UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Institut für Informatik

Statistische Konvergenzanalyse des RIP-Routingprotokolls

Diplomarbeit

zur Erlangung des Grades eines
Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Marcel Jakobs 204210335

Betreuer: Prof. Dr. Christoph Steigner, Institut für Informatik, AG Rechnernetze, Fachbereich 4: Informatik

Koblenz, im April 2010

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet 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

Inhaltsverzeichnis

1 Einleitung

Diese Ausarbeitung wurde von Marcel Jakobs im Rahmen einer Diplomarbeit bei Herrn Prof. Dr. Christoph Steigner erstellt.

Ein Netzwerk, wie beispielsweise das Internet, ist eine Menge von Netzen, die durch Router miteinander verbunden sind. Ein Router ist ein Computer, der mit mehreren Netzwerkschnittstellen ausgerüstet und an mehrere Netze angeschlossen ist, um zwischen diesen Pakete zu vermitteln. Man kann ein Netzwerk auch als Graph repräsentieren, wobei Router als Knoten und Netze als Kanten angesehen werden können. Diesen Graph nennt man die Topologie des Netzwerks.

Soll ein Paket in ein anderes Netz als das eigene gesendet werden, so wird es normalerweise dem sogenannten Default-Router gesendet. Dieser besitzt (wie jeder Router) eine Tabelle (die sogenannte Forwardingtabelle), die alle Netze enthält. Zusätzlich ist in der Tabelle der jeweilige Router eingetragen, über den das Netz am besten erreicht werden kann. So wird das Paket von einem Router zum nächsten geleitet, bis es das Zielnetz erreicht. Dabei schlägt jeder Router in seiner Tabelle nach, welches der nächste Router auf dem günstigsten Weg zum Zielnetz ist. Ein Routingprotokoll kümmert sich um den automatischen Austausch von Informationen zwischen den Routern, um die Forwardingtabelle aufzubauen und auf dem aktuellen Stand zu halten. Sind die Tabellen aller Router auf dem aktuellen Stand, so befindet sich das Netzwerk in einem konvergenten Zustand. Die Zeit, die benötigt wird, um die Forwardingtabelle aufzubauen beziehungsweise sie nach einer Änderung der Topologie zu aktualisieren, wird Konvergenzzeit genannt.

Das Routingprotokoll RIP ist ein bekanntes und gut erforschtes Distanzvektor-Protokoll. Jedoch gibt es bisher nur wenige Untersuchungen der Konvergenzeigenschaften (wie z.B. benötigte Zeit, um in einen konvergenten Zustand zu gelangen, oder das dabei erzeugte Trafficvolumen) dieses Protokolls.

In dieser Arbeit soll das Konvergenzverhalten von RIP in Abhängigkeit von Eigenschaften der Topologie experimentell untersucht werden. Ziel ist es, einen direkten Zusammenhang zwischen einer Konvergenzeigenschaft und einer oder mehreren Topologieeigenschaften zu ermitteln. Dadurch wäre es möglich, bereits aus der Topologie eines Netzwerks auf seine Konvergenzeigenschaften zu schließen.

In Kapitel 1 werden die Ziele dieser Arbeit sowie die verwendeten Begriffe definiert. Kapitel 2 führt in das zu untersuchende Routingprotokoll RIP ein. Das 3. Kapitel beschäftigt sich mit Simulationsprogrammen, die für diese Arbeit in Frage kommen. Die für diese Arbeit verwendete Testumgebung wird in Kapitel 4 erläutert. Einige im Vorfeld getroffene Annahmen und die daraus resultierenden Erwartungswerte werden in Kapitel 5 diskutiert. Kapitel 6 beschäftigt sich mit den getesteten Topologien. Um die Erwartungswerte aus Kapitel 5 zu untersuchen, wurden einige Messungen mit speziell ausgewählten Topologien in Kapitel 7 ausgewertet. Mit statistischen Verfahren wurden in Kapitel 8 Zusammenhänge zwischen je einer Topologieeigenschaft und einer Konvergenzeigenschaft ausgearbeitet. In Kapitel 8.7 werden genauere Berechnungsformeln unter Einbeziehung mehrerer Topologieeigenschaften vorgestellt. In Kapitel 9 werden die gewonnenen Erkenntnisse noch einmal zusammengefasst und ein Fazit daraus gezogen. Im Anhang (Kapitel 10) befindet sich die Bedienungsanleitung des für diese Arbeit implementierten Programms sowie diverse Verzeichnisse.

Die beiliegenden CD enthält diese Ausarbeitung, das für diese Arbeit implementierte Programm inklusive Dokumentation sowie alle Beispieldateien aus dieser Ausarbeitung. Daneben sind alle für diese Ausarbeitung erstellten gnuplot-Dateien beigelegt. Der komplette CD-Inhalt (außer dieser Ausarbeitung) ist auch online unter `http://github.com/zimon/zimulator` erhältlich.

1.1 Terminologie

Im Folgenden werden die in dieser Arbeit verwendeten Begriffe definiert.

Netzwerk-Terminologie

Netz: Ein *Netz* (auch *Subnetz* oder *lokales Netz* genannt) ist eine Menge von Rechnern, die direkt miteinander verbunden sind. In einem Netz kann jeder Rechner jeden anderen Rechner direkt über seine *IP-Adresse* erreichen. Um einen Rechner in einem anderen Netz zu kontaktieren, müssen die Netze über Router verbunden sein.

IP-Adresse: Eine *IP-Adresse* ist ein 32-Bit-Wert, der einen Rechner in einem Netzwerk eindeutig adressiert. Dafür wird das IP-Protokoll [Pos81] genutzt.

Routing: Als *Routing* wird das Festlegen von Verbindungswegen für Paketübertragungen in Rechnernetzen bezeichnet. Ein *Router* verbindet zwei oder mehr *Netze* miteinander. Mehrere Router tauschen Informationen aus, damit auch weiter entfernte Netze erreichbar sind.

Routingprotokoll: Ein *Routingprotokoll* ist ein Algorithmus, der eine Strategie für *Routing* beschreibt.

Netzwerk: Eine Menge von verbundenen *Netzen* und *Routern* wird als *Netzwerk* bezeichnet.

Device: Ein *Device* (oder auch *Interface*) entspricht in dieser Arbeit einer Netzwerkkarte oder einem ähnlichen Gerät, über das Pakete von einem *Router* oder *Endgerät* in ein *Netz* gesendet werden können. *Router* haben in der Regel mehrere Devices, für jedes angeschlossene *Netz* eins.

Topologie: Als *Topologie* wird der einem *Netzwerk* zugrunde liegende Graph bezeichnet, in dem *Router* als *Knoten* und *Netze* als *Kanten* angesehen werden. Solche Graphen werden in dieser Arbeit auf ihre Eigenschaften hin analysiert und versucht, das Konvergenzverhalten der entsprechenden Netzwerke mit ihren *Topologieeigenschaften* zu erklären.

Topologieeigenschaft: *Topologieeigenschaften* sind die bei einer graphentheoretischen Untersuchung errechneten Größen (siehe auch Kapitel 4.2 auf Seite 24).

Topologieklasse: Als *Topologieklasse* wird im Folgenden eine Menge von *Topologien* bezeichnet, die durch einen Algorithmus beschrieben werden können. Die in Topologieklassen zusammengefassten Topologien sind meist symmetrisch, besitzen also eine regelmäßige Struktur.

RIP: Das *Routing Information Protocol (RIP)* ist ein Distanzvektor-*Routingprotokoll*. Eine genauere Beschreibung des Protokolls und der zugehörigen Terminologie ist im Kapitel 2 „Das Routingprotokoll RIP“ auf Seite 17 zu finden.

NextHop: Der *NextHop* ist derjenige *Router*, an den alle Pakete für ein bestimmtes Zielnetz weiter geleitet werden. In der *Forwardingtabelle* ist für jedes Netz der entsprechende NextHop eingetragen.

Route: Eine *Route* ist streng genommen ein Pfad in einer *Topologie*. Hier wird jedoch gerade im Zusammenhang mit *Routing-* und *Forwardingtabellen* ein *Netz* inklusive *NextHop* beziehungsweise *Device*, über welches das Netz erreichbar ist, als Route bezeichnet. Im Allgemeinen gehört zu einer Route auch eine *Metrik*.

Metrik: Die *Metrik* ist eine Größe, die die Distanz zu einem entfernten *Subnetz* einer *Route* angibt.

Routingtabelle: Die *Routingtabelle* ist eine Tabelle, die vom *Routingprotokoll* verwaltet wird, und dient der Berechnung von besten Wegen durch die *Topologie*. Aus der Routingtabelle wird die *Forwardingtabelle* generiert.

Forwardingtabelle: Die *Forwardingtabelle* ist eine von Betriebssystem-Kernel verwaltete Tabelle mit *Routen* und den zugehörigen *Metriken*. Die Forwardingtabelle wird für die Vermittlung von Paketen genutzt.

Konvergenz: Ein *Netzwerk* ist genau dann *konvergent* (bzw. in einem *konvergenten Zustand*), wenn sich die *Topologie* nicht verändert und keine Routingpakete mehr versendet werden, die den Inhalt einer *Forwardingtabelle* verändern.

Konvergenzeigenschaft: *Konvergenzeigenschaften* sind Größen, die den Prozess eines *Netzwerks*, einen *konvergenten Zustand* zu erreichen, quantifizieren.

Konvergenzzeit: Die *Konvergenzzeit* ist eine *Konvergenzeigenschaft*, die angibt, wie viel Zeit ein *Netzwerk* benötigt, um von einem bestimmten Zustand in einen *konvergenten Zustand* zu wechseln.

Trafficvolumen: Das *Trafficvolumen* ist eine *Konvergenzeigenschaft*, die angibt, wie groß die von *Routern* versendete Datenmenge ist, bis ein *Netzwerk* von einem bestimmten Zustand einen *konvergenten Zustand* erreicht. In dieser Arbeit ist mit dem Begriff *Traffic* oder *Updatetraffic* das Trafficvolumen gemeint.

Payload-Traffic: Unter dem Begriff *Payload-Traffic* versteht man die versendete Datenmenge, die von Anwendungsprogrammen verursacht wird. Dazu zählen zum Beispiel Pakete des HTTP- oder FTP-Protokolls. Pakete von Routern sowie von Diensten, die keine Nutzdaten übertragen (wie zum Beispiel ARP oder Ping), werden nicht dazu gerechnet.

Definitionen der Simulationsbegriffe

Testfall: Ein *Testfall* ist eine genau definierte Folge von Ereignissen, die während der *Messung* einer *Topologie* ausgelöst werden. Dabei kann ein Testfall auch auf verschiedene Topologien angewendet werden.

Test-Script: Ein *Test-Script* ist eine Datei, die einen *Testfall* genau beschreibt.

Durchlauf: Ein *Durchlauf* ist die einmalige Durchführung eines *Testfalls* mit einer *Topologie*.

Messung: Eine Messung ist die Ermittlung eines *Ergebnisses* während oder nach einem *Durchlauf*.

Simulation: Eine *Simulation* ist eine Menge von *Messungen* von einem *Testfall* in einer *Topologie*.

Ergebnis: Ein *Ergebnis* ist eine Größe, welche das Resultat einer *Messung* quantifiziert.

Topologiebeschreibung: Eine *Topologiebeschreibung* ist eine Datei, die in einer speziellen Syntax geschrieben ist und eine *Topologie* repräsentiert. Bei der Analyse einer Topologie werden die *Topologieeigenschaften* ebenfalls in die Topologiebeschreibung geschrieben, damit Topologien leicht anhand verschiedener Kriterien gesucht werden können.

Ausführungsbeschreibung: Eine *Ausführungsbeschreibung* ist eine Datei, in der aufgeführt ist, wie viele *Durchläufe* ein *Test-Script* jeweils mit einer *Topologie* simulieren soll.

Simulationsordner: Im *Simulationsordner* werden alle Zwischen- und Endergebnisse sowie temporäre Dateien gespeichert. Der Name eines Simulationsordners setzt sich aus dem Namen des *Test-Scripts* und dem Namen der simulierten *Topologie* zusammen.

