Systemprogramm TCPdump für alle virtuellen Netze und initieren den RIP-Daemon, der selbständig beginnt Routinginformationen auszutauschen. Dieses Beispiel deaktiviert den RIP-Daemon des Routers r14 nach der Coldstart-Konvergenz, sodass sich das Netzwerk reorganisieren muss. Der Zeitpunkt des Ausfalls wird mit **gettime()** festgehalten. Nach einer neuen Wartezeit, in der das Netzwerk selbständig einen konvergenten erreichen soll, werden die TCPDumps und VNUML-Prozesse heruntergefahren und die Parser-Funktion des Zimulators gibt nach der Analyse der erhobenen Daten ein Ergebnis aus.
Sind Szenario- und Konfigurations-Dateien vorhanden, kann die Stapelverarbeitungsdatei erweitert werden.

```
[root@workstation ~]\$ cat simulations.txt
z75r routerausfall_r14 rip 10 2
```

Code 2.9: Beispiel einer Zimulator Stapelverarbeitungsdatei

Der Inhalt des Beispiels ist wie folgt zu verstehen:

- z75r = Name des Szenarios ohne Dateiendung *.xml[17] / *.zvf[18]

[17] Dateiformat geeignet für VNUML, bestehend aus Definitionen der Netze, VNUML-EXEC-Kommandos, den Virtellen Maschinen und deren Netzwerk-Verbindungen im XML-Format

[18] Dateiformat geeignet für Zimulator, bestehend aus Netzen und einer Auflistung der Netz-Verbindungen für jeden Router. Aus dieser Datei kann mit Hilfe von Zimulator mit weniger Aufwand eine XML-Datei erzeugt werden

- routerausfall_r14 = Name der Konfiguration ohne Dateiendung *.cfg
- rip = einzusetzendes Protokoll, start(scenario) startet den gleichnamigen Routing-Daemon[19]
- 10 = Anzahl, wie oft der Versuch erfolgreich durchgeführt werden soll
- 2 = Anzahl, wie oft der Versuch versagen darf (danach wird einfach mit der nächsten Zeile der simulations.txt weitergearbeitet)

Gestartet wird das Programm über:

```
1 [root@workstation ~]\$ ./zimulator -vs simulations.txt >> simulations.out
```

Code 2.10: Start des Zimulators

Es sollten für den Start die Parameter v (verbose mode für mehr Ausgaben) und s (start) verwendet werden. Die Ausgabe in eine Datei ist für spätere Fehlersuche und auch für folgende Zusatzwerkzeuge wichtig.
Für den Fall das VNUML abstürzt gibt es das Script `checkrunning.pl`. Es überprüft, ob die Standard-Ausgabedatei `simulations.out` länger als 12min nicht verändert wurde. Die Zeit lässt sich im Perlscript natürlich auch anpassen. Ist dies der Fall, wird das Programm und die Simulation heruntergefahren und neu gestartet. Mit `killsimulations.pl` kann man Zimulator im Falle eines Absturzes auch manuell herunterfahren.

Für effizientes Arbeiten mit Zimulator sollte man also 3 Fenster / Tabs für Prozesse offen halten, um den Fortschritt zu kontrollieren.

```
1 [root@workstation ~]\$ ./zimulator -vs simulations.txt >> simulations.out
2 [root@workstation ~]\$ ./checkrunning.pl
3 [root@workstation ~]\$ tail -f simulations.out
```

Code 2.11: Zimulator Monitoring

Genauere Informationen zum Zimulator mit mehr Beispielen gibt es in der Bedienungsanleitung [Jac10].

2.5.1 Troubleshooting

Wie auch beim XTPeer müssen die Linux-Einstellungen beim Zimulator präzise sein, da sonst Fehler auftreten die nicht über entsprechende Exceptions abgefangen werden. Meis-

[19] rip und rmti sind 1 Daemon, über den Konfigurations-<filetree>-Tag, gesetzt für jeden Router in den *.xml-Dateien, lässt sich festlegen ob eine Konfiguration mit oder ohne RMTI-Routinen genutzt wird (Kapitel 3.2.3

tens hängen Probleme direkt mit der VNUML-Umgebung zusammen, Fehler und Lösungen des Zimulators decken sich deshalb auch mit denen des XTPeers.

Kapitel 3

Evaluation des RMTI

Da einige Netze, die diese Arbeit testet, auch gewollt höhere Diameter als 15 aufweisen, um die Skalierbarkeit des modifizierten RIP bzw. RMTI zu untersuchen, wird auch eine leicht modifizierte Version von RIP verwendet. Klassisches RIP hat einen Infinity-Wert von 16, d.h. alle Routen, die mehr als 15 Hops aufweisen, werden als ungültig markiert. Der Algorithmus würde auf alternativen Routen warten und gar nicht in einen konvergenten Zustand übergehen lassen. Deshalb werden sowohl RIP als auch RMTI mit einem Infinity-Wert von 64 arbeiten, um jedoch fair zu bleiben werden die RIP-Untersuchungen in Fällen, wo es möglich ist, auch mit einem Wert von 16 wiederholt, da bei einer auftretenden CTI-Situation RIP sonst ungleich schlechter abschneiden würde. Zusätzlich wurde in der Vergangenheit versucht das Protokoll mit geringeren Timings arbeiten zu lassen, auch diese Variable soll berücksichtigt werden.
Da es bei den Messung nur um die Konvergenzzeit und Datenvolumina geht, sieht diese Arbeit davon ab Testfälle durchzuführen, in denen Artikulationen[1] oder Brücken[2] in einer Topologie ausfallen. Ausgefallene Elemente eines Netzwerks sollen durch Redundanz aufgefangen werden.

3.1 Testumgebung

Um die Leistungsfähigkeit des RMTI-Protokolls in größeren Netzwerken zu prüfen wurden diverse Netzwerkszenarien erstellt. Sie sind größen-klassifiziert in 25/50/75/100 Router pro Szenario, dazu gibt es ein paar Varianten mit mehr bzw. weniger Netzen, um Unterschiede im Diameter und in den Loops zu erzeugen, die sich unter Umständen auf die Konvergenz-

[1] Ein Router in einer Topologie heißt Artikulation, wenn dieser Router zwei Teilnetze verbindet. Sollte er ausfallen, zerfällt das Netz in zwei unverbundene Teilnetze und die Konvergenzzeit müsste für zwei neue Topologien berechnet werden.
[2] Das Gegenstück zu einer Artikulation, ein Netz das zwei Teilnetze miteinander verbindet, heißt Brücke

zeit auswirken. Unter VNUML werden zudem unterschiedliche Protokolle eingesetzt, um Coldstart Konvergenzen zu messen und die zeitlichen Unterschiede festzustellen. Voraussichtlich werden sich die Zeiten kaum unterscheiden, wichtiger sind jedoch die Zeiten bei simulierten Router- und Netzausfällen, zusätzlich dazu auch das Reaktionsverhalten bei provozierten CTI-Situationen. Als Werkzeug wird aufgrund von Speicherproblemen von XTPeer im Zusammenhang mit größeren Netzwerken (auch ab 25 Routern) ausschließlich der Zimulator verwendet.

3.2 Vorbereitungen

VNUML deklariert ein vordefiniertes Konfigurationsverzeichnis über den <filetree>-Tag. In diesem Verzeichnis befinden sich in der Regel eine Konfiguration für Zebra und den RIP-Daemon, sowie für alle anderen Routing-Daemons, die man verwenden möchte. In diesem Fall beschränkt sich die Beschreibung auf diese zwei Dateien. Die `zebra.conf` ändert sich in keinem der Versuche. Diese Dateien werden an alle Instanzen von VNUML vom Hostsystem übergeben. Standardmäßig befinden sich diese Dateien in einem Unterverzeichnis relativ zur XML-Datei, z.B. conf/.
Der Standard-Hostname ist für jede Maschine die Zebra verwaltet „zebra“ und das Standardpasswort lautet „xxxx“. Diese Werte können jedoch durch die Daemons überschrieben werden. Die Datei wird während des Starts des Szenarios in das instanzierte Dateisystem der jeweiligen virtuellen Maschine kopiert, ist diese gestartet wird diese Datei verwendet um Zebra zu starten. Ein EXEC-Tag in der VNUML-Datei, z.b. „start“, zeigt in der Regel auf einen solchen Befehl:

```
1 zebra -f /etc/quagga/zebra.conf -d
```

Code 3.1: Start des Zebra-Daemons

Der folgende Ausschnitt zeigt den Inhalt der `ripd.conf`, der Datei, die verwendet wird, um den RIP Daemon zu konfigurieren. Zusätzliche Parameter werden im Kapitel 3.2.3 erklärt.

```
[root@workstation ~]\$ cat ripd.conf
!
hostname ripd
password xxxx
!
router rip
network 10.0.0.0/8

!timers basic 10 30 20
timers basic 30 180 120
```

Code 3.2: Standard rip.conf

Die mit „!“ beginnenden Zeilen sind auskommentiert, spielen also für die Konfiguration keine Rolle. Hostname und Passwort sind die Standardwerte, die für die VM gesetzt werden. Im Falle von VNUML werden diese Hostnamen jedoch schlussendlich in die jeweiligen Routernamen umbenannt. Die Variable „Router“ legt das Routingprotokoll fest, in diesem Falle also „rip“. „network“ legt eine Maske für das virtuelle Netz fest, das verwendet wird. Jede der Schnittstellen, die mit dem virtuellen Router verbunden sind, erhält eine eindeutige IP Adresse aus diesem Pool. „timers basic“ legt fest, nach welchen RIP-Timings (siehe Kapitel 2.2) der Daemons agieren soll. Die Zahlen stehen für UPDATE TIMEOUT GARBAGETIME, in dieser Reihenfolge. Standardwerte sind 30 180 120 Sekunden, allerdings wird auch mit kleineren Werten gearbeitet, sodass alternative Konfigurationen auskommentiert in den Dateien im Anhang stehen könnten (nach [GNU09]).
Von der AG Rechnernetze zusätzlich implementierte Konfigurationsmöglichkeiten sind „mti“, „infinity“, „cti“ und „hello“. Es sind boolsche Variablen, können also 0 oder 1 sein, siehe Kapitel 3.2.3
Die Variable „cti“ sollte **nicht** in allen Routern, sondern nur in ausgesuchten virtuellen Maschinen eingesetzt werden. Der <filetree>-Tag wird in der Regel für jeden einzelnen Router in der XML-Datei gleich beschrieben, will man aber an einer bestimmten Stelle einen Count-To-Infinity-Effekt provozieren, sollte man für die bestimmte Stelle einen virtuellen Router mit einer ripd.conf versorgen, der mit dieser Option konfiguriert ist. „cti x y“ bewirkt, dass alle Pakete, die über das x. in der XML-Datei vordefinierte Netzwerkinterface empfangen werden, nicht über das y. Interface weitergegeben werden. Die Interfacenummerierung beginnt mit 1. Beispiel: „cti 3 1“ bedeutet, über das 3. Interface gelernte Informationen werden nicht über Interface 1 weitergegeben. Beispiele unterschiedlicher Konfigurationen in einer XML-Datei:

```
[root@workstation ~]\$ cat szenario.xml
[..]
<vm name="r1">
  <if id="1" net="net1">
    <ipv4 mask="255.255.255.0">10.0.1.1</ipv4>
</if>
       <forwarding type="ipv4" />
       <filetree root="/etc/quagga" seq="start">conf</filetree>
[..]
<vm name="r3">
  <if id="1" net="net2">
    <ipv4 mask="255.255.255.0">10.0.2.3</ipv4>
</if>
  <if id="2" net="net3">
    <ipv4 mask="255.255.255.0">10.0.3.3</ipv4>
</if>
       <forwarding type="ipv4" />
       <filetree root="/etc/quagga" seq="start">conf_cti21</filetree>
[..]
```

Code 3.3: Übergabe unterschiedlicher Konfigurationen im XML-Szenario

Entsprechend des Beispiels 3.3 relativ zum Speicherort der Datei szenario.xml sollten sich vor dem Start von VNUML die Verzeichnisse /conf und /conf_cti21 befinden. Die Dateien können unterschiedlich sein, auf diese Weise kann Router r1 anders reagieren als Router r3, wenn beispielsweise in der `rip.conf` im Verzeichnis /conf_cti21 der Parameter „cti 2 1“ eingetragen ist, wird vom Router r3 eine CTI-Situation ausgelöst, nachdem das erste mal die Infinity-Metrik registriert wurde.

3.2.1 Zimulator-Konfiguration

Der Zimulator wird über eine Konfigurationsdatei gesteuert, sie befindet sich unter `modules/Configuration.pm`. In dieser Datei werden unter anderem die Parameter für die XML-Szenarien gesetzt, die aus der ZVF-Datei[3] erzeugt werden. Wichtig sind hier die `object` Variablen, sie geben an wo sich systemrelevante Dateien befinden, wie groß der Offset (Abstand zwischen den IP-Adressen) sein muss und welches Dateisystem benutzt werden soll.

[3]Das vereinfachte Format des Zimulator-Simulationstools, siehe Kapitel 2.5

Anpassung der Configuration.pm

Der Variablen-Block der `modules/Configuration.pm` für diese Anwendung sieht wie folgt aus:

```
$object->{DTDPATH} = "/usr/local/share/xml/vnuml/vnuml.dtd";
$object->{SSH_KEY} = "/root/.ssh/id_rsa.pub";
$object->{MANAGEMENT_NET} = '192.168.0.0';
$object->{MANAGEMENT_NETMASK} = '16';
$object->{MANAGEMENT_NET_OFFSET} = '0';
$object->{VM_DEFAULTS} = " exec\_mode="mconsole\"";
$object->{FILESYSTEM} = "/usr/local/share/vnuml/filesystems/ripmti-hello.img";
$object->{KERNEL} = "/usr/local/share/vnuml/kernels/linux";
$object->{NET_MODE} = "virtual\_bridge";
$object->{ZEBRA_PATH} = "/sbin";
$object->{RIPD_PATH} = "/sbin";
$object->{OSPF_PATH} = "/sbin";
```

Code 3.4: Konfiguration des Zimulators (modules/Configuration.pm)

Das Zimulator-Tool wird anhand dieser Informationen eine XML-Datei erstellen, mit der VNUML eine Simulation starten kann, vorausgesetzt die Pfade sind korrekt eingetragen. Das Management-Netz ist ein klassisches 192.168.0.0/16, für den Fall dass man es auch in XTpeer verwenden möchte, denn noch ist das Netz in diesem Tool fest einprogrammiert und ein anderes Netz kann nicht ausgelesen werden (mehr Vor- und Nachteile gibt es im Review (5.1). Es ist sinnvoll den Net-Offset, also den Beginn des zu vergebenen IP-Adress-Bereichs auf 0 zu belassen, weil auf diese Weise mehr Router innerhalb des Adressraums Platz finden. Da mit größeren Netzen gearbeitet wird, werden sonst sowohl der XTpeer als auch der Zimulator Fehlermeldungen ausgeben, sowie eine bestimmte Grenze überschritten wird. Da dieses Netz auch in vielen lokalen Netzwerken Anwendung findet, sollte man, falls es zuhause probiert wird, unbedingt das eigene Netz umstellen, da sich sonst IP-Adressen überschneiden und das Szenario unbrauchbare Ergebnisse liefert. VNUML sieht hierfür keine Fehlermeldung vor.

3.2.2 Dateisystem

Das in den Simulationen dieser Arbeit verwendete Dateisystem ist im Vergleich zu dem, was man auf der VNUML-Webseite herunterladen kann etwas modifiziert. Der rip-daemon unterstützt nun eine erhöhte Metrik von 64 sowie die MTI-Erweiterung und auch die Hello-Nachrichten, auf letztere wird aber nur oberflächlich eingegangen. Das Dateisystem `ripmti-hello.img` (wurde auch in Kapitel 2.3 angesprochen) be-

sitzt keine Schnittstelle zum XTpeer[4], die RIP-Daemons arbeiten wie die normaler RIP-Router, doch durch die Eigenschaften des Zimulator-Tools (Kapitel 2.5) ist es weiterhin nutzbar. Dieses Dateisystem beinhaltet zudem ein paar nützliche implementierte „Schalter“, mit denen sich unterschiedliche Verhaltensweisen des RIP-Daemons einfacher konfigurieren lassen. Gesteuert werden diese Optionen über die `ripd.conf`.

3.2.3 Routerkonfiguration: ripd.conf (besondere Parameter)

Diese Datei steuert die Verhaltensweisen des RIP-Daemons, in der Regel werden für jedes VNUML-Szenario angepasste Konfigurationen an das Dateisystem weitergegeben, wenn die Simulation initiiert wird. Die Konfiguration wird mit dem <filetree>-Tag innerhalb der XML-Datei übergeben. Die Dateien `zebra.conf` und `ospf.conf`, eventuell auch weitere, werden in der Regel nie verändert, eventuelle optionale Einträge werden deshalb auch nicht weiter erläutert[5].

```
1 [root@workstation ~]\$ cat zebra.conf
2 !
3 hostname zebra
4 password xxxx
5 enable password xxxx
```

Code 3.5: Standard zebra.conf

Die wichtigsten Schalter der `ripd.conf` sind folgende[6]:

[4] Das Dateisystem wird parallel auch in anderen Diplomarbeiten verwendet, verschiedene Gründe sprachen dafür die XTpeer-Anbindung zu deaktivieren

[5] Inhalt der `zebra.conf` sind nur Default-Werte für den Hostnamen jeder virtuellen Maschine und für das Passwort, in den meisten fällen „xxxx“ (4mal x)

[6] Das „!“-Zeichen kommentiert Zeilen innerhalb der `ripd.conf` aus, in diesem Beispiel sind die MTI-Algorithmen für Router, die diese Konfiguration benutzen, aktiviert

mti 1

Schaltet die RMTI-Subroutinen aus oder ein, sind sie ausgeschaltet (mti 0), arbeiten herkömmliche RIP-Algorithmen.

```
1  [root@workstation ~]\$ cat ripd.conf
2  !
3  hostname ripd
4  password xxxx
5  !
6  router rip
7  network 10.0.0.0/8
8  timers basic 10 30 20
9  !timers basic 30 180 120
10 !mti 0
11 mti 1
```

Code 3.6: Steuerung des RIP-Daemons: MTI ein/ausschalten

In diesem Beispiel ist MTI aktiviert, die Timings für periodische Updates, Timeouts und Garbage-Collect sind 10 / 30 / 20 und der zu benutzende Routing-Daemon soll **rip** sein. Default Hostname, falls nicht überschrieben, soll „ripd“[7] und das Passwort „xxxx“ sein.

infinity 64

Diese Option manipuliert die Infinity-Metrik für den jeweiligen RIP-Router. Der Defaultwert ist 16. Damit das Routingprotokoll auch für größere Netze nutzbar ist, arbeitet die AG Rechnernetze mit einer Metrik von 64, diese Option macht es möglich auch unterschiedliche Metriken zu testen, um zu prüfen ob es starke Veränderungen in der Konvergenzzeit gibt. Welche Zahl letztendlich in der `ripd.conf` eingetragen wird, ist dem Nutzer überlassen.

[7] Wenn über VNUML gestartet, wird dieser Wert wird in der Regel immer über den Tag <vm name> überschrieben

```
[root@workstation ~]\$ cat ripd.conf
!
hostname ripd
password xxxx
!
router rip
network 10.0.0.0/8
timers basic 10 30 20
mti 1
infinity 64
```

Code 3.7: Steuerung des RIP-Daemons: Infinity-Metrik ändern

CTI-Generierung

Die Entstehung einer CTI-Situation (siehe auch Kapitel 2.2.1) geschieht durch Datenverlust bzw. starke Verzögerungen in der Übertragung. Es wurde eine Funktion implementiert, die einen solchen Datenverlust provoziert, ohne dass das gesamte Netz den „Ausfall“ sofort mitbekommt[8]. Die Erklärung erfolgt über ein Beispiel:

```
[root@workstation ~]\$ cat ripd.conf
!
hostname ripd
password xxxx
!
router rip
network 10.0.0.0/8
timers basic 10 30 20
!timers basic 30 180 120
cti 3 1
mti 1
```

Code 3.8: Steuerung des RIP-Daemons: CTI erzeugen

Die Option `cti 3 1` bewirkt, dass alle Router, die diese Konfiguration benutzen, Paketinformationen vom 3. Interface (die Nummerierung wird durch die Reihenfolge der Interface-Dekleration im XML-File definiert) nach dem Auftreten einer Infinity-Metrik nicht über das 1. Interface weitergeleitet. Die Route über das 3. Interface ist somit nur für die Router hinter Interface 2 sichtbar, zum Interface 1 hin gibt

[8]Notiz 1: Mit dem Befehl `./zimulator.pl -H routerausfall60_300-yscenario/rip_run_1/net*` lassen sich (H-Parameter) die Dumps in einer lesbaren Form betrachten. Das gewollte Problem, der CTI, wird dadurch auch sichtbar, denn imkrementierende Metriken bis auf 16 (oder höher) sind ein eindeutiges Indiz für einen CTI. Die Standard-Infinity-Metrik beim rip-hello-Image ist noch 16, in der ripd.conf reicht aber ein `infinity 64` für den gewünschten Effekt.

es aber Update-Anomalien, in vielen Fällen einen CTI[9]. Der CTI-Generator für RMTI hat nicht zu 100% funktioniert, Ergebnisse dieser Arbeit wurden mit einer frühen Beta-Version generiert, sodass sich Fehler eingeschlichen haben könnten. Aus diesem Grund wurden nicht alle CTI-Versuche im Kapitel 3.3.2 ausgewertet, weil sie keine neuen Erkenntnisse geliefert haben. Das Dateisystem, welches sich zur Zeit im Einsatz befindet gibt vielversprechendere Ergebnisse aus, Zimulator-Result-Files sind jedoch aus Zeitgründen hauptsächlich auf DVD und online zu finden.

3.3 Versuchs-Ergebnisse

Dieses Kapitel wird auf die Versuche mit unterschiedlichen Netzwerken eingehen, die Eckdaten[10] sind:

- 25 Router
- 35 Netze
- Diameter: 8

Die Topologie ist eine Kombination diverser Testnetze, die in vergangenen Arbeiten dazu verwendet wurden die Schleifenerkennung des RMTI-Protokolls zu testen, die Idee war ähnliche Situationen innerhalb einer Schleife des größeren Netzes zu provozieren und das Verhalten von RIP und RMTI in verschiedenen Konfigurationen zu beobachten.

3.3.1 Coldstart Zeiten

Coldstart bezeichnet die Zeit, die ein Netzwerk von Routern benötigt direkt nach dem ersten Start, also zu einem Zeitpunkt ohne Routing Table Entries, benötigt, um einen konvergenten Status zu erlangen. Das eingesetzte Protokoll identifiziert die Topologie sowie die kürzesten Pfade zwischen dem lokalen und allen anderen

[9] Notiz 2: Der undokumentierte Parameter `-t` prüft die provozierten CTI-Situationen und gibt aus bei welchem Run dies geschehen ist. `./zimulator.pl -t 5 8 yszenario.zvf` sucht somit alle Konfigurationen, mit denen das Szenario yszenario.zvf einmal gestartet wurde und durchsucht die TCP-Dumps. In diesem Falle sucht das Programm nach Situationen, in denen **Netz 5** eine **höhere Metrik als 8**, auch prüft es ob die Metrik dann auch noch weiter gestiegen ist. Bei höherem Infinity-Wert ist der Metrikwert entsprechend hoch anzusetzen.

[10] Genaue Spezifikationen, also einen Graphen und die Szenario-Relevanten Dateien, gibt es im Anhang Kapitel 6

Routern.
Legende:

- Protocol -> eingesetztes Protokoll für die Testreihe
- MFN -> most frequented net (engl.: meist frequentiertes Netz), das Netz mit dem höchsten Datenverkehr
- LFN -> least frequented net (engl.: am wengigsten frequentiertes Netz), das Netz mit dem niedrigsten Datenverkehr
- Total Traffic -> das gesamte gemessene Datenvolumen während der Testreihe von allen Netzen in Kilobytes
- Ø Traffic -> das durchschnittlich gemessene Datenvolumen pro Netz in Kilobytes[11].
- Time -> die Konvergenzzeit in Stunden (hh:mm:ss)

Die Tabelle zeigt

- Bestes Ergebnis
- Schlechtestes Ergebnis
- Durchschnittliches Ergebnis

25 Router, ripd Timing 30 180 120

Dies ist eine Zusammenfassung (Tabelle 3.1) aller Messergebnisse, im direkten Vergleich zwischen RIP und RMTI. Die jeweils erste Zeile zeigt die beste Zeit, die zweite den Mittelwert der kompletten Testreihe, der dritte den schlechtesten Wert. Die Messungen und Vergleiche geschahen mit den klassischen RIP-Timings 30 / 180 / 120.

In jeweils 50 Versuchen mit RIP und RMTI wurde ersichtlich, dass RMTI ca. 1sek langsamer als RIP bei unbekannter Topologie (=Coldstart) zu einem konvergenten Netz führt. Dafür ist die Netzwerklast ist in etwa gleich hoch. Ein so großer Unterschied kann jedoch auch durch eventuelle Peak-Ergebnisse kommen, wie im „schlechtesten“ Fall jeweils zu sehen ist. Mit mehr Versuchen / Testfall ist der Unterschied wahrscheinlich nicht ganz so groß. In beiden Fällen traten die sehr hohen

[11] Hinweis: `tcpdump` misst den Datenverkehr in byte, diese Zahl wurde durch 1024 geteilt, es können also mehr Nachkommastellen vorkommen als logisch denkbar wäre

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net2	net25	55.84765625	1.59564732	00:00:08.26 (best)
	net3	net25	56.75781250	1.62165178	00:00:20.26 (worst)
	net2	net25	56.29581705	1.60845191	00:00:12.61 (average)
rmti	net4	net25	54.17187500	1.54776785	00:00:06.28 (best)
	net2	net25	76.22265625	2.17779017	00:00:28.17 (worst)
	net2	net25	56.57779947	1.61650855	00:00:13.26 (average)

Tabelle 3.1: Ergebnis der Coldstart-Tests (25 Router, RIP-Timings 30/180/120)

Konvergenz-Zeiten jeweils 1-2x auf, ebenso die sehr schnelle Konvergenzzeit von 4 bzw. 7 Sekunden. Es besteht natürlich die Wahrscheinlichkeit eines Messfehlers, der sich aber bei steigender Versuchsanzahl relativieren sollte.

25 Router, ripd Timing 10 30 20

Diese Versuchsreihe arbeitet mit stark Verkürzten Timings, nämlich 10 / 30 / 20. D.h. nach einem Ausfall werden schneller ungültige Routen verworfen und die Konvergenzzeit sollte im allgemeinen etwas kürzer sein. Dadurch, dass die Updates in kürzeren Abständen folgen, ist aber zu erwarten, dass mehr Datenverkehr anfällt.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net2	net25	80.35937500	2.29598214	00:00:08.24 (best)
	net28	net8	84.60546875	2.41729910	00:00:15.28 (worst)
	net2	net8	81.16625976	2.31903599	00:00:10.69 (average)
rmti	net2	net25	78.03515625	2.22957589	00:00:08.23 (best)
	net2	net25	84.64062500	2.41830357	00:00:16.27 (worst)
	net2	net25	82.52897135	2.35797061	00:00:11.69 (average)

Tabelle 3.2: Ergebnis der Coldstart-Tests (25 Router, RIP-Timings 10/30/20)

Obwohl die Update-Timings nur ein drittel so lang sind, fällt dies bei der durchschnittlichen Konvergenzzeit nur marginal ins Gewicht. Eine Verbesserung von 1-2 Sekunden ist zu sehen. Die Häufigkeit der extrem kurzen Zeiten von 8 Sekunden kam selten (3mal) vor, die Wahrscheinlichkeit ist vorhanden, dass es Messfehler sind. Insgesamt kann man aber auch hier feststellen, dass RMTI-Ergebnisse bei einem Mittelwert von 50 Versuchen etwas langsamer agiert als RIP.

