# Bedienungsanleitung zum Simulationsscript zimulator.pl

Marcel Jakobs

2. Dezember 2009

# Inhaltsverzeichnis

# 1 Einleitung

Das Perl-Script „zimulator.pl“ ermöglicht die Autmatisierung von Simulationen des RIP-Protokolls unter VNUML. Die Konvergenzeigenschaften der Simulationen können dabei automatisch ausgewertet werden. Dabei kann der Ablauf der Simulation über eine Konfigurationsdatei sehr genau beschrieben werden. Die Messungen der Konvergenzeigenschaften werden durch die Analyse von automatisch generierten TCP-Dumps nach der eigentlichen Simulation durchgeführt. Daneben bietet das Script noch Optionen zum schnellen Erzeugen von VNUML-Konfigurationsdateien mittels einer eigenen Konfigurationsdatei oder per Zufall nur über die Angabe der Anzahl der Netze und Router. Darüber hinaus können Netzwerke (als png-Dateien) visualisiert werden und einige Eigenschaften der zugrunde liegenden Graphen berechnet werden.

## 1.1 Wichtige Hinweise

Die Simulationen werden in der Regel mit root-Rechten ausgeführt. Von daher sollte man stets Vorsicht walten lassen.

Diese Bedienungsanleitung sowie das Script selbst sind noch nicht fertig und werden momentan weiter entwickelt.

Die Dokumentation des Quelltextes ist veraltet. Sie wird demnächst auf den aktuellen Stand gebracht.

Dieses Script wurde nur unter Linux getestet.

Die neuesten Versionen gibt es im SVN: `https://svn.uni-koblenz.de/zimon/diplomarbeit/zimulator`

Hinweise, Fragen ... können an mich per Email (zimon@uni-koblenz.de) gerichtet werden.

# 2 Bedienungsanleitung

Mit dem Perl-Script `zimulator.pl` ist es möglich Simulationen mit VNUML-Netzwerken zu automatisieren und automatische Konvergenzmessungen durchzuführen.

## Übersicht

Um Simulationen zu Automatisieren, werden 3 Konfigurationsdateien benötigt:

- Eine *Topologiebeschreibung* in Form einer zvf-Datei. (Siehe auch Seite 9)
- Ein *Test-Script* in der die einzelnen Simulationsschritte definiert sind als .cfg-Datei. (Siehe auch Seite 12)

- Eine *Ausführungsbeschreibung*s-Datei (Name wird währscheinlich noch geändert), die beschreibt, welche Topologien mit welchen Test-Scipten wie oft Simuliert werden sollen. (Siehe auch Seite 16)

Bei allen Dateien werden Leerzeilen und Zeilen, die mit `#` beginnen (Kommentare) ignoriert. Die zugehörige VNUML xml-Konfigurationsdatei (*Szenario-Datei*) wird automatisch erstellt, falls sie nicht vorhanden ist. So wie man eine Szenario-Datei also das konkrete Netzwerk als die Instanz einer Toplogie betrachten kann, so kann man das *Szenario* also konkreten Simulationsprozess ebenfalls als Instanz einer Simulation ansehen.

Eine *Simulation* ist eine Kombination aus Topologiebeschreibung und Test-Script. Das Simulieren einer Simulation wird als *Durchlauf* (bzw. Run) bezeichnet. Das Ergebnis eines Durchlaufs (also dessen Konvergenzzeit, gesendete Pakete, ...) wird in einer *Ergebnisdatei* (resultfile) gespeichert. Bei mehreren Durchläufen wird das Ergebnis jeweils an die Datei angehängt, wobei jeder Durchlauf einer Zeile entspricht. Die Ergebnisdatei wird zusammen mit tcpdump-Dateien und anderen erstellten Dateien für diese Simulation in einem Ordner (*Simulationsordner*) gespeichert. Der Name des Ordners wird aus den Namen des Test-Scripts und der Topologiebeschreibungs-Datei zusammengesetzt (ohne Endungen und mit Bindestrich getrennt).

Die Datei `modules/Scenario.pm` ist eine Klasse, die eine Simulation repräsentiert. Sie besitzt Informationen über die Topologie und verfügt über Funktionen zum Steuern der Simulation.

### Funktionen des Scripts

Im Folgenden werden alle Funktionen von „zimulator.pl“ mit ihren Optionen aufgeführt.

Die Datei `zimulator.pl` verarbeitet die übergebenen Argumente und ruft die benötigten Funktionen auf.

**Syntax:** `./zimulator.pl MODE [FILE] [OPTIONS] [FILE(S)]`

### Die Modi des Programms:

`-s FILE` - Simuliert die Simulationen, die in der angegebenen Ausführungsbeschreibungs-Datei angegeben sind.

`-S [-T] [-F] [-o OUTPUT_FILE] TOPOLOGY_FILE(S)` - Berechnet die Konvergenzeigenschaften für alle Durchläufe aller Simulationen einer Topologie.

`-r ZVFFILE [-T] [-F] [-o OUTPUT_FILE] FILE1 FILE2 ...` - Berechnet die Konvergenzeigenschaften für die angegebenen tcpdump-Dateien zu einer gegebenen Topologie.

`-a FILE1 FILE2 ...` - Errechnet die durchschnittlichen Konvergenzzeiten von allen angegeben Ergebnisdateien.

`-z [-i] FILE1 FILE2 ...` - Analysiert die Topologie des Netzwerks und schreibt die Eigenschaften des Graphen in die zvf-Datei (es wird auch eine png-Datei angelegt). Kommentare in

der zvf-Datei gehen verloren

`-x FILE1 FILE2 ...` - Erzeugt VNUML xml-Dateien für alle angegebenen Topologiebeschreibungen.

`-g TYPE [-o OUTPUT_FILE] [-i] ARGUMENTS` - Generiert einen Graphen des Typs TYPE [1]. Es wird eine zvf-Datei mit den Eigenschaften des Graphen erzeugt.

`-C ZVFFILE` - Überpüft die angegebene zvf-Datei auf Sytaxfehler.

`-H FILE` - Gibt angegebene tcpdump-Dateien oder Ergebnisdateien in von Menschen lesbarer Form aus.

`-h` - Gibt die Hilfe aus.

**Mögliche Optionen:**

`-v` - Gibt zusätzliche Informationen aus.

`-c CONFIG_FILE` - Benutzt die angegebene Konfigurationsdatei

`-o OUTPUT_FILE` - Schreibt alle Ausgaben in die angegebe Datei (die Datei „-“ bezeichnet die Standardausgabe

`-T TIME` - Zieht nur Pakete zur Konvergenzberechnung heran, die vor dem Zeitpunkt TIME versendet wurden (im Modus -S kann die Angabe einer Zeit weg fallen, da diese aus der Ergebnisdatei entnommen werden kann).