Szenario: Als *Szenario* (oder *VNUML-Netzwerk*) wird in dieser Arbeit eine mittels VNUML (siehe Kapitel 3 „Auswahl des Simulationsprogramms“ auf Seite 19) gestartete Menge von UML-Rechnern bezeichnet. Dies ist das *Netzwerk*, mit dem *Messungen* durchgeführt werden. Oft wird auch die *VNUML-Beschreibungsdatei* als Szenario bezeichnet. Diese XML-Dateien werden hier jedoch *VNUML-Konfigurationsdateien* oder *VNUML-Beschreibungsdateien* genannt.

Ergebnisdatei: Die *Ergebnisdatei* (auch *Resultfile* genannt) enthält pro *Durchlauf* eine Zeile mit den *Topologieeigenschaften* der getesteten *Topologie* und den Ergebnissen der *Messung* (also den *Konvergenzeigenschaften* für diesen Durchlauf).

tcpdump: *tcpdump*[1] ist ein Programm, mit dem sich der Netzwerkverkehr eines *Devices* mitschneiden lässt.

tcpdump-Prozess: Ein *tcpdump-Prozess* ist eine Instanz des *tcpdump*-Programms auf einem *Device*. In den Simulationen dieser Arbeit wird pro *Netz* ein tcpdump-Prozess gestartet, sodass alle Netze überwacht werden.

[1] http://www.tcpdump.org

tcpdump-Datei: Die von einem *tcpdump-Prozess* mitgeschnittenen Pakete werden in eine *tcpdump-Datei* gespeichert. In dieser Datei sind die kompletten Pakete enthalten, die über das entsprechende *Netz* oder *Interface* gesendet wurden.

Graphentheoretische Begriffe und Definitionen

Im Folgenden werden einige verwendete Begriffe aus der Graphentheorie definiert.

Graph: Ein *Graph* $G = (V, E)$ ist eine Menge von n *Knoten* $V = v_1, v_2, \cdots, v_n$, die durch eine Menge von m *Kanten* $E \subseteq V \times V$ mit $E = e_1, e_2, \cdots, e_m$ verbunden sind.

Gerichteter Graph: Ein *Graph* heißt gerichtet, wenn seine *Kanten* eine Richtungsmarkierung besitzen. Es ist für die Objekte, die durch den Graphen repräsentiert werden, nur möglich in eine Richtung über die Kante zu laufen. In dieser Arbeit werden jedoch nur ungerichtete Graphen betrachtet.

Hypergraph: *Kanten*, die mehr oder weniger als zwei Enden besitzen, bezeichnet man als *Hyperkanten*. *Graphen* mit Hyperkanten werden als *Hypergraphen* bezeichnet.

Multigraph: *Graphen* mit *Mehrfachkanten* (auch *Multikanten* genannt) besitzen mehrere *Kanten* zwischen zwei *Knoten* v_a und v_b. *Graphen* mit Multikanten werden als *Multigraphen* bezeichnet.

Blatt: Ein *Blatt* b ist ein *Knoten*, der mit genau einer *Kante* verbunden ist. Der Begriff „Blatt“ ist eigentlich nur für *Bäume* definiert, wird in dieser Arbeit jedoch auf alle *Graphen* erweitert.

Innerer Knoten: Ein *innerer Knoten* v_i ist ein *Knoten*, der mit mehr als einer *Kante* verbunden ist. Die Menge aller inneren Knoten wird als V_i bezeichnet. Der Begriff „innerer Knoten“ ist eigentlich nur für *Bäume* definiert, wird in dieser Arbeit jedoch auf alle *Graphen* erweitert.

Da es in der Netzwerkterminologie keine Unterscheidung zwischen Routern mit einem Netz[2] und solchen mit mehreren Netzen gemacht wird, werden die Begriffe *Blatt* und *innerer Knoten* auch für die entsprechenden Router genutzt.

Pfad: Ein Pfad P ist eine alternierende Sequenz von miteinander verbundenen Knoten V_p und Kanten E_p. Sie beginnt und endet mit einem Knoten. Die Länge P_l eines Pfades ist durch folgende Formel definiert [Die06, Seite 7]:

$$P_l = m_p \tag{1}$$

mit m_p = Anzahl der Kanten in P.

Ein *Weg* ist ein Pfad, in dem kein Knoten mehrmals vorkommt.

[2] Es würde auch nicht viel Sinn ergeben, einen Router an nur ein Netz anzuschließen, da er keine Pakete vermitteln könnte. Siehe Kapitel 5.4 für eine Erklärung, warum hier trotzdem mit solchen Routern gearbeitet wird.

Distanz: Die *Distanz* (auch *Abstand* genannt) zwischen zwei Knoten v_a und v_b entspricht der Länge des kürzesten Pfades P zwischen ihnen. Das heißt wenn es keinen anderen Pfad zwischen v_a und v_b gibt, der kürzer ist als P.

Erreichbarkeit: Ein *Knoten* v_a ist von einem anderen *Knoten* v_b erreichbar, wenn es einen *Pfad* von v_b nach v_a gibt.

Zusammenhang: Ein Graph ist *zusammenhängend*, wenn jeder *Knoten* von jedem anderen *Knoten* aus erreichbar ist. In dieser Arbeit werden nur zusammenhängende Graphen betrachtet. Ein zusammenhängender Graph ohne Hyperkanten hat immer eine minimale Kantenzahl m_{min} für eine gegebene Knotenzahl mit:

$$m_{min} = n - 1 \tag{2}$$

Begründung:
Ein einzelner Knoten ist immer ein zusammenhängender Graph, auch bei $n - 1 = 0$ Kanten.
Zwei Knoten v_a und v_b brauchen genau $n - 1 = 1$ Kante, um verbunden zu werden.
Für jeden weiteren Knoten wird wieder eine weitere Kante benötigt, um sie mit dem vorhandenen Graph zu verbinden.

Für eine gegebene Kantenzahl kann ein zusammenhängender Graph (ohne Hyperkanten) entsprechend nur eine maximale Anzahl von Knoten n_{max} besitzen:

$$n_{max} = m + 1 \tag{3}$$

Begründung:
Eine Kante kann höchstens $m + 1 = 2$ Knoten miteinander verbinden.
Jede weitere Kante muss auf der einen Seite mit dem bereits vorhandenen Graphen mit n Knoten verbunden sein und kann somit nur einen weiteren – also $n + 1$ – Knoten mit dem Graphen verbinden.

Brücke: Eine *Brücke* ist eine *Kante*, bei deren Entfernung der *Graph* in zwei Teilgraphen zerfallen würde, also nicht mehr *zusammenhängend* wäre.

Artikulation: Eine *Artikulation* ist ein *Knoten*, bei dessen Entfernung der *Graph* in zwei Teilgraphen zerfallen würde, also nicht mehr *zusammenhängend* wäre.

Kreis: Ein *Pfad* P zwischen zwei *Knoten* v_a und v_b heißt *Kreis*, wenn gilt: $v_a = v_b$. Also wenn Start- und Endknoten identisch sind.

Kreiszahl: Die Anzahl der *Kreise* k in einem *Graphen* wird *Kreiszahl* oder *zyklomatische Zahl* genannt und entspricht der Anzahl der redundanten Kanten. Die zyklomatische Zahl k berechnet sich für *zusammenhängende Graphen* durch Formel (4) [Ber73, Seite 19f].

$$k = m - n + 1 \tag{4}$$

In *zusammenhängenden Hypergraphen* wird die zyklomatische Zahl durch Formel (5) [Ber73, Seite 391] berechnet.

$$k = \sum_i (|e_i| - 1) - n + 1 \tag{5}$$

mit $|e_i|$ = Anzahl der mit der Kante e_i verbundenen Knoten.

Baum: Bei *ungerichteten Graphen* ist ein *Baum* ein *Graph* ohne *Kreise*.

Kompletter Graph: Ein *kompletter Graph* (auch *Full-Mesh*, *Mesh* oder *Clique* genannt) ist ein *Graph*, bei dem jeder *Knoten* mit jedem anderen Knoten über eine eigene *Kante* verbunden ist. Die Anzahl der Kanten eines kompletten Graphen entspricht der maximal möglichen Anzahl von Kanten in einem Graph ohne *Multikanten* und lässt sich durch Formel (6) [Deo04, Seite 22] berechnen.

$$m = \frac{n \cdot (n-1)}{2} \tag{6}$$

Durchmesser: Der *Durchmesser* d eines *Graphen* entspricht der größten *Distanz* im *Graph*. Aus gegebener *Knoten*- und *Kanten*anzahl lässt sich der minimal (d_{min}, siehe Formel (7)) sowie maximal mögliche Durchmesser (d_{max}, siehe Formel (8)) für Graphen ohne Hyperkanten oder Multikanten berechnen. Der maximal (bzw. minimal) mögliche Durchmesser wird im Folgenden auch maximaler (bzw. minimaler) Durchmesser genannt.

$$d_{min} = \begin{cases} 1, & \text{für } \textit{komplette Graphen} \\ 2, & \text{für alle anderen } \textit{zusammenhängenden Graphen} \end{cases} \tag{7}$$

Begründung:
$d_{min} = 1$ *für komplette Graphen:* Da in einem kompletten Graph jeder Knoten mit jedem anderen Knoten verbunden ist, kann jeder Knoten auch von jedem anderen Knoten direkt – also mit einem Schritt – erreicht werden. Daher ist der Durchmesser bei kompletten Graphen immer 1.

$d_{min} = 2$ *für sonstige zusammenhängende Graphen:* Eine Menge von Knoten mit einer minimalen Anzahl von Kanten nach Formel (2), kann immer zu einer Sterntopologie geformt werden, bei der sich ein Knoten in der Mitte befindet und alle anderen Knoten durch je eine Kante mit dem mittleren Knoten verbunden sind. Diese Sterntopologien haben immer den Durchmesser 2. Jede weitere Kante kann nun zwischen zwei Knoten angebracht werden, ohne dabei den Durchmesser zu verändern. Erst bei der letzten möglichen Kante, welche den Graphen komplettiert, sinkt der Durchmesser auf 1.

$$d_{max} = n - \lfloor \frac{\sqrt{8 \cdot k + 1} + 1}{2} \rfloor = \lfloor n - \frac{\sqrt{8 \cdot (m - n + 1) + 1} + 1}{2} \rfloor \tag{8}$$

Herleitung:

Den maximalen Durchmesser bei n Knoten hat eine Reihe mit $m = n - 1$ Kanten. Fügt man weitere Kanten hinzu, so verringert sich der maximale Durchmesser wie es beispielhaft in Abbildung 1 dargestellt ist. Man kann jedes Mal, wenn sich der maximal mögliche Durchmesser verringert, eine Kante mehr hinzufügen, bis er sich erneut verringert.

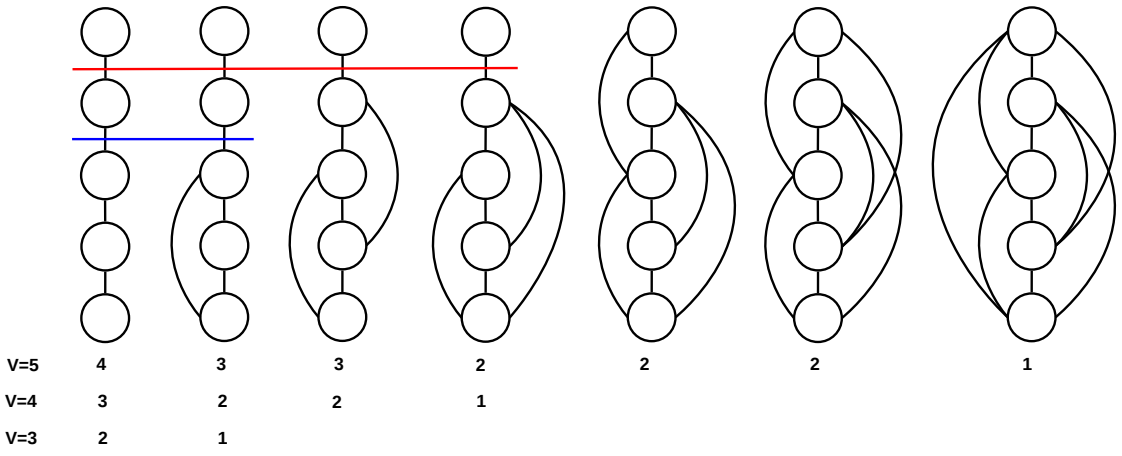

Abbildung 1: Verringerung des maximalen Durchmessers bei zunehmender Kantenzahl

Dieses Verhalten kann dadurch erklärt werden, dass durch das Hinzufügen einer Kante, die den maximalen Durchmesser senkt, ein Teilgraph komplettiert wird. Alle weiteren Kanten, die dann eingefügt werden können, ohne dass sich der maximal mögliche Durchmesser ändert, verbinden

einen neuen (im bisherigen Teilgraphen nicht vorhandenen) Knoten mit dem kompletten bisherigen Teilgraphen. Dieser neue Teilgraph hat einen Knoten mehr als der vorherige, wodurch eine Kante mehr eingefügt werden kann, bis sich der maximale Durchmesser erneut verringert. Abbildung 1 veranschaulicht, wie zuerst der Teilgraph mit 3 Knoten (unterhalb der blauen Linie) und dann der nächste Teilgraph mit 4 Knoten (unterhalb der roten Linie) komplettiert wird, bis schließlich der gesamte Graph komplettiert ist womit der maximale Durchmesser auf 1 sinkt.