50 Router, ripd Timing 30 180 120

- 50 Router
- 63 Netze
- Diameter: 22

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net1	net64	235.00390625	3.61544471	00:00:29.03 (best)
	net1	net64	223.59375000	3.43990384	00:00:47.09 (worst)
	net1	net64	215.53776041	3.31596554	00:00:36.38 (average)
rmti	net25	net64	213.89062500	3.29062500	00:00:28.11 (best)
	net1	net64	216.25781250	3.32704326	00:00:44.09 (worst)
	net1	net64	213.22981770	3.28045873	00:00:35.77 (average)

Tabelle 3.3: Ergebnis der Coldstart-Tests (50 Router, RIP-Timings 30/180/120)

Bei dieser Versuchsreihe sind keine besonderen Unterschiede aufgefallen. Nach 50 Wiederholungen hat der Mittelwert des RMTI bis zu 1-3 Sekunden schneller zu einem konvergenten Netz geführt als RIP.

50 Router, ripd Timing 10 30 20

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net1	net64	267.84375000	4.12067307	00:00:21.66 (best)
	net1	net64	366.01171875	5.63094951	00:00:35.62 (worst)
	net1	net64	290.38419596	4.46744916	00:00:26.96 (average)
rmti	net1	net64	278.96093750	4.29170673	00:00:24.15 (best)
	net1	net64	345.64843750	5.31766826	00:00:34.59 (worst)
	net1	net64	305.19858099	4.69536278	00:00:28.75 (average)

Tabelle 3.4: Ergebnis der Coldstart-Tests (50 Router, RIP-Timings 10/30/20)

Setzt man die Timer des Protokolls herunter, scheint RIP leistungsmäßig wieder 8% schneller zu sein als RMTI. Diese Testreihe wurde 50x wiederholt.

75 Router, ripd Timing 30 180 120

- 75 Router
- 79 Netze
- Diameter: 29

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net19	net9	248.80749421	3.14946195	00:00:35.13 (best)
	net19	net9	431.53545752	5.46247414	00:00:46.87 (worst)
	net19	net9	331.95587571	4.20197311	00:00:46.87 (average)
rmti	net19	net9	316.29687500	4.05508814	00:00:34.46 (best)
	net19	net9	831.08203125	10.65489783	00:02:33.25 (worst)
	net19	net9	342.55414870	4.39171985	00:00:48.36 (average)

Tabelle 3.5: Ergebnis der Coldstart-Tests (75 Router, RIP-Timings 30/180/120)

Die 75-Router-Topologie besteht aus 5 Reihen von jeweils 15 Routern, die an den Enden jeweils die nächstliegende Reihe verbinden. Die Ergebnisse von 50 Durchläufen zeigen, dass beim RMTI Anomalien auftreten können. Zwar besteht immer die Chance eines Messfehlers, aber die Konvergenzzeit von 2,5min bei einem Coldstart könnte durchaus auch mit einer Update-Anomalie von RMTI zusammenhängen. Die erhöhten Zeiten kamen in 50 Versuchen zwei mal vor. Die Verzögerungen durch RMTI entstehen durch das Zurückhalten von Routen durch den Careful-Modus, die bei einem Timeout-Timer von 180 Sekunden durchaus auch 180 Sekunden dauern können, abhängig von der Größe der topologischen Schleife. Dieses Phenomen trat nicht oft auf, man sollte die Auswirkungen jedoch näher untersuchen.

75 Router, ripd Timing 10 30 20

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net19	net9	411.04296875	5.26978165	00:00:28.28 (best)
	net19	net9	506.94921875	6.49934895	00:00:42.48 (worst)
	net19	net9	491.51325334	6.30145196	00:00:35.74 (average)
rmti	net19	net9	422.85156250	5.42117387	00:00:27.95 (best)
	net19	net9	529.20703125	6.78470552	00:00:38.69 (worst)
	net19	net9	490.27818080	6.28561770	00:00:34.54 (average)

Tabelle 3.6: Ergebnis der Coldstart-Tests (75 Router, RIP-Timings 10/30/20)

Bei verkürzten Timings kam die Update-Anomalie nicht vor, auch konvergierte RMTI schneller als RIP. Die Konvergenzzeit sank im vergleich zu den längeren RIP-Timings um 24 bzw. 30%, dafür erhöhte sich der gesamte Traffic um 33 bzw. 31%.

101 Router, ripd Timing 30 180 120

- 101 Router
- 131 Netze
- Diameter: 20

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net38	net1	1541.60546875	11.59101856	00:00:21.33 (best)
	net38	net1	1717.88671875	12.91644149	00:00:28.06 (worst)
	net38	net1	1534.93261718	11.54084674	00:00:24.54 (average)
rmti	net38	net3	1198,27230379	9.147116822	00:00:20.33 (best)
	net38	net3	2095.45156375	15.99581346	00:02:16.31 (worst)
	net38	net3	1467.45813412	11.20197048	00:00:28.33 (average)

Tabelle 3.7: Ergebnis der Coldstart-Tests (101 Router, RIP-Timings 30/180/120)

101 Router, ripd Timing 10 30 20

Bei der 101-Router-Topologie mit niedrigen RIP-Timings fielen keine Konvergenzzeiten auf, die besonders auffällig waren. Der verbrauchte Datenverkehr stieg im

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net38	net3	1339.57812500	10.07201597	00:00:18.96 (best)
	net38	net1	1581.96875000	11.89450187	00:00:30.10 (worst)
	net38	net3	1528.09695870	11.48945081	00:00:23.94 (average)
rmti	net38	net3	1198,27230379	9.147116822	00:00:16.96 (best)
	net38	net3	2095.45156375	15.99581346	00:00:39.87 (worst)
	net38	net3	1467.45813412	11.20197048	00:00:22.99 (average)

Tabelle 3.8: Ergebnis der Coldstart-Tests (101 Router, RIP-Timings 10/30/20)

Vergleich zu den langsamen RIP-Timings extrem hoch. Die TCP-Dumps der 101-Router-Topologie haben pro Netz ein Datenaufkommen von 40-60MByte, da die RIP-Pakete mit steigender Netz-Anzahl auch größer werden. Die 75-Router-Topologie hat mit unterschiedlich langen Timings auch ein unterschiedliches Datenaufkommen, je kürzer die Timings desto mehr Daten werden versendet. Bei der 101-Router-Topologie wurde ein solcher Unterschied nicht gemessen, die Konvergenzzeit war zwar insgesamt bei kürzeren Timings niedriger, aber es sind nicht mehr Daten angefallen. Eine mögliche Annahme wäre eventuell der Aufbau der zwei Topologien, bei der 101-Router-Topologie sind deutlich mehr Netze vorhanden, die zwar propagiert werden müssen, aber dafür stehts niedrigere Metriken benötigen und kürzere Wege entsprechen weniger Datenaufkommen.

3.3.2 Device-Failure Zeiten

Die folgenden Testreihen prüfen Szenarien, in denen, nachdem das Netzwerk einen konvergenten Status erlangt hat, jeweils ein Router ausfällt und wie es braucht erneut einen konvergenten Status zu erlangen. Vier Szenarien sind ja gegeben, Ausfallmöglichkeiten sind sehr viele da. Diese Arbeit beschränkt sich auf einige Standard- und spezielle Fälle.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net4	net7	75.50390625	2.15725446	00:00:22.02 (best)
	net27	net1	108.13281250	3.08950892	00:00:38.23 (worst)
	net4	net1	107.19154575	3.06261559	00:00:30.85 (average)
rmti	net27	net3	102.29687500	2.92276785	00:00:26.05 (best)
	net32	net3	339.69531250	9.70558035	00:01:49.73 (worst)
	net19	net9	119.79849137	3.42281403	00:00:34.57 (average)

Tabelle 3.9: Coldstart-Tests (25 Router, RIP-Timings 10/30/20), Ausfall von r2

In diesem Versuch die Konvergenzzeit nach dem Ausfall von Router r2, dem Router mit dem höchsten Traffic-Aufkommen der 25-Router-Topologie, gemessen. Sowohl bei RIP als auch RMTI wurden über 30 Wiederholungen durchgeführt, die Zeiten bei RIP blieben relativ stabil, bei RMTI fielen 1 Testlauf auf, bei denen die Konvergenzzeit über 90 Sekunden dauerte. Nähere Betrachtung hat gezeigt, dass kein CTI aufgetreten ist, die Routen-Speicherung wurde sehr wahrscheinlich durch den Strict-Modus von RMTI länger zurückgehalten als notwendig. Diese Mechanik ist sinnvoll, um CTI-Probleme zu verhindern, Routen die aber richtig sein könnten nicht unnötig zu verwerfen.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net21	net40	794.30078125	10.18334334	00:00:56.06 (best)
	net21	net18	978.44921875	12.54422075	00:01:08.57 (worst)
	net21	net18	910.44182477	11.67233108	00:01:03.00 (average)
rmti	net21	net18	961.03125000	12.32091346	00:01:01.01 (best)
	net21	net68	2282.21875000	29.2592147	00:02:45.14 (worst)
	net21	net18	1204.18428308	15.4382600	00:01:33.42 (average)

Tabelle 3.10: Coldstart-Tests (75 Router, RIP-Timings 10/30/20), Ausfall von r19, Delay-Problem

Dieses Beispiel war ebenfalls ein Routerausfall, diesmal bei der 75-Router-Toplogie. Die Konvergenzzeiten unterscheiden sich hier in den schlechtesten Fällen, in 2/30 Wiederholungen traten beim RMTI erneut starke Verzögerungen auf, ohne dass ein CTI aufgetreten ist. Rechnet man diese Ausnahme-Versuche mit ein, konvergiert der RMTI auch hier langsamer als RIP.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	xxx	xxx	xxx	xxx	00:14:49.29 (average
rmti	net32	net16	789.19994630	9.98987273	00:00:55.70 (best)
	net34	net16	3842.85115686	48.64368552	00:04:31.22 (worst)
	net2	net16	1152.20358408	14.58485549	ca. 00:01:21.32 (average)

Tabelle 3.11: Coldstart-Tests (75 Router, RIP-Timings 30/180/120), Ausfall von r2, CTI-Problem

Der CTI-Generator im VNUML-Image von 2009 arbeitet nicht sehr zuverlässig, da CTI-Erzeugungen nicht zu 100% erfolgreich waren. Versuche mit diesem Dateisystem haben aber letztendlich gezeigt, dass wenn Router 16 einen CTI provoziert, nachdem R2 ausfällt, das gesamte Netz während einer Zeit von 15min mit RIP nicht konvergiert. Es entstehen an Router 16 oft Falschinformationen, über die Timeout-Zeit hinaus weitergegeben werden und keine Konvergenz zulassen. Mit RMTI haben Versuche zwar zu einem konvergenten Netz geführt, manchmal traten jedoch auch erneut sehr hohe Konvergenzzeiten auf. Auch vergleichbare Versuche mit anderen Topologien haben dieses Problem aufgezeigt. Versuche mit dem neuen Dateisystem von September 2010 sind vielversprechender, aber es wurden nicht genug Versuche durchgeführt um einen ordentlichen Mittelwert festzustellen. Hier besteht noch Nachholbedarf. Es sind aber genügend Indizien vorhanden zu sagen, dass die CTI-Vermeidung auch in großen Netzwerken funktioniert.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net19	net9	436.25000000	5.66558441	00:00:41.20 (best)
	net19	net9	463.07421875	6.01395089	00:00:50.41 (worst)
	net19	net9	434.14404296	5.63823432	00:00:45.20 (average)
rmti	net19	net9	318.32421875	4.13408076	00:00:36.61 (best)
	net19	net9	417.24609375	5.41878043	00:00:45.37 (worst)
	net19	net9	382.12548828	4.96266867	00:00:40.59 (average)

Tabelle 3.12: Maze-Tests (75 Router, RIP-Timings 10/30/20), „Maze“-Test

Der „Maze“-Test[12] war eine Idee die Infinity-Metrik zu erreichen oder zumindest ein Netz zu kreiieren, dass zumindest einen Diameter von fast 64 hat. In der 75-Router-Topologie hatte es sich angeboten verbindende Netze zu streichen / ausfallen

[12]Maze, weil die Topologie einem Labyrinth ähnelt

zu lassen, um die 5 Reihen von jeweils 15 Routern nur durch ein Netz anstatt zwei Netze zu verbinden. Die Konvergenzzeit durch RIP hat gezeigt, dass sie stark ansteigt. RIP und RMTI nehmen sich nicht viel, die Tabelle zeigt dass mit kürzeren Zeiten RMTI etwas schneller konvergiert. Wenige Versuche mit längeren Timings schlugen entweder wegen Speicherproblemen fehl oder zeigten erneut einen Nachteil von RMTI gegenüber RIP.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net2	net8	90.20916565	2.57740473	00:00:09.25 (best)
	net2	net8	587.67614296	16,79074694	00:01:00.26 (worst)
	net2	net8	190.75581407	5,78047921	00:00:19.56 (average)
rmti	net16	net25	99.86398446	3.02618134	00:00:10.24 (best)
	net32	net3	126.97549587	3,84774229	00:00:13.02 (worst)
	net2	net8	108.83613932	3.29806482	00:00:11.16 (average)

Tabelle 3.13: CTI Situaionen, RMTI+RIP Misch-Netzwerk (25 Router, RIP-Timings 10/30/20)

Dieser Test wurde mit der 25-Router-Topologie mit kurzen RIP-Timings gemacht. Die Besonderheit ist der „Mix“ von Routern im RMTI-Ergebnis. Hier wurden nur bei r1, r2 und r3 die RMTI-Routinen aktiviert, alle anderen Router blieben Standard-RIP. Ziel des Versuchs war zu testen, ob wenige RMTI-Router einen Unterschied im RIP-Netzwerk bewirken können. In 20 Versuchen kam im RMTI-Netzwerk keine hohe Konvergenzzeit vor, man sollte also meinen, dass CTI-Situationen verhindert wurden. Während den RIP-Versuchen gab es allerdings nur zwei Ergebnisse, die über 50 Sekunden lagen und eine CTI-Situation zumindest bis zu einem Timeout bestand. Allerdings wurde bei den anderen Versuchen kein CTI erzeugt, dieser Versuch sollte deshalb mit einer aktuellen Version des RMTI-CTI-Generators wiederholt werden.
Weitere Ausfalltests mit Topologien mit 50 und mehr Routern haben in jeweils 10-20 Versuchen immer ein-zwei Fälle aufgewiesen, in denen RMTI deutlich länger brauchte, um das Netzwerk in einen konvergenten Zustand zu bringen, als es bei RIP der Fall war. Diese Tests sind erst kürzlich unternommen worden, das neuste Dateisystem wurde verwendet. Je größer Schleifen innerhalb einer Topologie werden, desto höher ist die Zeit, die RMTI zur Konvergenz benötigt, falls eine Route zurückgehalten wird, wenn sie den Simple Loop Test nicht bestanden hat (Careful Modus). Zwar ist diese Zeit durch den RIP Timeout nach oben beschränkt, aber pro

Hop in einer topologischen Schleife werden im Augenblick 5 Sekunden beanschlagt - der Wert ist noch nicht variabel bzw. über die `ripd.conf` einstellbar. Der Fehler tritt deshalb häufig in Versuchen mit verkürzten RIP-Timings auf, im Zusammenhang mit normalen RIP-Timings waren die Konvergenzzeiten zwar leicht verzögert, aber verlängerten sie in Einzelfällen nicht um bis zu 200%.

Protocol	MFN	LFN	Total Traffic (kb)	Ø Traffic (kb)	Time (Stunden)
rip	net2	net19	184.33984375	5.26685267	00:00:49.29 (best)
	net3	net16	395.21093750	11.29174106	00:01:49.25 (worst)
	net2	net16	220.78417968	6.30811941	00:01:04.18 (average)
rmti	net32	net16	198.87890625	5.68225446	00:00:55.70 (best)
	net34	net16	347.02734375	9.91506696	00:01:30.24 (worst)
	net2	net16	206.95200892	5.91291454	00:01:01.78 (average)

Tabelle 3.14: Ausfalltests (25 Router, RIP-Timings 10/30/20), Ausfall von r25, Artikulationsproblem

Dieses Ergebnis ist nur eine Demonstration für die Anomalien, die auftreten können, wenn man eine Artikulation des Graphen ausfallen lässt und dies mit dem Zimulator berechnen will. Router r25 ist zwar ein Router mit sehr wenig Datenaufkommen, ist aber die einzige Verbindung zu Router r20. In diesem Fall sind also gleich zwei Router für das Netz nicht mehr erreichbar, allerdings ist r20 weiterhin aktiv. r20 bildet also ein zweites Teilnetzwerk mit nur einem Router, aber es kann dazu führen dass Zimulator seltsame Ergebnisse ausgibt, weil nicht klar ist für welches der zwei Teilnetzwerke die Konvergenzzeit berechnet werden soll. Die Konvergenzzeit ist deshalb nicht sehr aussagekräftig (erwähnt auch in Kapitel 2.5).

Kapitel 4

XTPeer und EDIV

Da das Zimulator-Tool parallel zu dieser Arbeit entwickelt wurde, stand zunächst eine Aufgabe bevor den XTPeer auch mit verteilten Simulationen zu betreiben. VNUML-Simulation lassen sich über EDIV auf mehreren Rechnern simulieren, um die Last vieler virtueller Router zu verteilen, vor allem schwächere Rechner profitieren davon. Pro virtuelle Maschine ist je einmal der Kernel im Arbeitsspeicher, sowie Teile des Dateisystems, also grob geschätzt 15-40 Megabyte. Dies ist ablesbar über einen Prozessmanager, jede Kernel-Instanz einer virtuellen Maschine heisst [linux]. Misst man den Speicherverbrauch über ein Tool wie ps, top oder psmem[1]:

- 860.5 MiB + 2.7 MiB = 863.2 MiB linux (25x)
- 1.6 GiB + 2.6 MiB = 1.6 GiB linux (50x)
- 2.5 GiB + 2.6 MiB = 2.5 GiB linux (75x)
- 3.3 GiB + 2.7 MiB = 3.3 GiB linux (101x)

Dies ist der Verbrauch der 4 Simulationen, die in dieser Arbeit untersucht wurden, jeweils mit 25-50-75 und 101 Routern. Will man zusätzlich dazu also auch den XTPeer nutzen, um die Simulation auszuwerten, bleibt unter Umständen nicht viel Arbeitsspeicher übrig. Allerdings sollte man beachten, dass Linux für die Prozesse auch die SWAP-Partition verwendet, der XTPeer tut dies jedoch nicht. Die Lauffähigkeit von XTPeer wird also nicht beeinträchtigt, die Performanz jedoch schon.
Nutzt man EDIV, so könnte man die Simulation auf anderen Rechnern laufen lassen,

[1]Es handelt sich um ein Python-Script, dass den Speicherverbrauch aller Prozesse in Megabyte ausrechnet, z.B. `./psmem.py |grep linux` - erhältlich unter http://unixlive.editboard.com/general-linux-admin-stuff-f3/how-much-ram-is-used-per-program-t5.htm

während die ganze Leistung der Workstation XTPeer zur Verfügung gestellt wird. Es bestehen allerdings diverse Probleme, die es nicht ohne weiteres zulassen EDIV und XTPeer gemeinsam zu nutzen.

4.1 Probleme und Voraussetzungen

EDIV und XTPeer funktionieren nicht auf Anhieb miteinander. XTPeer benötigt einen direkten Zugriff auf die Virtuelle Maschine, um mit dem SLserver Verbindung aufzunehmen. Der SLserver ist ein Daemon, der Kontroll- und Lesezugriffe auf den rip-Daemon ermöglicht. Über dieses Programm lassen sich im XTPeer komfortabel Router manipulieren oder über präparierte Konfigurationsdateien gezielte Ausfälle einzelner Netze erzielen, um das resultierende Verhalten in Echtzeit zu beobachten.

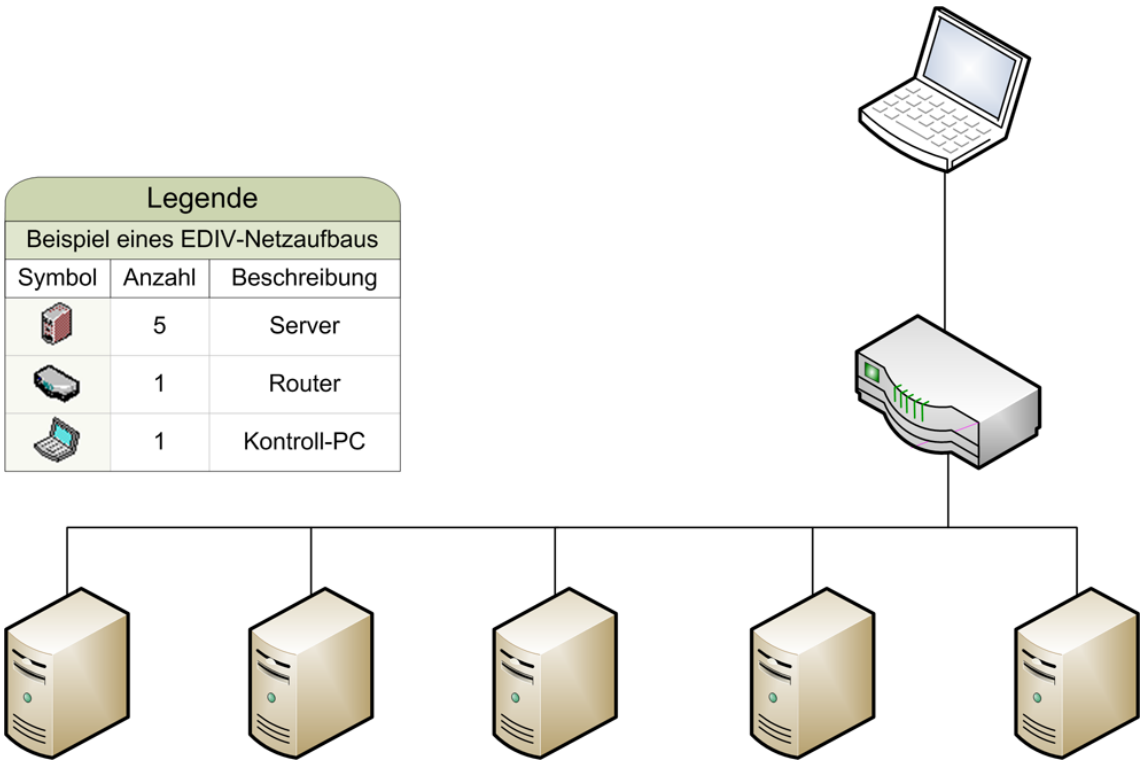


Abbildung 4.1: Ausgangssituation des EDIV-XTPeer-Problems (Graph)

Es kann eine beliebig hohe Anzahl von Servern herangezogen werden, um eine Simulation im Cluster zu starten. Im Beispiel 4.1 sind es 5 Server, die bei einem Szenario von 50 virtuellen Routern jeweils 10 hosten würden. Die in diesem Fall 10 Teilstücke der XML-Datei, welche diese jeweils 10 Router definieren, würden über SSH an die Server gesendet und lokal über VNUML gestartet.

Die Verbindung zwischen den physikalischen Servern geschieht dann über ein 802.1q VLAN bzw. ein. Informationen welche virtuellen Maschinen von welchem Server gehosted werden, werden in eine Datenbank geschrieben, auf die der Kontroll-Rechner Zugriff hat. Laut Dokumentation ist auch ein Segmentierungsalgorithmus teil des Kontrollrechners, da zur Zeit aber nur der Round-Robin-Algorithmus[2] lauffähig ist, habe ich diesen aus den Grafiken gelassen. Über diese Algorithmen wird definiert, welchem Server welche Anzahl von virtuellen Maschinen zugewiesen wird. Der RoundRobin-Algorithmus weist jedem Server dieselbe Anzahl von VMs zu. Startet man nun eine Simulation in EDIV, erzielt ein Ergebnis ähnlich des Beispiels 4.2.

Die virtuellen Maschinen sind untereinander über ein VLAN verbunden, welches ein geswitchtes Netz emuliert. Die Teilnetze der in diesem Fall 5 VNUML-Simulationen (je eine pro Server) bilden ein Ganzes, ohne dass zusätzliche Parameter in den Daemons des Dateisystems oder in der Szenariodatei getätigt werden müssen. Der Zugriff wird von EDIV über SSH-Tunnels realisiert. Diese Art der Verbindung ist für XTPeer allerdings nicht brauchbar, da auf diese Weise nicht direkt auf einen Router zugriffen werden kann (Beispiel: `ssh r44` um auf den Router r44 zuzugreifen), sondern nur indirekt über einen ssh-Port des Servers, auf dem die Maschine gestartet wurde (Beispiel: `ssh netuml04.uni-koblenz.de:64033`).

Für einen Beispielfall von 25 Routern, simuliert auf 5-Server-Cluster, zeigt die Tabelle 4.1 die Belegung von IP-Adressen, SSH-Ports und welcher Router auf welchem Rechner gehosted wird. Die IP-Adressvergabe geschieht über einen zugegebenermaßen seltsamen Algorithmus, die Router selbst werden logisch auf die Rechner verteilt. Auf dem 1. Server befinden sich die Router #1,6,11,16 und 21, entsprechend verteilt sind die übrigen 20 Router. Nach dem Start von EDIV über

```
1 [root@workstation ~]\$ ediv_ctl.pl -t 25r.xml -vBZ
```

Code 4.1: Start einer Simulation über EDIV

gibt EDIV den Zugang zu jeder virtuellen Maschine über einen SSH-Befehl an. Die Ports werden zufällig ausgewählt, erhalten jedoch eine feste Zuweisung in einer mySQL-Datenbank, welcher Tunnel (in der Form von z.B. `netuml01.uni-koblenz.de:64014`)

[2]Bereitgestellt über das perl-modul Math::Round, erhältlich über http://search.cpan.org/~grommel/Math-Round-0.05/Round.pm oder über das Software-Repository der jeweiligen Linux-Distribution, bei Fedora 11 ist es `perl-math-round`, bei Ubuntu `libmath-round-perl`.