`-F TIME` - Zieht nur Pakete zur Konvergenzberechnung heran, die nach dem Zeitpunkt TIME versendet wurden (im Modus -S kann die Angabe einer Zeit weg fallen, da diese aus der Ergebnisdatei entnommen werden kann).

`-i` - Generiert ein PNG Bild der Topologie(n).

## Installation und Konfiguration

Im folgenden werden die einzelnen Schritte der Installation und Konfiguration von VNUML und dem Simulationsscript `zimulator.pl` auf Debian basierten Linuxdistributionen beschrieben. Getestet wurde zimulator.pl auf Ubunu 9.04 und Fedora.

### Installation von VNUML

Als erstes muss das benötigte Paket „bridge-utils“ mit folgendem Befehl installiert werden:

[1] Die möglichen Typen und ihre Argumente sind in Kapitel 2 auf Seite 23 beschrieben.

```
sudo apt-get install bridge-utils
```

Dieses wird benötigt um die virtuellen Netze an das Hostsystem weiter leiten zu können damit entsprechende tcpdump-Dateien vom Hostssystem aus arbeiten können.

Als nächstes wird VNUML installiert. Dazu öffnet wird mit root-Rechten die Datei /etc/apt/-sources.list editiert und folgende Zeile hinzugefügt:

```
deb http://jungla.dit.upm.es/~vnuml/debian binary/
```

Die folgenden 3 Befehle installieren dann VNUML und den Kernel:

```
sudo apt-get update
sudo apt-get install vnuml
sudo apt-get install linux-um
```

Nun kann man von der Seite http://www.dit.upm.es/vnumlwiki/index.php/Download ein Dateisystem herunter laden. Im Abschnitt „Root filesystems“ werden 2 Dateisysteme angeboten, wobei sich das kleinere („minimal root_fs“) für größere Simulationen besser eignet, da für jede simulierte Maschine eine Kopie des Dateisystems in den Arbeitsspeicher geladen wird. Um den RMTI Algorithmus zu verwenden kann man statt dessen auch das Dateisystem von Frank Bohdanowicz von der Seite http://www.uni-koblenz.de/~bohdan/vnuml/ nutzen. Das Dateisystem wird dann mit root-Rechten unter einem geeigneten Namen in das Verzeichnis `/usr/share/vnuml/filesystems` kopiert. Das folgende Kommando kopiert das minimal root_fs (die Versionsnummer und damit der Dateiname kann sich mit der Zeit ändern) unter dem Namen „mini_fs“ in das entsprechende Verzeichnis:

```
sudo cp n3vlr-0.11-vnuml-v0.1.img /usr/share/vnuml/filesystems/mini_fs
```

### Installation und Konfiguration von zimulator.pl

Das Script „zimulator.pl“ benötigt einige CPAN Bibliotheken sowie das Programm GraphViz [2]. Diese Pakete können mit folgendem Befehl installiert werden:

```
sudo apt-get install graphviz libgraphviz-perl
```

Die Bibliothek „Graph“ muss über CPAN installiert werden, da der Dijkstra-Algorithmus der Ubuntu-Version (Paket libgraph-perl unter Ubuntu 9.04) nicht funktioniert.

```
sudo cpan
install Graph
exit
```

Beim ersten Start von CPAN werden einige Fragen gestellt. Die meisten Fragen kann man mit

[2] Siehe http://www.graphviz.org/

ENTER bestätigen. Man sollte jedoch den Kontinent, das Land und den zu verwendenden Server angeben. Es kann auch sein, dass alles automatisch eingerichtet wird.

Im Moment gibt es nur ein Subversion (SVN) Verzeichnis als Bezugsquelle für das Script. Wer Zugang dazu haben möchte, kann sich bei mir melden (zimon@uni-koblenz.de). Das neueste Programm liegt im Verzeichnis `zimulator`. Da SVN nicht standardmäßig installiert ist, kann man dieses mit folgendem Befehl nachholen:

```
sudo apt-get install subversion
```

Um die neueste Revision auszuchecken nutzt man folgenden Befehl:

```
svn co https://svn.uni-koblenz.de/zimon/diplomarbeit/zimulator
```

Als Zugangsdaten werden die Daten der Uni-Kennung benutzt.

Nun sollte man zuerst die wichtigsten Konfigurationsparameter des Programms einrichten. Dazu öffnet man in einem Editor die Datei `modules/Configuration.pm`. Darin werden die wichtigsten Einstellungen festgelegt wie Pfade zu Programmen und Dateien.

Eine gängige Konfiguration könnte z.B. so aussehen (hier wird das minimal root_fs Image genutzt, welches wie auf Seite 6 beschrieben „mini_fs“ genannt wurde):

Der erste Abschnitt legt fest, wie aus zvf-Dateien die Topologiebeschreibungen generiert werden.