Eine Perl-Funktion, die für gegebene Knoten- und Kantenzahlen den jeweils maximalen Durchmesser errechnet, ist Quelltext 1 zu entnehmen.

```
sub maxDiameter{
    my ($n,$m) = @_; # Parameter: Anzahl der Knoten und Kanten

# Kanten, die eingefuegt werden muessen, damit sich der max. Durchmesser
# verringert:
    my $x = 1;

    my $mm = $n-1; # Bisher berechnete Anzahl an Kanten
    my $maxd; # Maximaler Durchmesser

    # Undefiniert, wenn nicht zusammenhaengend oder Multigraph:
    return -1 if $m < $mm or $m > ($n*($n-1))/2;

    for($maxd = $v-1;$maxd>=1;$maxd--){ # maxd wird immer verringert,
        for(my $j=0;$j<$x;$j++){ # wenn die maximale Anzahl der Kanten
                    # erreicht ist, die den max. Durchmesser nicht senken

     # Fertig, wenn bisher berechnete Kanten den Gesamtkanten entsprechen:
            return $maxd if $m == $mm;

            $mm++; # Anzahl der berechneten Kanten erhoehen
        }

        # Nach jedem Durchlauf kann eine Kante mehr hinzugefuegt werden:
        $x++;
    }
}
```

Quelltext 1: Berechnung des maximalen Durchmessers

Der entscheidende Parameter ist die Anzahl der Kanten, die einer Reihe hinzugefügt werden, also die zyklomatische Zahl k. Demnach wird eine Funktion $f(k)$ gesucht, die die Änderungen des maximal möglichen Durchmessers beschreibt, sodass Formel (9) gilt. In Tabelle 1 wurden die Funktionswerte von $f(k)$ aufgelistet.

$$d_{max} = n - f(k) \tag{9}$$

Da die Komplettierung von Teilgraphen jeweils den maximalen Durchmesser senken, liegt es nahe, die entsprechende Funktion (siehe Formel (6)) genauer zu betrachten. Wie Tabelle 2 zeigt,

gibt Formel (6) genau die Stellen an, an denen sich der maximale Durchmesser verringert. Somit würde die Umkehrfunktion fast der gesuchten Funktion entsprechen. Um die Werte zwischen den Änderungen des maximalen Durchmessers zu berechnen, müssen diese abgerundet werden. Formel (10) entspricht der Umkehrfunktion zu Formel (6). Da nur positive Zahlen sinnvoll sind, ergibt sich aus n_1 die gesuchte Formel, welche für Tabelle 1 die Stellen berechnet, an denen sich der Funktionswert ändert. Aus den oben beschriebenen Gründen wurde n_1 noch abgerundet und die Anzahl der Kanten durch die zyklomatische Zahl ersetzt, woraus sich Formel (11) ergibt. Schließlich wurde sie in Formel (9) eingesetzt, um die endgültige Formel (8) für die Berechnung des Durchmessers zu erhalten.

k	0	1	2	3	4	5	6	7	8	9	10	...
$f(k)$	1	2	2	3	3	3	4	4	4	4	5	...

Tabelle 1: Tabellarische Funktion zur Änderung des maximalen Durchmessers

n	1	2	3	4	5	6	7	8	9	10	...
$g(n)$	0	1	3	6	10	15	21	28	36	45	...

Tabelle 2: Tabellarische Funktion zu Formel (6)

$$n_1 = \frac{\sqrt{8 \cdot m + 1} + 1}{2}, n_2 = -\frac{\sqrt{8 \cdot m + 1} - 1}{2} \tag{10}$$

$$f(k) = \lfloor \frac{\sqrt{8 \cdot k + 1} + 1}{2} \rfloor \tag{11}$$

Clusterkoeffizient: Der *Clusterkoeffizient cc* ist das Verhältnis von Anzahl der *Kanten* m in einem *Graphen* zur Anzahl der möglichen *Kanten*, die der Graph ohne *Multikanten* enthalten kann (siehe Formel (6)). Er berechnet sich durch

$$cc = \frac{m}{\frac{n \cdot (n-1)}{2}} = \frac{2m}{n \cdot (n-1)} \tag{12}$$

Quellen: [Ber73], [Die06], [Deo04]

Statistische Begriffe

Durchschnitt: Als *Durchschnitt* wird in dieser Arbeit der arithmetische Mittelwert aus mehreren Messergebnissen bezeichnet.

Regressionsanalyse: Die *Regressionsanalyse* beschäftigt sich mit der statistischen Analyse von Beziehungen zwischen einer abhängigen und (einer oder mehreren) unabhängigen Variablen.

Methode der kleinsten Quadrate: Die *Methode der kleinsten Quadrate* ist ein Instrument der *Regressionsanalyse*. Bei dieser Methode wird versucht, eine Funktion zu finden, dessen Funktionsgraph möglichst nahe bei den Messergebnissen liegt. Die beste Funktion definiert man als diejenige, deren *mittlerer quadratischer Fehler* minimal ist.

Approximierende Funktion: Die Funktion, die durch die *Methode der kleinsten Quadrate* berechnet wurde, wird im Folgenden *approximierende Funktion* genannt.

Varianz: Die *Varianz* ist ein Maß für die Streuung von Messwerten und wird berechnet, indem man den *Durchschnitt* der quadrierten Abstände der Messwerte zum Mittelwert bildet. Für klassische reellwertige Funktionen entspricht die Varianz der *mittleren quadratischen Abweichung (MSE)*, weshalb beide Begriffe in dieser Arbeit synonym verwendet werden.

Standardabweichung: Die *Standardabweichung* ist eines der wichtigsten Maße bei der *Regressionsanalyse* und gibt die Genauigkeit der approximierenden Funktion an. Sie entspricht der Quadratwurzel der Varianz. Da hier von normalverteilten Messwerten ausgegangen werden kann, beschreibt die Standardabweichung einen Bereich um die approximierende Funktion, in dem näherungsweise 68% aller Messwerte liegen.

Quellen: [Hä67]

Verwendete Symbole

In dieser Arbeit werden folgende Symbole verwendet:

- V – Menge der Knoten in einem Graphen
- n – Anzahl der Knoten in einem Graphen
- E – Menge der Kanten in einem Graphen
- m – Anzahl der Kanten in einem Graphen
- d – Durchmesser eines Graphen
- B – Anzahl der Blätter eines Graphen
- V_i – Menge der inneren Knoten eines Graphen
- n_i – Anzahl der inneren Knoten eines Graphen
- t_s – Startzeitpunkt einer Messung
- t_e – Endzeitpunkt einer Messung
- t – Konvergenzzeit eines Durchlaufs oder einer Simulation
- t_{approx} – Durch eine Funktion approximierte Konvergenzzeit
- cc – Clusterkoeffizient
- T – Trafficvolumen eines Durchlaufs oder einer Simulation
- T_{approx} – Durch eine Funktion approximiertes Trafficvolumen
- T_a – Durchschnittliches Trafficvolumen pro Netz eines Durchlaufs oder einer Simulation
- $T_{a_{approx}}$ – Durch eine Funktion approximiertes durchschnittliches Trafficvolumen pro Netz
- k – Anzahl der Kreise in einem Graphen (zyklomatische Zahl)

2 Das Routingprotokoll RIP

Das Routing Information Protocol (RIP) ist ein dynamisches Routingprotokoll für lokale Netzwerke. Es ist ein Distanzvektor-Protokoll, welches häufig für Lehre und Forschung verwendet wird. Der Algorithmus selbst stammt von Ford und Fulkerson [FF62] während die Berechnung der kürzesten Wege auf R.E. Bellman [Bel57] zurück geht. Daher werden Distanzvektorprotokolle auch Bellman-Ford-Protokolle genannt. Bei diesem Algorithmus sendet jeder Router alle direkt angeschlossenen Netze mit der Metrik 1 zu all seinen Nachbarroutern. Bekommt ein Router eine Route zugesandt, so wird in der Routingtabelle nachgeschlagen, ob sich der Weg zu diesem Netz durch die neue Route verkürzen würde. Ist dies der Fall, so trägt er die Route in seine Routingtabelle ein und sendet die Information an alle anderen seiner direkten Nachbarn mit inkrementierter Metrik. Die Route wird dabei auch in die Forwardingtabelle übertragen.

Metrik: Die *Metrik* bezeichnet die Länge des kürzesten *Pfades* zu einem *Netz*. Die Metrik 16 entspricht dabei einer unendlichen Entfernung und kennzeichnet ein Netz als unerreichbar.

RIP kennt folgende Arten von Paketen die mittels UDP [Pos80] übertragen werden:

Request: Mit einem *Request* wird eine *Response-Nachricht* angefragt, welche die gesamte oder in speziellen Fällen einen Teil der *Forwardingtabelle* enthält. Normalerweise werden Request-Pakete verschickt, wenn ein *Router* gerade erst zum *Netzwerk* hinzugekommen ist und seine Routing- und Forwardingtabelle aufbauen möchte. In diesem Fall wird die Request-Nachricht per Multicast übertragen. Es ist jedoch auch möglich, mit Requests einen einzelnen Router nach einer speziellen Route zu fragen.

Periodisches Update: Ein *Periodisches Update* ist eine *Response-Nachricht*, die regelmäßig gesendet wird, wenn der *Update Timer* abgelaufen ist. Es enthält die gesamte *Forwardingtabelle* des *Routers* abzüglich der *Routen*, die von demjenigen Router gelernt wurden, an den das Paket gesendet wird.

Triggered Update: Das *Triggered Update* ist eine *Response-Nachricht*, welche versendet wird, wenn eine neue *Route* bekannt wird oder sich die Metrik einer Route ändert. Steht ein *Periodisches Update* an, so wird das Triggered Update zurück gehalten und zusammen mit dem Periodischen Update versendet, um Bandbreite zu sparen.

Daneben gibt es noch folgende Timer:

Update Timer: Jedes Mal, wenn der *Update Timer* abläuft, wird ein *Periodisches Update* versendet. Er beträgt als Standard 30 Sekunden, wurde in den Simulationen zu dieser Arbeit jedoch auf 10 Sekunden verkürzt, um Ergebnisse zu erlangen, die sich mit anderen Routingprotokollen besser vergleichen lassen. Es wurde angenommen, dass diese Änderung für die Coldstart-Tests, bei denen keine Topologieänderungen auftreten, die Messergebnisse nicht beeinflusst.

Der Update Timer dient vor allem dazu, den anderen Routern mitzuteilen, dass der Router noch läuft und die Verbindung funktioniert. Des Weiteren kann man so sicher stellen, dass alle Informationen weiter gegeben werden, auch wenn ein Triggered Update verloren geht.

Timeout Timer: Der *Timeout Timer* ist normalerweise 180 Sekunden lang und wird von einer *Response-Nachricht* zurückgesetzt. Läuft er ab, so wird die zugehörige *Route* auf die *Metrik* 16 – also unerreichbar gesetzt. Der Timeout Timer dient der Erkennung von Topologieänderungen. Insbesondere Ausfälle von Routern, Devices oder Kabeln sollen damit erkannt werden.