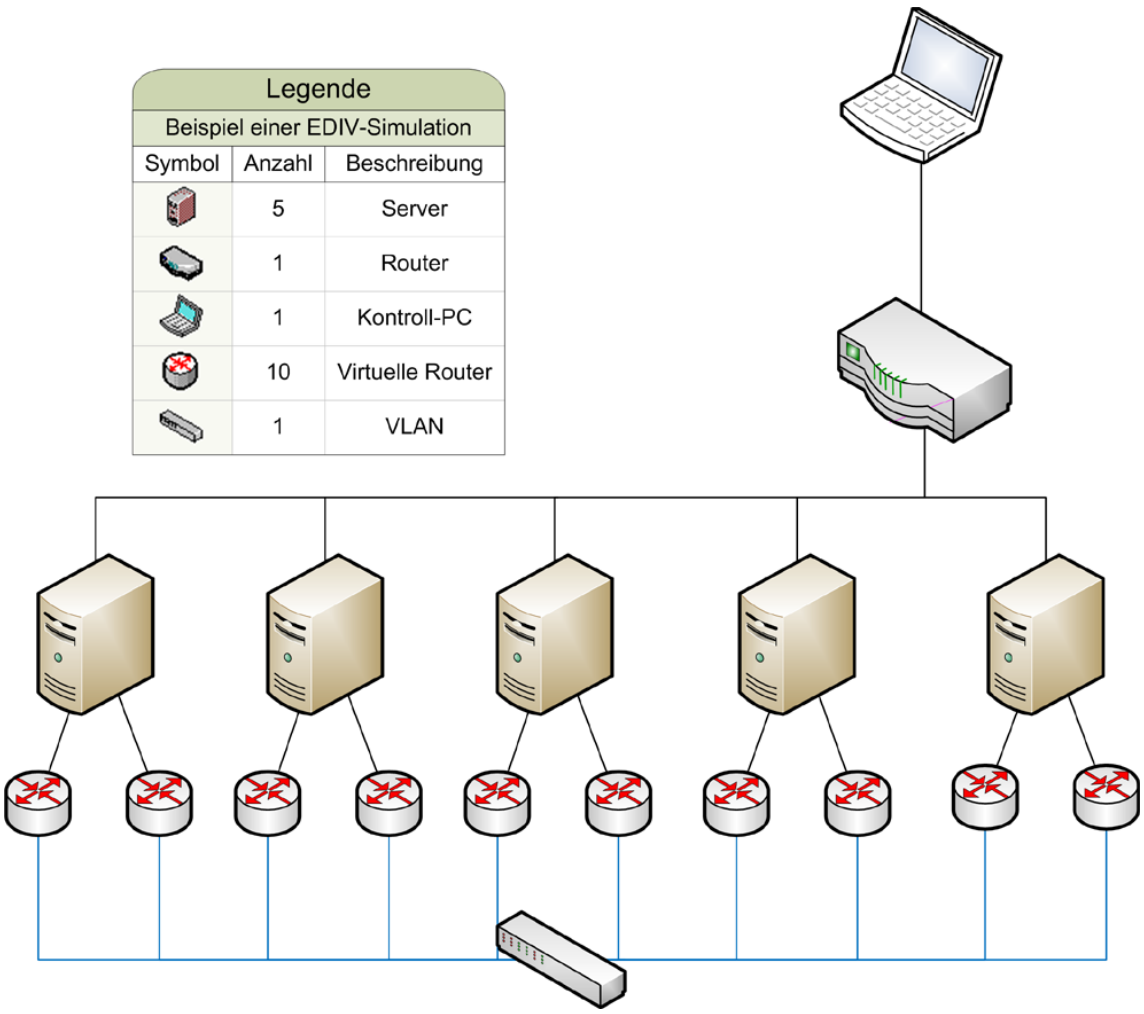


Abbildung 4.2: EDIV-Simulation gestartet (Graph)

Server	Router	ssh Port	management ip	vhost ip
netuml01	r1	64001	192.168.0.21	192.168.0.22
	r6	64023	192.168.0.25	192.168.0.26
	r11	64014	192.168.0.29	192.168.0.30
	r16	64016	192.168.0.33	192.168.0.34
	r21	64017	192.168.0.37	192.168.0.38
netuml02	r2	64015	192.168.0.1	192.168.0.2
	r7	64022	192.168.0.5	192.168.0.6
	r12	64002	192.168.0.9	192.168.0.10
	r17	64005	192.168.0.13	192.168.0.14
	r22	64020	192.168.0.17	192.168.0.18
netuml03	r3	64021	192.168.0.61	192.168.0.62
	r8	64011	192.168.0.65	192.168.0.66
	r13	64000	192.168.0.69	192.168.0.70
	r18	64007	192.168.0.73	192.168.0.74
	r23	64008	192.168.0.77	192.168.0.75
netuml04	r4	64006	192.168.0.41	192.168.0.42
	r9	64010	192.168.0.45	192.168.0.46
	r14	64024	192.168.0.49	192.168.0.50
	r19	64018	192.168.0.53	192.168.0.54
	r24	64013	192.168.0.57	192.168.0.58
netuml05	r5	64019	192.168.0.21	192.168.0.22
	r10	64012	192.168.0.25	192.168.0.26
	r15	64003	192.168.0.29	192.168.0.30
	r20	64004	192.168.0.33	192.168.0.34
	r25	64009	192.168.0.37	192.168.0.38

Tabelle 4.1: Belegung von IP-Adressen, SSH-Ports und VNUML-Maschinen in EDIV (25 Router)

zu welcher virtuellen Maschine führt. Die SL-Client-Server-Architektur ist jedoch zu dieser Art von Verbindung noch inkompatibel. Ein Work-Around sollte die schnellere Alternative sein. Mit Hilfe von IP-Tunneling sollte die direkte Verfügbarkeit der Router wieder möglich werden, und zwar nicht nur über die IP-Adresse, sondern auch über den DNS-Eintrag in der `/etc/hosts` Datei unter Linux. Diese wird nämlich auch von VNUML genutzt, nur EDIV trägt hier keine Informationen ein, die der XTPeer erwartet. VNUML wird schließlich nur auf den Cluster-Rechnern lokal gestartet, die Einträge in der `/etc/hosts` sind demnach auch nur für den Teil der Simulation aktuell, für die der jeweilige Rechner auch Ressourcen bereit stellt. Sollte der Kontrollrechner teil des Clusters sein, so erhält dieser durch VNUML ebenfalls Teileinträge der gesamten Simulation in der lokalen hosts-Datei.

4.2 Lösungsansatz

IP-Tunneling für virtuelle Maschinen von VNUML-Simulationen ist das kreieren von jeweils zwei Verbindungen zwischen dem Kontroll-Rechner und der virtuellen Maschine, eine vom Kontroll-Rechner zur VM und eine von der VM zum Kontroll-Rechner. Auf diese Weise wird jeder Cluster-Server selbst zum Router, ohne dass es die VNUML-Simulation beeinträchtigt.
Der erste Schritt ist das erstellen eines Tunnels auf jedem beteiligten Rechner, beispielhaft wird mit den folgenden Kommandos ein Tunnel auf dem Kontroll-Rechner „workstation“ und ein weiterer auf dem Cluster-Server „netuml01“ erstellt:

```
1 [root@workstation ~]\$ ip tunnel add l2uml01 mode gre local 141.26.68.21 remote 141.26.70.109 ttl
      255
2 [root@workstation ~]\$ ip link set l2uml01 up
3 [root@workstation ~]\$ ifconfig l2uml01 192.168.1.101
4 [root@workstation ~]\$ route add -host 192.168.1.111 dev l2uml01
5 [root@netuml01 ~]\$ ip tunnel add tunl_uml01 mode gre local 141.26.70.109 remote 141.26.68.21 ttl
      255
6 [root@netuml01 ~]\$ ip link set tunl_uml01 up
7 [root@netuml01 ~]\$ ifconfig tunl_uml01 192.168.1.111}
8 [root@netuml01 ~]\$ route add -host 192.168.1.101 dev tunl_uml01
9 \caption{Befehlsabfolge: IP Tunnel localhost to remote}
```

Code 4.2: Erstellung eines IP-Tunnels in beide Richtungen

Das Ergebnis ist in Bild 4.3 zu sehen. Es bestehen zwei Tunnels zwischen den Rechnern, fügt man nun die IP-Adressen der Virtuellen Maschinen, die auf dem Cluster-Server gehosted werden, zu beiden Tunneln hinzu, ist die Verbindung in eine

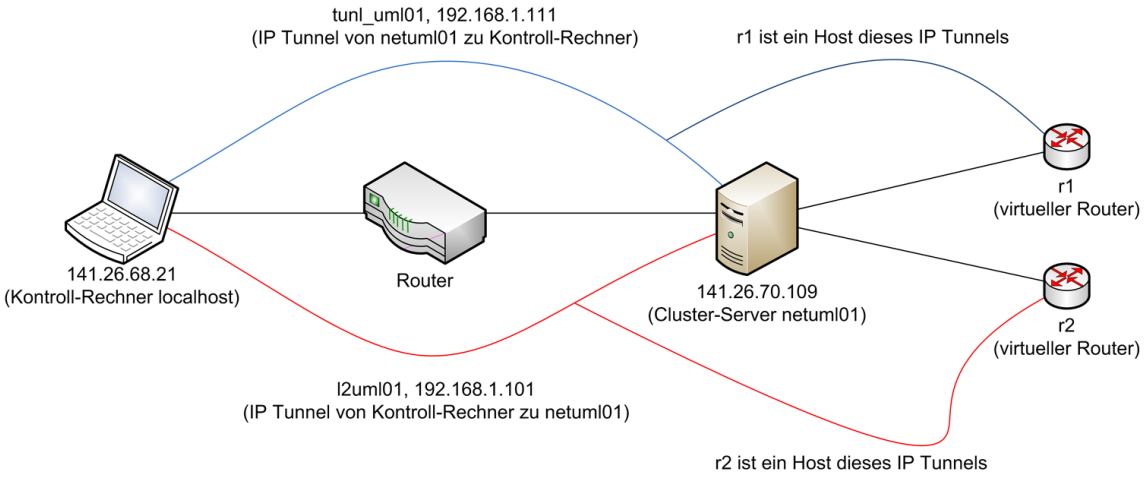


Abbildung 4.3: Verbindungsschema nach Erstellung der IP-Tunnels

Richtung, vom Kontroll-PC zu den virtuellen Maschinen, komplett. Die hinzuzufügenden IP-Adressen sind je Maschine die vhost-IP und die managementnetwork-IP (siehe auch Kapitel 2.3. Sollte man jetzt versuchen eine virtuelle Maschine vom Kontroll-PC zu erreichen, wäre der next-hop für die Adresse richtigerweise der Server, auf dem sie gehosted ist, jedoch ist es noch nicht möglich Pakete zurück zu senden. Der virtuellen Maschine fehlt noch der Eintrag in der Routing Tabelle, um den Kontroll-PC zu erreichen.
Um diesen Eintrag zu erzeugen gibt es zwei Möglichkeiten:

- ein Zugriff auf jede virtuelle Maschine über den Cluster-Server, der angibt, dass über den IP Tunnel „tunl_uml01“ ein Rechner mit der IP-Adresse 141.26.68.21 erreichbar ist, also der Kontroll-PC)
- ein Eintrag in der XML-Datei, die von EDIV gestartet wird, die jener virtuellen Maschine eine Route vorgibt, die ab dem Start Gültigkeit hat

Die erste Lösung wäre ein zweifach geschachtelter SSH-Befehl, der die Cluster-Server anweisen sollte über SSH auf den gehosteten virtuellen Maschinen je eine Route zu einem IP Tunnel zu erstellen. Die zweite Lösung erschien etwas unkomplizierter, weshalb diese auch gewählt wurde.
Ein solcher Eintrag ist abhängig vom Cluster-Setup und von der Anzahl der Router. Es gilt also herauszufinden wie EDIV die IP-Adressen verteilt, auf welchem Host welcher virtuelle Rechner läuft und entsprechend *vor* dem Start der Simulation diesen Eintrag in die XML-Datei zu schreiben.

4.3 P.E.T. Mini-Script

Herausgekommen ist ein umfangreiches Script, geschrieben in Bash ([OSP09a]), das die Erstellung der Tunnels anhand der EDIV-Konfigurationsdatei `/usr/local/etc/ediv/cluster.conf` und einigen Parametern in den ersten Zeilen des Scripts selbst übernimmt, sowie die XML-Szenario-Dateien und die `/etc/hosts` modifiziert. Einige Hilfsfunktionen sind ebenfalls Teil der Lösung, da diverse Handgriffe öfter notwendig sind und mit einem Script viel repitative Schreibarbeit abgenommen wird. Dieses Kapitel geht kurz auf einige der interessanteren Funktionen ein, die auch im Source-Code ebenfalls erklärt werden. Hilfsfunktionen werden nur aufgelistet.
Die aus der Arbeit hervorgegangenen Hilfs-Scripts wurden zu einem zusammengefasst. Dazu gehörten die Erstellung von IP Tunnels, das Hinzufügen von Hosts zu den Tunneln oder auch das Einpflegen von SSH-public-keys, um zum einen Fehler zu korrigieren, zum anderen Szenarien auf entfernten Rechnern ohne Passwort zu starten, denn jede Verbindung zum Cluster würde unter anderen Umständen eine Passwortabfrage bedeuten. Schon ab 25 virtuellen Routern wäre dies viel Tipparbeit. Theoretisch sollte dies VNUML selbst leisten, da EDIV jedoch VNUML-Instanzen auf unterschiedlichen Rechnern startet, müssen die SSH-Keys aller Cluster-Rechner für den Kontroll-Rechner bekannt sein. Die wichtigsten Funktionen werden nun vorgestellt, für den Gesamtüberblick ist der Sourcecode auch kommentiert. Die meisten Funktionen sind noch nicht auf Benutzerfreundlichkeit getrimmt, es ist ein proof-of-concept Script, wenn am Ende der Arbeit (oder danach) noch Zeit ist, wird daran noch gearbeitet. Die vorgestellten Funktionen sind nur pro forma mit einer „Gefahrenstufe“ versehen, da man in VNUML (und auch mit EDIV) nahezu immer als root-user eingeloggt ist, kann man in bestimmten Situationen bei falschem Umgang Datenverlust herbeiführen.

Hilfsfunktion: SSH Key Import

- wird gestartet über `./pet.sh -ssh`
- importiert ssh keyfiles in das vordefinierte VNUML-Image
- ungefährlich

Diese Funktion ist im Code unter *import_ssh_key()* zu finden. Der Dateiname bzw. Pfad des Arguments <keyfile> sollte den Public Key beinhalten, der auch in den XML-Dateien angegeben wurde. Dieser kann in RSA oder DSA kodiert sein, es schadet auch nicht beide Keys auf diese weise zu importieren. Das Script mountet das

Script das eingestellte VNUML-Dateisystem in das Verzeichnis `\%VNUMLdir\%/filesystems /mntpoint/`, fügt den Inhalt der Datei in die Datei |root/.ssh/authorized_keys| des Images und unmounted es wieder. Das VNUML-Dateisystem wird über den Eintrag *VNUMLIMAGE* in der Script-Datei `pet.sh` festgelegt. Wenn das lokale modifizierte VNUML-Dateisystem mit allen notwendigen SSH-Keys versorgt ist, muss noch manuell an alle Cluster-Rechner verteilt werden (oder man führt den 2. Schritt auf allen Rechnern nocheinmal aus).

Hilfsfunktion: EDIV Logs einsehen

- wird gestartet über `./pet.sh -logview`
- öffnet in einer Linux GUI ein Popup, dass alle Logs eines Szenarios einsehbar macht
- ungefährlich

Diese Funktion ist im Code unter *logview()* zu finden. EDIV legt für jeden Rechner, der Teil des Clusters ist, eine Logdatei im Verzeichnis `/tmp/` an. Diese wird benannt nach Rechnernamen + Netzwerkdomäne + Namen des Szenarios. Die Netzwerknamen stehen in der EDIV Konfigurationsdatei `/usr/local/etc/ediv/cluster.conf`, eine Unterfunktion dient als Parser dieser Datei, ließt die Rechnernamen aus (Variablenname „host“) und nutzt diese um die Dateien zu identifizieren. Ist in der EDIV-Konfiguration ein host auskommentiert, ignoriert die Funktion diese Zeile[3]. Sind alle Informationen gesammelt öffnet das Script ein neues `xterm`-Fenster und eröffnet über `screen` mehrere Tabs, in der jeweils ein Logfile zu sehen ist. Die Logfiles sind über `tail` geöffnet, Änderungen sind also in Echtzeit sichtbar. Die Tabs lassen sich über **F7** und **F8** wechseln[4].

Hauptfunktion: Tunnel erstellen

- wird gestartet über `./pet.sh -build_tunnel`
- erstellt IP-Tunnels passend für das XML-Szenario
- ungefährlich

[3] alle aktiven Hosts **müssen** erreichbar sein, da sonst das Script nicht korrekt arbeiten kann. Die unveränderte default-Datei von EDIV wird wahrscheinlich dafür sorgen, dass das Script „hängt“, hier hilft nur CTRL-C

[4] Voraussetzung hierfür ist, dass die -installssh Funktion auf dem Rechner schon ausgeführt wurde

Diese Funktion ist im Code unter *tunnel_puncher()* zu finden. Über diese Funktion werden IP Tunnels lokal und auf den Cluster-Rechnern erstellt, die Informationen welche Tunnel erstellt werden sollen, zieht aus der EDIV-Konfiguration geholt. Einer Parserfunktion liest die Variable „host“ aus der Datei `/usr/local/etc/ediv/cluster.conf` und stellt die IP-Adresse dieser Rechner fest. Abhängig von der Anzahl der Rechner läuft eine Schleife durch, die jeweils folgende Kommandos (Abbildung ähnlich, IP-Adressen sind lokal abhängig) ausführt:

```
1 [root@workstation ~]\$ ip tunnel add l2uml mode gre local 141.96.3.44 remote 141.96.3.22 ttl 255
2 [root@workstation ~]\$ ip link set l2uml
3 [root@workstation ~]\$ ifconfig l2uml 192.168.100.1
4 [root@workstation ~]\$ route add -host 192.168.101.1 dev l2uml
```

Code 4.3: Hinzufügen eines IP Tunnels (lokal)

Durch diese Kommandoserie wird lokal ein IP Tunnel erstellt, in diesem Beispiel mit dem Namen *l2uml*. Dem Tunnel wird die IP Adresse 192.168.100.1 zugewiesen, das andere Ende des Tunnels, einer der Rechner, der in der EDIV `cluster.conf` definiert wurde, wird diesem IP Tunnel als „host“ hinzugefügt, er wird die IP Adresse 192.168.101.1 erhalten. In diesem Beispiel ist die IP Adresse 141.96.3.22 dem Cluster-Rechner und 141.96.3.44 dem Localhost zugeordnet. Die IP-Adressen der Tunnel sind in der `pet.sh` vorkonfiguriert, können aber geändert werden, z.b. wenn sie durch eine eigene Netzwerkkonfiguration schon belegt sind. Die Variablen heißen TUNL_LOC_TO_UML[xx] (Richtung: Kontrollrechner zum Cluster) und TUNL_UML_TO_LOC[xx] (Richtung: Cluster zum Kontrollrechner). Es sind nur 5 Tunnel-Adressen voreingestellt, sollten mehr Rechner für den Cluster zur Verfügung stehen, können analog dazu auch weitere Adressen eingetragen werden. Die Adressen werden analog zur Reihenfolge der Hosts, die in der `cluster.conf` stehen, zugeteilt.

```
1 [root@workstation ~]\$ ssh -2 -o 'StrictHostKeyChecking no' -X 141.96.3.22 -l root 'ip tunnel add
      uml2l mode gre local 141.96.3.22 remote 141.96.3.44 ttl 255'
2 [root@workstation ~]\$ ssh -2 -o 'StrictHostKeyChecking no' -X 141.96.3.22 -l root 'ip link set
      uml2l'
3 [root@workstation ~]\$ ssh -2 -o 'StrictHostKeyChecking no' -X 141.96.3.22 -l root 'ifconfig uml2l
      192.168.101.1'
4 [root@workstation ~]\$ ssh -2 -o 'StrictHostKeyChecking no' -X 141.96.3.22 -l root 'route add -host
      192.168.100.1 dev l2uml'
```

Code 4.4: Hinzufügen eines IP Tunnels (remote)

Diese Kommandos sind analog zu denen, die lokal ausgeführt werden, nur dass sie jeweils eine Verbindung in die andere Richtung schlagen. Sie werden über SSH an den passenden Cluster-Rechner gesendet, vor allem für diese Funktion ist es deshalb wichtig, dass die SSH-Keys richtig eingerichtet sind (Kapitel 4.3). Für den Fall, dass es noch nicht standardmäßig aktiv ist, aktiviert das Programm generell auf allen beteiligten Rechnern IP Forwarding.

```
[root@workstation ~]\$ sysctl -q -w net.ipv4.conf.all.rp_filter=0
[root@workstation ~]\$ sysctl -q -w net.ipv4.conf.all.forwarding=1
```

Code 4.5: aktivieren von IP Forwarding

Hauptfunktion: Szenario-Datei patchen

- wird gestartet über `./pet.sh -patch`
- Szenario wird verändert, Backup-Hinweis beachten

Diese Funktion erstellt eine Kopie der festgelegten Szenario-Datei und modifiziert diese dann. Der Dateiname des Szenarios ist dann „patched_<xml-szenario>“, abhängig davon welche Datei man als Argument übergeben hat.
Es ist notwendig dem virtuellen Router eine Route zum Kontrollrechner bekannt zu machen, diese führt über die IP Tunnels. EDIV verteilt im RoundRobin-Modus die Router intuitiv (r1 an Rechner 1, r2 an Rechner 2 ... etc.) an den Cluster, die IP-Adressvergabe ist jedoch untypisch. PET immitiert die Adressvergabe von EDIV und weisst über die Nebenfunktion `weird_ip_algorithm()` die zu vergebenen Adressen der Clusterarchitektur entsprechend zu. Die IP-Vergabe ist also vor dem Start des Szenarios bekannt, die richtige Route kann also vorher in die XML-Datei eingepflegt werden. Dazu fügt diese Funktion in jeden <vm>-Tag einen <route>-Tag ein, der ein CIDR-32-Netz (also ein Netz mit nur einer IP-Adresse) zu einem Gateway weiterleitet, der Gateway ist in diesem Fall ein IP Tunnel. Die Einträge haben in etwa folgende Form:

```
[root@workstation ~]\$ cat ranomized7564-mti.xml
# [..]
<route type="ipv4" gw="192.168.0.0">192.168.1.101/32</route>
# [..]
```

Code 4.6: EDIV-Patch: Zusätzlicher Eintrag für die XML-Szenarios

Zusätzlich werden in diesem Schritt alle vhost- und management-IP-Adressen, die die virtuellen Router nach dem Start haben werden, den IP Tunnels zugewiesen und entsprechende Einträge in der `/etc/hosts` vorgenommen, damit die virtuellen Router nach dem Start „wie gewohnt“ mit Befehlen wie z.B. `ssh r1` erreichbar sind.

Hauptfunktion: PET beenden

- wird gestartet über `./pet.sh -dev\_destroy`
- macht alle PET-Modifikationen rückgängig
- mit Vorsicht zu genießen

Damit über EDIV erstellte Szenarien mit XTPeer betrachtet werden können, müssen IP Tunnels erstellt, Hosts hinzugefügt, VNUML-Szenarien gepatched und EDIV-Daten auf alle Rechner übertragen werden. Das kann Probleme bereiten, wenn man hintereinander unterschiedliche Szenarien oder Cluster-Konfigurationen analysieren will. Diese Funktion startet die Kernel-Module neu, die TUN/TAP-Devices (also alle VNUML-Schnittstellen) steuern, um diese aus dem System zu entfernen. Zusätzlich werden alle VLANs deaktiviert und temporäre Dateien entfernt, die EDIV zurückgelassen haben könnte[5]. Probleme dabei sind jedoch nur bei SUSE Linux Enterprise Server aufgetreten, fehlerhaftes Beenden der Simulation muss also kein Fehler bei EDIV sein. Alle beteiligten Rechner, also Kontroll- und Cluster-PCs, sollten nach dieser Funktion wieder im Ursprungszustand sein. Die Lösung für heruntergefahrene eth0-Schnittstellen war im Falle des SLES-Clusters ein 5min-Cronjob[6], der „ifconfig eth0 up“ ausführte.

[5] EDIV terminiert sehr selten „sauber“, da während des Prozesses die eth0-Schnittstellen der Cluster heruntergefahren werden können

[6] Zeitbasierter Prozess-Scheduler für Linux. Zu vordefinierten Zeiten werden in regelmäßigen Abständen Prozesse ausgeführt
Wiki-Link: http://en.wikipedia.org/wiki/Cron

Kapitel 5

Fazit und Ausblick

Mit dem Abschluss der Arbeit sollen die Ergebnisse und Erfahrungen während der Diplomarbeit noch einmal zusammenfassend dargestellt und bewertet werden. Neben dem Fazit wird im Hinblick auf weiterführende Arbeiten beschrieben, welche Probleme in Zukunft zu lösen sind, um die Performanz des RMTI-Algorithmus und der Diagnose-Programme zu verbessern.