Constants.pm

```
# Package: Constants
#
# Hier werden alle Konstanten zur Konfiguration des Scripts definiert

package Constants;
require Exporter;
@ISA    = qw(Exporter);
@EXPORT = qw();

use strict;
use warnings;

#
# Constants: Konstanten zur generierung der VNUML XML-Datei
#
# SSH_KEY - Pfad zum oeffentlichen SSH-Schluessel
# MANAGEMENT_NET - Adresse des Management Netzes
# MANAGEMENT_NETMASK - Netzmaske des Management Netzes
# MANAGEMENT_NET_OFFSET - Offset, das auf jede IP aufaddiert wird
# VM_DEFAULTS - Attribute fuer das Tag "vm_defaults"
#        (Standard: " exec_mode=\"mconsole\"")
# FILESYSTEM - Pfad zum VNUMLDateisystem
# KERNEL - Pfad zum VNUML-Kernel
#
# Die naechste Konstante nur aendern, wenn man genau weiss, was man tut!!!
```

```
# NET_MODE - Typ der virtuellen Netze (Standard: "virtual_bridge")
# ZEBRA_PATH - Pfad zu Zebra im Dateisystem (nur der Pfad ohne / am Ende)
# RIPD_PATH - Pfad zu RIP im Dateisystem (nur der Pfad ohne / am Ende)
# OSPF_PATH - Pfad zu OSPF im Dateisystem (nur der Pfad ohne / am Ende)
use constant SSH_KEY => "/root/.ssh/id_rsa.pub";
use constant MANAGEMENT_NET   => '192.168.0.0';
use constant MANAGEMENT_NETMASK   => '24';
use constant MANAGEMENT_NET_OFFSET => '100';
use constant VM_DEFAULTS => " exec_mode=\"mconsole\"";
use constant FILESYSTEM => "/usr/share/vnuml/filesystems/mini_fs";
use constant KERNEL => "/usr/share/vnuml/kernels/linux";
use constant NET_MODE => "virtual_bridge";
use constant ZEBRA_PATH => "/usr/lib/quagga";
use constant RIPD_PATH => "/usr/lib/quagga";
use constant OSPF_PATH => "/usr/lib/quagga";

#
# Constants: VNUML Ausfuehrungs-Konfiguration
#
# VNUML_PATH - Pfad zum VNUML-Programm (nur der Pfad ohne / am Ende)
# VNUML_START_PARAMETERS - Parameter zum Starten von VNUML
# VNUML_EXEC_PARAMETERS - Parameter zum ausfuehren von Tags
# VNUML_STOP_PARAMETERS - Parameter zum Beenden von VNUML
use constant VNUML_PATH => '/usr/bin';
use constant VNUML_START_PARAMETERS => '-w 300 -Z -B -t';
use constant VNUML_EXEC_PARAMETERS => '-x';
use constant VNUML_STOP_PARAMETERS => '-P';

#
# Constants: Sonstige Konstanten
#
# MAXFAIL_DEFAULT - Maximale Anzahl der Fehler bis Abbruch der Simulation,
#         falls dies nicht in der Aufuehrungsbeschreibung angegeben ist
# MAXRUN_DEFAULT - Maximale Anzahl der Durchlaeufe falls dies nicht in der
#         Aufuehrungsbeschreibung angegeben ist
# LOGFILE - Datei, in die VNUML-Ausgaben gespeichert werden
#         "/dev/null/" um keine Logdatei zu verwenden
# RAW_TCPDUMP - 1 um tcpdump-Dateien im RAW-Format zu speichern.
#         0 um sie im HEX-Format zu speichern
# VISUALIZE_NET_NAMES - 1 um die Namen der Netze in die PNG-Datei zu
#         schreiben
use constant MAXFAIL_DEFAULT => 5;
use constant MAXRUN_DEFAULT => 15;
use constant LOGFILE => 'logfile.log';
use constant RAW_TCPDUMP => 0;
use constant VISUALIZE_NET_NAMES => 1;

1;
```

## Simulationen Automatisieren

### Topologiebeschreibung mittels ZVF-Dateien

XML ist ein für den Menschen schwer lesbares Format, bei dessen manueller Bearbeitung schnell Fehler entstehen können. Bei den Konfigurationsdateien von VNUML müssen darüber hinaus auch viele Teile mehrmals geschrieben werden, wodurch leicht Copy'n'Paste Fehler entstehen können. Um die Topologie des Netzes zu beschreiben reichen aber die Informationen, welche Netze existieren, welche Router existieren und wie diese miteinander verbunden sind. Alle restlichen Informationen können automatisch generiert werden.

Daher wurde ein Dateiformat entwickelt, welches leicht von Menschen zu handhaben ist und in ASCII-Klartext nur die benötigten Informationen enthält. Dieses Dateiformat kann durch das Script in eine Szenario-Datei (VNUML-Konfigurationsdatei) umgewandelt werden. Die Konfigurationsdatei muss die Endung .zvf besitzen um vom Script erkannt zu werden.

Um aus einer zvf-Datei eine VNUML-Konfigurationsdatei zu generieren wird das Script mit folgenden Parametern aufgerufen:

```
./zimulator.pl -x {DATEI}
```

Es können also auch mehrere Dateien gleichzeitig verabeitet werden.

**Beispiel:**

```
./zimulator.pl -x *.zvf
```

generiert die VNUML-Konfigurationsdateien aus allen zvf-Dateien im aktuellen Verzeichnis.

Beim Starten einer Simulation werden die xml-Dateien automatisch generiert, falls sie noch nicht existieren.

### Syntax der Topologiebeschreibung

Es wurde versucht, die Syntax möglichst einfach zu halten.

Leerzeilen und Zeilen, die mit einem Rautenzeichen (`#`) beginnen werden ignoriert. In der ersten Zeile werden die Netze durch Komma getrennt aufgelistet. Diese müssen mit „net“ beginnen und fortlaufen nummeriert sein. Z.B. net1,net2,... In den restlichen Zeilen wird jeweils der Name des Routers angegeben. Dieser muss mit r beginnen. Z.B. r1 oder r2 Nach dem Router kommt ein Leezeichen und dahinter eine mit Komma separierte Liste der an diesen Router angeschlossenen Netze.

**Beispiel:** Dreiecks-Topologie

In der Graphentheorie bezeichnet ein Dreieck den kleinstmöglichen Kreis der mit einem einfachen

Graphen erstellt werden kann. Dieser besteht wie in Abbildung 1 zu sehen ist aus drei Knoten, welche mit 3 Kanten verbunden sind. Das folgende Listing beschreibt diese Topologie in der zvf Syntax:

dreieck.zvf

```
net1,net2,net3

r1 net1,net3
r2 net1,net2
r3 net2,net3
```

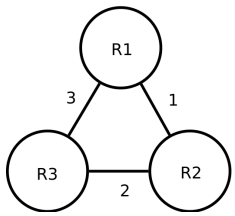


Abbildung 1: Listing und graphische Darstellung der Dreiecks-Toplogolie

Die globalen Einstellungen für die VNUML-Konfigurationsdatei, der Netztyp und der Aufbau eines einzelnen Routers sowie das Netz, aus dem die IP-Adressen und Netze generiert werden, können über die Konfigurationsdatei (`modules/Configuration.pm`) gesteuert werden.

Zusätzlich kann man die Variablen `$global`, `$net` und `$router` in der Funktion `toZVF()` der Datei `modules/Toplology.pm` anpassen, wo die Funktion zum generieren von Szenario-Dateien definiert ist.

Zur Generierung der IP- und Netzadressen wird standardmäßig das Netz 10.0.0.0/16 herangezogen. Die Netzadressen werden dann entsprechend der eingegebenen Netznummern angelegt: net1 wird zu 10.0.1.0/24 und net12 zu 10.0.12.0/24. Somit sind bis zu 254 Netze möglich. Die Routernummern entsprechen dem letzten Byte der IP-Adresse wodurch ebenfalls maximal 254 Router möglich sind. Durch diese Konvention kann an der IP-Adresse sofort das Netz und der Router abgelesen werden. Demnach würde der Router r3 im obigen Beispiel die beiden Adressen 10.0.2.3 und 10.0.3.3 erhalten.

Hier sind noch einige Beispiele für Topologien aufgeführt. Eine genauere Beschreibung des Aufbaus und der Syntax von zvf-Dateien befindet sich in Kapitel 2 auf Seite 9.