Garbage Collection Timer: Der *Garbage Collection Timer* beginnt zu laufen, sobald der *Timeout Timer* abgelaufen ist. Die Timerzeit beträgt 120 Sekunden. Wenn der Garbage Collection Timer abgelaufen ist, wird die *Route* endgültig gelöscht. Bis dahin verbleibt sie mit der *Metrik* 16 in der *Routingtabelle*.

In der Praxis wurde RIP nahezu vollständig von OSPF [Moy98] und IS-IS [Ora90] abgelöst. Trotzdem sind Distanzvektorprotokolle weiterhin auch von praktischer Bedeutung, da es mit ihnen möglich ist, Policies [SL03] zu verwenden und diese zu propagieren. Des Weiteren wird im Bereich der mobilen ad-hoc-Netzwerke mit Distanzvektorprotokollen wie DSDV [PB94] oder AODV [PBRD03] geforscht.

Quellen: [Boh08], [Mal98]

3 Auswahl des Simulationsprogramms

Es wurden verschiedene Simulationsprogramme auf ihre Eignung für diese Diplomarbeit hin untersucht. Dabei wurde vor allem auf folgende Eigenschaften Wert gelegt:

- Die Software sollte Zugriff auf Routinginformationen bieten, um zumindest den Zeitpunkt der Konvergenz und damit die Konvergenzzeit bestimmen zu können.
- Die Software sollte unter Linux laufen, um die Vorteile der dort bereits vorhandenen Werkzeuge nutzen zu können.
- Die Software sollte möglichst ohne graphische Oberfläche auskommen, um die Simulationen auf mehreren Rechnern parallel durchführen zu können. Dabei soll auf die Rechner per ssh[3] zugegriffen werden.
- Die Software sollte per Script gesteuert werden können, um die Simulationen zu automatisieren
- RIP sollte bereits implementiert sein
- Die Software sollte möglichst unter einer freien Lizenz stehen. Dies lässt die Möglichkeit offen, den Quelltext abzuändern, wenn dies nötig sein sollte. Des Weiteren ist freie Software, vorteilhaft um diese Arbeit nachvollziehen und weiterführen zu können.
- Die Software sollte möglichst große Netze simulieren können.
- Die Software sollte möglichst realistische Ergebnisse liefern.

Es können grob zwei konkurrierende Ziele bei Simulationsprogrammen unterschieden werden. Die Simulation großer Netzwerke durch starke Abstraktion und eine möglichst große Genauigkeit der Simulation durch eine möglichst geringe Abstraktion. Bei stark abstrahierenden Simulationsprogrammen werden nur bestimmte Aspekte oder Teile der eigentlichen Netzwerkübertragung simuliert während andere Aspekte vernachlässigt werden. Dadurch ist die Simulation sehr ressourcenschonend und kann auch sehr große Netzwerke simulieren. Bei einem möglichst niedrigen Abstraktionsgrad werden die Netzwerkprotokolle möglichst bis ins Detail simuliert, was recht aufwändig ist und somit nur relativ kleine Netzwerke zulässt. Diese werden dafür umso genauer simuliert. Somit stehen die beiden Anforderungen, große Netze simulieren zu können und eine große Realitätsnähe zu erzielen, im Widerspruch zueinander. In dieser Arbeit wurde der Genauigkeit der Vorzug gegeben und die daraus resultierende Einschränkung der Netzgröße in Kauf genommen. Mit entsprechender Hardware wird es in Zukunft möglich sein, auch sehr große Netze mit geringer Abstraktion zu simulieren.

3.1 SSFNET

Der Netzwerksimulator SSFNET (Scalable Simulation Framework Network Models)[4] ist ein in Java geschriebener Simulator, der speziell für große Netzwerke ausgelegt ist. Der Programmkern ist proprietär und wurde von der DARPA, der Renesys Corporation sowie der Universität Dartmouth entwickelt, während viele Implementationen von Routingprotokollen unter einer freien Lizenz stehen. Um auch sehr große Netzwerke (mit mehr als 1000 Routern) simulieren zu können,

[3] genauer: OpenSSH – http://www.openssh.com
[4] http://www.ssfnet.org

muss entsprechend stark abstrahiert werden. Daher wird nur das Layer 3 (IP) des OSI-Modells und höher simuliert.

Die Beschreibung der Topologie und der Simulation wird in einer Beschreibungsdatei (DML-Datei) hinterlegt. Diese verfügt über die Möglichkeit, sich eigene Komponenten wie Router oder gar komplexe Teilnetze zu definieren und diese dann beliebig oft zu instanziieren. Dadurch können auch sehr große Simulationen schnell erstellt und übersichtlich dargestellt werden.

Zu jedem Netz kann man eine Menge Parameter definieren, wie zum Beispiel die Bitrate und Latenzzeit, wodurch reale Netze gut simuliert werden können.

Damit die Quellen unter neueren Java-Versionen (Versionen höher als Java 1.4) funktionieren, müssen die Verwendungen des Bezeichners „enum“ geändert werden, da dieser bei neueren Java-Versionen ein reserviertes Schlüsselwort ist. Dies kann z.B. mit dem Linux-Befehl aus Quelltext 2 erfolgen, der jedes Auftreten von „enum“ durch „enumer“ ersetzt.

```
find -name "*.java" -exec sed -i 's/enum/enumer/g' "{}" \;
```

Quelltext 2: Umbennenung des Bezeichners enum nach enumer für SSFNET

Danach kann man beginnen, Simulationsdateien zu schreiben. Etwas ungewöhnlich ist, dass zwei Netze unter SSFNET in der Regel durch zwei Router verbunden werden statt nur durch einen.

Die Protokolle IP, ICMP, TCP, UDP, BGPv4, OSPFv2, HTTP und FTP sind bereits vorhanden, jedoch gibt es keine Implementierung des RIP-Protokolls für SSFNET. Somit müsste erst eine RIP-Implementierung erstellt werden, um dieses Simulationsprogramm für diese Arbeit nutzen zu können.

Quelle: [Sch04]

3.2 NS2

Der Network Simulator 2 (NS2)[5] ist ein freier, in C++ geschriebener Netzwerksimulator, der alle Netzwerkschichten bis hin zur Bitübertragungsschicht simuliert. Auch Satellitenübertragungen und WLAN werden unterstützt. Der Simulator ist ereignisgesteuert, was bedeutet, dass jedes Paket und jede Aktion die Simulation beeinflussen. Dadurch wird nur Rechenleistung zur Verarbeitung der Ereignisse benötigt und so können recht große Netzwerke simuliert werden.

Es gibt einige Implementierungen von Routingprotokollen für ns2. Ein Distanzvektorprotokoll ähnlich zu RIP ist bereits implementiert. Der Update Timer sowie die Metrik für „unerreichbar’ sind zwar anders als bei RIP, können jedoch geändert werden [ns2b].

[5] http://www.isi.edu/nsnam/ns/

Die Installation ist recht umständlich. Es gibt zwar ein „all-in-one“-Paket, doch es müssen trotzdem einige Abhängigkeiten manuell installiert werden. Unter Debian-basierten Linux-Distributionen wie Ubuntu kann dies mit dem Befehl aus Quelltext 3 erreicht werden.

```
sudo apt-get install build-essential automake autoconf libxmu-dev
```

Quelltext 3: Installation der Abhängigkeiten für ns2

Damit auch das Animationsprogramm NAM (Network AniMation) funktioniert, ist ein Patch nötig.

Es kann zwar ausgegeben werden, welche Pakete von welchem Rechner zu welchem anderen Rechner gesendet werden, jedoch ohne den Inhalt der Pakete. Somit lässt sich keine Aussage darüber treffen, welche Routen nun mit welchen Metriken verbreitet werden. Ein Versuch, den Quelltext mit vertretbarem Aufwand so abzuändern, dass die benötigten Informationen beim Empfang eines Paketes ausgegeben werden, scheiterte. Dazu ist eine Einarbeitung in die gesamte Programmstruktur notwendig.

Somit ist NS2 für diese Arbeit zwar nutzbar, jedoch nur mit relativ hohem Aufwand.

Quellen: [Kni05], [ns2a]

3.3 Cisco Packet Tracer

Es gibt eine proprietäre Simulationssoftware von Cisco namens „Cisco Packet Tracer“[6], mit der sich jedoch nur die Router von Cisco simulieren lassen. Diese ist ausschließlich über eine graphische Oberfläche zu bedienen. Auch die Topologien müssen mit dieser Oberfläche erstellt werden. Dadurch ist diese Software umständlich zu handhaben und schwer zu automatisieren. Auch der Zugriff auf die Routinginformationen, die zur Bestimmung der Konvergenzzeit benötigt werden ist umständlich und nicht automatisierbar. Die Beschreibungen der Simulationen bzw. Topologien werden in einer Binärdatei gespeichert und sind somit nur schwer auslesbar.

Daher scheidet Cisco Packet Tracer als Simulationsprogramm für diese Arbeit aus.

3.4 VNUML

Virtual Network User-Mode-Linux (VNUML)[7] ist ein freies Simulationsprogramm auf Basis von virtuellen Maschinen. Die virtuellen Maschinen sind User-Mode-Linux-Instanzen (UML)[8]. UML ist eine Portierung des Linux-Kernels auf sich selbst. Das Netzwerk wird durch eine XML-Datei beschrieben. Für jeden Router wird ein virtueller Rechner gestartet, auf dem die Quagga Routing

[6] http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html
[7] http://www.dit.upm.es/~vnuml
[8] http://user-mode-linux.sourceforge.net

Suite[9] mit ihren Protokollimplementierungen läuft. Die Vernetzung der virtuellen Maschinen wird über TUN/TAP realisiert. TUN/TAP sind Kernel-Treiber für virtuelle Netzwerkgeräte, mit denen Programme wie mit einem Netzwerkinterface kommunizieren können. Dabei werden mit TUN Layer 3 Geräte simuliert, während TAP auf Layer 2 arbeitet [Kra]. Diese werden über virtual bridges [vir] an das Hostsystem gekoppelt. Die so erzeugten Devices können mit dem Programm tcpdump überwacht werden, welches alle Pakete mitschneiden und in eine Datei speichern kann. Diese tcpdump-Dateien können hinterher von verschiedenen Programmen ausgewertet werden.

Durch die Verwendung von virtuellen Maschinen wird eine extrem niedrige Abstraktion und somit ein sehr realistisches Verhalten erzeugt. Man kann jedoch nur relativ kleine Netzwerke simulieren, weil für jede virtuelle Maschine eine UML-Instanz gestartet und eine Kopie der Änderungen an den Dateisystemen im Arbeitsspeicher gehalten werden muss. Die Dateisysteme nutzen die Copy-on-write-Technik (COW), was bedeutet, dass nur für die Änderungen zusätzlicher Speicherplatz gebraucht wird. Für jedes Netz muss zusätzlich ein tcpdump-Prozess gestartet werden. Die Automatisierbarkeit ist aufgrund der Textbasiertheit und durch die Möglichkeit, Befehle für verschiedene Router vorzudefinieren, auch ohne eigene Scriptsprache gegeben.

Die Entscheidung fiel auf VNUML, da es die meisten der Anforderungen sehr gut erfüllt. Auch wenn nur kleinere Netzwerke simuliert werden können, so werden diese dafür sehr realistisch simuliert. Die Automatisierbarkeit sowie die Möglichkeit, alle gesendeten Pakete mit allen zugehörigen Informationen zur Verfügung zu haben, birgt einen großen Vorteil. Durch die Simulation mit einem modernen Mehrkernprozessor mit vier Kernen und acht GiB Ram wird der Overhead durch die Virtualisierung weitestgehend aufgefangen. Da die einzelnen Router also UML Instanzen eigene Prozesse sind, können diese sehr gut auf die Kerne verteilt werden.

Durch die Verwendung von virtuellen Maschinen kann man eigentlich nicht mehr von einer Simulation im engeren Sinne sprechen. Dies kann damit begründet werden, dass eine Simulation immer eine Abstraktion darstellt, die bei virtuellen Maschinen sehr gering ausfällt. Daher wird der Begriff „Simulation“ im Zusammenhang mit VNUML im weiteren Sinne als eine unter Laborbedingungen durchgeführte Reihe von Messungen mit gleichen Parametern und bekannter Umgebung verstanden.