5.1 Review: Zimulator vs. XTPeer

Beide Programme lassen sich zur Untersuchung von RIP und RMTI verwenden, allerdings haben beide Lösungen Vor- und Nachteile.
XTPeer hat 2010 zwei Updates erhalten. Ein „Facelift“ der Darstellung von Topologien wurde implementiert, diese wird nun über JGraph dargestellt und ist etwas intuitiver als vorher. Desweiteren lassen sich Topologien, die man manuell innerhalb des XTPeers zu einer logischen Struktur verschieben kann, auch in der angepassten Form abspeichern. Standardmäßig setzte vorher XTPeer nach Start des Szenarios die Router in eine Reihe, was zweckmäßig, aber nicht übersichtlich war. Jetzt wird die angepasste Struktur über Koordinaten in XML-Kommentaren gespeichert, die das Szenario nicht ändern, aber die Ansicht im XTPeer erhält. Das zweite Update ist eine statistische Datenauswertung, die bestimmt, wie oft in einem Szenario CTI-Probleme aufgetreten bzw. nicht aufgetreten sind und gibt auch Balkendiagramme aus. Die erhobenen Daten werden in einer MySQL-Datenbank gespeichert.
Funktional wurde XTPeer in Kapitel 2.4 genauer beschrieben, es lassen sich Router und Netze live im Betrieb über die Oberfläche konfigurieren, deaktivieren oder aktivieren, Routing-Table-Entries auslesen und über eine Konfigurationsdatei auch

Ausfallszenarien präzise wiederholen, um zum Beispiel Auswirkungen eines CTI zu beobachten. Netz-, Routerausfälle sowie auch CTI-Situationen werden in den Tabellen sichtbar hervorgehoben, weswegen XTPeer sich nicht nur zur Analyse, sondern auch für Demonstrationszwecke eignet. Auch Metrik-Graphen für alle Netze, assoziiert zu allen Routern, sind jederzeit abrufbar. 3 unterschiedliche Algorithmen sind anwählbar um im aktuell geladenen Netzwerk topologische Schleifen auszumachen und Konfigurationen vorzuschlagen, wie man CTI-Situationen provozieren kann, die sogenannten Generatoren.
Problematisch wird diese Vielfalt an dargestellten Informationen in größeren Netzwerken. Gerade beim Thema „RMTI in großen Netzwerken“ stößt man mit XTPeer an ein Speicherproblem. Die erhobenen Daten werden komplett im Arbeitsspeicher gespeichert und je schneller diese erzeugt werden, desto weniger Zeit bleibt bis XTPeer den freien Arbeitsspeicher komplett aufgebraucht hat und zum Absturz des XTPeers führt. Im Falle von 25-Router-Szenarien waren dies zwischen 20-25 Minuten Arbeitszeit bis zum Programmabsturz, mit steigender Router/Netz-Zahl verringerte sich die Zeit deutlich (101-Router Szenarien sorgten nach 4 Minuten für vollen Arbeitsspeicher). Durch die Nutzung von EDIV+PET (Kapitel 4) kann man die Arbeitszeit auf circa 8-12 Minuten für diese Netzwerkgröße erhöhen. Dodurch wird der Arbeitsspeicher von VNUML freigegeben, allerdings ist dies immernoch keine ausfallsichere Arbeitsmethode. Ein weiteres Problem ist die Übersicht, denn die Reiter-Darstellung in XTPeer ist nicht sortierbar, sodass es mehr Zeit in Anspruch nimmt, bis man einen bestimmten Router und darunter ein bestimmtes Netz gefunden hat. Große Netze haben zudem 20-40 unterschiedliche topologische Schleifen in verschiedener Größe, diverse Versuche haben gezeigt, dass die Generator-Algorithmen mit großen Netzen überfordert sind und das Programm zum Absturz bringen. Solange die Speicherauslastung bei XTPeer nicht behoben wird, indem die erhobenen Daten nicht vollständig im Speicher behalten, sondern nach der Verarbeitung ausgelagert werden, kann man den XTPeer nicht für die Analyse großer Netzwerke mit mehr als 30 virtuellen Maschinen verwenden. Angemerkt sei auch, dass das Netz 192.168.0.0/16 fest im XTPeer als Management-Netz für VNUML-Maschinen einprogrammiert ist, andere Netzwerkmasken werden nicht akzeptiert.
Zimulator ist ein neues in Perlscript geschriebenes Programm, welches in der Lage ist, die angefallenen Daten der RIP-Daemons über einen Parser zu analysieren. Es gibt keine Möglichkeit direkt in den Programmablauf einzugreifen, alles muss vor der Simulation konfiguriert werden. Ausfälle und Messungen werden über eine Konfigurationsdatei festgelegt. Der Ausgabe-String wird in einer Textdatei

gespeichert, die man mit Zimulator interpretieren kann (Traffic, Konvergenzzeit, Topologie-Informationen etc). Die Analyse geschieht asynchron zum Ablauf von VNUML, die Daten werden mit Hilfe von TCPDump auf der Festplatte festgehalten und erst nach der Simulation über einen Parser ausgewertet. Genauer wird die Arbeitsweise Zimulator in Kapitel 2.5 und 8.2 beschrieben. Mit Zimulator können die errechneten Ergebnisse auch ausgewertet werden, ein ausgegebener Result-String zeigt so die besten, schlechtesten und mittleren Konvergenzzeiten einer Reihe von Simulationen. Topologien können über die Perl-Library GraphViz leicht als Bilddatei ausgegeben werden, auch wenn dies nicht immer optimal aussieht. Zwar legt Zimulator Startsequenzen für diverse im Dateisystem implementierte Protokolle an, aber einen Protokollparser gibt es nur für das RIP Protokoll. Ein großer Vorteil von Zimulator ist auch die autonome Wiederholbarkeit von Versuchen. Die Abfolge der Simulationen wird über eine Stapelverarbeitungsdatei gesteuert, die pro Zeile vorschreibt, welches Szenario mit welcher Konfiguration und welchem Protokoll wie oft wiederholt werden soll. Für ausgewählte Szenarien lassen sich so mehrere Versuche direkt hintereinander durchführen, ohne den PC dauerhaft beaufsichtigen zu müssen.

Die Idee RIP-Pakete aus den TCP-Dumps für die Analyse zu benutzen ist gut, aber die Methode ist nach wie vor indirekt und kann zu Fehlmessungen führen, wenn die TCP-Dump-Dateien nicht korrekt gespeichert werden. Der Parser errechnet die Differenz zweier Zeitstempel, die Zeitstempel werden innerhalb der TCP-Dump-Dateien lokalisiert, der erste ist der First-Timestamp, der entweder zu Beginn einer Messung oder nach einem Device-Ausfall (Router oder Netz) gesetzt wird. Der Last-Timestamp wird ermittelt, indem Änderungen der RIP-Pakete verglichen werden. Sendet RIP keine ändernden Update-Informationen mehr, wird angenommen, dass das Netz zu diesem Zeitpunkt konvergent ist und der Last-Timestamp wird gesetzt. Da das Programm der reinen Analyse dient und längst nicht so viel Entwicklungsarbeit geleistet wurde wie bei XTPeer offieriert Zimulator keinen „Eyecandy“, also keine benutzerfreundliche GUI. Die Konfiguration ist für große Netze etwas umständlich, die Infinity-Metrik muss an 3 Stellen des Zimulators auf 64 geändert werden, damit diese auch übernommen wird (zu ändernde Dateien sind `zimulator.pl`, `.zimulatorc` und `/modules/Configuration.pm`).

Ein weiteres Problem entsteht, wenn die Netzwerke größer werden. Die Wahrscheinlichkeit eines Programmabsturtzes ist nicht so hoch wie bei XTPeer, aber bei mehr als 100 Routern ist die Verarbeitung der TCPDump-Dateien anfällig für Abstürze. In aktuellen Versuchen hat ein Mitschnitt eines simulierten Netzes 50 Megabytes, diese

Datenmenge muss der Parser in einem Anwendungsbeispiel für 133 Netze durchgehen, was hohe Anforderungen an das Hostsystem stellt (CPU und Arbeitsspeicher). Das Problem ist hier nicht das Programm, sondern wahrscheinlich das Betriebssystem. Diese Art von Fehler trat in 200 Versuchen ungefähr 12x auf. Das Risiko kann man nicht vermeiden, aber mit zusätzlichen Ressourcen verringern. Die obere Grenze der aktuellen Testmaschine mit 8 Gigabytes Arbeitsspeicher waren eben gerade etwa 100 Router, bei der 3,5 Gigabytes für VNUML und vermutlich 3 Gigabytes Arbeitsspeicher für die Verarbeitung der TCP-Dumps verwendet werden.

Eigenschaft	**XTPeer**	**Zimulator**
Obere Grenze Topologie-Größe[1]	20 Router (30 mit EDIV)	100 Router
GUI	✓	✗
Vorkonfigurierte Szenarien	✓	✓
Stapelverarbeitung	✗	✓
Konvergenz-Analyse	✓ (synchron)	✓ (asynchron)
Manipulation der Daemons	✓	✓
CTI-Analyse	✓	✗
Traffic-Analyse	✗	✓
Zusammenfassung der Ergebnisse	✓ [2]	✓ [3]
EDIV-Kompatibel	✓ [4]	✗
Frei wählbares Management-Netz	✗	✓

Im Vergleich ist Zimulator allein für die Feststellung von Konvergenzzeiten und Datenaufkommen das bessere, aber nicht perfekte, Werkzeug. Die Simulationen können vorbereitet werden und auch im aktiven Prozess noch hinzugefügt werden. Ergebnisse lassen sich einfach über eine Unterfunktion des Zimulators auswerten und auch graphisch darstellen. Die Funktionen des XTPeers sind über die GUI optisch

[1] Gemessen mit 8Gigabyte Arbeitsspeicher
[2] XTPeer-Ergebnisse sind nur visuell vergleichbar
[3] Zimulator Result-Strings können zusammengefasst und in Diagramm-Grafiken übersetzt werden
[4] nur mit PET Routing Tunnels, siehe Kapitel 4

aufbereitet und für Demonstrationen und Analysen von CTI-Problemen geeignet. Die visuelle Darstellung unterstützt den Benutzer darin, die Zusammenhänge zu verstehen. Allerdings benötigt die Darstellung vor allem im Zusammenhang mit großen Netzwerken so viel Speicher, dass eine längere Beobachtung eines Netzes gegenwärtig nicht möglich ist. Auch bei XTPeer können Simulationen über eine Konfigurationsdatei gesteuert werden, um vergleichbare Ergebnisse zu erhalten. Diese müssen jedoch zu jeder Zeit überwacht werden. Als wissenschaflicher Nachweis dienen bei Zimulator die TCP-Dump-Dateien und bei XTPeer ein eigenes Dateiformat, in dem die Routing-Tabellen und weitere Daten gespeichert werden. Bei Zimulator ist jedoch die die Nachvollziehbarkeit bei Ausfall-Szenarien nicht gegeben. Löst eine Konfiguration einen Ausfall aus, so wird der Zeitpunkt des Ausfalls zwar gespeichert, bei der erneuten Auswertung kommt es aber noch zu Fehlern. Zimulator kann TCP-Dump-Dateien erneut auswerten. Dateien, die einen Ausfall aufweisen, können aber nicht neu berechnet werden, weil der richtige Timestamp nicht übergeben wird. Jedoch könnte man den Code dahingehend modifizieren, dass dieser fehlende Timestamp nach der Registrierung einer Ausfallroute (im der TCP-Dump-Datei als Infinity-Metrik erkennbar) gespeichert wird.
Bei Coldstart-Konvergenzen gibt es dieses Problem nicht. Bei XTPeer lassen sich Versuche und Ergebnisse 1:1 aus den gespeicherten Dateien herauslesen. Weitere Kritikpunkte an Zimulator sind nicht vollständig implementierte Fehler-Ausgaben. So erhält man eine missverständliche Ausgabe, die auf einen Programmfehler hinweist, wenn man Zimulator mit leerer Stapelverarbeitungsdatei aufruft, was einen Benutzer verwirren könnte. Auch Abstürze von VNUML werden nur rudimentär an den Benutzer weitergegeben. Das Programm ist sehr zweckmäßig, allerdings muss man sich deutlich mehr Wissen aneignen, um Zimulator zu benutzen, als es bei XTPeer der Fall ist. Hauptkritikpunkt bei XTPeer ist neben dem Speicherproblem auch die Übersicht. Ein Vorschlag wäre es, ab einer bestimmten Netzwerkgröße kaskadierende Menüs für die Router zu implementieren, die Reiterdarstellung lohnt sich ab circa 15 Routern nicht mehr.
Um in die Thematik einzusteigen, ist es empfehlenswert, den XTPeer zu nutzen. Die Nutzung ist intuitiv und die Ergebnisse sind schnell begreifbar. Wer ernsthaft Versuche mit unterschiedlichen Topologien machen möchte, ist ebenfalls mit XTPeer gut beraten. Wenn die Versuchsanzahl höher wird oder man den PC automatisch nur nach Ergebnissen suchen lassen will, ist Zimulator attraktiver. Je größer die Netzwerktopologie ist, desto weniger übersichtlich wird sie in XTPeer dargestellt. Sobald der Arbeitsspeicher voll ist stürzt das Programm ab, weil die Daten auch nach der

Verarbeitung im Arbeitsspeicher gehalten werden und diesen damit blockieren. Deshalb ist bei größeren Netzwerken Zimulator die bessere Wahl, weil der Arbeitsspeicher nicht so stark belastet wird wie bei XTPeer. Das indirekte Auslesen des Zimulators von RIP-Daten aus TCPDump-Dateien ist jedoch ein komplizierter und dadurch auch fehleranfälliger Vorgang, der näher untersucht werden muss, am besten dann, wenn man beide Programme unter gleichen Bedinungungen einsetzen kann, die bei großen Netzen unmöglich zu bewerkstelligen sind.

5.2 RIP vs. RMTI in großen Netzwerken

Die Messergebnisse in Kapitel 3.3.1ff sind vollständig über den Zimulator errechnet worden. Die Unterschiede in den Konvergenzzeiten zeigen, dass RMTI bei den Standard-Aufgaben entweder gleichschnell oder bis zu 20% langsamer arbeitet als RIP. Standard-Aufgaben sind zum einen die Verteilung von Routing-Informationen über regelmäßige Updates, zum anderen Ausfall-Informationen über Triggered Updates. Der Grund, warum es in diversen Fällen langsamer konvergiert, ist mit hoher Wahrscheinlichkeit eine Eigenschaft des Careful-Modus von RMTI, der in größeren Schleifen Updates länger als notwendig zurück hält, bis entweder die RIP Timeout Time oder der Careful-Value[5] von RMTI erreicht ist, sodass eventuell richtige Routen erst deutlich später in die Tabelle eingetragen werden[6]. In einer Schleife ist dies im schlimmsten Falle eine Zeit von [Umfang des Loops] * 5 Sekunden.
Die ersten Ergebnisse, bei denen dieser starke Unterschied zu beobachten war, war ein VNUML-Dateisystem von September 2009. Dieses Dateisystem wurde auch in den meisten Tests verwendet (vorgestellt in Kapitel 2.3, ripmti-hello.img). Ergebnisse einer weiterentwickelten Version, die den Caraful-Modus einsetzt, liegen zwar vor, sind aber nur zum Teil in die Messergebnisse eingeflossen. Diese haben gezeigt, dass das Problem mit verschachtelten und/oder großen topologischen Schleifen, auch weiterhin vorkommen kann.
Versuche mit großen Netzen wurden vor allem für Coldstarts, auch mit den sogenannten Short Timings, also RIP-Timings von 10/30/20 Sekunden, getestet. Diese und auch die Ausfalltests haben gezeigt, dass der Careful-Modus zu langen Konvergenzzeiten führen kann, wenn RMTI Updates verzögert, die den Simple Loop Test

[5] Zeitspanne, die RMTI abwartet, falls eine Route den Simple Loop Test nicht besteht, bevor sie verworfen wird

[6] Vorraussetzung dafür ist, dass die zurückgehaltene Route, während sie noch in der Tabelle steht, erneut empfangen wird. Falls dies nicht passiert, wird dieser Eintrag verworfen

nicht bestehen.
Verzögerte Routen in größeren Netzen können ebenfalls die Konvergenzzeit verlängern, da mit einem größeren Loop auch der Careful-Value deutlich erhöht wird. Die getesteten Szenarien haben alle mehr als eine mögliche topologische Schleife, Loops mit mehr als 6 Routern erzeugen einen Careful-Value von 6 * 5sek = 30sek, da pro Hop 5 Sekunden gewartet wird. Das könnte Testergebnisse mit bis zu 300% höherer Konvergenzzeit erklären.
Mitte 2010 wurden neue Modifikationen von RMTI in ein neues Dateisystem implementiert, die eventuell diese Schwäche des Algorithmus lösen oder abschwächen könnten. Die aktuelle Version des RMTI arbeitet ebenfalls mit dem Careful-Modus, jedoch wäre es für zukünftige Untersuchungen sehr sinnvoll, die Zeit pro Hop zum Zurückhalten der Route, die vom Algorithmus abgelehnt wird, entweder über eine Einstellung oder abhängig von der Simple Loop Metrik zu reduzieren. Vor allem in Tests mit kurzen Timings war die Wartezeit höher als die Timeout-Zeit, was teilweise zu hohen Konvergenzzeiten führte. Allerdings kamen diese Ergebnisse nicht oft vor, man müsste die entsprechenden Szenarien noch einmal genauer untersuchen und testen, ob der Careful-Modus regelmäßig zu diesen Problemen führt oder ob es sich um einen Messfehler durch das Zimulator-Tool handelt.
Die variable Infinity-Metrik des modifizierten RIP-Daemons RMTI macht das Protokoll skalierbar, denn die Eliminierung der Routing-Loops funktioniert auch in großen Netzwerken, wie z.B. in Kapitel 3.11 auch zu sehen ist. Die Funktion, die diese CTIs generiert, war allerdings nicht zu 100% erfolgreich. Bei Netzwerken mit mehreren Schleifen ist jedoch auch zu beobachten, dass die RMTI-Konvergenzzeit insgesamt etwas höher ist als bei RIP. Diese Beobachtung ist protokollabhängig und steht nicht in Zusammenhang mit der Performanz des Host-Systems. Es wurde mit einem Intel Core2Duo und einem Intel Core2Quad gearbeitet, weitere Leistungsdaten der beiden Host-Systeme waren identisch. Die Ergebnisse haben sich, was Konvergenzzeit und Datenaufkommen angeht, ergänzt. Die Unterschiede der benutzten Topologien unterstützen die These, dass mit einer steigenden Anzahl von Netzen und Schleifen die Konvergenzzeiten von RMTI stärker ansteigen als bei RIP. Es treten zwar keine Routing-Loops auf, dafür aber in manchen Fällen eine starke Verzögerung hervorgerufen durch Careful-Values. Eventuell wurde das Problem durch die Implementierung von „Hello“-Nachrichten, einer Methode, die benachbarte Router in regelmäßigen kurzen Abständen Nachrichten austauschen lässt, um die Konnektivität zu prüfen, schon gelöst. Tests mit dieser Methode müssten in großen Netzwerken ebenfalls noch durchgeführt werden.

5.3 Die Nutzung von EDIV mit XTPeer

Mit Hilfe des Scripts P.E.T. können nun beliebige Szenarien angepasst werden, um mit XTPeer auch auf ausgelagerte VNUML-Szenarien (gestartet über EDIV) zugreifen zu können. Nichtsdestotrotz besteht weiterhin das Problem, dass XTPeer enorm viel Arbeitsspeicher verbraucht, weil die gesammelten Daten der RIP-Daemons vollständig darin gespeichert werden. In Zukunft müsste also am XTPeer gearbeitet werden, sodass Daten entweder vom Arbeitspeicher in eine Datenbank ausgelagert werden oder die verarbeiteten Daten in komprimierterer Form im Arbeitsspeicher bleiben. Der aktuelle Verbrauch des XTPeers erhöht sich mit steigender Router- und Netz-Anzahl nahezu exponentiell und Versuche können nur wenige Minuten andauern, bevor das Programm mangels freiem Speicher abstürzt.
Das P.E.T.-Programm selbst greift stark in die von Linux bereitgestellte Netzwerkumgebung ein. Neben den von VNUML verwalteten TUN/TAP-Devices werden die von EDIV benötigten VLANs und die von PET verwalteten IP Tunnel erzeugt. Damit diese funktionieren, wird auch die Datei `/etc/hosts` verändert. Unter den meisten Distributionen stellt es kein Problem dar, aber unter Suse Linux Enterprise Server führte ein Herunterfahren der verteilten Simulationen auch zum Herunterfahren der primären Netzwerk-Schnittstelle. Wenn diese Lösung weiter verwendet werden soll, empfielt es sich, eine andere Distribution für den Cluster zu verwenden. Die Vorteile von EDIV sind vor allem ein performanterer Start von Szenarien, ohne die Quelldateien stark verändern zu müssen. VNUML-Prozesse werden mit EDIV parallel auf beliebig vielen Rechnern gestartet. Die notwendigen Veränderungen werden von P.E.T. übernommen. Abhängig davon, wie sich XTPeer entwickelt, kann diese Lösung später erneut benutzt werden, um größere Simulationen mit EDIV durchzuführen. Durch die Nutzung von EDIV ist es nicht notwendig, einen in der Anschaffung teuren, sehr performanten Rechner zu verwenden, um sehr große Szenarien starten und auswerten zu können. Es ist möglich, diese auch auf mehrere kleine Rechner zu verteilen.

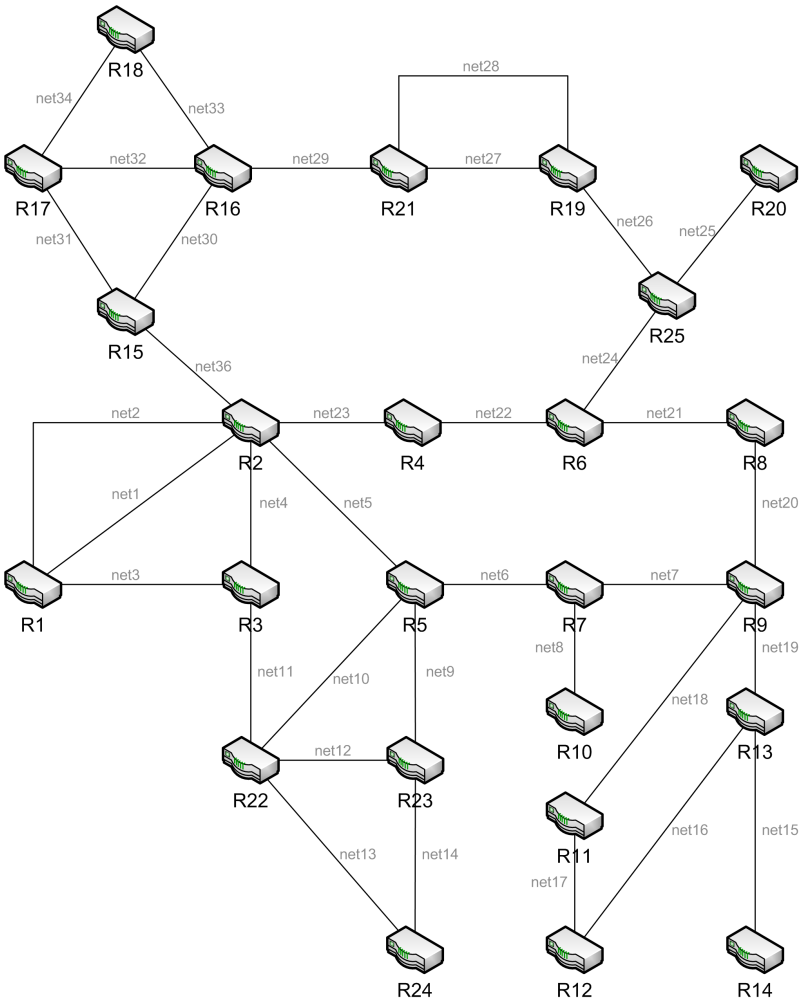
Kapitel 6

Anhang A: Topologien und Konfigurationen

Dieses Kapitel zeigt Graphen und benutzte ZVF-Files (Zimulator-Steuerdateien) für die benutzten Szenarien. Die Messergebnisse wurden mit unterschiedlichen Timings, RIP-Konfigurationen und Ausfallsimulationen erhoben, die in Kapitel 3 verglichen verglichen. Alle Dateien befinden sich auch auf der beiliegenden DVD.

6.1 25 Router Szenario

Graph des Szenarios:



Legende		
Szenario mit 25 Routern & 35 Netzen		
Symbol	Anzahl	Beschreibung
	25	Router

Abbildung 6.1: Graph der 25-Router-Simulation

Zimulator-Datei:

```
net1,net2,net3,net4,net5,net6,net7,net8,net9,net10,net11,net12,net13,net14,net15,
    net16,net17,net18,net19,net20,net21,net22,net23,net24,net25,net26,net27,net28,
    net29,net30,net31,net32,net33,net34,net35

r1 net1,net2,net3
r2 net1,net2,net4,net5,net23,net35
r3 net3,net4,net11
r4 net23,net22
r5 net5,net6,net9,net10
r6 net21,net22,net24
r7 net6,net7,net8
r8 net20,net21
r9 net7,net20,net18,net19
r10 net8
r11 net17,net18
r12 net17,net16
r13 net15,net16,net19
r14 net15
r15 net35,net30,net31
r16 net30,net29,net33,net32
r17 net31,net32,net34
r18 net33,net34
r19 net27,net28,net26
r20 net25
r21 net29,net28,net27
r22 net11,net10,net12,net13
r23 net12,net9,net14
r24 net13,net14
r25 net24,net25,net26
```

Code 6.1: Zimulator Datei der 25-Router-Simulation

6.2 50 Router Szenario

Graph des Szenarios:

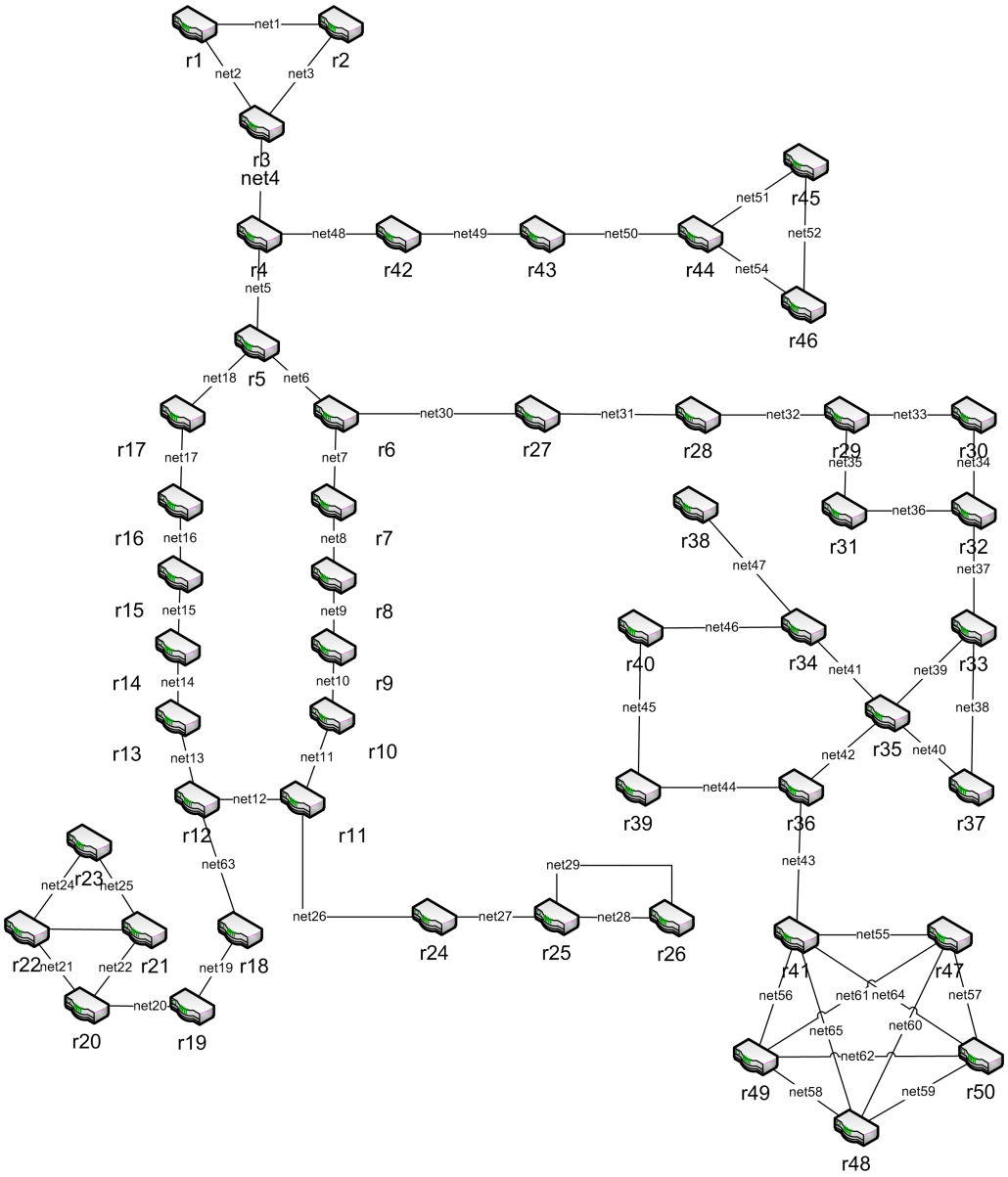


Abbildung 6.2: Graph der 50-Router-Simulation

Zimulator-Datei:

```
net1,net2,net3,net4,net5,net6,net7,net8,net9,net10,net11,net12,net13,net14,net15,
    net16,net17,net18,net19,net20,net21,net22,net23,net24,net25,net26,net27,net28,
    net29,net30,net31,net32,net33,net34,net35,net36,net37,net38,net39,net40,net41,
    net42,net43,net44,net45,net46,net47,net48,net49,net50,net51,net52,net53,net54,
    net55,net56,net57,net58,net59,net60,net61,net62,net63,net64,net65

r1 net1,net2
r2 net1,net3
r3 net2,net3,net4
r4 net4,net5,net48
r5 net5,net6,net18
r6 net6,net30,net7
r7 net7,net8
r8 net8,net9
r9 net9,net10
r10 net10,net11
r11 net11,net12,net26
r12 net12,net13,net63
r13 net13,net14
r14 net14,net15
r15 net15,net16
r16 net16,net17
r17 net17,net18
r18 net63,net19
r19 net19,net20
r20 net20,net21,net22
r21 net22,net23,net25
r22 net21,net23,net24
r23 net24,net25
r24 net26,net27
r25 net27,net28,net29
r26 net28,net29
r27 net30,net31
r28 net31,net32
r29 net32,net33
r30 net33,net34
r31 net35,net36
r32 net34,net36,net37
r33 net37,net38,net39
r34 net41,net47,net46
r35 net39,net41,net40,net42
r36 net42,net44,net43
r37 net38,net40
r38 net47
r39 net44,net45
r40 net45,net46
r41 net43,net55,net56
r42 net48,net49
r43 net49,net50
r44 net50,net51,net53,net54
r45 net51,net52
```

```
48 r46 net52,net53,net54
49 r47 net57,net55,net60,net61
50 r48 net58,net59,net60,net65
51 r49 net56,net58,net61,net62
52 r50 net57,net59,net62,net64
```