**Y-Toplogie**

Eine gängige Toplogie ist die Y-Topologie, welche aus 4-5 Routern besteht und die ungefähre Form eines Y besitzt. Die Konvergenzzeit beträgt etwa 5-10 Sekunden. (Also sollte man mindestens 30 Sekunden warten, wenn man sicher gehen möchte, dass das Netzwerk Konvergent ist)

y_topologie.zvf

```
net1,net2,net3,net4,net5

r1 net1,net3
r2 net1,net2
r3 net2,net3,net4
r4 net4,net5
r5 net5
```

Das zugehörige Bild zu dieser Topologie sieht dann so aus:

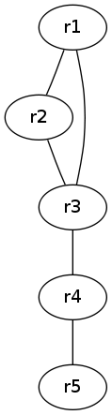


Abbildung 2: Visualisierung der Y-Topologie

**square3 Topologie**

Eine weitere oft genutzte Topologie ist square3. Sie besteht aus einem Quadrat von 3 mal 3 Routern. Die Konvergenzzeit beträgt etwa 5-10 Sekunden.

square3.zvf

```
net1,net2,net3,net4,net5,net6,net7,net8,net9,net10,net11,net12

r1 net1,net3
r2 net1,net2,net4
r3 net2,net5
r4 net3,net6,net8
r5 net4,net6,net7,net9
r6 net5,net7,net10
r7 net8,net11
r8 net9,net11,net12
r9 net10,net12
```

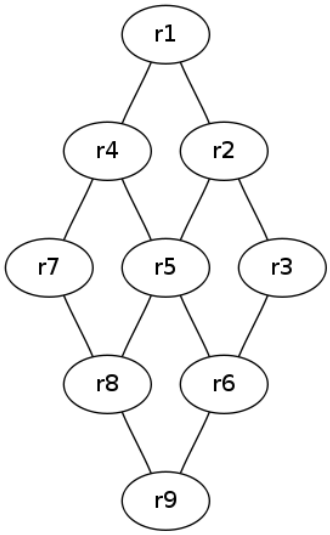


Abbildung 3: Visualisierung der square3-Topologie

Abbildung 3 zeigt die Visualisierung der Topologie.

### Test-Script

Um automatisiert Simulationen durchzuführen wird für jedes Test-Script eine Datei mit der Endung `.cfg` erstellt. Diese beschreibt die einzelnen Schritte eines Durchlaufs.

Dabei wird in jede Zeile ein Befehl geschrieben. Die Funktion `runScenario` in der Datei `modules/Simulator` interpretiert die Befehle nacheinander und ruft entsprechende Methoden der Klasse `modules/Scenario.pm` auf.

Ein Beispiel für ein Test-Script befindet sich auf Seite **??**.

Im Folgenden werden die möglichen Befehle beschrieben:

**Befehlsübersicht:**

- start() - Startet VNUML-Szenario, Routingdaemon oder tcpdump-Prozesse
- stop() - Stoppt VNUML-Szenario, Routingdaemon oder tcpdump-Prozesse
- gettime() Hält aktuellen Zeitpunkt fest zum späteren Messen der Konvergenzzeit
- sleep(x) Wartet x Sekunden
- execute(ROUTER,COMMAND,FILE) - Führt den Befehl COMMAND auf Router ROUTER aus und speichert das Ergebnis mit Zeitpunkt in die Datei FILE (bzw. hängt es an)
- disable() - Deaktiviert Router oder Device eines Routers (auch zufällige Auswahl ist möglich)
- enable() - Aktiviert den zuletzt deaktivierten Router/Device oder einen angegebenen falls mehrere deaktiviert wurden.

**start() und stop()** Mit dem Startbefehl werden VNUML, die tcpdump-Prozesse und die Routingdaemons gestartet. Für eine genauere Konfiguration können die einzelne Programme separat gestartet werden. Dies wird mit den Parametern angegeben.

Mit dem Parameter `scenario` wird vnuml mit dem im der Topologie entsprechenden Szenario gestartet. 2 Sekunden später wird der Zebra- Daemon auf allen Routern gestartet. Der Parameter `tcpdump` startet für jedes Netz einen tcpdump-Prozess und leitet die Ausgaben in entsprechende Dateien (netXXX.dump mit XXX=Nummer des Netzes) um. Mit dem Parameter `protocol` wird der im globalen Teil definierte Routingdaemon gestartet.

Um alle 3 Aktionen in der oben genannten Reihenfolge mit jeweils 3 Sekunden Abstand zu starten kann man den Parameter `all` verwenden.

Der Stop-Befehl hat die gleichen Parameter wie der Start-Befehl. Er stoppt die entsprechenden Dienste bzw. Programme. (`stop(protocol)` ist noch nicht implementiert. Der Routingdaemon wird zusammen mit dem Scenario per `stop(scenario)` gestoppt.)

**sleep()**

Der Befehl sleep erwartet als Parameter eine ganze Zahl, welche die Sekunden angibt, die gewartet werden sollen bis der nächste Befehl ausgeführt wird.

Dieser Befehl ist vor allem dazu gedacht, abzuwarten bis das Netzwerk Konvergent ist, bevor die Simulation beendet wird und die tcpdump-Dateien ausgewertet werden.

**Hinweis:**

Man sollte in jedem Fall darauf achten diese Zeit groß genug zu wählen um sicher zu gehen, dass das Netzwerk auch Konvergent ist bevor die Simulation beendet wird. Am besten Probiert man aus, wie lange ein Netzwerk ungefähr braucht um die Zeit großzügig bemessen zu können.

**execute()**

Der Befehl execute kann beliebige Befehle auf einem angegebenen Router ausführen und bei Bedarf die Ausgabe des Befehls in eine angegebene Datei auf dem Hostsystem schreiben. Als ersten Parameter erwartet execute() den Namen des Routers, auf dem der Befehl ausgeführt werden soll. Das zweite Argument gibt den Befehl selber an. Der dritte Parameter ist optional und gibt die Datei an, in der die Ausgabe gespeichert werden soll. Dabei wird auch immer der Zeitpunkt mit in die Datei geschrieben, wann der Befehl ausgeführt wurde. Wird die Datei mehrfach verwendet, so wird die Ausgabe jeweils angehängt.

Der Befehl lässt sich gut verwenden um iptables-Befehle an Router zu senden oder sich zu bestimten Zeitpunkten die Forwardingtabelle eines Routers ausgeben zu lassen.