Quellen: [Boh08], [GFR+04]

[9] http://quagga.net

4 Testumgebung

Da für die Untersuchung von Konvergenzeigenschaften mehrere verschiedenartige Tests mit verschiedenen Topologien durchgeführt werden müssen, wurde ein Programm benötigt, das in der Lage ist, diese Messungen zu automatisieren.

Zur Automatisierung der Simulationen wurde ein Perl-Programm geschrieben, welches VNUML mit einem gegebenen Szenario startet und den Durchlauf sowie die Messung der Konvergenzeigenschaften durchführt. Dabei kann der Ablauf eines Durchlaufs über ein Test-Script sehr genau beschrieben werden. Die Messungen werden durch die Analyse von automatisch generierten tcpdump-Dateien nach dem eigentlichen Durchlauf durchgeführt. Sie können auch unabhängig vom Durchlauf mit bereits vorhandenen tcpdump-Dateien durchgeführt werden. Mittels Regressionsanalyse kann mit Hilfe der Programme gnuplot[10] und maxima[11] eine Funktion approximiert werden. Daneben bietet das Programm noch Optionen zum schnellen Erzeugen von VNUML-Konfigurationsdateien mittels einer eigenen Konfigurationsdatei (der Topologiebeschreibung) oder per Zufall nur über die Angabe der Anzahl der Netze und Router. Darüber hinaus können Topologien visualisiert und einige ihrer Eigenschaften berechnet werden. Die Ergebnisdateien des Messungen können durch gnuplot graphisch aufbereitet und als Diagramm dargestellt werden.

Das Perl-Programm ist teilweise etwas umfangreicher als für diese Arbeit nötig. Dies soll die Wiederverwendbarkeit sicherstellen. Des Weiteren wurde dadurch die Möglichkeit offen gehalten, weitere Simulationen durchzuführen, falls sich dies als nötig erwiesen hätte.

4.1 Messmethode

Um eine möglichst hohe Flexibilität und Automatisierbarkeit zu erreichen, wurden die Topologiebeschreibung und das Test-Script getrennt. Dadurch kann man mit einer Topologie verschiedene Testfälle durchlaufen oder Messungen eines Testfalls mit vielen verschiedenen Topologien durchführen. Welcher Testfall wie oft mit welcher Topologie durchgeführt werden soll, wird in der Ausführungsbeschreibung definiert.

Um die Simulationen so wenig wie möglich zu beeinflussen, wurde jeder Durchlauf in eine Online-Phase und eine Offline-Phase aufgeteilt (siehe Abbildung 2). Die Online-Phase entspricht dabei der eigentlichen Simulation, während dieser der gesamte Updatetraffic aufgezeichnet wird. Die Auswertung wird nach jedem Simulationslauf in der Offline-Phase ausgeführt. Durch diese Trennung soll eine noch höhere Genauigkeit der Messergebnisse erreicht werden.

Der Ablauf eines jeden Durchlaufs beginnt mit dem Starten des VNUML-Netzwerks und des Quagga Zebra Daemons. Danach wird für jedes Netz ein tcpdump-Prozess gestartet, der den gesamten Verkehr des Netzes aufzeichnet und in eine tcpdump-Datei schreibt. Nun werden im Allgemeinen die RIP-Daemons gestartet und eine Zeit lang gewartet, bis diese konvergent sind. Der weitere Verlauf hängt vom Test-Script ab. Es können Router oder Devices heruntergefahren oder Zeitpunkte gespeichert werden. Am Ende werden die tcpdump-Prozesse gestoppt und

[10] http://www.gnuplot.info
[11] http://maxima.sourceforge.net

das VNUML-Netzwerk heruntergefahren. Nach dieser Online-Phase folgt die Offline-Phase, in der der gespeicherte Traffic ausgewertet wird und die Konvergenzzeit sowie weitere Konvergenzeigenschaften berechnet werden. Dafür wird auch die Topologie auf ihre Eigenschaften hin analysiert. Die Ergebnisse werden in einer Ergebnisdatei gespeichert, bevor der nächste Durchlauf gestartet wird. Sind alle Durchläufe abgeschlossen, können die Ergebnisse noch gemittelt werden.

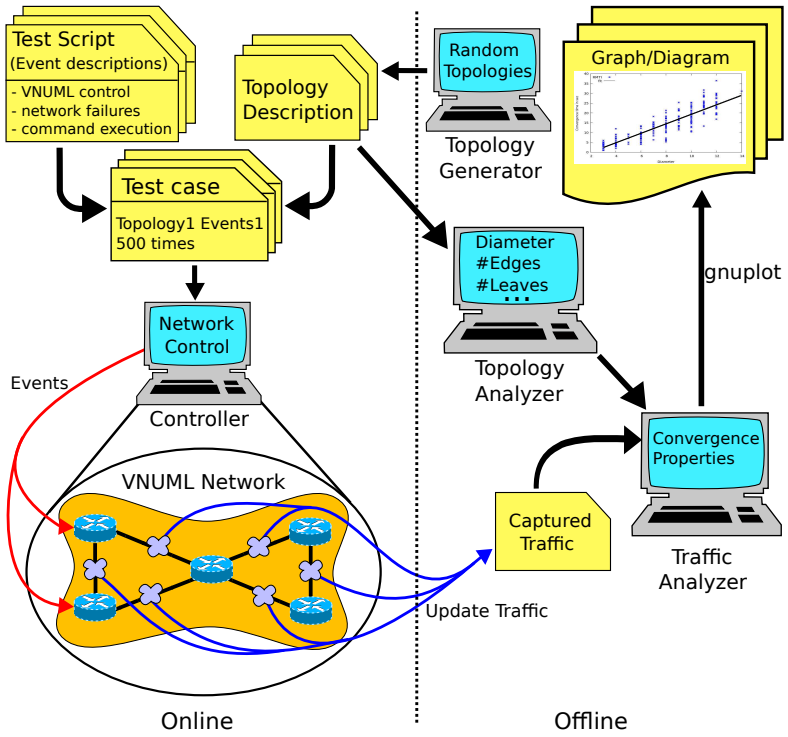

Abbildung 2: Schematische Darstellung der Testumgebung

4.2 Topologiebeschreibung und Analyse

XML ist ein für den Menschen schwer lesbares Format, bei dessen manueller Bearbeitung schnell Fehler entstehen können. Bei den VNUML-Konfigurationsdateien müssen darüber hinaus auch viele Teile mehrmals geschrieben werden, wodurch leicht „Copy and 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 Klartext nur die benötigten Informationen enthält. Diese Topologiebeschreibung kann durch das Programm in eine VNUML-Konfigurationsdatei umgewandelt werden. Dabei wird neben einem Syntaxcheck auch ein Plausibilitätscheck durchgeführt, der doppelt definierte sowie ungenutzte Router und Netze erkennt und entsprechende Warnungen ausgibt.

Topologiebeschreibungen können entweder manuell in einer einfachen Syntax beschrieben oder automatisch durch den Topologiegenerator des Programms generiert werden. Der Topologiegenerator kann automatisch zufällige Topologien nach bestimmten Kriterien wie der Anzahl der Knoten und Kanten oder symmetrische Topologien bestimmter Topologieklassen generieren. Um manuell erstellte Topologiebeschreibungen zu überprüfen oder zufällig generierte Graphen zu betrachten, kann das Programm mit Hilfe des Tools GraphViz[12] jede Topologie auch graphisch darstellen. Dies funktioniert jedoch nicht, wenn die Topologie Hyperkanten enthält.

Im Programm werden Topologien als Graphen durch das Perl-Modul `Graph`[13] repräsentiert. Die Datenstruktur ist dabei kantenbasiert. Aus diesen Graphen lassen sich wiederum Topologiebeschreibungen oder VNUML-Konfigurationsdateien generieren.

Um die Zusammenhänge zwischen Topologieeigenschaften und Konvergenzeigenschaften von RIP zu untersuchen, werden alle getesteten Topologien auch graphentheoretisch analysiert. Im Folgenden wird die Berechnung der verschiedenen graphentheoretischen Eigenschaften erläutert.

Hinweis: Durch die Tatsache, dass ein Netz mit mehr oder auch weniger als zwei Routern verbunden sein kann, müssen solche Netze im Graph als Hyperkanten modelliert werden, wodurch die Verwendung von Hypergraphen erzwungen wird. Dadurch funktionieren einige der Algorithmen des `Graph`-Moduls jedoch nicht mehr und es müssen teilweise andere (allgemeinere) Formeln verwendet werden. Daher wurden einige Algorithmen angepasst, sodass sie auch mit Hyperkanten arbeiten können. Die Anpassung wird jeweils bei der Beschreibung des Algorithmus bzw. der Formel angegeben. Trotzdem wurde versucht, Hypergraphen in dieser Arbeit zu vermeiden um Problemen vorzubeugen.

Berechnung des Durchmessers

Der Durchmesser entspricht der maximalen Metrik innerhalb eines konvergierten Netzwerks.

Es wird der Dijkstra-Algorithmus des Perl-Moduls `Graph` verwendet, um von jedem Knoten zu jedem anderen Knoten den kürzesten Pfad zu berechnen. Die Kantengewichte werden dabei alle auf 1 gesetzt, da in dieser Arbeit alle Netze die gleiche Bandbreite und Latenz besitzen. Der längste der errechneten kürzesten Pfade ist dann der Durchmesser. Da Dijkstra mit Hyperkanten umgehen kann, war hier keine Anpassung notwendig. (Die Komplettierung jedes Teilgraphen, bestehend aus der jeweiligen Hyperkante und allen damit verbundenen Knoten, würde aber auch funktionieren.)

Berechnung des Clusterkoeffizienten

Der Clusterkoeffizient gibt den Grad der Vernetzung eines Graphen an. Ein kompletter Graph hat einen Clusterkoeffizient von 1, während alle anderen Graphen einen $cc < 1$ besitzen.

[12] http://www.graphviz.org
[13] http://search.cpan.org/~jhi/Graph-0.91/lib/Graph.pod

Der Clusterkoeffizient ist definiert durch Formel (12) (siehe Seite 14). Der Clusterkoeffizient wird mit dieser Formel auch bei Hypergraphen berechnet, da das Programm keine Unterscheidung macht.

Berechnung der Artikulationen

Die Berechnung der Artikulationen ist praktisch, um bei der Deaktivierung eines zufälligen Routers das Netz nicht in zwei Teilnetze zerfallen zu lassen. Daher werden die Artikulationen errechnet, damit man sie vom zufälligen Deaktivieren ausschließen kann.

Für die Berechnung von Artikulationen wird eine bereits implementierte Funktion eines Perl-Moduls `Graph` genutzt (siehe Anhang B auf Seite 86).

Bei Hypergraphen wird vor der Berechnung von Artikulationen für jede Hyperkante der Teilgraph aus allen mit ihr verbundenen Knoten komplettiert. Dann funktioniert der Algorithmus wie gewünscht. Die Ersetzung von Hyperkanten mit kompletten Teilgraphen der mit der Hyperkante verbundenen Knoten hat dabei keinen Einfluss darauf, ob ein Knoten eine Artikulation ist oder nicht.

Begründung:
Ist ein Knoten v_a über eine Hyperkante mit zwei (oder mehr) weiteren Knoten v_b und v_c verbunden, so existiert diese Kante nach der Entfernung des Knotens v_a weiter und die beiden Knoten v_b und v_c sind immer noch verbunden. Wird aus allen mit der Hyperkante verbundenen Knoten v_a, v_b und v_c ein kompletter Teilgraph gebildet, so sind v_b und v_c bei der Entfernung des Knotens v_a ebenfalls weiterhin verbunden. Somit hat eine Ersetzung von Hyperkanten durch komplette Teilgraphen keine Auswirkungen darauf, ob ein Knoten eine Artikulation ist.

Berechnung der zyklomatischen Zahl

Die zyklomatische Zahl gibt die Anzahl der Kreise in einem Graphen an. Dies ist äquivalent zur Anzahl der redundanten Netze.

Die Berechnung der zyklomatischen Zahl k erfolgt nach Formel (4). Für Hypergraphen wird Formel (5) verwendet.