Code 6.2: Zimulator Datei der 50-Router-Simulation

6.3 75 Router Szenario

Graph des Szenarios:

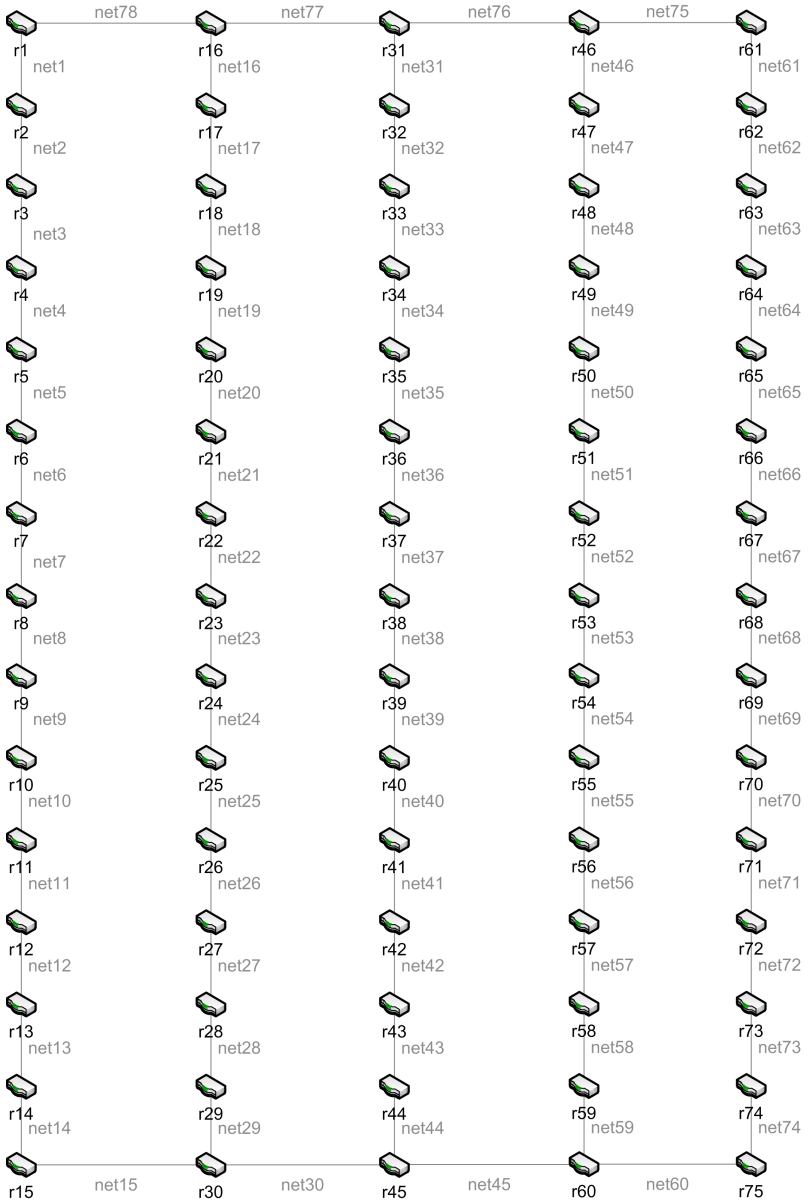


Abbildung 6.3: Graph der 75-Router-Simulation

Zimulator-Datei:

```
net1,net2,net3,net4,net5,net6,net7,net8,net9,net10,net11,net12,net13,net14,net15,
    net16,net17,net18,net19,net20,net21,net22,net23,net24,net25,net26,net27,net28,
    net29,net30,net31,net32,net33,net34,net35,net36,net37,net38,net39,net40,net41,
    net42,net43,net44,net45,net46,net47,net48,net49,net50,net51,net52,net53,net54,
    net55,net56,net57,net58,net59,net60,net61,net62,net63,net64,net65,net66,net67,
    net68,net69,net70,net71,net72,net73,net74,net75,net76,net77,net78

r1 net1,net78
r2 net2,net1
r3 net3,net2
r4 net4,net3
r5 net5,net4
r6 net6,net5
r7 net7,net6
r8 net8,net7
r9 net9,net8
r10 net10,net19
r11 net11,net10
r12 net12,net11
r13 net13,net12
r14 net14,net13
r15 net15,net14
r16 net16,net77,net78
r17 net17,net16
r18 net18,net17
r19 net19,net18
r20 net20,net19
r21 net21,net20
r22 net22,net21
r23 net23,net22
r24 net24,net23
r25 net25,net24
r26 net26,net25
r27 net27,net26
r28 net28,net27
r29 net29,net28
r30 net30,net29,net15
r31 net31,net77,net76
r32 net32,net31
r33 net33,net32
r34 net34,net33
r35 net35,net34
r36 net36,net35
r37 net37,net36
r38 net38,net37
r39 net39,net38
r40 net40,net39
r41 net41,net40
r42 net42,net41
r43 net43,net42
r44 net44,net43
```

```
47 r45 net45,net44,net30
48 r46 net46,net76,net75
49 r47 net47,net46
50 r48 net48,net47
51 r49 net49,net48
52 r50 net50,net49
```

Code 6.3: Zimulator Datei der 75-Router-Simulation

6.4 101 Router Szenario

Graph des Szenarios:

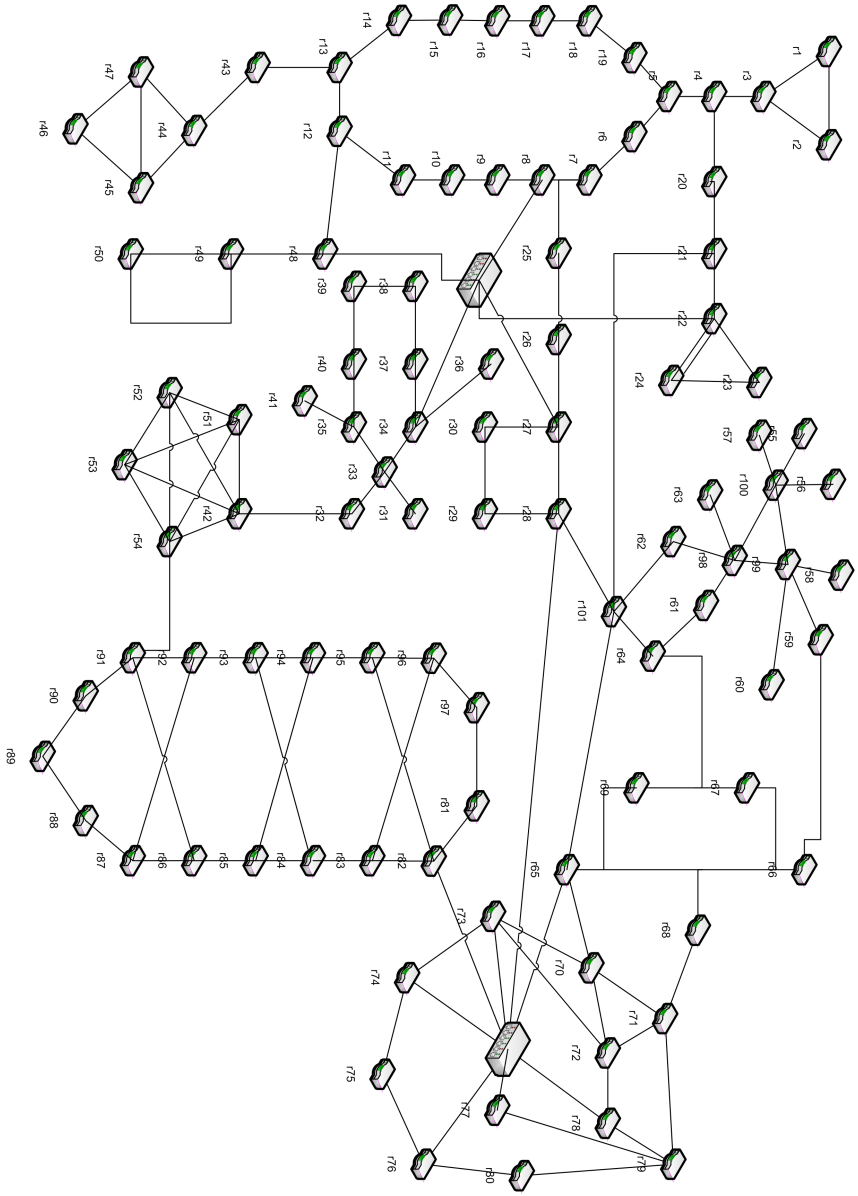


Abbildung 6.4: Graph der 101-Router-Simulation

Zimulator-Datei:

```
net1,net2,net3,net4,net5,net6,net7,net8,net9,net10,net11,net12,net13,net14,net15,
    net16,net17,net18,net19,net20,net21,net22,net23,net24,net25,net26,net27,net28,
    net29,net30,net31,net32,net33,net34,net35,net36,net37,net38,net39,net40,net41,
    net42,net43,net44,net45,net46,net47,net48,net49,net50,net51,net52,net53,net54,
    net55,net56,net57,net58,net59,net60,net61,net62,net63,net64,net65,net66,net67,
    net68,net69,net70,net71,net72,net73,net74,net75,net76,net77,net78,net79,net80,
    net81,net82,net83,net84,net85,net86,net87,net88,net89,net90,net91,net92,net93,
    net94,net95,net96,net97,net98,net99,net100,net101,net102,net103,net104,net105,
    net106,net107,net108,net109,net110,net111,net112,net113,net114,net115,net116,
    net117,net118,net119,net120,net121,net122,net123,net124,net125,net126,net127,
    net128,net129,net130,net131,net132,net133

r1 net1,net2
r2 net1,net3
r3 net2,net3
r4 net4,net5,net6
r5 net6,net7,net8
r6 net7,net9
r7 net9,net10,net11
r8 net11,net12,net13
r9 net13,net14
r10 net14,net15
r11 net15,net16
r12 net16,net17,net18
r13 net18,net19,net20
r14 net20,net21
r15 net21,net22
r16 net22,net23
r17 net23,net24
r18 net24,net25
r19 net8,net25
r20 net5,net26
r21 net26,net27,net28
r22 net12,net28,net29,net30,net31
r23 net29,net32
r24 net32,net30,net31
r25 net10,net33
r26 net33,net34
r27 net12,net34,net35,net36
r28 net27,net36,net37,net38
r29 net37,net39,net40
r30 net35,net39
r31 net40,net41,net42,net43
r32 net42,net44,net45
r33 net43,net44,net46,net47
r34 net12,net46,net48,net54
r35 net47,net52,net53
r36 net54
r37 net48,net49
r38 net49,net50
r39 net50,net51
```

```
42 r40 net51,net52
43 r41 net53
44 r42 net45,net55,net56,net57,net58
45 r43 net19,net125
46 r44 net125,net126,net127
47 r45 net126,net128,net129
48 r46 net129,net130
49 r47 net127,net128,net130
50 r48 net12,net17,net131
51 r49 net131,net132,net133
52 r50 net132,net133
53 r51 net55,net59,net60,net61
54 r52 net56,net61,net62,net63
55 r53 net57,net60,net63,net64
56 r54 net58,net59,net62,net64,net65
57 r55 net99
58 r56 net100
59 r57 net101
60 r58 net103
61 r59 net104,net106
62 r60 net105
63 r61 net94,net98
64 r62 net91,net93
65 r63 net95
66 r64 net92,net98,net108
67 r65 net90,net38,net107,net113
68 r66 net106,net107
69 r67 net107,net108
70 r68 net107,net109
71 r69 net107,net108
72 r70 net110,net113,net114,net115
73 r71 net109,net110,net111,net112
74 r72 net111,net115,net116,net117
75 r73 net38,net114,net116,net118
76 r74 net38,net118,net119
77 r75 net119,net120
78 r76 net38,net120,net121
79 r77 net38,net123
80 r78 net38,net117,net124
81 r79 net112,net122,net123,net124
82 r80 net121,net122
83 r81 net66,net67
84 r82 net38,net67,net68,net69
85 r83 net69,net70,net71
86 r84 net71,net72,net73
87 r85 net73,net74,net75
88 r86 net75,net76,net77
89 r87 net77,net78,net79
90 r88 net79,net80
91 r89 net80,net81
92 r90 net81,net82
93 r91 net65,net76,net82,net83
94 r92 net78,net83,net84
95 r93 net72,net84,net85
```

```
96  r94 net74,net86,net87
97  r95 net68,net87,net88
98  r96 net41,net70,net88,net89
99  r97 net66,net89
100 r98 net93,net94,net95,net96,net97
101 r99 net96,net102,net103,net104,net105
102 r100 net97,net99,net100,net101,net102
103 r101 net27,net90,net91,net92
```