Das folgende Beispiel führt den Befehl `route -n` auf dem Router r1 aus und hängt dessen Ausgabe an die Datei `route_r1.txt` an:

**Beispiel:**

```
execute(r1,route -n,route_r1.txt)
```

**disable() und enable()**

Um einen Router oder ein Device eines Routers zu deaktivieren, kann der Befehl disable() verwendet werden. Er dient zur Simulation von Ausfällen. Um einen Router zu deaktivieren ruft man den Befehl mit dem Namen des Routers als Parameter auf. Soll ein zufälliger Router ausfallen, so kann der Parameter `random` genutzt werden. Der Router wird bei jedem Durchlauf neu berechnet, so dass man Aussagen über Konvergenzzeiten treffen kann, wenn ein beliebiger Router ausfällt (vorrausgesetzt es wurden genügend Durchläufe gemacht).

Wenn man einen Router deaktiviert, der im entsprechenden Graphen einer Artikulation entspricht, so kann es passieren, dass zwei kleine Teilnetze entstehen, die parallel viel schneller Konvergieren als das Ursprüngliche große Netz. Da dies die Messergebnisse verfälschen würde werden solche Router ignoriert. (noch nicht implementiert)

Problematisch ist es auch, wenn Router deaktiviert werden, die an nur ein Netz angeschlossen sind. Da die getesteten Routingalgorithmen Netzbasiert arbeiten bleibt das Netzwerk in einem solchen Fall Konvergent, da weiterhin alle Netze erreichbar sind. Daher werden bei der zufälligen Auswahl des zu deaktivierenden Routers solche Router ignoriert. (noch nicht implementiert)

Möchte man weitere Router vom Herunterfahren mittels des random-Parameters ausschließen, so kann man diese in eckigen Klammern mit Komma getrennt angeben. Im folgenden Beispiel soll ein zufälliger Router deaktiviert werden, die beiden Router r1 und r2 sollen davon aber ausgeschlossen werden: `disable(random[r1,r2])`.

Um ein Device eines Routers zu deaktivieren kann man die Nummer des Devices als zweiten Pa-

rameter angeben. Der erste Parameter gibt dann den Rechner an, auf dem das Device deaktiviert werden soll. Auch hier kann der Parameter `random` in beliebiger Kombination genutzt werden.

**Beispiele:**

- disable(r1)
- disable(random)
- disable(random[r1,r2])
- disable(r1,2)
- disable(r1,random)
- disable(r1,random[1])
- disbale(random,2)
- disable(random[r1,r2],1)
- disable(random,random)
- disable(random[r1,r2],random)
- disable(random,random[1])
- disable(random[r1,r2],random[1])

Um einen deaktivierten Router wieder zu aktivieren gibt es den Befehl enable(). Wird er ohne Parameter aufgerufen, so wird der zuletzt deaktivierte Router (bzw. das zuletzt deaktivierte Device) wieder aktiviert. Möchte man einen anderen Router (bzw. ein anderes Device) wieder aktivieren, so kann man (bis auf random) die gleichen Parameter nutzen wie bei disable.

**gettime()**

Der Befehl gettime speichert die zum Zeitpunkt seines Aufrufs aktuelle Zeit. Diese wird später als Startzeitpunkt zur Berechnung der Konvergenzzeit genutzt. Dieser Befehl kennt keine Parameter. Er kann genutzt werden um zu messen wie lange ein Netzwerk benötigt, bis es nach dem Ausfall eines Routers wieder Konvergent ist. Dazu führt man den Befehl direkt vor oder nach dem Befehl disable aus.

Im Folgenden werden zwei Beispiele für Test-Scripte aufgeführt. **Konvergenzzeitbestimmung**

Als erstes ein recht simples Beispiel, welches lediglich die Konvergenzzeit einer Topologie bestimmt. Es wird VNUML gestartet und danach die Forwardingtabelle des Routers r1 mit dem Befehl `execute(r1,route -n,route_r1.txt)` in die Datei `route_r1.txt` geschrieben. Danch werden die tcpdump-Prozesse und das Routingprotokoll gestartet und 60 Sekunden abgewartet. Schließlich werden die tcpdump-Prozesse sowie der Durchlauf selbst beendet.

Die zugehörige Datei „konvergenzzeit60sec.cfg“ sieht folgendermaßen aus:

konvergenzzeit60sec.cfg

```
start(scenario)
sleep(3)
execute(r1,route -n,route_r1.txt)
start(tcpdump)
start(protocol)
sleep(60)
execute(r1,route -n,route_r1.txt)
stop(tcpdump)
stop(scenario)
```

**Routerausfall**

Dieses Beispiel lässt einen zufälligen Router ausfallen und misst die Zeit bis das Netzwerk wieder Konvergent ist. Nach dem Start des Szenarios, der tcpdump-Prozesse und des Routingdaemons wird 60 Sekunden gewartet bis das Netz Konvergent ist. Nun wird ein zufälliger Router mit dem Befehl `disable(random)` deaktiviert. Dann wird die aktuelle Zeit mit `gettime()` gespeichert um sie später als Startzeitpunkt für die Konvergenzzeitberechnung zu nutzen. Schließlich wird noch 5 Minuten gewartet bis das Netzwerk wieder Konvergent ist und dann der Durchlauf beendet.

Die zugehörige Datei „routerausfall60_300.cfg“ sieht folgendermaßen aus:

routerausfall60_300.cfg

```
start(scenario)
sleep(3)
start(tcpdump)
start(protocol)
sleep(60)
disable(random)
gettime()
sleep(300)
stop(tcpdump)
stop(scenario)
```

**Ausführungsbeschreibung**

Die Ausführungsbeschreibungs-Datei besitzt die folgende Syntax:

```
TOPOLOGIENAME SIMULATIONSNAME PROTOKOLL DURCHLÄUFE FEHLER
```

wobei der Topologiename der zvf- bzw. xml-Datei ohne Endung entspricht. Der Simulationsname ist der Name des Test-Scripts (ohne Endung). Das Protokoll entspricht dem auszuführenden Tag der VNUML xml-Datei für das zu startende Protokoll. Das Feld „DURCHLÄUFE“ gibt an, wie

oft diese Simulation wiederholt werden soll. „FEHLER“ gibt die maximale Anzahl der Fehler an, nach denen mit dieser Simulation abgebrochen werden soll. Ein Fehler entsteht z.B. wenn ein Dienst nicht startet oder die errechnete Konvergenzzeit negativ ist.