Berechnung der Blätter

Da die meisten Routingprotokolle netzbasiert arbeiten, würde ein Ausfall eines Routers, der nur an ein Netz angeschlossen ist, keine Folgen haben. Das Netzwerk bleibt in einem solchen Fall konvergent. Daher sollte zum Beispiel bei einem zufälligen Routerausfall kein Blatt ausfallen. Die Blätter könnten auch bei den nachfolgenden Untersuchungen eine Rolle spielen. Aus diesen Gründen wurde eine Methode implementiert, die alle Blätter eines Graphen bestimmt.

Funktionsweise:

Zuerst wird ein Hash angelegt, welches für jeden Knoten die Anzahl der mit ihm verbundenen Kanten angibt. Diese ist zu Anfang für jeden Knoten Null. Nun wird in einer Schleife über alle Kanten iteriert wobei von jeder Kante alle Knoten extrahiert werden, mit denen sie verbunden ist. Für jeden extrahierten Knoten wird nun dessen Wert im Hash inkrementiert. Danach ist in dem Hash für jeden Knoten die Anzahl der mit ihm verbundenen Kanten gespeichert.

Abschließend wird jeder Knoten aus dem Hash in eine Liste gespeichert, der mit genau einer Kante verbunden ist. Die daraus resultierende Liste ist die Liste der Blätter des Graphen.

Die entsprechende Funktion ist in Quelltext 4 dargestellt.

```
sub _getLeaves {
    my $object = shift;
    my $g = $object->{GRAPH};
    my @edges = $g->edges(); # erstelle Array mit Kanten
    my %routers; # erstelle Hash
    foreach my $edge (@edges) { # iteriere ueber Kanten
        my @rs = @{$edge}; # erstelle Array mit Routern der Kante
        foreach my $r (@rs) { # iteriere ueber Router
            $routers{$r}++; # inkrementiere Hash-Werte
        }
    }
    my @result = (); # erstelle Erbegnisliste
    foreach my $r (keys %routers) { # iteriere ueber Hash
        # wenn Kantenzahl=1, fuege Router zur Ergebnisliste hinzu:
        push(@result,$r) if $routers{$r} == 1;
    }

    return @result;
}
```

Quelltext 4: Code für die Berechnung der Blätter eines Graphen

Die Methode kann auch mit Hyperkanten umgehen.

4.3 Test-Script

Das Test-Script besteht aus einer sequentiellen Folge von Anweisungen, die den Verlauf der Simulation steuern. Dabei können alle erdenklichen Ereignisse wie zum Beispiel Router- oder Deviceausfälle ausgelöst werden. Es stehen eine Reihe von Befehlen zur Verfügung, die im Test-Script genutzt werden können. Mit einer `sleep()`-Anweisung können beliebig lange Pausen zwischen den Befehlen eingebaut werden. Damit kann man zum Beispiel warten, bis ein Netzwerk den konvergenten Zustand erreicht hat, bevor man einen Router oder ein Device ausfallen lässt. Dies ist wichtig um die Akkuratheit der Ergebnisse sicherzustellen. Eine wesentliche Erweiterung des Funktionsumfangs wird durch die `execute()`-Funktion eröffnet, mit der jegliches installierte Linuxkommando oder -programm auf beliebigen Routern zu (mittels `sleep()`-Befehl) festgelegten

Zeitpunkten ausgeführt werden kann. So können mit dem Tool tc[14] (traffic control) Paketverluste oder Übertragungsverzögerungen verursacht werden. Ebenso kann jedes UNIX-Tool zur Generierung von Payload-Traffic genutzt werden, um die Performance von Routingprotokollen unter Last zu untersuchen. Die Ausgaben der Programme können mit einem Zeitstempel versehen in eine Datei gespeichert werden.

Eine Übersicht über alle Befehle sowie Beispiele für Test-Scripte befinden sich in der Bedienungsanleitung auf Seite 69.

4.4 Konvergenzanalyse

Zur Bestimmung der Konvergenzeigenschaften wird der mittels tcpdump mitgeschnittene Updatetraffic aller Netze analysiert. Der Startzeitpunkt t_s kann dabei je nach gewünschter Messung auf unterschiedliche Weise festgelegt werden:

- nach dem Zeitstempel des ersten Pakets
- durch Speichern der Uhrzeit zu einem im Test-Script bestimmten Zeitpunkt während der Simulation
- nach einem beliebig wählbaren Zeitstempel

Dadurch kann man neben Coldstart-Tests die Konvergenzzeit nach einem Router- oder Deviceausfall bestimmen.

Der Endzeitpunkt t_e der Konvergenzmessung wird berechnet, indem das Programm für jeden Router eine Forwardingtabelle verwaltet und dann jedes gesendete Updatepaket analysiert. Dabei wird jede Berechnung, die ein Router durchgeführt hat, wiederholt und die Forwardingtabelle für den jeweiligen Router angepasst. Der Endzeitpunkt t_e, an dem das Netzwerk zum letzten Mal in einen konvergenten Zustand übergeht, entspricht dem Zeitstempel des Pakets, welches zum letzten Mal eine der Forwardingtabellen ändert. Für diese Berechnung wird auch die Topologiebeschreibung benötigt, um zuordnen zu können, welche Pakete an welche Router gesendet werden. Um die Konvergenzzeit bis zu einem früheren als den letzten konvergenten Zustand zu ermitteln, lässt sich ein Zeitstempel angeben. Alle Pakete, die älter sind als dieser Zeitstempel, werden dann für diese Berechnung ignoriert. Die Konvergenzzeit t entspricht nun der folgenden Gleichung:

$$t = t_e - t_s \tag{13}$$

t_k = Konvergenzzeit
t_s = Startzeitpunkt
t_e = Endzeitpunkt

Das gesamte Volumen des Updatetraffics während der Konvergenzzeit kann nun bestimmt werden, indem die Größe aller Pakete, die während des Konvergenzzeitraums versendet wurden addiert werden. Darüber hinaus werden noch weitere Konvergenzeigenschaften berechnet wie

[14] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

durchschnittlicher Traffic pro Netz, Paketanzahl, Paketanzahl pro Netz und einige mehr. Eine komplette Auflistung der berechneten Konvergenzeigenschaften befindet sich in der Bedienungsanleitung auf Seite 75.

4.5 Aufbereitung der Ergebnisse

Um grundsätzliche Aussagen über die Konvergenzeigenschaften in Abhängigkeit zur Topologie treffen zu können, benötigt man große Mengen an Messergebnissen. Der hohe Grad der Automatisierung des Programms erlaubt eine einfache Generierung der benötigten Menge an Ergebnissen. Diese können statistisch ausgewertet werden. So kann man in einem ersten Auswertungsschritt die Durchschnitte der Konvergenzeigenschaften zu einer Simulation berechnen. Dabei wird der jeweils höchste und niedrigste Wert verworfen, um mögliche Ausreißer zu reduzieren und einen besseren Mittelwert zu erhalten. Des Weiteren ist es möglich, mittels Regressionsanalyse approximierende Funktionen zu berechnen sowie Häufigkeitsverteilungen zu erstellen. Daraus wiederum kann man grundsätzliche Zusammenhänge zwischen Topologie- und Konvergenzeigenschaften ableiten. Für die Regressionsanalyse wurden die Programme gnuplot und maxima genutzt. Die Syntax der Ergebnisdateien wurde auf diese Verwendung hin optimiert, sodass die Nutzung dieser Programme erleichtert wird. Mittels der Methode der kleinsten Quadrate [Hä67] können aus den Ergebnissen approximierende Funktionen berechnet werden, die bei genügend Messwerten genaue Aussagen über Zusammenhänge von Topologie- und Konvergenzeigenschaften ausdrücken. Mittels gnuplot ist es zusätzlich möglich, die Messergebnisse sowie die approximierenden Funktionen und Häufigkeitsverteilungen als Graphen darzustellen. Die Ergebnisdateien sind so aufgebaut, dass man sehr einfach verschiedene Topologie- und Konvergenzeigenschaften gegeneinander auftragen kann. Dadurch kann man sich sehr schnell einen ersten Eindruck über die Ergebnisse verschaffen. Dies ermöglicht ein dynamisches Arbeiten, da man mögliche Zusammenhänge schnell erkennen und entsprechende Tests starten kann, um diese Zusammenhänge zu stützen oder zu falsifizieren.

5 Annahmen und Erwartungswerte

Um einen Zusammenhang zwischen der Topologie eines Netzwerks und dessen Konvergenzeigenschaften herzustellen, wurden einige Topologieeigenschaften näher betrachtet. Im Folgenden werden Vermutungen und Annahmen diskutiert und beschrieben, zu welchen Erwartungswerten sie führten.

5.1 Konvergenzzeiten und Trafficvolumen

Die wichtigste Konvergenzeigenschaft ist die Konvergenzzeit, da während dieser Zeit Pakete nicht oder nicht optimal zugestellt werden können und das Netzwerk somit nicht voll funktionstüchtig ist. Daher ist es für jedes Routingprotokoll sehr wichtig, diese Zeit möglichst gering zu halten. Eine weitere wichtige Konvergenzeigenschaft ist der Traffic, da die meisten Abrechnungsmodelle darauf beruhen. Dabei ist egal, ob nun viele kleine oder wenige große Pakete verschickt wurden. Der Gesamttraffic hängt mit Sicherheit von der Größe des Netzes ab, da in größeren Netzen mehr Routingpakete versendet werden müssen. Daher ist der durchschnittliche Traffic pro Netz möglicherweise interessanter für den Vergleich mit anderen Routingalgorithmen. Gesamttraffic und Traffic pro Netz hängen über die Anzahl der Netze zusammen. Somit kann man für eine Topologie immer vom Gesamttraffic den durchschnittlichen Traffic pro Netz berechnen, indem man ihn durch die Anzahl der Netze teilt.

5.2 Durchmesser

Da der Durchmesser die maximale Distanz (in der Topologie) ist, die die Routingpakete zurücklegen müssen, müsste er eine wichtige Rolle für die Konvergenzzeit des Netzwerks spielen. Um diese Annahme zu stützen, sollten verschiedene Topologien getestet werden, die bei steigendem Durchmesser nur geringfügige Änderungen der Kanten- und Knotenzahlen aufweisen. Diese Ergebnisse sollten mit Messungen an Topologien verglichen werden, bei denen eine wesentliche Änderung der Kanten- und Knotenzahl mit einem möglichst konstantem Durchmesser einhergeht. Wenn man eine Reihe von Routern erstellt, so sollte die Konvergenzzeit mit der Anzahl der Router in der Reihe und damit zum Durchmesser des Graphen wachsen. In einem kompletten Graph müsste die Konvergenzzeit unabhängig von der Anzahl der Router und Netze fast konstant sein. Ebenso in einem Stern, in dem n-1 Router mit einem in der Mitte befindlichen Router verbunden sind.

5.3 Die Anzahl der Knoten

Die Anzahl der Knoten eines Graphen entspricht der Anzahl der Router. Da jeder Router (außer Blätter, dazu mehr in Kapitel 5.4) auch Routingpakete versendet, müsste zumindest die Anzahl der Pakete und damit der Traffic mit steigender Routeranzahl zunehmen. Denkbar ist auch, dass die Konvergenzzeit mit steigender Anzahl der Router unabhängig vom Durchmesser wächst, da jeder Router eine gewisse Zeit zur Berechnung braucht. Um diese Annahmen zu überprüfen,

müssten Topologien getestet werden, die bei gleichem Durchmesser und gleicher Kantenzahl unterschiedlich viele Knoten besitzen.

5.4 Blätter und innere Knoten

Blätter spielen voraussichtlich eine gesonderte Rolle bei der Messung von Konvergenzeigenschaften. Router, die an genau einem Netz angeschlossen sind, können keine Netze propagieren. Somit senden solche Router weder getriggerte noch periodische Updatepakete. Andere Router senden jedoch ihre Updatepakete über alle Interfaces, weshalb Stub-Netze[15] und Blätter (bzw. die Netze an denen Blätter angeschlossen sind) für die Konvergenzeigenschaften gleichwertig sein müssten. Die Router, die den Blättern entsprechen, werden in dieser Arbeit genutzt, um solche Stub-Netze zu analysieren und dabei Hypergraphen[16] möglichst zu vermeiden, da diese sehr schlecht von der verwendeten Graphenbibliothek (siehe Kapitel 4.2) unterstützt werden. Ein weiterer Grund für die Betrachtung von Blättern und inneren Knoten ist, dass die verwendete Funktion zur Erzeugung zufälliger Graphen keine Hypergraphen erzeugt, dafür jedoch Graphen mit Blättern.