Code 6.4: Zimulator Datei der 101-Router-Simulation

Kapitel 7

Anhang B: Programmcode

In diesem Kapitel befindet sich der Inhalt des Bashscripts pet.sh, welches verwendet wird um IP-Tunnel zwischen lokalem Rechner und VNUML-Maschinen im EDIV-Cluster herzustellen und die Szenarien, die dafür verwendet werden sollen, zu modifizieren. Es wird in Kapitel 4.3 erklärt. In Kapitel 8.4 befindet sich ein HowTo, wie man es einsetzen kann.

```
#/bin/bash
# creating ip routing tunnels via sh-script
### P.E.T. -- Punching EDIV Tunnels

### BEGIN OF EDIT AREA #
# Path information for EDIV and VNUML

# scenariofile which has to run with P.E.T.
SCENARIOFILE="ediv_testing_25.xml"
VNUMLPATH="/usr/local/share/vnuml/"
EDIVCONFIGPATH="/usr/local/etc/ediv/cluster.conf"
VNUMLIMAGE="mini_fs"              # File Name of the VNUML-image that you use for simulations
SCREENRC_INSTALLED="true"         # if set to false, the option -screen_install will work, so the logview function can
                                  # be used properly. True cancels the -screen_install process, so it won't accidently
                                  # damage an existing .screenrc.
# local_ip = adress of your localhost
LOCAL_IP="141.26.68.21"           # rnetquad
# local_if = interface to be used by localhost, in EDIV-speaking: the controlling computer (eth0, wlan0 etc)
LOCAL_IF="eth0"
# linux domain of all cluster computers, that are defined as "host" in EDIV's cluster.conf
DOMAIN="uni-koblenz.de"            # no subdomains or backslashes
# ip adresses, defining tunnels from localhost to cluster pc #x (variable)
```

```
23 TUNL_LOC_TO_UML[0]="192.168.1.101"
24 TUNL_LOC_TO_UML[1]="192.168.1.102"
25 TUNL_LOC_TO_UML[2]="192.168.1.103"
26 TUNL_LOC_TO_UML[3]="192.168.1.104"
27 TUNL_LOC_TO_UML[4]="192.168.1.105"
28 # ip adresses, defining tunnels from cluster pc #x to localhost (variable)
29 TUNL_UML_TO_LOC[0]="192.168.1.111"
30 TUNL_UML_TO_LOC[1]="192.168.1.112"
31 TUNL_UML_TO_LOC[2]="192.168.1.113"
32 TUNL_UML_TO_LOC[3]="192.168.1.114"
33 TUNL_UML_TO_LOC[4]="192.168.1.115"
34 # Segmenation-algorithm, which is configured in cluster.conf. Only RoundRobin ist
       supported. Just an user-information
35 # that does not hold any importance. If you think you're funny, type in "doesn't
       matter"
36 SEGMENTATION="round robin"
37
38 ### END OF EDIT AREA #
39 # The script begins here, unless you're modifying the code, don't tamper with
       forces you don't understand ;)
40 ### PLEASE DO NOT CHANGE ANYTHING FROM HERE ON #
41
42 # 1) Variables used to control several functions
43 SSHKEYURL="leer"                  # URL (e.g. http://www.uni-koblenz.de/~zealot/
       id_dsa_kali.pub)
44                                   #  of the controlling PC for EDIV. Default is "
                                         leer", for no key.
45 SSHKEYFILE="leer"                 # Name of a local key-file, in the same directory
        as the pet-script.
46                                   # (e.g. id_dsa_kali.pub). Default is "leer", for
                                         no key. Both Variables will be filled
47                                   # during script runtime
48 ARGUMENT_TWO="leer"       # I really hated the editing-stuff in this file, so now
       it should all be done over
49                  # arguments
50
51 # 2) Subroutines and mini-functions
52 PETVERSION="pet v0.09 for EDIV"    # Version, what else?
53                                                                                #
54 # 2.1) creates an Array of all PCs that are part of the EDIV-cluster. Information
       is being parsed from the ediv.conf
55 #      looks for entries like "host = netuml02"
56 getclustermembers() {
57   if [ ! -f $EDIVCONFIGPATH ]; then
58     echo
59     echo -e "\e[1;31mError"
60     echo -e "\t\033[0myour EDIV configuration file was not found. check for typos
           in your pet configuration ... *abort*"
61     echo
62     exit 1
63   fi
64   TEMPCLUSTER=$(more +10 $EDIVCONFIGPATH |grep "^host = " |awk -F "host = " '{print
         $2}')
```

```
65    COUNTER=0
66    for i in $TEMPCLUSTER
67    do
68      CLUSTERMEMBER[$COUNTER]=$i
69      COUNTER=`expr $COUNTER + 1`
70    done
71  }
72
73  # 2.2) helping the user to CREATE a set of ssh-keys. These keys can be copied into
        the VNUML image in order to fix the
74  #       infamous "public key denied" error when connecting to a VM (--> 2.3 )
75  create_ssh_key() {
76    echo
77    echo -e "\033[1;4;34m$PETVERSION\033[0m - How to create SSH-Keys"
78    echo
79    echo -e "\t1. install the openssh-client pakage (e.g. debian: apt-get install
          openssh-client)"
80    echo -e "\t2. create RSA and DSA keys using"
81    echo -e "\t\tssh-keygen -t rsa"
82    echo -e "\t\tssh-keygen -t dsa"
83    echo -e "\t\thit ENTER until the process is finished. New keyfiles will be
          located at /root/.ssh/"
84    echo -e "\tWhich one you use is pretty much doesn't matter. DSA *signs* faster,
          but RSA is better in *verifying*."
85    echo
86    echo -e "\tBe sure you use an existing key in each VNUML-Scenario (XML-Tag: <
          ssh_key>/root/.ssh/id_dsa.pub</ssh_key>)"
87    echo
88    exit 1
89  }
90
91  # 2.3) imports a ssh-key-file (local or from URL) into the VNUML-image you chose in
        line 18. Virtual machines can be
92  #       accessed without passwords (imperative for EDIV-Scenarios) and works around
       the "public key denied" bug
93  import_ssh_key() {
94    echo
95    echo -e "\033[1;4;34m$PETVERSION\033[0m - SSH key import function for VNUML
          filesystems"
96    echo -e "\tvnuml-path:                    \033[0;33m$VNUMLPATH\033[0m"
97    echo -e "\tused filesystem:               \033[0;33m$VNUMLIMAGE\033[0m"
98    echo -e "\tsupplied secondary argument:   \033[0;33m$ARGUMENT_TWO\033[0m"
99    echo -e "\tyou are about to write your public key into your vnuml filesystem"
100   echo -e "\tyou will be able to access your virtual machines without a password
          and fix the public key denied error"
101   echo -e "\tplease know what you're doing ;)"
102   echo -e "\tif you dont't, press CTRL-C now - otherwise press ENTER to proceed."
103   read
104
105   if [ $ARGUMENT_TWO = "leer" ]; then
106    echo
107    echo -e "\e[1;31mError\033[0m"
108    echo -e "\tinvalid key-file - use ./pet.sh -ssh <keyfile> to insert the key into
           your VNUML image."
```

```
109     echo
110     exit 1
111   fi
112   sleep 1
113   echo "mounting mini-image"
114   if [ ! -d $VNUMLPATH/filesystems/mntpoint/ ]; then
115     mkdir $VNUMLPATH/filesystems/mntpoint/
116   fi
117   mount -o loop $VNUMLPATH/filesystems/$VNUMLIMAGE $VNUMLPATH/filesystems/mntpoint/
118   sleep 1
119   if [ ! -f $VNUMLPATH/filesystems/$VNUMLIMAGE ]; then
120     echo
121     echo -e "\e[1;31mError"
122     echo -e "\t\033[0myour mini-Image from VNUML is missing or the path is not
          correct"
123     echo -e "\tcheck your configuration in the first lines of this script"
124     echo -e "\t...aborting"
125     echo
126     exit 1
127   fi
128   echo "... creating backup of authorized_keys, an existing backup will be purged"
129   sleep 1
130   rm -f $VNUMLPATH/filesystems/mntpoint/root/.ssh/authorized_keys_BACKUP
131   echo "... creating backup of authorized_keys"
132   sleep 1
133   cp $VNUMLPATH/filesystems/mntpoint/root/.ssh/authorized_keys $VNUMLPATH/
          filesystems/mntpoint/root/.ssh/authorized_keys_BACKUP
134   echo "... adding key to the authorized_keys file"
135   sleep 1
136   cat /root/.ssh/download/$SSHKEYFILE >> $VNUMLPATH/filesystems/mntpoint/root/.ssh/
          authorized_keys
137   echo "... removing temporary files"
138   sleep 1
139   rm -f /root/.ssh/download/$SSHKEYFILE
140   echo "unmounting mini-image"
141   echo
142   umount $VNUMLPATH/filesystems/mntpoint/
143   sleep 1
144   echo -e "\e[1;32mSuccess\033[0m"
145   echo -e "\tYou successfully added a public key to the vnuml-image."
146   echo -e "\tThe computer owning that key should now be able to access every
          virtual machine started with that filesystem."
147   echo -e "\tIf that's not the case, try restarting the ssh-server."
148   echo
149   exit 1
150 }
151
152 # 2.4) Function to create a customized .screenrc for user root. It allows tabbed
      browsing though simulation logs
153 #       written by EDIV. Not vital, but makes control easier. .screenrc design by
      Marcel Jacobs (zimon@uni-koblenz.de)
154 #       Function is only used by "install_logview_screenrc", if the user decides to
      do so
155 install_logview_screenrc_execute() {
```

```
156   echo "# kill startup message" >> /root/.screenrc
157   echo "startup_message off" >> /root/.screenrc
158   echo "# detach on hangup" >> /root/.screenrc
159   echo "autodetach on" >> /root/.screenrc
160   echo "# define a bigger scrollback, default is 100 lines" >> /root/.screenrc
161   echo "defscrollback 2048" >> /root/.screenrc
162   echo "# shell" >> /root/.screenrc
163   echo "shell /bin/bash" >> /root/.screenrc
164   echo "# Automated Screensessionconfiguration:" >> /root/.screenrc
165   echo "# What programs are started where, if a new screensession is started." >> /
          root/.screenrc
166   echo "#screen -t Allgemein 0 bash" >> /root/.screenrc
167   echo "#screen -t Uni 1 /home/zimon/bin/unibash" >> /root/.screenrc
168   echo "#screen -t top 2 top" >> /root/.screenrc
169   echo "# use F7 and F8 to cycle trough the windows" >> /root/.screenrc
170   echo "bindkey -k k7 prev" >> /root/.screenrc
171   echo "bindkey -k k8 next" >> /root/.screenrc
172   echo "bindkey -k k1 detach" >> /root/.screenrc
173   echo "bindkey -k k2 screen" >> /root/.screenrc
174   echo "#term rxvt" >> /root/.screenrc
175   echo "# putty bindings" >> /root/.screenrc
176   echo "#bindkey \"^[OC\" next" >> /root/.screenrc
177   echo "#bindkey \"^[OD\" prev" >> /root/.screenrc
178   echo "#bindkey \"^[OD\" prev" >> /root/.screenrc
179   echo "#bindkey \"^[OD\" prev" >> /root/.screenrc
180   echo "#termcap stuff" >> /root/.screenrc
181   echo "#termcapinfo xterm ti@:te@" >> /root/.screenrc
182   echo "# An alternative hardstatus to display a bar at the bottom listing the" >>
          /root/.screenrc
183   echo "# windownames and highlighting the current windowname in blue. (This is
          only" >> /root/.screenrc
184   echo "# enabled if there is no hardstatus setting for your terminal)" >> /root/.
          screenrc
185   echo "hardstatus on" >> /root/.screenrc
186   echo "hardstatus alwayslastline" >> /root/.screenrc
187   echo "hardstatus string \"%{.bW}%-w%{.rW}%n %t%{-}%+w %=%{..G} %H %{..Y} %d.%m.%y
           %c:%s \"" >> /root/.screenrc
188   echo "#hardstatus string \"%{rw} * | %H * $LOGNAME | %{bw}%c %D | %{-}%-Lw%{rw
          }%50>%{rW}%n%f* %t %{-}%+Lw%<\"" >> /root/.screenrc
189   echo "#hardstatus string \"%{=b}%-w%{=bru}%n %t%{-}%+w\"" >> /root/.screenrc
190   echo "# Switch visuell bell off" >> /root/.screenrc
191   echo "vbell off" >> /root/.screenrc
192   echo
193   echo -e "\e[1;32mSuccess\033[0m"
194   echo -e "\t.screenrc has been imported"
195   echo -e "\tRemember that the SCREENRC_INSTALLED variable is still set 'false', if
          no more changes are required"
196   echo -e "\tyou should switch it back to 'true'"
197 }
198
199 # 2.5) Function control function, which let's you decide if you want that colorful
      screen-setup.
200 #       copies the generated .screenrc into /root/. Can be called with ./pet.sh -
      installscreen
```

```
201 install_logview_screenrc(){
202   if [ "$SCREENRC_INSTALLED" == "true" ]; then
203     echo
204     echo -e "\033[1;4;34m$PETVERSION\033[0m - shiny new .screenrc installation"
205     echo
206     echo -e "\taccording to your configuration the .screenrc addition has already
          been installed."
207     echo -e "\tIf you wish to copy the .screenrc to /root once again, set the
          SCREENRC_INSTALLED variable manually to 'false'"
208     echo
209     exit 1
210   else
211     echo
212     echo -e "\033[1;4;34m$PETVERSION\033[0m - shiny new .screenrc installation"
213     echo -e "\tused configuration:           \033[0;33m./$CONFIG_FILE\033[0m"
214     echo
215     echo -ne "\tchecking for existing /root/.screenrc "
216     echo -ne ". "
217     sleep 1
218     echo -ne ". "
219     sleep 1
220     echo -ne ". "
221     sleep 1
222     if [ -f /root/.screenrc ]; then
223       echo -e "\033[0;31mfile exists\033[0m"
224       echo -e "\tif you wish to replace it type 'y' (indeed, without capslock). You
            can watch logs with your own .screenrc, but then"
225       echo -ne "\tyou have to set the SCREENRC_INSTALLED variable manually to 'true
          '. Replace /root/.screenrc? [y/n]: "
226       read TEMP2
227       if [ $TEMP2 = y ]; then
228     echo
229     echo -e "\tmv /root/.screenrc /root/.screenrc.OLD"
230     rm -fr /root/.screenrc.OLD
231     mv /root/.screenrc /root/.screenrc.OLD
232     touch /root/.screenrc
233     echo -ne "\twriting new .screenrc "
234     echo -ne ". "
235     sleep 1
236     echo -ne ". "
237     sleep 1
238     echo -ne ". "
239     sleep 1
240     install_logview_screenrc_execute
241       else
242     echo
243     echo -e "\t.screenrc was not harmed ... *abort*"
244     echo
245     exit 1
246       fi
247     else
248       echo -e "\033[0;32mfile does not exists\033[0m"
249       echo -e "\tit will be created now"
250       touch /root/.screenrc
```

```
251       echo
252       echo -ne "\twriting new .screenrc "
253       echo -ne ". "
254       sleep 1
255       echo -ne ". "
256       sleep 1
257       echo -ne ". "
258       sleep 1
259       install_logview_screenrc_execute
260       echo
261       exit 1
262     fi
263   fi
264   exit 1
265 }
266
267 # 2.6) This function can be called with ./pet.sh -logview. It opens a new terminal
      (xterm) with the currently active
268 #       EDIV logs.
269 logview() {
270   getclustermembers
271   echo
272   echo -e "\033[1;4;34m$PETVERSION\033[0m - Logviewer"
273   echo -e "\tvnuml-path:                    \033[0;33m$VNUMLPATH\033[0m"
274   echo -e "\tused filesystem:               \033[0;33m$VNUMLIMAGE\033[0m"
275   echo -ne "\tused cluster-members:         \033[0;33m"
276   TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
277   for (( c=0; c<=$TEMP; c++ ))
278   do
279     if [ "$c" != "$TEMP" ]
280     then
281       echo -ne "${CLUSTERMEMBER[$c]}, "
282     else
283       echo -e "${CLUSTERMEMBER[$c]}\033[0m"
284     fi
285   done
286   echo
287   echo -e "\tthis function opens an Xterminal with tabs (navigate with F7 and F8).
        It requires a modified .screenrc."
288   echo -ne "\tchecking for existing /root/.screenrc "
289   echo -ne ". "
290   sleep 1
291   echo -ne ". "
292   sleep 1
293   echo -ne ". "
294   sleep 1
295   if [ -f /root/.screenrc -a $SCREENRC_INSTALLED = false ]; then
296     echo -e "\033[0;31mfile exists, yet your configuration states it's not the
          right one\033[0m"
297     echo -e "\tuse the \e[1m-installscreen\e[m to install and/or set the
          SCREENRC_INSTALLED variable manually to 'true'"
298     echo
299     exit 1
300   elif [ -f /root/.screenrc -a $SCREENRC_INSTALLED = true ]; then
```

```
301       echo -e "\033[0;32mexists, good boy ;)\033[0m"
302       echo
303     fi
304     echo -e "\tkilling existing screen sessions"
305     killall screen
306     sleep 1
307     echo
308     echo -e "\tif not exists: creating logfiles"
309     TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
310     for (( c=0; c<=$TEMP; c++ ))
311     do
312       touch /tmp/${CLUSTERMEMBER[$c]}.${DOMAIN}_log
313       echo -e "\t\ttouch /tmp/${CLUSTERMEMBER[$c]}.${DOMAIN}_log"
314     done
315     sleep 1
316     echo -e "\tstarting screen"
317     nohup xterm -bg black -fg grey88 -geometry 125x30 -e "screen -S logview -t ${
          CLUSTERMEMBER[0]} tail -f /tmp/${CLUSTERMEMBER[0]}.${DOMAIN}_log" &
318     echo -e "\t\tnohup xterm -bg black -fg grey88 -geometry 125x30 -e "screen -S
          logview -t ${CLUSTERMEMBER[0]} tail -f /tmp/${CLUSTERMEMBER[0]}.${DOMAIN}_log
          " &"
319     sleep 2
320     TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
321     for (( c=1; c<=$TEMP; c++ ))
322     do
323       screen -S logview -X screen -t ${CLUSTERMEMBER[$c]} tail -f /tmp/${
            CLUSTERMEMBER[$c]}.${DOMAIN}_log
324       echo -e "\t\tscreen -S logview -X screen -t ${CLUSTERMEMBER[$c]} tail -f /tmp/$
            {CLUSTERMEMBER[$c]}.${DOMAIN}_log"
325       sleep 1
326     done
327     rm nohup.out -rf
328     echo
329     echo -e "\tfinished. You should see a new window with ${#CLUSTERMEMBER[@]} logs"
330     exit 1
331   }
332
333   # 2.7) EDIV allocates routers in a logical way, but its ip addresses are awkwardly
        distributed. This function
334   #       implements the correct order to "forsee" which router get's which ip address
        , so right routes can be pasted
335   #       into the scenario files.
336   weird_ip_algorithm() {
337   #  right now P.E.T. can determine ip addresses for up to 7 cluster members. If you
        want to extend the compability
338   #  simulate a scenario on a larger cluster and determine which host got which ip
        adresses.
339     getclustermembers
340     echo
341     echo -e "\t\tNumber of Clustermachines: \033[0;32m${#CLUSTERMEMBER[@]}\033[0m"
342     echo -en "\t\tIP-adresses allocated to the Clustermembers should be in following
          order: \033[0;32m"
343     case "${#CLUSTERMEMBER[@]}" in
344   #   actually all cases have one and the same array, only in 3 versions. C.I.C. is
```

```
    the array for text output queues
#   C.I.C.R. is the array which is used in functions and C.I.A. is an unfinished attempt to combine both. Maybe
#   I'll finish this nasty hardcoding at the very end, right now it's working.
      1     )   CLUSTER_IP_CHAOS=( 1 )
            CLUSTER_IP_CHAOS_REVERSE=( 1 )
            CLUSTER_IP_ALLOC=( 0 );;
      2     )   CLUSTER_IP_CHAOS=( 1 2 )
            CLUSTER_IP_CHAOS_REVERSE=( 2 1 )
            CLUSTER_IP_ALLOC=( 1 0 );;
      3     )   CLUSTER_IP_CHAOS=( 3 1 2 )
            CLUSTER_IP_CHAOS_REVERSE=( 2 1 3 )
            CLUSTER_IP_ALLOC=( 1 0 2 );;
      4     )   CLUSTER_IP_CHAOS=( 4 3 1 2 )
            CLUSTER_IP_CHAOS_REVERSE=( 2 1 3 4 )
            CLUSTER_IP_ALLOC=( 1 0 2 3 );;
      5     )   CLUSTER_IP_CHAOS=( 3 4 5 1 2 )
            CLUSTER_IP_CHAOS_REVERSE=( 2 1 5 4 3 )
            CLUSTER_IP_ALLOC=( 1 0 4 3 2 );;
      6     )   CLUSTER_IP_CHAOS=( 6 3 4 5 1 2 )
            CLUSTER_IP_CHAOS_REVERSE=( 2 1 5 4 3 6 )
            CLUSTER_IP_ALLOC=( 1 0 4 3 2 5 );;
      7     )   CLUSTER_IP_CHAOS=( 4 7 3 5 6 1 2 )
            CLUSTER_IP_CHAOS_REVERSE=( 2 1 6 5 3 7 4 )
            CLUSTER_IP_ALLOC=( 1 0 5 4 2 6 3 );;
  esac
  for (( i = 0 ; i < ${#CLUSTER_IP_CHAOS[@]} ; i++ ))
  do
    echo -en "${CLUSTER_IP_CHAOS[$i]} "
  done
  GATEWAYADDRESSARRAY=( )
  TUNNELADDRESSARRAY=( )
  echo -e "\033[0m"
  echo
}

# 2.8) For routing perposes, ip tunnels need to know the vhost and management host ip adress for each virtual machine
#       vhost_ip_numbers and mhost_ip_numbers create those ip-adresses from 192.168.0.0/16 in groups of four
#       right now XTpeer only supports this network, so it was hardcoded here as well

# 2.8.1) Creation of the array vhost_ip - unsorted, offset = 0 required in the scenario file
vhost_ip_numbers() {
  c_netz1=0
  d_netz1=2
  for (( c=0; c < $ROUTERCOUNT; c++ ))
  do
    vhost_ip[$c]="192.168.$c_netz1.$d_netz1"
    if [ $d_netz1 -ge 251 ]; then
      d_netz1=`expr $d_netz1 - 251`
      c_netz1=`expr $c_netz1 + 1`
```

```
393     else
394       d_netz1=`expr $d_netz1 + 4`
395     fi
396   done
397 }
398
399 # 2.8.2) Creation of the array mhost_ip - unsorted, offset = 0 required in the
      scenario file
400 manage_ip_numbers() {
401   c_netz2=0
402   d_netz2=1
403   for (( c=0; c < $ROUTERCOUNT; c++ ))
404   do
405     mhost_ip[$c]="192.168.$c_netz2.$d_netz2"
406     if [ $d_netz2 -ge 250 ]; then
407       d_netz2=`expr $d_netz2 - 250`
408       c_netz2=`expr $c_netz2 + 1`
409     else
410       d_netz2=`expr $d_netz2 + 4`
411     fi
412   done
413 }
414
415 # 2.8.3) corresponding to EDIVs ip address allocation, vhost_ip + mhost_ip array is
       being sorted                                    #
416 vhost_ip_adress_sort() {
417   count=`expr $ROUTERCOUNT / ${#CLUSTERMEMBER[@]}`
418   count_test=`expr $ROUTERCOUNT % ${#CLUSTERMEMBER[@]}`
419   if [ $count_test -ne 0 ]; then
420     count=`expr $count + 1`
421   fi
422   vhost_counter=0
423   for (( cluster_array_counter=0; cluster_array_counter < ${#CLUSTERMEMBER[@]};
        cluster_array_counter++ ))
424   do
425     while_counter=${CLUSTER_IP_ALLOC[$cluster_array_counter]}
426     while [ $while_counter -lt $ROUTERCOUNT ]
427     do
428       vhost_ip_sorted[$while_counter]=${vhost_ip[$vhost_counter]}
429       mhost_ip_sorted[$while_counter]=${mhost_ip[$vhost_counter]}
430       while_counter=$[$while_counter+${#CLUSTERMEMBER[@]}]
431       vhost_counter=$[$vhost_counter+1]
432     done
433   done
434 #  WHICH_HOST=${CLUSTER_IP_CHAOS[$TEMP2]}
435 }
436
437 # 2.9) Following function patches a VNUML scenario file to create ip tunnles from
      the remote side of the network
438 #       in order to be able to connect to this pc, the virtual machines must have a
      route from the VM to the hosting
439 #       cluster pc, which has the correct route to this pc. It is compatible to
      zimulator formed XML-files.
440 #       Each Router will receive an entry like: <route type="ipv4" gw
```

```
    ="192.168.0.94">192.168.1.102/32</route>
tunnel_allocation() {
  getclustermembers
  echo
  echo -e "\033[1;4;34m$PETVERSION\033[0m - Tunnel-Allocation / Scenario-Patch"
  echo -e "\tvnuml-path:                    \033[0;33m$VNUMLPATH\033[0m"
  echo -e "\tused filesystem:               \033[0;33m$VNUMLIMAGE\033[0m"
  echo -e "\tdesignated scenario-file       \033[0;33m$SCENARIOFILE\033[0m"
  echo -ne "\tused cluster-members:         \033[0;33m"
  TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
  for (( c=0; c<=$TEMP; c++ ))
  do
    if [ "$c" != "$TEMP" ]
    then
      echo -ne "${CLUSTERMEMBER[$c]}, "
    else
      echo -e "${CLUSTERMEMBER[$c]}\033[0m"
    fi
  done
  echo
  echo -e "\tparsing \033[0;33m$SCENARIOFILE\033[0m ..."
  ROUTERCOUNT=`grep '<vm name' -c $SCENARIOFILE`
  echo -e "\t\tNumber of Routers: \033[0;32m$ROUTERCOUNT\033[0m"
  NETWORKCOUNT=`grep '<net ' -c $SCENARIOFILE`
  echo -e "\t\tNumber of Networks: \033[0;32m$NETWORKCOUNT\033[0m"
  ROUTINGSCOUNT=`grep '<route ' -c $SCENARIOFILE`
  echo -ne "\t\tNumber of already existing Routes: \033[0;32m$ROUTINGSCOUNT\033[0m"
  echo -e " (if this is larger than zero, the Scenario might already be patched)"
  echo -e "\t\tSegmentation Algorithm: \033[0;32m$SEGMENTATION\033[0m"
  echo
  echo -e "\tThis function patches the scenario in order to use the created tunnles
      ."
  echo -e "\tThe virtual machines need tunnel-routes, too, in order to grant access
      "
  echo -ne "\tto the control computer (this computer). You shouldn't use it on
      scenarios which are already patched. Press ENTER to continue."
  read
  weird_ip_algorithm
  vhost_ip_numbers
  manage_ip_numbers
  vhost_ip_adress_sort
  echo
  echo -e "\tCreating Backup (\033[0;32mbackup_$SCENARIOFILE\033[0m)"
  cp $SCENARIOFILE backup_$SCENARIOFILE
  rm /etc/hosts.BACKUP -f
  cp /etc/hosts /etc/hosts.BACKUP


# 2.9.1) vHosts are allocated to ip tunnels
  TEMP=${#CLUSTERMEMBER[@]}
  echo
  echo -e "\tAdding hosts to the tunnels"
  echo
  echo -e "\tAdding routes to /etc/hosts"
```

```
491   for (( c=0; c < $ROUTERCOUNT; c++ ))
492   do
493
494 # 2.9.2) create an array of strings to be inserted as vnuml-route
495 #        choose an cluster array from the "weird_ip_algorithm" function
496 #        calculate which virtual machine is hosted by which host
497     VNUML_ROUTE[$c]="<route type=\"ipv4\" gw="
498     if [ $TEMP == 1 ]; then WHICH_HOST=0
499     else TEMP2=`expr $c % $TEMP`
500     WHICH_HOST=${CLUSTER_IP_CHAOS[$TEMP2]}
501     fi
502 #        note to self: value of array element = gateway $TUNL_LOC_TO_UML - 1,
        because gateway
503 #        incrementation begins from 1, not 0
504 #DEBUG    echo $TEMP2
505 #    COUNTER43=`expr $WHICH_HOST - 1`
506     TUNL_COUNTER=`expr $TEMP2 + 1`
507 #        note to self: insert vHost ip as gateway to the cluster host, not the
        controling computer
508 #ERROR        = {vhost_ip[$COUNTER42]}
509     MHOST_DIRTY=${mhost_ip_sorted[$c]}
510 #        note to self: complete the string with all collected information
511 #ERROR    VNUML_ROUTE[$c]="${VNUML_ROUTE[$c]}\"$MHOST_DIRTY\">${TUNL_LOC_TO_UML[
        $COUNTER43]}/32</route>"
512     VNUML_ROUTE[$c]="${VNUML_ROUTE[$c]}\"$MHOST_DIRTY\">${TUNL_LOC_TO_UML[$TEMP2
            ]}/32</route>"
513
514 #    echo "echo" >> tunnel_$SCENARIOFILE.sh
515     AUSGABE_COUNTER=`expr $c + 1`
516 #DEBUG    echo "adding route for r$AUSGABE_COUNTER :: ${VNUML_ROUTE[$c]}"
517 #    echo "echo R$AUSGABE_COUNTER @ netuml$WHICH_HOST" >> tunnel_$SCENARIOFILE.sh
518 #    echo "route add -host ${vhost_ip_sorted[$c]} dev l2uml$WHICH_HOST" >>
        tunnel_$SCENARIOFILE.sh
519 #    echo "route add -host ${mhost_ip_sorted[$c]} dev l2uml$WHICH_HOST" >>
        tunnel_$SCENARIOFILE.sh
520 #    echo "route add -host ${mhost_ip_sorted[$c]} dev l2uml$TUNL_COUNTER"
521     route add -host ${mhost_ip_sorted[$c]} dev l2uml$TUNL_COUNTER
522 #DEBUG    echo "route add -host ${vhost_ip_sorted[$c]} dev l2uml$TUNL_COUNTER"
523     route add -host ${vhost_ip_sorted[$c]} dev l2uml$TUNL_COUNTER
524     echo "${vhost_ip_sorted[$c]} r$AUSGABE_COUNTER" >> /etc/hosts
525 #DEBUG    echo "hosts: ${vhost_ip_sorted[$c]} r$AUSGABE_COUNTER"
526   done
527   for (( c=0; c<$ROUTERCOUNT; c++))
528   do
529   ROUTERLINES=$(grep '\<forwarding type="ipv4" />' $SCENARIOFILE -n |awk '{print $1
          }' |sed 's/://')
530   COUNTER=0
531   for i in $ROUTERLINES
532   do
533     TEMP=`expr $i - 1`
534     ROUTERLINE[$COUNTER]=$TEMP
535     COUNTER=`expr $COUNTER + 1`
536   done
537   rm tempfile.xml -f
```

```
538   sed -e "${ROUTERLINE[$c]}a${VNUML_ROUTE[$c]}" $SCENARIOFILE > tempfile.xml
539   rm $SCENARIOFILE
540   mv tempfile.xml $SCENARIOFILE
541  done
542  exit 1
543 }
544
545 # 2.10) dead_device_destroyer deactivates every TUN/TAP and bridge device on every computer in your network
546 #       including all cluster members activated in the cluster.conf and the controling pc. This might be neccessary
547 #       because EDIV doesn't always terminate properly. tunnels by PET or TAPs from VNUML can corrupt new simulations
548 #       EDIV has it's own function, EDIV_CLUSTER_CLEANUP.PL, but it doesn not always work at 100%. DDD is very
549 #       destructive, it reloads kernel modules. So be careful.
550 dead_device_destroyer() {
551   getclustermembers
552   echo
553   echo -e "\033[1;4;34m$PETVERSION\033[0m - Dead Device Destroyer"
554   echo -e "\tvnuml-path:                    \033[0;33m$VNUMLPATH\033[0m"
555   echo -e "\tused filesystem:               \033[0;33m$VNUMLIMAGE\033[0m"
556   echo -ne "\tused cluster-members:         \033[0;33m"
557   TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
558   for (( c=0; c<=$TEMP; c++ ))
559   do
560     if [ "$c" != "$TEMP" ]
561     then
562       echo -ne "${CLUSTERMEMBER[$c]}, "
563     else
564       echo -e "${CLUSTERMEMBER[$c]}\033[0m"
565     fi
566   done
567   echo
568   echo -e "\tSometimes EDIV fails to delete created TUN/TAP and bridge devices, which isn't very nice (especially SUSE systems .. crap)"
569   echo -e "\tIt will disable ALL TUN/TAPs and ALL bridges, so if you have custom interfaces"
570   echo -e "\t(other than lo, eth0 or wlan0) do NOT use this function!"
571   echo -e "\tPET will access all cluster-members and search&destroy the devices by un- and reloading the 'tun' and 'bridge' kernel modules"
572   echo
573   echo -e "\tAre you sure you want to do this?"
574   echo -ne "\tThen press ENTER to continue: "
575   read
576 #  MAGICWORD=PLEASE
577   echo -ne "\t. "
578   sleep 1
579   echo -ne ". "
580   sleep 1
581   echo -ne ". "
582   sleep 1
583   if [ 1 == 1 ]; then  # earlier here was a magic-word-check which was annoying after some uses. ENTER will sufice
```

```
584     echo -e "beginning start sequence\033[0m"
585   else
586     echo -e "you did not say the magic word *sob* - aborting.\033[0m"
587     exit 1
588   fi
589   echo
590   sleep 1
591   echo -e "\tLet the destruction commence!"
592   echo
593   for (( c=0; c<${#CLUSTERMEMBER[@]}; c++ ))
594   do
595     NUMBER=`expr $c + 1`
596     echo -e "\tconnecting to: \033[0;33m${CLUSTERMEMBER[$c]}\033[0m"
597     echo -e "\t\tkilling VM-kernel processes, deleting temporary data"
598     echo -e "\tshutting down tunnels"
599     ip tunnel del l2uml${NUMBER}
600     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root "ip tunnel
          del tunl_uml${NUMBER}"
601     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'killall
          linux'
602     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rm /tmp/
          conf* -rf'
603     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rm /tmp/*
          log -f'
604     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rm /tmp
          /*.xml -f'
605     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rm /root
          /.vnuml/simulations/* -Rf'
606     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rm /root
          /.vnuml/networks/* -Rf'
607     echo -e "\t\tshutting down TUN/TAP driver"
608     echo -e "\t\tssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l
          root 'rmmod tun'"
609     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rmmod tun
          -f'
610     echo -e "\t\treloading TUN/TAP driver"
611     echo -e "\t\tssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l
          root 'modprobe tun'"
612     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'modprobe
          tun'
613     echo -e "\t\tdestroying all bridges"
614     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'rmmod
          bridge -f'
615     echo -e "\t\treloading bridge module"
616     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root 'modprobe
          bridge'
617     echo -e "\t\tremoving VLANs"
618     echo
619     TEMPDESTROY=$(ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l
          root 'ifconfig' |awk '/eth/{print $1}')
620     for i in $TEMPDESTROY
621     do
622     if [ "$i" = "eth0" ] || [ "$i" = "$TEMPDESTROY[@]" ]; then
623       echo -e "\tskipping eth0"
```

```
624     else
625       ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTERMEMBER[$c]} -l root "vconfig
              rem $i"
626     fi
627     done
628   done
629   echo -e "\tfinished. Your cluster has been cleansed ;)"
630   echo
631   echo -e "\tNotice: Some distributions (suse ... suxx) have difficulties with this
        removing-script. Removing the VLANs created"
632   echo -e "\tby EDIV causes the eth0 interface to shut down, too. If that happens,
       create a cronjob that performs 'ifconfig eth0 up'"
633   echo -e "\tevery 5 minutes or so. There seems to be no other workaround."
634   echo
635   exit 1
636 }
637
638 # 2.11) creates tunnels, local and remote, but does not add any hosts, names of the
       tunnels are hard-coded.
639 #         Tunnel Localhost to netumlxx  == l2umlxx
640 #         Tunnel netumlxx to Localhost  == tunl_umlxx
641
642 tunnel_puncher() {
643 #   if [ $TUNNEL_TAG == "false" ]; then
644 #     echo -e "\033[1;4;34m$PETVERSION\033[0m - Tunnel Puncher"
645 #     echo -e "\t-t parameter was not set ... *abort*"
646 #     echo
647 #     exit 1
648 #   fi
649   getclustermembers
650   echo
651   echo -e "\033[1;4;34m$PETVERSION\033[0m - Tunnel Puncher"
652   echo -e "\tvnuml-path:                    \033[0;33m$VNUMLPATH\033[0m"
653   echo -e "\tused filesystem:               \033[0;33m$VNUMLIMAGE\033[0m"
654   echo -ne "\tused cluster-members:         \033[0;33m"
655   TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
656   for (( c=0; c<=$TEMP; c++ ))
657   do
658     if [ "$c" != "$TEMP" ]
659     then
660       echo -ne "${CLUSTERMEMBER[$c]}, "
661     else
662       echo -ne "${CLUSTERMEMBER[$c]}\033[0m"
663     fi
664   done
665   echo -e " (${#CLUSTERMEMBER[@]})"
666   echo
667   echo -e "\tif the orange information is correct, just press ENTER to continue ..
       or abort with CTRL+C"
668   read
669   echo
670   if [ ${#CLUSTERMEMBER[@]} -lt 2 ]; then
671     echo -e "\t${#CLUSTERMEMBER[@]} were found. This script and EDIV works for
          physical networks with MORE THAN ONE member."
```

```
672     echo -e "\tuse VNUML if you have only one computer at your disposal ... *abort*
            "
673     echo
674     exit 1
675   fi
676   sysctl -q -w net.ipv4.conf.all.rp_filter=0              # activate ip forwarding
677   echo -e "\tsysctl -w net.ipv4.conf.all.rp_filter=0"
678   sysctl -q -w net.ipv4.conf.all.forwarding=1
679   echo -e "\tsysctl -w net.ipv4.conf.all.forwarding=1"
680   modprobe ipip                                           # activate neccessary
          modules
681   echo -e "\tmodprobe ipip"
682   modprobe ip_gre
683   echo -e "\tmodprobe ip_gre"
684   echo
685   sleep 1
686   echo -e "\tcreating tunnels, hardcoded name conventions:"
687   echo -e "\t\tlocalhost = tunnel on THIS computer"
688   echo -e "\t\tnetnumlXX = tunnels on remote computers"
689   echo
690   sleep 1
691   NUMBER=0
692   for (( c=0; c<=$TEMP; c++ ))
693   do
694     NUMBER=`expr $c + 1`
695     ip tunnel add l2uml${NUMBER} mode gre local $LOCAL_IP remote ${CLUSTER_PC_IP[$c
          ]} ttl 255
696     ip link set l2uml${NUMBER} up
697     ifconfig l2uml${NUMBER} ${TUNL_LOC_TO_UML[$c]}
698     route add -host ${TUNL_UML_TO_LOC[$c]} dev l2uml${NUMBER}
699     echo -e "\ttunnel localhost to netuml${NUMBER} --finished--"
700     echo
701     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTER_PC_IP[$c]} -l root "sysctl -q
           -w net.ipv4.conf.all.rp_filter=0"
702     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTER_PC_IP[$c]} -l root "sysctl -q
           -w net.ipv4.conf.all.forwarding=1"
703     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTER_PC_IP[$c]} -l root "ip tunnel
           add tunl_uml${NUMBER} mode gre local '${CLUSTER_PC_IP[$c]}' remote '
          $LOCAL_IP' ttl 255"
704     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTER_PC_IP[$c]} -l root "ip link
          set tunl_uml${NUMBER} up"
705     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTER_PC_IP[$c]} -l root "ifconfig
          tunl_uml${NUMBER} '${TUNL_UML_TO_LOC[$c]}'"
706     ssh -2 -o 'StrictHostKeyChecking no' -X ${CLUSTER_PC_IP[$c]} -l root "route add
           -host '${TUNL_LOC_TO_UML[$c]}' dev tunl_uml${NUMBER}"
707     echo -e "\ttunnel netuml${NUMBER} to localhost --finished--"
708     echo
709   done
710   echo -e "\t${#CLUSTERMEMBER[@]} tunnels were created. If you got errors like \"
          SIOCADDRT: Die Datei existiert bereits\" you executed this script before."
711   echo -e "\tCheck with 'ifconfig', that all tunnels exist like planned, then
          continue with ./pet.sh -patch"
712   echo
713   exit 1
```

```
714 }
715
716 # 2.12) after tunnel-creation and allocation, even after adding hosts to the
        tunnels, you are prepared to launch
717 #       this function is rather obsolete, because EDIV can do the same. But it is
        bunt
718 #       additionally you can check if anything is missing
719 start_scenario() {
720   getclustermembers
721   echo
722   echo -e "\033[1;4;34m$PETVERSION\033[0m - i can haz scenarioz"
723   echo -e "\tvnuml-path:                    \033[0;33m$VNUMLPATH\033[0m"
724   echo -e "\tused filesystem:               \033[0;33m$VNUMLIMAGE\033[0m"
725   echo -e "\tdesignated scenario-file       \033[0;33m$SCENARIOFILE\033[0m"
726   echo -ne "\tused cluster-members:         \033[0;33m"
727   TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
728   for (( c=0; c<=$TEMP; c++ ))
729   do
730     if [ "$c" != "$TEMP" ]
731     then
732       echo -ne "${CLUSTERMEMBER[$c]}, "
733     else
734       echo -e "${CLUSTERMEMBER[$c]}\033[0m"
735     fi
736   done
737   echo
738   echo -e "\tThis function launches EDIV, the scenario is defined in your
        configuration file ($SCENARIOFILE). If you want to launch a different"
739   echo -e "\tone, edit your actual configuration. CTRL-C is the way to go then ...
        if the information above is correct, press ENTER."
740   read
741   echo -ne "\tanalyzing data . "
742   sleep 1
743   echo -ne ". "
744   sleep 1
745   echo -ne ". "
746   echo
747   ROUTERCOUNT=`grep '<vm ' -c $SCENARIOFILE`
748   echo -e "\tNumber of Routers: \033[0;32m$ROUTERCOUNT\033[0m"
749   sleep 1
750   NETWORKCOUNT=`grep '<net ' -c $SCENARIOFILE`
751   echo -e "\tNumber of Networks: \033[0;32m$NETWORKCOUNT\033[0m"
752   sleep 1
753   echo -e "\tNumber of Clustermachines: \033[0;32m${#CLUSTERMEMBER[@]}\033[0m"
754   sleep 1
755   echo -e "\tSegmentation Algorithm: \033[0;32m$SEGMENTATION\033[0m"
756   sleep 1
757   echo -ne "\tIdentified IP tunnels: \033[0;32m"
758   NUMBER=0
759   TUNLTMP=""
760   for (( c=0; c<=$TEMP; c++ ))
761   do
762     NUMBER=`expr $c + 1`
763     if [ "$c" != "$TEMP" ]
```

```
764     then
765       TUNLTMP=$(ifconfig | grep l2uml${NUMBER} | awk '{print $1}')
766       if [ "$TUNLTMP" = "" ]; then
767     echo -ne "$TUNLTMP"
768       else
769     echo -ne "$TUNLTMP, "
770       fi
771     else
772       TUNLTMP=$(ifconfig | grep l2uml${NUMBER} | awk '{print $1}')
773       if [ "$TUNLTMP" = "" ]; then
774     TUNLTMP="\033[0;31mnot found"
775       fi
776       echo -e "$TUNLTMP\033[0m"
777     fi
778   done
779   if [ "$TUNLTMP" = "\033[0;31mnot found" ]; then
780     echo -e "\t\tNo IP-Tunnels have been found (may be a false warning). If you
          still want to continue press ENTER, but you won't be able to use XTpeer
          tough."
781     echo -e "\t\tCheck all of your clustermembers defined in your EDIV-config,
          maybe one or more of them are offline."
782     echo -ne "\t\tI recommend you abort with CTRL-C and create some tunnels first
          (\e[1m./pet -t -c\e[m or something)."
783     read
784   fi
785   echo -e "\tEDIV will segment the scenario into following parts, PET hopes you
        have configured a management-network with 192.168.0.0/16"
786   echo -e "\t(note to self: implement a parsing-check for this ...)"
787   echo
788   echo -ne "\tpress ENTER to continue, if the information above is correct. "
789   read
790   echo -e "\tstarting EDIV ..."
791   echo -e "\tediv_ctl.pl -t -s $SCENARIOFILE -vB"
792   ediv_ctl.pl -t -s $SCENARIOFILE -vB
793   echo
794 exit 1
795 }
796
797 # 2.13) pops up after typing in ./pet.sh -help
798
799 zomg_help_me() {
800   echo
801   echo -e "\033[1;4;34m$PETVERSION"
802   echo -e "\033[0muse it to create ip tunnels for your EDIV scenario in order to be
        able to work with the XTpeer"
803   echo
804   echo -e "\t use \e[1m-build_tunnel\e[m to start creating tunnles AFTER you
        checked all settings in your config-section"
805   echo -e "\t\tIt's important to create Tunnels BEFORE you start any scenarios or
        add hosts."
806   echo -e "\t\tPET will try to add hosts to these tunnels, if those don't exist
        your PET will be sad!"
807   echo -e "\t use \e[1m-patch\e[m to add routes to the designated scenario file.
        These routes are the remote connection"
```

```
808   echo -e "\t\tto your ip tunnels. this function will also add non-virtual hosts to
          your tunnel-network."
809   echo -e "\t use \e[1m-start\e[m to start the scenario specified in your
          configuration, that you want EDIV to run."
810   echo -e "\t use \e[1m-help\e[m to see this screen."
811   echo
812   echo -e "\t use \e[1m-dev_destroy\e[m to cleanse the cluster. Sometimes EDIV
          leaves several unused TUN/TAPs and VLANs on every cluster member."
813   echo -e "\t\twhich could interfere with new scenarios. Use this function with
          care."
814   echo -e "\t use \e[1m-ssh\e[m to import a public SSH key into your VNUML
          filesystem, which hosts all your virtual machines."
815   echo -e "\t use \e[1m-create_key\e[m to read how to create a new set of ssh-keys
          for your root-user."
816   echo -e "\t use \e[1m-logview\e[m to open an Xterminal with Tabs (navigate with
          F7 and F8). it requires a modified .screenrc. use -logview for more intel."
817   echo -e "\t\tmight be too invasive for screen-addicts."
818   echo -e "\t use \e[1m-installscreen\e[m to copy a nice looking .screenrc to /root
          , observing logs of your EDIV-Scenario will be easier that way"
819   echo
820   echo -e "\033[0;32mExample: ./pet.sh -build_tunnel yscenario.xml"
821   echo -e "\t\033[0m begins creating tunnels on localhost and every cluster
          computer you have activated in cluster.conf"
822   echo
823   exit 1
824 }
825
826 temp_function_extremenet() {
827     ip tunnel del l2uml01
828     ip tunnel del l2uml02
829     ip tunnel del l2uml03
830     ip tunnel del l2uml04
831     ip tunnel del l2uml05
832     ssh -2 -o 'StrictHostKeyChecking no' -X 141.26.70.109 -l root 'ip tunnel del
            tunl_uml01'
833     ssh -2 -o 'StrictHostKeyChecking no' -X 141.26.70.110 -l root 'ip tunnel del
            tunl_uml02'
834     ssh -2 -o 'StrictHostKeyChecking no' -X 141.26.70.111 -l root 'ip tunnel del
            tunl_uml03'
835     ssh -2 -o 'StrictHostKeyChecking no' -X 141.26.70.112 -l root 'ip tunnel del
            tunl_uml04'
836     ssh -2 -o 'StrictHostKeyChecking no' -X 141.26.70.113 -l root 'ip tunnel del
            tunl_uml05'
837 exit 1
838 }
839
840
841 # Program functions
842 #
843 # 3) main program, with a root-user check and a case box, which reads parameters in
        order to call the right function
844 # checks if user is root go on, if not -> abort
845 if [ "$(id -u)" != "0" ]; then
846   echo
```

```
847   echo -e "\033[1;4;34m$PETVERSION"
848   echo -e "\033[0muse it to create ip tunnels for your EDIV scenario in order to be
          able to work with the XTpeer."
849   echo
850   echo -e "\033[0;32myou have to be root to create ip tunnels or messing with your
          ssh keys. Aborting\033[0m"
851   echo
852   exit 1
853 fi
854
855 # gets all IP-Adresses from Cluster-Members.
856 #echo -e "\tgetting IP-Adresses"
857 getclustermembers
858 TEMP=`expr ${#CLUSTERMEMBER[@]} - 1`
859 for (( c=0; c<=$TEMP; c++ ))
860 do
861    CLUSTER_PC_IP[$c]=`ping -c 1 ${CLUSTERMEMBER[$c]} |awk '{print $3} NR>0{exit};0'
          |sed -e 's/(//g' |sed -e 's/)//g'`
862 #DEBUG   echo -e "${CLUSTERMEMBER[$c]} hat IP-Adresse ${CLUSTER_PC_IP[$c]}"
863 done
864
865 # if nil parameters are provided, help pops up
866 if [[ -z "$@" ]]; then
867   echo
868   echo -e "\033[1;4;34m$PETVERSION"
869   echo -e "\033[0muse it to create ip tunnels for your EDIV scenario in order to be
          able to work with the XTpeer"
870   echo
871   echo -e "\t use \e[1m-build_tunnel\e[m to start creating tunnles AFTER you
          checked all settings in your config-section"
872   echo -e "\t\tIt's important to create Tunnels BEFORE you start any scenarios or
          add hosts."
873   echo -e "\t\tPET will try to add hosts to these tunnels, if those don't exist
          your PET will be sad!"
874   echo -e "\t use \e[1m-patch\e[m to add routes to the designated scenario file.
          These routes are the remote connection"
875   echo -e "\t\tto your ip tunnels. this function will also add non-virtual hosts to
          your tunnel-network."
876   echo -e "\t use \e[1m-start\e[m to start the scenario specified in your
          configuration, that you want EDIV to run."
877   echo -e "\t use \e[1m-help\e[m to see ALL possible parameters of this script."
878   echo
879   echo -e "\033[0;32mExample: ./pet.sh -build_tunnel"
880   echo -e "\t\033[0m begins creating tunnels on localhost and every cluster
          computer you have activated in cluster.conf"
881   echo
882   exit 1
883
884 # abort conditions, pretty much only a check if more than one argument is present
885 elif [[ $# > 2 ]]; then
886   echo
887   echo -e "\033[1;4;34m$PETVERSION\033[0m"
888   echo -e "\tc'mon, it's only a script, please use a maximum of 2 arguments"
889   echo -e "\t\033[0;32mExample:\033[0m ./pet.sh -logview or ./pet.sh -ssh id_rsa.
```

```
      pub"
890   echo
891   exit 1
892 elif [ 1 == 1 ]; then
893
894 # parses all arguments, known arguments trigger stuff (OBSOLETE, for now)
895 #  for x in $@
896 #  do
897 #    case "$x" in
898 #      -t    )    TUNNEL_TAG="true";;
899 #      -P    )    DESTROY_TAG="true";;
900 #      -c    )    CUSTOMCONFIG_TAG="true";;
901 #    esac
902 #  done
903 if [[ $# = 2 ]]; then
904   ARGUMENT_TWO="$2"
905 fi
906
907 # ruft Funktionen auf, die direkt ausgewählt werden können
908   for x in $@
909   do
910     case "$x" in
911       -help      )    zomg_help_me;;
912       -start         )    start_scenario;;
913       -ssh       )    import_ssh_key;;
914       -create_key    )    create_ssh_key;;
915       -logview       )    logview;;
916       -installscreen     )    install_logview_screenrc;;
917       -dev_destroy   )    dead_device_destroyer;;
918       -build_tunnel      )        tunnel_puncher;;
919       -patch             )    tunnel_allocation;;
920       -udkill            )        temp_function_extremenet;;
921     esac
922   done
923 fi
924
925
926 # fool's check, but also OBSOLETE ;)
927 #if [ $TUNNEL_TAG == "true" ] && [ $DESTROY_TAG == "true" ]; then
928 #  echo
929 #  echo "creating AND destroying tunnels at the same time. Humanity never seizes to
      amaze me ... *abort*"
930 #  echo
931 #  exit 1
932 #fi
933 echo -e "\033[1;4;34m$PETVERSION"
934 echo -e "\033[0muse it to create ip tunnels for your EDIV scenario in order to be
     able to work with the XTpeer"
935 echo
936 echo -e "\tif you see this screen, you called an unexpected function or combination
     "
937 echo -e "\t use ./pet.sh \e[1m-help\e[m to see a list of supported parameters"
938 echo
939 exit 1
```