Ruft man das Script nun mit folgenden Parametern auf (wobei simulations.txt die Ausführungsbeschreibungs-Datei ist), so werden die Simulationen der Reihe nach abgearbeitet:

```
sudo ./zimulator.pl -s simulations.txt
```

Nach jedem Durchlauf wird die Datei erneut eingelesen und mit dekrementierter Anzahl von Durchläufen wieder zurück geschrieben. Somit zeigt die Datei gleichzeitig an, an welcher Stelle sich die Simulation gerade befindet. Durch dieses Vorgehen ist es auch möglich während der Simulation die Anzahl der Durchläufe oder die zu simulierenden Szenarien zu ändern. Ein weiterer Vorteil dieser Methode ergibt sich daraus, dass bei einem Problem (es kann vorkommen, dass VNUML hängen bleibt) alles durch ein spezielles Script [3] gestoppt und ohne die Ausführungsbeschreibungs-Datei ändern zu müssen die Simulationen durch erneutes Aufrufen des obigen Befehls fortgesetzt werden können.

Die Funktion `simulate()` in der Datei `modules/Simulator.pm` interpretiert und manipuliert die Ausführungsbeschreibungs-Datei und ruft für jeden Durchlauf die Funktion runScenario auf, welche das Test-Script parst.

Die folgende Ausführungsbeschreibungs-Datei „simulations.txt“ lässt jede Topologie in jeder Simulation 10 mal laufen. Wenn pro Simuation mehr als 5 Fehler auftreten wird sie abgebrochen.

simulations.txt

```
y_scenario konvergenzzeit60sec rip 10 5
square3 konvergenzzeit60sec rip 10 5

y_scenario routerausfall60_300 rip 10 5
square3 routerausfall60_300 rip 10 5
```

### Neustart der Simulationen bei Problemen

Da es aus verschiedenen Gründen immer wieder vorkommen kann, dass VNUML hängen bleibt, wurde ein kleines Perl-Script geschrieben, welches die Wiederaufnahme der Simulationen erleichtert: `killsimulation.pl`. Wenn VNUML bei obigem Aufruf stehen bleibt, so ruft man das Script mit folgenden Argumenten auf:

```
sudo ./killsimulation.pl simulations.txt
```

Dieses beendet `zimulator.pl`, `vnumlparser.pl` sowie alle laufenden tcpdump-Prozesse. Die LOCK-Datei im vnuml-Verzeichnis des Users wird gelöscht, damit VNUML wieder startet. Daraufhin wird der Ausführungsbeschreibung der Name des aktuellen Szenarios entnommen

[3] Siehe nächstes Kapitel

und dieses mit `vnumlparser.pl` gestoppt. Danach kann man zimulator.pl erneut (wie oben beschrieben mit `-r simulations.txt`) ausführen. Es wird die abgebrochene Simulation wiederholt und danach normal weiter gearbeitet.

## Auswertung der Ergebnisse

Alle Ergebnisse einer Simulation werden in einem eigenen Verzeichnis (dem Simulationsordner) gepeichert, welches folgenden Aufbau besitzt: `simulationsname-topologiename`.

Dort wird auch die Ergebnisdatei abgelegt, die den Namen der Toplogie mit der Endung `.txt` erhält. Aus den tcpdump-Dateien eines Durchlaufs kann errechnet werden, wie lange die Konvergenzzeit war (von Anfang an, ab oder bis zu einem bestimmten Zeitpunkt). Des weiteren wird die Anzahl der versendeten Routingpakete in diesem Zeitraum und der dadurch erzeugte Traffic ausgewertet.

Aus diesen Daten werden automatisch Statistiken erzeugt (durchschnittliche Anzahl der Pakete pro Netz, Netz mit dem meisten/wenigsten Traffic, ...) Diese Statistiken werden zusammen mit den Topologieeigenschaften in die Ergebnisdatei geschrieben, welche pro Durchlauf die folgenden Werte jeweils mit einem Leerzeichen getrennt enthält:

- Durchmesser der getesteten Topologie
- Anzahl der Knoten der Topologie
- Anzahl der Blätter
- Anzahl der inneren Knoten
- Anzahl der Kanten
- Clusterkoeffizient
- Konvergenzzeit des Durchlaufs
- Anzahl der Routingpakete
- Trafficvolumen der Routingpakete
- Anzahl der durchschnittlichen Pakete pro Netz
- Durchschnittlicher Updatetraffic por Netz
- Anzahl der Pakete im Netz, welches die wenigsten Pakete versendet hat
- Netz, das die wenigsten Pakete versendet hat
- Anzahl der Pakete im Netz, welches die meisten Pakete versendet hat
- Nummer des Netzes, das die meisten Pakete versendet hat
- Traffic des Netzes mit dem geringesten Traffic
- Nummer des Netzes mit dem geringsten Traffic
- Traffic des Netzes mit dem höchsten Traffic
- Nummer des Netzes mit dem höchsten Traffic
- Router oder Device das ausgefallen ist (0 wenn es keinen Ausfall gab)
- Zeit des Router- oder Deviceausfalls

- Name der getesteten Topologiedatei
- Name des getesteten Protokolls
- Nummer des Durchlaufs
- Erster Zeitstempel
- Letzter Zeitstempel

Wird eine Simulation mehrmals mit den gleichen Parametern simuliert, so können am Ende mit dem Modus `-a` die maximalen, minimalen und durchschnittlichen Werte errechnet werden. Die Ausgabe hat eine ähnliche Form wie die Ergebnisdatei:

- Durchmesser der getesteten Topologie
- Anzahl der Knoten der Topologie
- Anzahl der Blätter
- Anzahl der inneren Knoten
- Anzahl der Kanten
- Clusterkoeffizient
- Längste Konvergenzzeit eines Durchlaufs
- Kürzeste Konvergenzzeit eines Durchlaufs
- Durchschnittliche Konvergenzzeit aller Durchläufe
- Anzahl der Pakete im Durchlauf mit den meisten Paketen
- Anzahl der Pakete im Durchlauf mit den wenigsten Paketen
- Durchschnitt der Gesammtpakete
- Durchschnitt der MAXAVERAGEPACKETS
- MINAVERAGEPACKETS
- Durchschnitt der Pakete pro Netz
- Höchster Gesammttraffic
- Geringster Gesammttraffic
- Durchschnittlicher Gesammttraffic
- Durchschnittlicher Traffic pro Netz
- Name der getesteten Topologie

Es besteht weiterhin die Möglichkeit, tcpdump-Dateien im Nachhinein erneut auzuwerten. Dafür gibt es die Option `-r`. mit der Option `-T` kann ein Zeitpunkt angegeben, bis zu dem die Dumps ausgewertet werden sollen. Alle Pakete, die zu einem späteren Zeitpunkt versendet worden sind werden ignoriert. Das bietet die Möglichkeit, eine Simulation mit einem Routerausfall zu testen und danach die Konvergenzzeiten von Anfang an bis zum ersten konvergenten Zustand berechnen zu lassen. Somit sind weniger Simulationen notwendig.