Die geringere Anzahl an Paketen bei Topologien mit vielen Blättern müsste ein geringeres Trafficvolumen zur Folge haben, als wenn alle Knoten innere Knoten wären. Möglicherweise haben Topologien mit vielen Blätter auch eine geringere Konvergenzzeit. Daher sind wahrscheinlich die inneren Knoten maßgeblicher an Traffic und Konvergenzzeit beteiligt. Blätter müssten somit geringer gewichtet oder sogar vernachlässigt werden können.

Wie bei den Knoten könnte man diese These testen, indem man Zufallstopologien erstellt, die bei konstantem Durchmesser und konstanter Kantenzahl unterschiedlich viele Blätter und Knoten haben wobei auch die Gesamtanzahl der Knoten konstant bleiben sollte.

5.5 Kanten

Die Anzahl der Kanten müsste zumindest eine Auswirkung auf den Traffic haben. Da jede Kante einem Netz entspricht, welches an jeden Router weiter gesendet wird, sind viele Pakete auch um einen Eintrag pro Netz länger. Über Netze, die an zwei oder mehr Router angeschlossen sind, werden in der Regel auch Updatepakete versendet. Diese erhöhen den Traffic zusätzlich.

Somit ist es wahrscheinlich, dass ein Zuwachs von Kanten zwar die Zahl der Pakete und den Traffic erhöht, die Konvergenzzeit jedoch eher geringfügig beeinflusst.

5.6 Clusterkoeffizient

Da die Anzahl der Netze pro Router mit steigendem Clusterkoeffizient zunimmt, könnte man davon ausgehen, dass auch die Zahl der Pakete und somit der Traffic zunimmt. Dies würde auch

[15] Stub-Netze sind Netze, die nur an einen Router angeschlossen sind.
[16] Vor allem mit Hyperkanten, die mit nur einem Knoten verbunden sind.

eine erhöhte Konvergenzzeit begünstigen. Jedoch sinkt mit zunehmendem Clusterkoeffizienten auch der Durchmesser, was die Konvergenzzeit senken müsste.

Auf der einen Seite steigt mit dem Clusterkoeffizient die Anzahl der Netze pro Router, was darauf schließen lässt, dass auch die Anzahl der Pakete steigt. Dies könnte eine höhere Konvergenzzeit verursachen. Jedoch kann man Netze generieren, die bei gleichem Clusterkoeffizient sehr viele oder sehr wenige Netze enthalten, wodurch der Clusterkoeffizient keine Aussage über den Traffic zuließe. Auf der anderen Seite sinkt der Durchmesser mit steigendem Clusterkoeffizienten. Der Durchmesser bei einem Clusterkoeffizient von 1 ist immer 1. Somit könnte die Konvergenzzeit bei höheren Clusterkoeffizient sinken. Dadurch würde der Clusterkoeffizienten jedoch nur indirekt Einfluss nehmen und man könnte Netze mit beliebigem Durchmesser und einem Clusterkoeffizienten kleiner 1 generieren, die entsprechend langsam konvergieren.

Aus diesem Grund wird der Einfluss des Clusterkoeffizienten nur statistisch in Kapitel 8.5 auf Seite 51 untersucht werden, da er voraussichtlich keine (direkte) Abhängigkeit zu einer Konvergenzeigenschaft hat.

5.7 Kreise

Da die Anzahl der Kreise jederzeit aus den vorhandenen Informationen berechnet werden kann, kann immer eine entsprechende Auswertung stattfinden. Da aufgrund von Formel (4) die Anzahl der Kanten immer größer ist als die Anzahl der Kreise, kann von der zyklomatischen Zahl direkt auf die minimale Anzahl der Kanten geschlossen werden. Daher könnten die Konvergenzeigenschaften für Kreise Ähnlichkeiten zu denen für Kanten aufweisen. Dies wäre jedoch irrelevant, da in diesem Fall immer noch die Kanten ausschlaggebend wären.

Daher wird die Auswirkungen der zyklomatischen Zahl auf die Konvergenzeigenschaften nur statistisch in Kapitel 8.6 auf Seite 52 untersucht.

6 Getestete Topologien

Für das Kapitel 7 wurden Topologien oder Topologieklassen verwendet, die voraussichtlich Aussagen über Zusammenhänge einer bestimmten Topologieeigenschaft zu Konvergenzeigenschaften zulassen. Diese Topologien werden im entsprechenden Abschnitt vorgestellt oder wurden bereits in den Annahmen (Kapitel 5) erwähnt.

In den Kapiteln 8 und 8.7 werden die Ergebnisse von über 300 Topologien mit insgesamt über 5000 Durchläufen statistisch untersucht. Diese sind zum größten Teil zufallsgeneriert. Jedoch sind dort auch einige symmetrische Topologien enthalten. Aufgrund des hohen Rechenaufwands wurden die zu testenden Topologien auf maximal 50 Router und 100 Kanten beschränkt.

Um aussagekräftige Ergebnisse zu erhalten, ist es wichtig, dass eventuelle Zusammenhänge zwischen einzelnen Topologieeigenschaften berücksichtigt werden. Diese sind einerseits durch graphentheoretische Gesetzmäßigkeiten bedingt, wie zum Beispiel Formel (2) und (6), welche die minimal und maximal mögliche Anzahl an Kanten für eine gegebene Anzahl von Knoten festlegen. Andererseits können solche Zusammenhänge durch die Wahl der getesteten Topologien entstehen. Wenn alle getesteten Topologien einen steigenden Durchmesser bei zunehmender Knotenzahl besitzen, so wird der Durchmesser ähnliche Konvergenzeigenschaften wie die Knotenzahl besitzen. Solche von der Topologieauswahl bedingten Zusammenhänge gilt es möglichst auszuschließen, was leider nicht immer möglich ist. Daher werden solche Zusammenhänge dort untersucht, bei denen es wichtig ist, um keine falschen Schlüsse zu ziehen.

6.1 Vergleichstopologien

Im Kapitel 7 werden Simulationen mit symmetrischen Topologien beschrieben, mit denen sich die Vermutungen aus Kapitel 5 vermutlich stützen oder falsifizieren lassen. Da solch symmetrische Topologien in der Realität selten vorkommen, wurden zum Vergleich dazu einige andere Topologien gegenübergestellt, die zum einen zum Testen von Routingalgorithmen genutzt werden. Zum anderen wurden die beiden realen Topologien des Internet 2 und des Arpanets von 1972 simuliert. Diese beiden realen Topologien wurden auch als Vergleichstopologien für die Kapitel 8 und 8.7 herangezogen.

Crown: Crown (siehe Abbildung 3) ist eine Topologieklasse, bei der ein innerer Kreis aus N Knoten von einem äußeren Kreis mit $2 * N$ Knoten umgeben ist. Dabei ist jeder Knoten des inneren Kreises mit jedem zweiten Knoten des äußeren Kreises verbunden. Sie wurde bereits für andere Konvergenzuntersuchungen genutzt [LVPS08].

Square: Square (siehe Abbildung 4) ist eine Topologieklasse bestehend aus $N * N$ Knoten, die in einem Quadrat angeordnet und horizontal sowie vertikal vernetzt sind. Diese Topologie wurde ebenfalls schon für andere Konvergenzuntersuchungen genutzt [LVPS08].

Internet 2: Internet 2 ist eine Netzwerkgemeinschaft in den USA, die ein Hochgeschwindigkeitsnetz für Forschungszwecke betreibt. Mittlerweile sind 330 Forschungseinrichtungen Mitglied in der Gemeinschaft und gibt es auch internationale Anbindungen. Die Community zählt über 60.000 Institutionen und Firmen in 50 verschiedenen Ländern [int09]. Das Backbone des Netz-

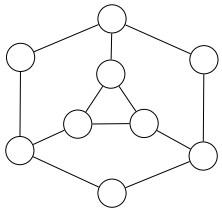

Abbildung 3: Die Crown3-Topologie

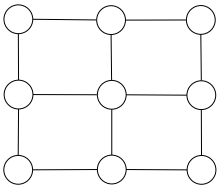

Abbildung 4: Die Square3-Topologie

werks in den USA besteht aus 12 Routern mit 16 Netzen, dessen Topologie in dieser Arbeit genutzt wird [GS07]. Im Folgenden wird mit Internet 2 dieses Netzwerk gemeint und nicht die Gemeinschaft. Die Topologie des Internet 2 Backbones (siehe Abbildung 5) wird hier als „Internet2“ bezeichnet.

Arpanet von 1972: Das Arpanet war das erste Netzwerk, welches von der ARPA (jetzt DARPA) zusammen mit einigen Universitäten in den 60er Jahren aufgebaut wurde. Aus dem Arpanet (siehe Abbildung 6) ist das heutige Internet entstanden. Die Topologie wird hier als „Arpa72“ bezeichnet [FKK72].

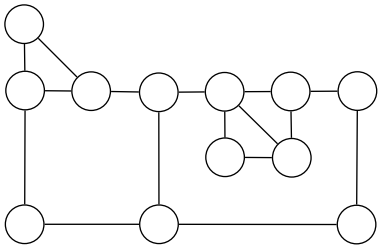

Abbildung 5: Topologie des Internet 2 am 29. April 2007

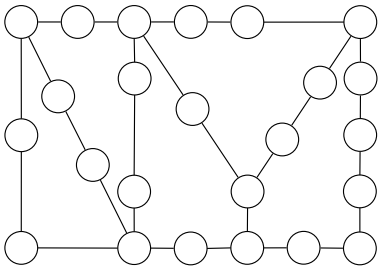

Abbildung 6: Topologie des Arpanet von 1972

Name	V	E	D	cc	t	T	T_a
Internet2	12	16	5	0.2424	12.9 s	19 KiB	1.2 KiB
Arpa72	24	28	8	0.1014	18.6 s	55.7 KiB	2 KiB
CrownN	$3 \cdot N$	$4 \cdot N$	-	$\frac{8 \cdot N}{3 \cdot N \cdot (3 \cdot N - 1)}$	-	-	-
SquareN	N^2	$2 \cdot N \cdot (N-1)$	$2 \cdot N - 2$	$\frac{4 \cdot N \cdot (N-1)}{N^2 \cdot (N^2-1)}$	-	-	-

Tabelle 3: Tabelle der Vergleichstopologien

Tabelle 3 gibt die getesteten Vergleichstopologien mit den wichtigsten Daten an. Alle Vergleichstopologien haben keine Blätter.

7 Messungen mit ausgewählten Topologien

Im Folgenden wird untersucht, welchen Einfluss die verschiedenen Topologieeigenschaften auf die Konvergenzeigenschaften haben. Dafür wurden sogenannte „Coldstart“-Tests durchgeführt, bei denen die Konvergenzzeit (und davon ausgehend die anderen Konvergenzeigenschaften, wie in Kapitel 4.4 auf Seite 28 beschrieben) vom Start der Routingdaemons bis zum ersten konvergenten Zustand gemessen wurde. Dies sollte von der Konvergenzzeit her dem Worst Case eines Netzausfalls entsprechen, da die Informationen über die gesamte Topologie verbreitet werden müssen. Gleichzeitig dürfte die Konvergenzzeit dieses Testfalls aber auch der Konvergenzzeit beim Hinzufügen eines neuen Netzes im Worst Case entsprechen, da die Information über das neue Netz im Worst Case auch den gesamten Durchmesser zurücklegen muss. Diese Zeiten sind natürlich abzüglich der Zeit, die von den Routern benötigt werden um die Topologieänderung zu erkennen. Dies führt zu genaueren Ergebnissen, da die Erkennungszeit vom Zufall abhängt und stark schwankt.