Code 7.1: PET IP Tunnel Script

Kapitel 8

Anhang C: HowTos

In diesem Teil der Dokumentation befinden sich diverse Tutorials, um Teile der Arbeit nachzuvollziehen. Um sich den Umgang mit verteilten Szenarien zu vereinfachen habe ich zudem auch ein kleines Programm (P.E.T) geschrieben, welches die meisten Arbeitsschritte automatisiert. Ich habe diese Informationen zusammengetragen, um sich etwas praxisorientierter in das Thema einzuarbeiten.

8.1 VNUML Installationshilfe

Da das verwendete Diagnose-Tool XTPeer bei größeren Netzwerken sehr viel Arbeitsspeicher benötigt, wird auf den Laborrechnern entsprechend mit 64bit-Betriebssystemen gearbeitet. Das Tutorial wird nur kurz auf die Unterschiede zu 32bit-Systemen eingehen, bei der Pfadgebung, z.B. bei perl libraries, sollten diese jedoch trivial sein. Diese Installationsanleitung ist ergänzend zu der bereits vorhandenen Anleitung der Uni Madrid[1] zu nutzen. Für die Messungen für diese Arbeit wurde Fedora 11 und Fedora 13 64bit genutzt, besondere Gründe dafür gab es keine. Die meisten Voraussetzungen decken sich mit denen in der offiziellen Anleitung.

8.1.1 VNUML Basis-Programme

Verwendete Pakete unter Fedora Linux 13 64bit (Goddard):

- perl
- perl-devel

[1] Official VNUML Installation Guide
http://www.dit.upm.es/vnumlwiki/index.php/Installation

- perl-XML-DOM
- expat
- libxml2
- readline-devel
- fuse-devel
- cpan
- bridge-utils
- flex
- byacc
- bison
- uml-utils (source)
- VNUML v1.8.9 (source)
- diverse perl-module über CPAN

`cpan` (Comprehensive Perl Archive Network)[Hie99] ist ein nützliches Tool um Perl-Module zu installieren, auch wenn sie nicht im Repository der jeweiligen Linux-Distribution zu finden sind. VNUML bringt die meisten Module von sich aus im Paket mit, es ist in der Regel aber besser sie in der neusten Version über `cpan` zu installieren. Die Pakete `expat`, `libxml2`, `vconfig`, `xterm`, `bridge-utils` und `screen` sind ausdrücklich benötigt, damit VNUML lauffähig ist, speziell zu `xterm` ist zu sagen, dass man davon ausgeht die Szenarien aus einer X-Konsole initiiert, die virtuellen Maschinen werden über X-Terminals zugänglich (via Popup). Das Paket `vlan` gibt es im Fedora-Repository unter dem Namen `vconfig`.

```
1 [root@workstation ~]\$ yum install perl perl-devel cpan libxml2 expat xterm screen bridge-utils fuse
      -devel vconfig
```

Code 8.1: VNUML-Installation: Linux-Paketliste

VNUML benötigt die `uml-utilities`, die, wie der Name schon sagt, eine Sammlung von UML-Tools zur Verfügung stellen. Die gibt es nicht mehr im Repository von Fedora und muss deshalb von Hand kompiliert werden. Das Paket gibt es auf der Webseite http://user-mode-linux.sourceforge.net.

```
[root@workstation ~]\$ wget http://user-mode-linux.sourceforge.net/uml_utilities_20070815.tar.bz2
[root@workstation ~]\$ tar -jxvf uml_utilities_20070815.tar.bz2
[root@workstation ~]\$ cd tools-20070815
[root@workstation ~]\$ make all
[root@workstation ~]\$ make BIN_DIR=/usr/local/bin install
```

Code 8.2: VNUML-Installation: Installation der UML Utilities

Unter Umständen müssen vorher noch die Pakete `gcc` und `readline-devel` über `yum` installiert werden, damit der `make`-Befehl funktioniert.
Das eigentliche VNUML-Paket gibt es auf der Webseite http://www.dit.upm.es/vnuml. Die aktuelle Version ist VNUML v1.8.9 (22.05.2009). Zusätzlich zu diesem Paket ist ein UML-Kernel und ein Filesystem für die virtuellen Maschinen notwendig. Ein richtiges Paket gibt es nur für Debian-Systeme, für alle anderen Distributionen nutzt man die Source-Dateien.

```
[root@workstation ~]\$ wget http://prdownloads.sourceforge.net/vnuml/vnuml_1.8.9.orig.tar.gz?
    download
[root@workstation ~]\$ tar -zxvf vnuml_1.8.9.orig.tar.gz
[root@workstation ~]\$ cd vnuml-1.8.9/
[root@workstation ~]\$ ./configure
```

Code 8.3: VNUML-Installation: Download und Konfiguration

Der Befehl `./configure` prüft, ob alle Voraussetzungen für VNUML vom Betriebssystem erfüllt werden. In den meisten Fällen müssen jedoch einige Perl-Module nachinstalliert werden. Dies geschieht wie angesprochen am besten über `cpan`.
Möglichkeit 1: Man überlässt VNUML die Installation der Perl-Libraries.

```
[root@workstation ~]\$ cpan Module::Build
[root@workstation ~]\$ ./configure -with-build_module
```

Code 8.4: VNUML-Installation: Auto-Installation der Perl-Module

Möglichkeit 2: In Ausnahmefällen funktioniert diese Methode aber nicht, es gilt dann die in der Fehlermeldung genannten Module einzeln nachzuinstallieren. `cpan` kann über Parameter oder über die eigene Eingabekonsole bedient werden:

```
[root@workstation ~]\$ cpan
cpan[1]> install Term::ReadKey
/* Installationsdialoge */
cpan[2]> exit
```

Code 8.5: VNUML-Installation: CPAN Konsole

Die Syntax Installation über Parameter ist beispielsweise:

```
[root@workstation ~]\$ cpan Term::ReadKey
[root@workstation ~]\$ ./configure
/* hoffen dass kein Fehler kommt, wenn einer kommt, das entsprechende Modul nachinstallieren */
```

Code 8.6: VNUML-Installation: VNUML Konfiguration

Hinweis: Zur Zeit ist die aktuelle Version des Moduls Net::Pcap v0.16. Diese ist allerdings unbenutzbar (broken) und wurde auch schon länger nicht mehr aktualisiert. Für VNUML ist es daher empfehlenswert die noch funktionierende v0.14 zu nutzen.

```
[root@workstation ~]\$ wget http://www.tcpdump.org/release/libpcap-1.0.0.tar.gz
[root@workstation ~]\$ tar -zxvf libpcap-1.0.0.tar.gz
[root@workstation ~]\$ yum install flex bison
[root@workstation ~]\$ cd libpcap-1.0.0
[root@workstation ~]\$ ./configure
[root@workstation ~]\$ make && make install
[root@workstation ~]\$ wget http://search.cpan.org/CPAN/authors/id/S/SA/SAPER/Net-Pcap-0.14.tar.gz
[root@workstation ~]\$ tar -zxvf Net-Pcap-0.14.tar.gz
[root@workstation ~]\$ cd Net-Pcap-0.14
/* Den Pfad für die gerade installierten pcap libraries muss man natürlich anpassen! */
[root@workstation ~]\$ perl Makefile.PL INC=-I/usr/local/include/pcap LIBS='-L/usr/lib64/libpcap -
    lpcap'
[root@workstation ~]\$ make && make install
```

Code 8.7: VNUML-Installation: Installation von Pcap v0.14

Im Falle eines gerade installierten Fedora 11 Linux[2]

[2]Innerhalb der `cpan`-Umgebung 'help' f+r weitere Informationen eingeben, speziell um den force-mode benutzen zu können falls nötig

```
[root@workstation ~]\$ cpan YAML
[root@workstation ~]\$ cpan Error Exception::Class XML::DOM XML::Checker NetAddr::IP Term::ReadKey
    Net::IPv6Addr Net::Pcap
/* Anfragen des Programms einfach mit [ENTER] bestätigen */
/* Vereinzelt kann es passieren dass Tests des Moduls fehlschlagen, in diesen Fällen einfach über '
    force' installieren */
```

Code 8.8: VNUML-Installation: Quick-CPAN

Unabhängig welche der zwei Möglichkeiten man benutzt hat: wenn das `./configure` erfolgreich durchgelaufen ist, kompiliert und installiert man VNUML mit

```
[root@workstation ~]\$ make && make install
```

Code 8.9: VNUML-Installation: Abschließende VNUML-Installation

8.1.2 UML-Kernel

Jeder von VNUML simulierte Router wird als virtual machine (VM) gestartet, die mit einem Linux-Betriebssystem arbeitet. der UML-Kernel wird von jeder dieser Maschinen verwendet. Das (hoffentlich) erfolgreich installierte VNUML (8.1.1) sollte ein Verzeichnis `/usr/local/share/vnuml` bzw. `/usr/share/vnuml` angelegt haben. Die ausführbaren perl-Scripts sollten in `/usr/local/bin` bzw. `/usr/bin` zu finden und somit von überall aus aufrufbar sein. Um die Kompatiblität mit Online-Tutorials[3] zu sichern, sollte man im Falle einer Installation in `/usr/local/share/vnuml` einen SymLink anlegen, der auf das Verzeichnis `/usr/share/vnuml` zeigt. Dies wird zukünftige Schritte vereinfachen.

```
[root@workstation ~]\$ ln -s /usr/local/share/vnuml/ /usr/share/vnuml
[root@workstation ~]\$ ln -s /usr/local/share/xml/ /usr/share/xml
```

Code 8.10: VNUML-Installation: SymLinks zur Kompatiblität

Im VNUML-Verzeichnis sollten sich nun 3 Unterverzeichnisse befinden

```
[root@workstation ~]\$ ls -l /usr/share/vnuml/
drwxr-xr-x. 2 root root 4096 11. Jun 02:10 examples
drwxr-xr-x. 2 root root 4096 11. Jun 02:14 filesystems
drwxr-xr-x. 2 root root 4096 11. Jun 02:13 kernels
```

Code 8.11: VNUML-Installation: Filetree des VNUML-Verzeichnis

[3] VNUML-Tutorials unter `/usr/share/vnuml/examples` oder
http://www.dit.upm.es/vnumlwiki/index.php/Tutorial

Unter `/examples` sind diverse vorinstallierte Übungsszenarien zu finden, mit denen man erste Gehversuche mit VNUML unternehmen kann. Eine Anleitung dazu findet sich unter http://www.dit.upm.es/vnumlwiki/index.php/Tutorial. Unter `/filesystems` werden die Dateisysteme für die VNUML-VMs abgespeichert (nächstes Kapitel). In das Verzeichnis `/kernels` sollten der Ordnung halber alle Kernels für VNUML installiert werden. Die offizielle Seite[4] bietet mittlerweile einen für UML-Architektur kompilierten Kernel in der Version 2.6.28.10-1m (vom 22.05.2009) an, den man mit einem 64bit Betriebssystem allerdings nicht nutzen kann. Die virtuellen Maschinen haben damit eine Absturzrate von ca. 95%, obwohl der Kernel ohne VNUML problemlos bootet. Vielleicht gibt es in Zukunft weniger Probleme damit, aber zu diesem Zeitpunkt ist der Kernel v2.6.18.1-bb2-xt-4m (vom 28.12.2007) weiterhin empfehlenswert. Um den Zugriff von Szenario-Dateien darauf zu vereinfachen, sollte auch hier ein SymLink genutzt werden:

```
1 [root@workstation ~]\$ tar -zxvf linux-um_2.6.18.1-bb2-xt-4m.orig.tar.gz
2 [root@workstation ~]\$ cd linux-um-2.6.18.1-bb2-xt-4m
3 [root@workstation ~]\$ mv linux-2.6.18.1-bb2-xt-4m /usr/share/vnuml/kernels
4 [root@workstation ~]\$ cd /usr/share/vnuml/kernels
5 [root@workstation ~]\$ ln -sf linux-um-2.6.18.1-bb2-xt-4m linux
6 /* Der Rest des entpackten .tar.gz's wird nicht weiter benötigt */
```

Code 8.12: VNUML-Install: UML-Kernel

Damit dieser Kernel auch in den Szenarien korrekt erkannt wird, muss in jedem XML-Szenario der korrekt Pfad angegeben sein.

```
1 <kernel>/usr/local/share/vnuml/kernels/linux</kernel>
```

Code 8.13: VNUML-Installation: XML-Tag des Kernels

8.1.3 VNUML Dateisystem

Für erste Tests mit VNUML-Szenarien sollte ein Dateisystem über die offizielle Seite[5] bezogen werden, um mit den Beispielszenarien Erfahrungen zu sammeln. Um auch die MTI-Erweiterungen zu nutzen, muss man auf ein Dateisystem der Uni-Koblenz zurückgreifen. Hier gibt es zwei Versionen, eines mit, ein anderes ohne XTPeer-Erweiterung. Abhängig davon, wie man Untersuchungen durchführen

[4] VNUML Download Seite:
http://www.dit.upm.es/vnumlwiki/index.php/Download
[5] http://www.dit.upm.es/vnumlwiki/index.php/Download

will, gilt es das passende Dateisystem zu wählen. Ein Dateisystem ohne XTPeer-Erweiterung findet zum Beispiel Anwendung für Simulationen, die mit dem Zimulator durchgeführt werden sollen. Eingebunden wird das Dateisystem folgendermaßen (empfohlen):

```
[root@workstation ~]\$ mv ripmti-hello.img /usr/share/vnuml/filesystems
/* RMTI-Image der Uni Koblenz */
[root@workstation ~]\$ mv root_fs_tutorial_6.img /usr/share/vnuml/filesystems
/* Offizielles Image */
[root@workstation ~]\$ cd /usr/share/vnuml/filesystems
[root@workstation ~]\$ ln -sf ripmti-hello.img root_fs_mti
[root@workstation ~]\$ ln -sf root_fs_tutorial_6.img root_fs
[root@workstation ~]\$ ln -sf root_fs_tutorial_6.img root_fs_tutorial
```

Code 8.14: VNUML-Install: Dateisystem

Das Erstellen eines SymLinks verringert auch hier die Gefahr von Schreibfehlern innerhalb der XML-Datei. In der Regel sollte VNUML nun lauffähig sein, im Unterverzeichnis `/examples` der VNUML-Installation finden sich kleine Beispiel-Dateien, die sich z.B. über

```
[root@workstation ~]\$ vnumlparser.pl -t /usr/share/vnuml/examples/tutorial_root1.xml -vB
```

Code 8.15: VNUML-Install: Start einer Simulation

starten lassen. Die Virtuellen Maschinen, die in der xml-Datei tutorial_root1.xml definiert wurden, werden nun gestartet. Je nach Rechenkapazität kann das mehrere Minuten dauern. Die definierten Namen der VMs werden in der Datei `/etc/hosts` den neuen IP-Adressen zugewiesen, solange die Simulation aktiv ist. So lässt sich z.B. über `ssh r1` der Router r1 ansteuern. Während der Parameter „-t“ die Simulation startet, beendet der Parameter „-P“ dieselbe[6]. Über den Parameter „-x“ lassen sich die vordefinierten EXEC-Kommendos initiieren, z.B.:

```
[root@workstation ~]\$ vnumlparser.pl -x start@/usr/share/vnuml/examples/tutorial_root1.xml -vB
```

Code 8.16: VNUML-Install: Start eines EXEC-Commands

Mehr Informationen, wie es ab hier weiter geht, findet man in Kapitel 2.3 oder online @ http://www.dit.upm.es/vnumlwiki/index.php/Tutorial.