Mit der Option `-F` lässt sich ein Zeitpunkt angeben, der in der Berechnung als Startzeitpunkt für die Konvergenzzeit verwendet wird. Alle vorher gesendeten Pakete werden ignoriert.

Um mehrere Durchläufe gleichzeitig zu analysieren gibt es die Optionen `-S` die statt der tcpdump-Dateien selbst die Topologiedatei übergeben bekommen. Dabei werden die tcpdump-Dateien aller mit dieser Topologie simulierten Durchläufe automatisch gesucht und ausgewertet.

**Beispiele:**

`./zimulator -r beispiel.zvf simulation-topologie/rip_run_1/net*` - wertet die Dumps des ersten Durchlaufs der Simulation „simulation-topologie“ aus.

`./zimulator -r beispiel.zvf -F 123456789 simulation-topologie/rip_run_1/net*` - wertet die Dumps des ersten Durchlaufs der Simulation „simulation-topologie“ ab dem Zeitpunkt 123456789 aus.

`./zimulator -r beispiel.zvf -T 123456789 simulation-topologie/rip_run_1/net*` - wertet die Dumps des ersten Durchlaufs der Simulation „simulation-topologie“ bis zum Zeitpunkt 123456789 aus.

`./zimulator -S beispiel.zvf` - wertet die Dumps aller Durchläufe aller Simulationen der topologie „beispiel“ aus und überschreibt die Ergebnisdateien.

`./zimulator -S beispiel.zvf -o newresultfile.txt` - wertet die Dumps aller Durchläufe aller Simulationen der Topologie „beispiel“ aus und generiert die Ergebnisdatei „newresultfile.txt“.

`./zimulator -S beispiel.zvf -F 0 -o newresultfile.txt` - wertet die Dumps aller Durchläufe aller Simulationen der Topologie „beispiel“ aus, wobei der Startzeitpunkt der Auswertung der Ergebnisdatei entnommen wird und generiert die Ergebnisdatei „newresultfile.txt“.

`./zimulator -S beispiel.zvf -T 0 -o newresultfile.txt` - wertet die Dumps aller Durchläufe aller Simulationen der Topologie „beispiel“ aus, wobei der Endzeitpunkt der Auswertung der Ergebnisdatei entnommen wird und generiert die Ergebnisdatei „newresultfile.txt“.

**tcpdump**

Beim Starten der tcpdump-Prozesse wird zuerst ein ifconfig ausgeführt um alle Netze ausfindig zu machen. Für jedes Netz, (also jedes Device, das mit „net“ anfängt) wird nun ein tcpdump-Prozess gestartet, der mit dem Namen des Netzes mit der Endung .dump gespeichert wird. Die dafür benötigten Funktionen sind in der Datei `modules/Scenario.pm` definiert. Für jeden Durchlauf werden die Dumps nach der Simulation in ein eigenes Verzeichnis verschoben (protocol_run_x). Wird eine Simulation mit gleicher Topologie und gleichem Protokoll noch einmal mit der gleichen Durchlauf-Nummer ausgeführt, so wird eine fortlaufende Zahl angehängt. z.B.

```
rip_run_1
rip_run_2
rip_run_1_1
rip_run_1_2
```

Somit werden die Dumps niemals versehentlich überschrieben. Dies ist oft sehr praktisch, wenn die Simulation abbricht. Dann kann die Simulation einfach neu gestartet werden.

Die Dumps können entweder im RAW-Format gespeichert werden oder im Hexadezimalformat ohne den Link-Header (dies entspricht der Option -x von `tcpdump`). Welches Format genutzt werden soll, kann in der Datei `modules/Constants.pm` angegeben werden.

### Berechnung der Konvergenzeigenschaften

Die Berechnung der Konvergenzzeit benötigt neben den tcpdump-Dateien aller Netze die Topologiebeschreibung des Netzwerks. Für jeden Router werden die tcpdump-Pakete der an den Router angeschlossenen Netze chronologisch sortiert. Für jeden Router wird eine Forwardingtabelle anhand der Dumps aufgebaut. Der Zeitpunkt des letzten Pakets, welches die Forwardingtabelle eines Routers verändert hat wird als Konvergenzzeitpunkt genspeichert. Das erste RIP-Paket aller Dumps wird als Startzeitpunkt der Konvergenzmessung herangezogen. Die Differenz beider Zeitstempel bildet die Konvergenzzeit.

Die Anzahl der Pakete sowie der Traffic werden aus den RIP-Paketen der tcpdump-Dateien berechnet, die zwischen den beiden während der Konvergenzzeitmessung erzeugten Zeitstempeln liegen. Die tcpdump-Dateien geben die Länge der Pakete ohne Header aus. Daher ist auf diese Länge noch der IP-Header von 20 Byte [2] und der UPD Header von 8 Byte [1] aufzuaddieren.

Die Funktionen dazu werden in der von `modules/Parser.pm` abgeleiteten Klasse (hier `modules/RIPParser.pm`) definiert.

### Statisitsche Analyse der Messergebnisse

Die Ergebnisse der Simulationen können mit dem Programm gnuplot graphisch dargestellt werden.

Wichtig ist, dass die Werte durch Leerzeichen voneinander getrennt sind. Um eine solche Datei zu visualisieren wird noch eine Plotdatei benötigt, die Durchmesser, Knotenanzahl, Kantenanzahl oder Clusterkoeffizient auf der x-Achse gegen Konvergenzzeit, Paketanzahl oder Traffic auf der y-Achse darstellt. Eine minimale Plotdatei, welche den Durchmesser und die Konvergenzzeit berücksichtigt könnte so aussehen:

```
plot "messungen.txt" using 1:5
```

Dadurch wird für jeden Messwert die erste Spalte auf der x-Achse und die fünfte Spalte auf der y-Achse aufgetragen. Für 3-dimensionale Graphen wird die Funktion splot verwendet. Im nächsten Beispiel wird die Paketanzahl in Zusammenhang mit Knotenanzahl und Kantenanzahl gebracht:

```
splot "messungen.txt" using 2:3:6
```

Die in dieser Arbeit genutzten Plotdateien haben noch einige weitere Einstellungen um z.B. die Achsen zu beschriften oder die Legende anders zu positionieren. Sie sind auf der beiliegenden CD enthalten oder online unter XXX zu finden.