[6]die Simulation wird eigentlich durch „-d“ beendet, hinterlässt aber zu oft Daten und Devices, die bei neuen Simulationen stören. „-P“ steht für **purge**, es beendet und entfernt alle temporären Dateien

8.2 Zimulator

Die umfangreichen Tests dieser Arbeit wurden mit dem Zimulator-Tool durchgeführt, um diese nachzuvollziehen gibt es hier zusätzlich zur Anleitung[Jac10] noch eine auf Fedora zugeschnittenes HowTo:

Zimulator ist wie VNUML in Perl geschrieben, die Paketabhängigkeiten decken sich weitestgehend. Zusätzlich wird GraphViz benötigt, ein Programm welches Zimulator ermöglicht Graphen von Szenarien als Bilddatei auszugeben.

```
1 [root@workstation ~]\$ yum install graphviz graphviz-devel graphviz-perl
2 [root@workstation ~]\$ cpan Graph GraphViz Test::More
```

Code 8.17: Zimulator-Install: Auflösung der Paket-Abhängigkeiten

Die AG Rechnernetze hat für alle Projekte, die mit Studien- und Diplomarbeiten zu tun haben, ein GIT-System eingerichtet (http://git.uni-koblenz.de). Auch der Zimulator ist hier zu finden.

```
1 [root@workstation ~]\$ yum install git
2 [root@workstation ~]\$ git clone git://git.uni-koblenz.de/zimulator/zimulator.git
```

Code 8.18: Zimulator-Install: Installation des Zimulators

Nach der Installation sollte sich im aktuellen Pfad ein neues Verzeichnis „zimulator“ befinden. Um zu testen, ob alle Funktionen ordnungsgemäß laufen sollte man als root-user einen Test durchführen. Wichtig ist das Endergebnis, da auch Fehlerausgaben getestet werden - wenn am Ende „all tests successful“ ausgegben wird, war die Installation erfolgreich. Falls nicht, fehlen entsprechende Perl-Module, die man über `cpan` installieren sollte.

```
1 [root@workstation ~]\$ ./zimulator --testall
```

Code 8.19: Zimulator-Install: Funktionstest

Die aktuelle Version (September 2010) nutzt noch Einstellungen, die in drei Dateien vorzunehmen sind.

- .zimulatorrc

- zimulator.pl
- modules/Configuration.pm

Die wichtigsten Einstellungen stehen in der versteckten Datei `.zimulatorrc`.

```
[root@workstation ~]\$ cat .zimulatorrc
VISUALIZE_NET_NAMES = "1"
MAXFAIL_DEFAULT = "5"
VNUML_START_PARAMETERS = "-w 300 -Z -B -t"
MANAGEMENT_NET = "192.168.0.0"
GARBAGE_TIMER = "20"
VNUML_EXEC_PARAMETERS = "-x"
VNUML_PATH = "/usr/local/bin"
OSPF_PATH = "/usr/lib/quagga"
INFINITY_METRIC = "64"
MANAGEMENT_NET_OFFSET = "0"
KERNEL = "/usr/local/share/vnuml/kernels/linux"
VM_DEFAULTS = "exec_mode="mconsole""
VNUML_STOP_PARAMETERS = "-P"
NET_MODE = "virtual_bridge"
RAW_TCPDUMP = "0"
MAXRUN_DEFAULT = "15"
DTDPATH = "/usr/local/share/xml/vnuml/vnuml.dtd"
FILESYSTEM = "/usr/local/share/vnuml/filesystems/mini_fs"
ZEBRA_PATH = "/usr/lib/quagga"
SSH_KEY = "/root/.ssh/id_rsa.pub"
MANAGEMENT_NETMASK = "24"
LOGFILE = "logfile.log"
TIMEOUT_TIMER = "30"
RIPD_PATH = "/usr/lib/quagga"
```

Code 8.20: Zimulator: Konfiguration (.zimulatorrc)

Die meisten Werte können in vielen Fällen auf Default belassen werden. Standardmäßig ist als Infinitiy-Metrik 16, der RIP-Standard, eingestellt, da sich diese Arbeit aber mit RMTI-Vergleichen beschäftigt wurde die Metrik in den meisten Fällen auf 64 gestellt. Die Pfade zu DTD, Zebra- und RIP-Daemon sowie zum Filesystem können sich je nach Linux-Distribution leicht unterscheiden. Der Offset zum Management-Network steht standardmäßig auf 100, das ist für Netzwerke mit mehr als 40 Netzen nicht mehr angemessen. Es sollte ein Wert *MANAGEMENT_NET_OFFSET* = „*0*“ eingetragen werden.

Ähnliche Einstellungen stehen auch in der `modules/Configuration.pm`. Zwar ist die Konfiguration über die `.zimulatorrc` meistens ausreichend, aber in Einzelfällen scheint es vorzukommen, dass die Default-Werte der `Configuration.pm` stärker gewichtet werden

und so falsche Einstellungen übergeben werden (eventuell noch ein Bug).

```
[root@workstation ~]\$ cat modules/Configuration.pm
/* Ab Zeile 139 */

# Constants: VNUML XML-file constants
$object->{DTDPATH} = "/usr/local/share/xml/vnuml/vnuml.dtd";
$object->{SSH_KEY} = "/root/.ssh/id_rsa.pub";
$object->{MANAGEMENT_NET} = '192.168.0.0';
$object->{MANAGEMENT_NETMASK} = '24';
$object->{MANAGEMENT_NET_OFFSET} = '0';
$object->{VM_DEFAULTS} = "exec_mode=\"mconsole\"";
$object->{FILESYSTEM} = "/usr/local/share/vnuml/filesystems/mini_fs";
$object->{KERNEL} = "/usr/local/share/vnuml/kernels/linux";
$object->{NET_MODE} = "virtual_bridge";
$object->{ZEBRA_PATH} = "/usr/lib/quagga";
$object->{RIPD_PATH} = "/usr/lib/quagga";
$object->{OSPF_PATH} = "/usr/lib/quagga";

# Constants: VNUML execution configuration
$object->{VNUML_PATH} = '/usr/local/bin';
$object->{VNUML_START_PARAMETERS} = '-w 300 -Z -B -t';
$object->{VNUML_EXEC_PARAMETERS} = '-x';
$object->{VNUML_STOP_PARAMETERS} = '-P';

# Constants: Misc
$object->{MAXFAIL_DEFAULT} = 5;
$object->{MAXRUN_DEFAULT} = 15;
$object->{LOGFILE} = 'logfile.log';
$object->{RAW_TCPDUMP} = 0;
$object->{VISUALIZE_NET_NAMES} = 1;
$object->{TIMEOUT_TIMER} = 30;
$object->{GARBAGE_TIMER} = 20;
$object->{INFINITY_METRIC} = 64;
```

Code 8.21: Zimulator: Konfiguration (Configuration.pm)

Abhängig vom VNUML-Dateisystem muss man auf Konstanten wie $object->ZEBRA_PATH = "/usr/lib/quagga"; achten, denn die Pfade zu den Zebra- und RIP-Daemons können variieren. Meistens ist in den Dateisystemen bereits eine Path-Variable vorhanden, sodass die Daemons pfadlos aufgerufen werden können. In diesem Fall kann man die Konstante für ZEBRA_PATH und RIP_PATH einfach leer lassen. Die Infinity-Metrik ($object->INFINITY_METRIC = 64;) muss auf 64 stehn.

```
1 $object->{ZEBRA_PATH} = "";
2 $object->{RIPD_PATH} = "";
3 $object->{OSPF_PATH} = "";
```

Code 8.22: Zimulator: Zebra- und RIP Daemon Einstellungen, falls Pathes gesetzt sind

In der Datei `zimulator.pl` befinden sich auch Konfigurationselemente, die in wenigen Fällen ausgelöst werden. Auch hier gilt es die Metrik auf 64 anzupassen, um den RMTI zu emulieren. Die Default-Metric steht nämlich in einigen wenigen Tests sonst auf 16. Die Einstellung befindet sich in Zeile 256:

```
1 $configuration->setOption("INFINITY_METRIC",64);
```

Code 8.23: Zimulator: Konfiguration (zimulator.pl

Der Zimulator erstellt über den Parameter „-x“ aus *.zvf-Dateien VNUML-freundliche XML-Dateien. Damit diese auf das gegebene System zugeschnitten sind, muss die obenbeschriebene Konfiguration präzise stimmen. Hinzu kommt, dass jedem virtuellen Rechner, der nach dem start durch VNUML generiert wird, die Konfigurationen für Zebra und RIP vom Host-System übergeben werden (siehe Kapitel 3.2. Ab hier sollte man sich mit der Anleitung zurecht finden –> [Jac10].

8.3 EDIV Installations-Tutorial

Das Programm EDIV (spanisch: Escenarios DIstribuidos con VNUML, englisch: Distributed Scenarios using VNUML) ist eine Script-Sammlung, um VNUML-Szenarien auf einen Computer-Cluster verteilt zu starten. Im Internet gibt es bereits eine englischsprachige Anleitung[7], die allerdings primär die Debian-Linux Distribution behandelt. Dieses Tutorial soll ergänzend dazu Hinweise geben, welche Paketabhängigkeiten zu beachten sind, geht aber auch auf aufgetretene Fehler ein, um Lösungsansätze für viele Probleme zu bieten, selbst wenn man eine andere Distribution benutzt.
Verwendete Pakete unter Fedora Linux 11 64bit (Leonidas):

- alle Pakete aus dem VNUML Installations-Tutorial (8.1)
- xterm

[7] Official EDIV Installation Guide
http://www.dit.upm.es/vnumlwiki/index.php/InstallationEDIV

- screen
- vconfig
- EDIV v0.9.4

```
[root@workstation ~]\$ wget http://downloads.sourceforge.net/project/vnuml/ediv/0.9.4/ediv_0.9.4.
    orig.tar.gz?use_mirror=garr
[root@workstation ~]\$ tar -zxvf ediv_0.9.4.orig.tar.gz
[root@workstation ~]\$ cd ediv-0.9.4/
```

Code 8.24: EDIV-Install: Download und Installation 1

Im Verzeichnis befindet sich eine Datei Makefile.pl, in die der korrekte Pfad zu Perl eingetragen werden muss.

```
[root@workstation ~]\$ perl -V
```

Code 8.25: EDIV-Install: Download und Installation 2

Die @INC variable, in der Regel am Ende der Ausgabe, zeigt diverse Pfade zu Perl-Modul-Verzeichnissen. Einer dieser Pfade muss in die erste Zeile der *Makefile*-Datei eingetragen werden (z.B.: 'MODULES_INSTALL_DEST=/usr/local/share/perl/5.10.0'). Als Editor nutzen viele auch `vi` anstatt `nano`. Es werden auch noch ein paar Perl-Module benötigt:

```
[root@workstation ~]\$ cpan AppConfig Math::Round
```

Code 8.26: EDIV-Install: Perlmodule

Falls noch nicht vorher installiert, kann man an dieser Stelle den MySQL-Server installieren.

```
[root@workstation ~]\$ nano Makefile
[root@workstation ~]\$ yum install mysql-server
[root@workstation ~]\$ make install
[root@workstation ~]\$ make modules-install
```

Code 8.27: EDIV-Install: Download und Installation 3

Falls Fehlermeldungen auftauchen fehlt unter Umständen ein installierter Compiler. Die Fehlermeldungen am Ende der Ausgabe bieten Hinweise was noch nachzu-

installieren ist. EDIV verwendet eine MySQL-Datenbank zur Verwaltung des Clusters. Es ist empfohlen die MySQL-Root-Logindaten zu nutzen, da über verschiedene EDIV-Werkzeuge Datenbanken angelegt und entfernt werden. Alternativ können natürlich auch andere User mit entsprechenden Zugriffsrechten verwendet werden. Sollte das MySQL-Root-Passwort nicht bekannt sein, so kann man ein neues anlegen mit:

```
1 [root@workstation ~]\$ /etc/init.d/mysqld start
2 [root@workstation ~]\$ mysqladmin -u root password NEUESPASSWORT
```

Code 8.28: EDIV-Install (optional): Neues MySQL Root-Passwort

Nun können Änderungen in der Konfigurationsdatei vorgenommen werden.

```
1 [root@workstation ~]\$ nano /usr/local/etc/ediv/cluster.conf
```

Code 8.29: EDIV-Install: Editieren der Konfigurationsdatei cluster.conf

Die Datenbank-Logindaten sind in der Section **[db]** einzutragen. Wenn die Zugriffsrechte für die MySQL-Datenbank ausreichend sind, kann das Script-Programm `ediv\_db\_create.pl` (vom root-user jederzeit aufrufbar) eine für EDIV geeignete Datenbankstruktur anlegen. Section **[vlan]** kann man mit default-Werten belassen werden, die Werte steuern die Benennung der VLANs, die die über die Cluster-Rechner verteilten VNUML-Szenarien verbinden. Wenn der Wert *first = 100* ist, werden die von EDIV erzeugten VLANs eth0.100, eth0.101, ... etc. genannt. Falls entsprechend große Szenarien erstellt werden, die den Rahmen sprengen, kann man diese Werte erhöhen. VLANs werden von EDIV verwendet, um die virtuellen Netze des VNUML-Szenarios, die auf verschiedene Cluster-Rechner zeigen, zu verbinden.
Die Section **[cluster]** beschreibt die Rechner, die Teil des Clusters sind. Jeder aktive Rechner wird als „host“ defininert. Stehen diverse Rechner nicht zur Verfügung können sie mit # auskommentiert werden, dabei müssen die Einstellungen unter der Cluster-Section nicht editiert werden. Nur Rechner die in der Cluster-Section aktiv sind, werden auch genutzt. Desweiteren wird definiert welches VNUML-Management-Network und welcher Segmentierungsalgorithmus benutzt werden soll. Als Management-Network trägt man in der Regel 192.168.0.0/16 ein, da dies vom Diagnose-Tool XT-Peer vorausgesetzt wird. Die Segmentierung legt eine Methode fest, nach der die einzelnen virtuellen VNUML-Maschinen auf den Cluster verteilt werden. Hier sollte der RoundRobin-Algorithmus beibehalten werden (default), da andere Alogrithmen

scheinbar nicht korrekt implementiert wurden. Desweiteren wird nur dieser eine Algorithmus vom PET-Script (näher beschrieben in Kapitel 4.3 und Kapitel 8.4) unterstützt. RoundRobin verteilt an jeden der Cluster-Rechner eine gleiche Anzahl von virtuellen Maschinen. Die Rechner im Cluster werden mit host = *Bezeichner* deklariert, der *Bezeichner* ist jeweils eine Untersektion der Konfigurationsdatei, in der definiert wird wieviel Arbeitsspeicher und CPU-Zeit verwendet werden darf und wieviele Virtuelle Maschinen maximal auf dem Rechner gleichzeitig laufen dürfen (Wert 0 entsprecht *unendlich*). Die Variable `ifname` definiert das Netzwerkinterface, welches für die Netzwerkkommunikation, also auch für die VLANs, verwendet werden soll.

```
1  [db]
2  type = mysql
3  name = ediv
4  host = localhost
5  port = 3306
6  user = root
7  pass = [entfernt]
8
9  [vlan]
10 first = 100
11 last = 199
12
13 [cluster]
14 host = netuml01
15 host = netuml02
16 host = netuml03
17 host = netuml04
18 host = netuml05
19 default_segmentation = RoundRobin
20 mgmt_network = 192.168.0.0
21 mgmt_network_mask = 16
22
23 [netuml01]
24 mem = 2048
25 cpu = 90
26 max_vhost = 10
27 ifname = eth0
28
29 [netuml02]
30 mem = 2048
31 cpu = 90
32 max_vhost = 10
33 ifname = eth0
34
35 [netuml03]
36 mem = 2048
37 cpu = 90
```

```
38 max_vhost = 10
39 ifname = eth0
40
41 [netuml04]
42 mem = 2048
43 cpu = 90
44 max_vhost = 10
45 ifname = eth0
46
47 [netuml05]
48 mem = 2048
49 cpu = 90
50 max_vhost = 10
51 ifname = eth0
```

Code 8.30: EDIV-Install: Die Konfigurationsdatei cluster.conf

In der Datei `cluster.conf` stehen auch diverse zusätzliche Erklärungen, die hier weggelassen wurden. Auf der DVD befindet sich eine Kopie der Datei im **ediv**-Ordner.

Der Rechner, von dem das VNUML-Szenario über EDIV gestartet wird, wird Kontrollrechner genannt. Ob mit oder ohne Hilfe des P.E.T.-Scripts, die Kommunikation zwischen dem Kontrollrechner und den Virtuellen Maschinen geschieht über SSH.

8.4 Nutzung des Hilfsscripts P.E.T. mit EDIV

Ursprünglich war es ein Teilziel dieser Arbeit die Diagnosesoftware XTPeer auch für verteilt simulierte Szenarien nutzbar zu machen. Die Implementierung dieses Hilfsscripts wird näher in Kapitel 4 beschrieben. Dieses Kapitel beschreibt, die man es benutzt und was es bei der Konfiguration zu beachten gibt.
Um ein Szenario zu starten muss man diverse vorbereitende Schritte durchführen. Diese geschehen sowohl auf dem Kontrollrechner als auch auf den Cluster-Rechnern, es empfielt sich deshalb eine Kopie des Scripts auf allen beteiligten Rechnern zu haben.

8.4.1 Schritt 0: Download

Das Hilfsscript ist der Webseite http://git.uni-koblenz.de erhältlich. Aktuell ist **Version 0.09**, welches in mehreren Teilschritten IP-Tunnels zu allen beteiligten Maschinen aufbaut. Es umfasst nur eine Datei, `pet.sh`.

```
[root@workstation ~]\$ git clone git://git.uni-koblenz.de/pet/pet.git
```

Code 8.31: Installation von PET über GIT

Die sh-Datei sollte möglichst in das Verzeichnis verschoben oder kopiert werden, in dem die Szenarien gespeichert sind. Sie muss zudem ausführbar sein, was nicht unbedingt gegeben ist. Danach sollte das Script über `./pet.sh` startbar sein. Falls nur ein schwarzer erscheint ist entweder die EDIV cluster.conf-Datei nicht richtig konfiguriert oder die Einstellungen in den ersten Zeilen der `./pet.sh` nicht korrekt. In den auskommentierten Zeilen gibt es Hinweise was zu tun ist.

```
[root@workstation ~]\$ chmod a+x pet.sh
[root@workstation ~]\$ ./pet.sh -help
```

Code 8.32: PET ausführbar machen, falls notwendig

8.4.2 Schritt 1: SSH Keys installieren

Das VNUML-Dateisystem wird von jeder virtuellen Maschine genutzt. Um ohne Passwort-Abfrage von jedem Rechner darauf zuzugreifen, muss der Public Key des Kontrollrechners in der Datei `/root/.ssh/authorized_keys` jedes Cluster-Rechners, der für VNUML/EDIV genutzt werden soll, eingetragen sein. Es ist sinnvoll ein und dasselbe Dateisystem auf allen Rechnern im Cluster zu haben. Es ist möglich dies über PET zu tun, aber da die Architektur der Images nicht immer dasselbe sein muss empfielt sich der manuelle Weg.

```
[root@workstation ~]\$ cd /usr/local/vnuml/filesystems
[root@workstation ~]\$ mkdir mntpoint
[root@workstation ~]\$ mount -o loop rmti-0.99.img mntpoint/
[root@workstation ~]\$ cd mntpoint/root/.ssh/
[root@workstation ~]\$ cat /root/.ssh/id_rsa.pub >> authorized_keys
[root@workstation ~]\$ cat /root/.ssh/id_dsa.pub >> authorized_keys
[root@workstation ~]\$ cd ../../../
[root@workstation ~]\$ umount mntpoint
```

Code 8.33: Beispiel: Einfügen des lokalen public keys

Es kann auch vorkommen, dass die Datei authorized_keys gar nicht genutzt wird, weil diese Option im VNUML-Image auskommentiert ist. In diesem Fall sollte man innerhalb des Images einen Blick in die `sshd.conf` im etc/-Verzeichnis werfen, die Option vereinfacht das Arbeiten mit EDIV enorm. Neben dem lokalen Public-Key

sollten auch die Public Keys der Cluster-Maschinen in die Datei authorized_keys eingetragen werden. Analog zum Beispiel 8.33 geht man über das Programm Secure-Copy `scp` wie folgt vor:

```
[root@workstation ~]\$ cd /usr/local/vnuml/filesystems
[root@workstation ~]\$ mount -o loop rmti-0.99.img mntpoint/
[root@workstation ~]\$ cd mntpoint/root/.ssh/
[root@workstation ~]\$ scp root@netuml01:/root/.ssh/id_rsa.pub remote_key1.pub
[root@workstation ~]\$ cat remote_key1.pub >> authorized_keys
[root@workstation ~]\$ scp root@netuml01:/root/.ssh/id_dsa.pub remote_key2.pub
[root@workstation ~]\$ cat remote_key2.pub >> authorized_keys
[root@workstation ~]\$ rm remote_key* -rf
[root@workstation ~]\$ cd ../../../
[root@workstation ~]\$ umount mntpoint
```

Code 8.34: Beispiel: Einfügen eines remote public keys (vom Rechner netuml01)

Sind alle Keys im VNUML-Image, sollte man es an alle Cluster-Rechner kopieren, dies geht auch über SCP, beispielsweise:

```
[root@workstation ~]\$ ssh netuml01
[root@netuml01 ~]\$ cd /usr/local/share/vnuml/filesystems
[root@netuml01 ~]\$ scp root@workstation:/usr/local/share/vnuml/filesystems/rmti-0.99.img .
```

Code 8.35: Beispiel: Kopieren über SCP

Wenn alle Rechner im Cluster inkl. des Kontrollrechners ein identisches Dateisystem mit allen SSH-Keys haben kann Schritt 2 begonnen werden[8].

8.4.3 Schritt 2: Szenario XTPeer-kompatibel machen

In v0.09 des Scripts ist es noch notwendig das Szenario, das gestartet werden soll, in einer der ersten Zeilen der Datei `pet.sh` zu definieren, ebenso die Pfade zum Dateisystem und zur `cluster.conf`. Die Kommandos sind in dieser Reihenfolge durchzuführen, was genau passiert steht hier [4.3].

```
[root@workstation ~]\$ ./pet -create_tunnels
```

Code 8.36: IP Tunnels gemessen an Clusterkonfiguration erstellen

[8]Es gibt auch eine Funktion für Schritt 1 im PET-Script, mit der man allerdings mit wenigen Fehlern viel kaputt machen konnte. Das Tutorial ist sicherer

```
[root@workstation ~]\$ ./pet -patch
```

Code 8.37: Szenario modifizieren und Hosts hinzufügen

`-patch` legt Backups der original-Datei an, wenn der Cluster korrekt konfiguriert wurde, werden Hosts den IP-Tunnels hinzugefügt, die später mit dem Start des Szenarios direkt untereinander kommunizieren können. Das EDIV-Szenario wird dadurch nicht beeinträchtigt.

```
[root@workstation ~]\$ service mysql start
[root@workstation ~]\$ ediv_ctl -t -s <szenario-name> -vB
[root@workstation ~]\$ ediv_ctl -x start -s <szenario-name> -vB
[root@workstation ~]\$ ediv_ctl -x rip -s <szenario-name> -vB
```

Code 8.38: Szenario starten

Das Szenario sollte nun für XTPeer zugänglich sein und ohne Einschränkungen nutzbar sein.

8.4.4 Schritt 3: Simulation beenden

```
[root@workstation ~]\$ ediv_ctl -P <scenario-name> -vB
[root@workstation ~]\$ ./pet.sh -dev_destroy
```

Code 8.39: Szenario beenden

Während des Vorgangs werden die einzelnen VNUML-Teile des Clusters und die VLANs heruntergefahren und die Datenbankeinträge gelöscht. Die Funktion `-dev_destroy` startet auf allen betroffenen Rechnern das TUN/TAP-Kernel-Modul neu und entfernt so die eventuelle EDIV/VNUML-Rückstände und IP Tunnel.

Literaturverzeichnis

[Boh08] Bohdanowicz, Frank: *Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons*. Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, Mai 2008. [Online; Stand 09. Januar 2010].

[FB10] Frank Bohdanowicz, Harald Dickel, Christoph Steigner: *Hidden Potentials of the Distance Vector Approach*. Diplomarbeit, Universität Koblenz, Mai 2010. [Abgerufen am 09. September 2010].

[GNU09] GNU: *Zebra – ripd*. Internetquelle, http://manticore.2y.net/doc/zebra/ripd.html, 2009. Abgerufen am 31.03.2009.

[Hed88] Hedrick, C.: *Routing Information Protocol*. Internetquelle, http://www.faqs.org/rfcs/rfc1058.html, Juni 1988. Abgerufen am 01.05.2010.

[Hie99] Hietaniemi, Jarkko: *CPAN - Comprehensive Perl Archive Network*. Internetquelle, http://www.cpan.org/, Februar 1999. Abgerufen am 01.05.2010.

[Jac10] Jacobs, Marcel: *Bedienungsanleitung zum Simulationsscript zimulator.pl*. Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, April 2010. [Online; Stand 9. September 2010].

[Kob09] Kober, Manuel: *Evaluation & Convergence-Analysis of RIP with metric based Topology Investigation* . Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, August 2009. [Online; Stand 19. März 2010].

[Ltd09] Ltd, JGraph: *JGraph - Visualize Everything*. Internetquelle, http://www.jgraph.com/, 2009. Abgerufen am 06.12.2009.

[Mad08] MADRID, UNIVERSITÄT: *VNUML Wiki*. Internetquelle, http://www.dit.upm.es/vnumlwiki/index.php/Main_Page, 2008. Abgerufen am 06.12.2009.

[Mal97] MALKIN, G.: *Routing Information Protocol*. Internetquelle, http://www.faqs.org/rfcs/rfc2080.html, Januar 1997. Abgerufen am 01.05.2010.

[Mal98] MALKIN, G.: *RIP Version 2*. Internetquelle, http://www.faqs.org/rfcs/rfc2453.html, November 1998. Abgerufen am 01.05.2010.

[ORe98] OREILLY, TIM: *XML, A Technical Instruction to XML*. Internetquelle, http://www.xml.com/pub/a/98/10/guide0.html, 1998. Abgerufen am 06.12.2009.

[OSP09a] OPEN-SOURCE-PROJEKT: *Bash Reference Manual*. Internetquelle, http://www.gnu.org/software/bash/manual/bashref.html, 2009. Abgerufen am 09.12.2009.

[OSP09b] OPEN-SOURCE-PROJEKT: *Quagga Homepage*. Internetquelle, http://www.quagga.net/about.php, 2009. Abgerufen am 09.12.2009.

[Sch99] SCHMID, ANDREAS: *RIP-MTI - Minimum-effort loop-free distance vector routing algorithm*. Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, September 1999.

[Tan00] TANENBAUM, ANDREW S.: *Computernetzwerke*. Pearson Studium, 2000.