## Graphentheoretische Analyse von Toplogien

Mit Hilfe der CPAN-Bibliothek „Graph“ ist das Script in der Lage, Graphen aus Topologiebeschreibungen zu erstellen und graphentheoretisch zu analysieren. Es können zvf Dateien eingelesen und in einen Graphen umgewandelt werden und aus Graphen wieder zvf-Dateien erstellt werden. Zudem können einige Eigenschaften der Graphen bestimmt werden, die einen praktischen Nutzen bei der Analyse von Routingprotokollen nahe legen.

Alle Funktionen und Datenstrukturen für die Analyse und Darstellung von Graphen sind in der Datei `modules/GraphTools.pm` definiert (in der auch die Biblitiothek `Graph` eingebunden ist).

Um eine zvf-Datei zu analysieren ruft man das Script mit folgender Syntax auf:

```
./zimulator.pl -z {DATEI}
```

Dabei werden die folgenden graphentheoretischen Eigenschaften berechnet:

- Durchmesser $d$
- Anzahl der Knoten $V$
- Anzahl der inneren Knoten $V_i$
- Blätter
- Anzahl der Kanten $E$
- Artikulationen
- Clusterkoeffizient $cc$
- Anzahl der Kreise (zyklomatische Zahl) $k$

Es können also auch mehrere Dateien gleichzeitig analysiert werden. Das Ergebnis der Analyse wird auf der Konsole ausgegeben sowie als Kommentar der zvf-Datei geschrieben. Alle bisherigen Kommentare gehen dabei jedoch verloren.

Die verwendeten Module einige Einschränkungen bei der Verwendung von Hypergraphen. Diese können z.B. nich mittels GraphViz dargestellt werden.

**Hinweis:** Das Script wurde noch nicht mit Hyperkanten deren Knotenanzahl kleiner als 2 ist getestet.

### Visualisierung von Graphen

Bei der Analyse der Graphen wird automatisch eine png-Datei mit einem Bild des Graphen erstellt. Dieses wird durch die das Programm GraphViz [4] berechnet. Es können die Namen der Netze an die Kanten geschrieben werden. Dies kann man in der Datei `modules/Constants.pm` festlegen.

## Generierung von Topologien

Mit dem Script können über die Option `-g` symmetrische und zufällige Topologien generiert werden. Dabei wird eine zvf-Datei automatisch zusammen mit einer graphischen Darstellung der Topologie als png erstellt und beide Dateien unter einem automatisch generierten oder optional angegebenen Namen gespeichert. (Angabe von Namen noch nicht implementiert)

Die generierten Graphen werden automatisch analysiert und deren Eigenschaften als Kommentare in die zvf-Datei geschrieben. Die verwendeten Funktionen sind in der Datei `modules/GenerateTopology.pm` definiert, die viele Funktionen aus `modules/GraphTools.pm` sowie der CPAN Bibliothek Graph verwendet.

Die generelle Syntax zum Generieren von Topologien ist:

```
./zimulator.pl -g TYPE ARGUMENTS [NAME]
```

Im folgenden werden die möglichen Typen von Topologien mit ihren Argumenten aufgelistet:

| Name | Argumente | Beschreibung |
|---|---|---|
| row | V | Eine Reihe von V Routern. |
| circle | V | Ein Kreis aus V Routern. |
| star2 | V | Ein Router, an den V-1 andere Router direkt angeschlossen sind. |
| star | D R | Ein Router, an den R Reihen der Länge D/2 angeschlossen sind. |
| square | N | Ein Quadrat aus N x N Routern. |
| crown | N | Eine CrownN-Topologie (Siehe auch Seite TODO) |
| circlex_rowy | X Y | Ein Kreis aus X Routern verbunden mit einer Reihe aus Y Routern |
| random | V E | Eine zufällige Topologie aus V Routern verbunden mit E Netzen. |

### Zufällige Topologien

Die Funktionen zur Generierung zufälliger Graphen sind im CPAN-Modul Graph implementiert, welches in der Datei `modules/GraphTools.pm` eingebunden ist. Es werden keine Netze mit Hyperkanten oder Multikanten erstellt.

Die Aufrufsyntax

[4] Siehe http://www.graphviz.org/

```
./zimulator.pl -g random V E
```

wobei V die Anzahl der Knoten angibt, E die Anzahl der Kanten und mit NAME der Name der zu erzeugendenen Dateien angegeben wird. Das folgende Beispiel erstellt eine zufällige Topologie „beispiel“ mit 24 Knoten und 32 Kanten. Es werden die Dateien test.png und test.zvf erzeugt.

**Beispiel:**

```
./zimulator.pl -g random 9 15
```

kann folgende Dateien erzeugen:

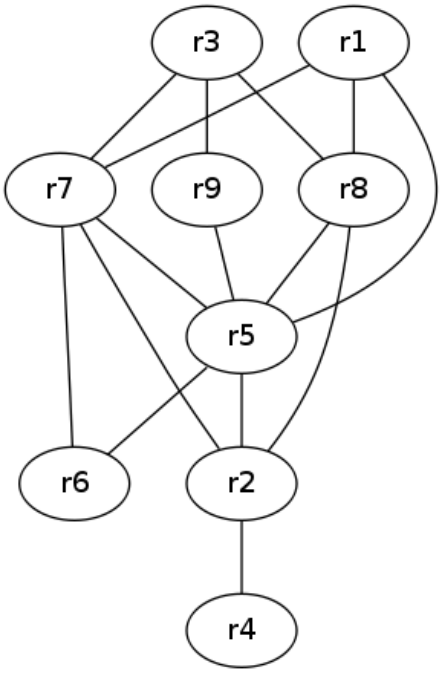


Abbildung 4: Visualisierung der erzeugten Zufallstopologie

random9_15.zvf

```
# random9_15
#
# diameter:                       3
# number of cycles:               7
# clustering coefficient:         0.416666666666667
# leaves:                         r4
# articulations:                  r2

net1,net2,net3,net4,net5,net6,net7,net8,net9,net10,net11,net12,net13,net14
,net15

r1 net2,net3,net5
r2 net10,net11,net13,net15
r3 net1,net7,net14
r4 net11
r5 net4,net5,net8,net9,net10,net12
r6 net4,net6
r7 net2,net6,net7,net12,net13
r8 net3,net9,net14,net15
r9 net1,net8
```

# 3 Pläne

Folgende Features sind noch geplant:

- Bugfixes,TODOs,...
- Dokumentation auf aktuellen Stand bringen
- Automatische Generierung von symmetrischen Netzen wie square, Kreisen, Reihen,...

# 4 Verzeichnisse

# Abbildungsverzeichnis

# Literatur

[1] Postel, J.: *RFC768 - User Datagram Protocol (UDP).* IETF, August 1980.

[2] Postel, J.: *RFC791 - Internet Protocol.* IETF, September 1981.