Für die Diagramme in diesem Kapitel wurden für jede Topologie 15 Durchläufe durchgeführt. Die Ergebnisse wurden jeweils gemittelt und pro Topologieklasse mit Linien verbunden. Somit stellt jede Linie den Durchschnitt aus 15 Durchläufen der jeweiligen Topologieklasse dar. Jeder Topologieklasse wurde eine Farbe zugeordnet, die über alle Diagramme in diesem Kapitel gleich bleibt. Für einige Untersuchungen wurden Zufallstopologien mit bestimmten Eigenschaften erstellt, die alle blau dargestellt sind (wie auch die Linien der Row-Topologie).

7.1 Durchmesser

In diesem Abschnitt wird der Einfluss des Durchmessers auf die Konvergenzeigenschaften untersucht. Dafür wurden einige Simulationen mit ausgewählten Topologien durchgeführt, wie sie im Kapitel 5.2 auf Seite 30 beschrieben wurden.

Konvergenzzeit

Die Annahme, dass der Durchmesser eine wichtige Rolle für die Konvergenzzeit spielt, soll nun durch Simulationen untersucht werden.

Zuerst wurden Netzwerke simuliert, in denen die Router eine Reihe bilden. Bei dieser „Row“ genannten Topologie sollte mit jedem weiteren Router die Konvergenzzeit ansteigen. Die Anzahl der Knoten und Kanten sollte bei dieser Topologie eine sehr untergeordnete Rolle spielen, da hier jeweils der maximal mögliche Durchmesser für die entsprechende Anzahl von Knoten erreicht wird. Da der Durchmesser der Topologie eines Netzwerks mit jedem weiteren Router wächst, sollte die Konvergenzzeit ebenfalls zunehmen.

Bei einer kreisförmigen Topologie („Circle“) würde man erwarten, dass die Konvergenzzeit aufgrund des halben Durchmessers bei gleicher Routeranzahl (und dadurch auch fast gleicher Netzanzahl, da Kreise genau ein Netz mehr haben als Reihen) auch etwa halb so groß ist.

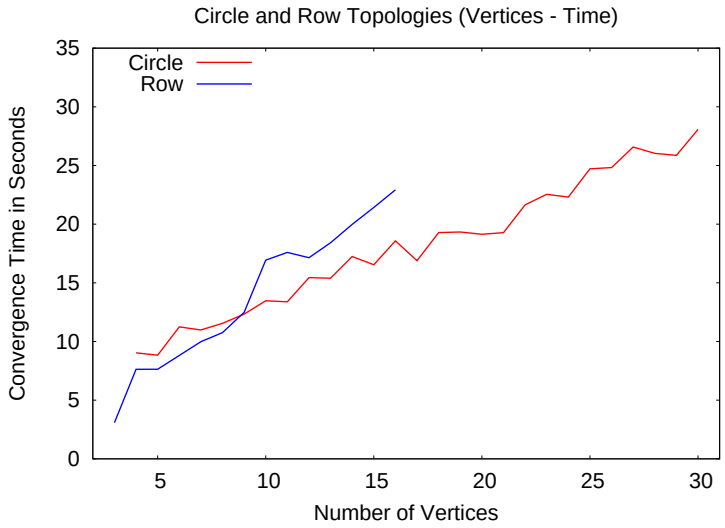

Abbildung 7: Konvergenzzeiten der Circle- und Row-Topologien mit zunehmender Knotenzahl

Die Ergebnisse (siehe Abbildung 7) zeigen, dass die Konvergenzzeit bei Row-Topologien nicht immer doppelt so hoch ist wie bei Circle-Topologien. Jedoch stimmen die Ergebnisse dahingehend mit den Annahmen überein, dass die Row-Topologien eine höhere Steigung bei zunehmender Knotenzahl haben als die Circle-Topologien.

Wie die Messergebnisse (siehe Abbildung 8) zeigen, wachsen auch die Konvergenzzeiten der Vergleichstopologien mit steigendem Durchmesser anscheinend linear an.

Des Weiteren wurden Simulationen mit konstantem Durchmesser überprüft, welche keine signifikanten Unterschiede in der Konvergenzzeit zueinander besitzen sollten, wenn der Durchmesser ausschlaggebend für die Konvergenzzeit ist. Zum einen wurden komplette Graphen mit $d = 1$ („Complete“) simuliert. Aufgrund der schnell wachsenden Kantenzahl (siehe Formel (6)) konnten jedoch nicht viele Complete-Topologien getestet werden. Zum anderen Sterntopologien, bei denen ein Router über je ein Netz mit allen anderen Routern verbunden wurde. Diese Topologien haben den Durchmesser 2 (daher werden sie im Folgenden „Star2“ genannt).

Die Simulationen mit konstantem Durchmesser hatten wie erwartet eine annähernd konstante Konvergenzzeiten, unabhängig von der Anzahl der Router oder Netze. Die Star2-Topologien sind mit einer durchschnittlichen Konvergenzzeit von ca. 9.2 Sekunden jedoch ungefähr ein Drittel langsamer als die Complete-Topologien mit ca. 6 Sekunden durchschnittlicher Konvergenzzeit.

Die konstante Konvergenzzeit hängt auch mit dem verwendeten Algorithmus zusammen. Bei einer Star2-Topologie sendet der mittlere Router an jeden anderen Router alle angeschlossenen Netze, während alle anderen Router nur empfangen. Dadurch wächst die Konvergenzzeit pro weiterem Router nicht merklich an (siehe auch Abbildung 10). Bei Complete-Topologien sieht es sehr ähnlich aus. Hier sendet jeder Router einmal alle angeschlossenen Netze an die anderen Router, die jedoch wiederum keine der Informationen weiter senden. Dadurch ist auch hier

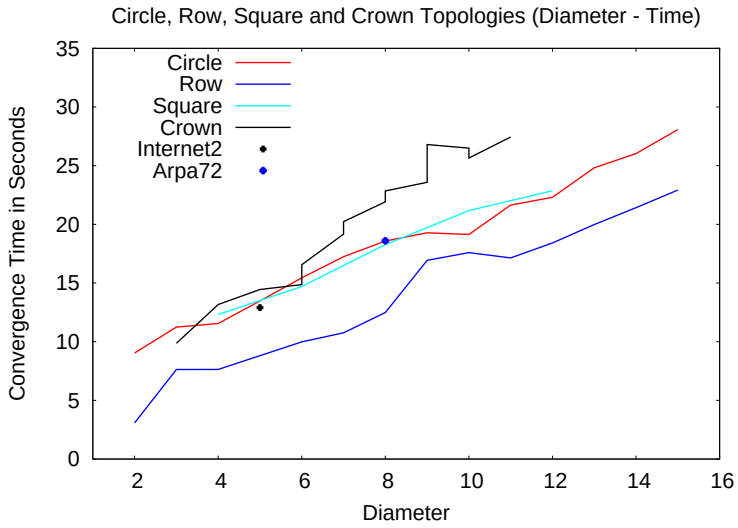

Abbildung 8: Konvergenzzeiten der verschiedener Topologien mit steigendem Durchmesser

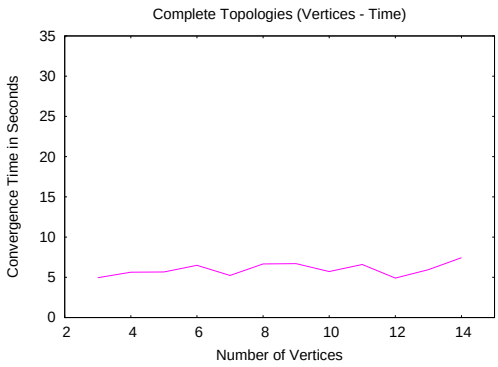

Abbildung 9: Konvergenzzeiten der Complete-Topologien mit steigender Knotenzahl

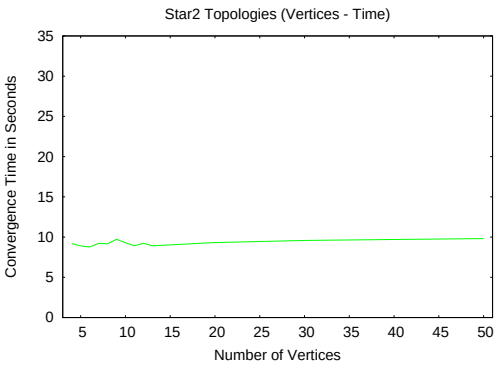

Abbildung 10: Konvergenzzeiten der Star2-Topologien mit steigender Knotenzahl

nur ein „Schritt“ notwendig, wodurch die Anzahl der Router für die Konvergenzzeit irrelevant wird. Lediglich das Trafficvolumen müsste steigen. Die Ergebnisse sind in Abbildung 9 dargestellt. Diese Topologien besitzen jeweils ein Minimum und ein Maximum an Kanten und stellen daher Grenzwerte dar. Aufgrund der Funktionsweise des RIP- Algorithmus und der damit sehr gut erklärbaren Konvergenzzeiten, können diese Topologieklassen jedoch auch als Ausnahmen angesehen werden.

Die Tendenz ist recht eindeutig. Die Konvergenzzeit steigt fast linear mit dem Durchmesser an. Dieses Verhalten wird im Kapitel 8.1 auf Seite 46 weiter untersucht.

Trafficvolumen

Ausgehend von den Annahmen aus Kapitel 5 (Seite 30) sollte der Durchmesser keinen großen Einfluss auf den Traffic haben. Dies wird auch von den Ergebnissen (siehe Abbildung 11) bestätigt.

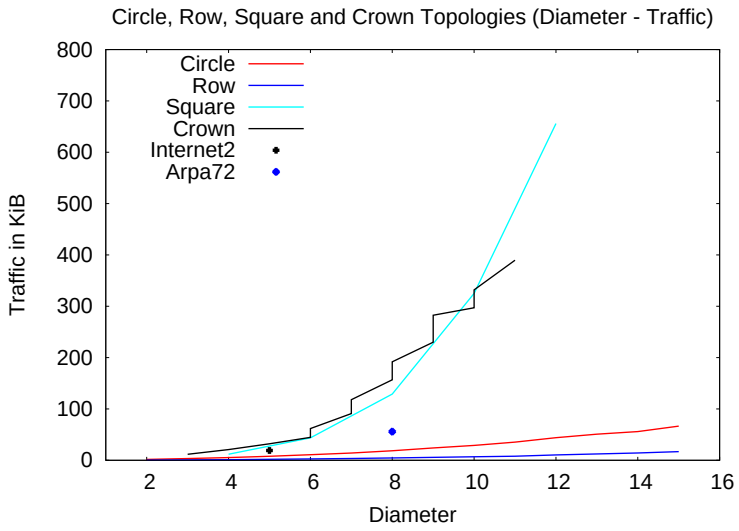

Abbildung 11: Trafficvolumen der verschiedener Topologien mit steigendem Durchmesser

Während die Row- und die Circle-Topologien eine sehr geringe Steigung des Trafficvolumens mit wachsendem Durchmesser haben, steigt der Traffic bei den Crown- und Square Topologien stark an.

Die extremen Unterschiede im Trafficvolumen zwischen den Topologien lassen darauf schließen, dass dieses nicht vom Durchmesser beeinflusst wird. Es scheint eher mit der Anzahl der Knoten oder Kanten zusammen zu hängen und wird in den nächsten Kapiteln weiter untersucht werden.

7.2 Kanten

Für die Untersuchung, welchen Einfluss die Anzahl der Netze auf das Konvergenzverhalten von RIP hat, wurden zufällige Netze mit neun Knoten und 10–25 Kanten getestet, die alle den Durchmesser 3 besitzen. Die Annahmen aus Kapitel 5.5 gehen von einem Einfluss auf den Updatetraffics aus, wohingegen die Konvergenzzeit unbeeinflusst bleiben sollte. Aus jedem der kompletten Graphen wurde eine Kante entfernt, sodass sie nun wie auch die Star2-Topologien einen Durchmesser von 2 besitzen (sie werden daher im Folgenden „Complete-1“ genannt). Erwartungsgemäß sollten beide Topologien gleich schnell konvergieren. Auch der Traffic sollte mit zunehmender Anzahl der Netze bei beiden Topologieklassen gleich schnell steigen.

Konvergenzzeit

Die Ergebnisse der Konvergenzzeiten entsprechen fast genau den Erwartungswerten. Wie in den Abbildungen 12 und 13 zu sehen, bleibt die Konvergenzzeit bei zunehmender Anzahl von Kanten nahezu konstant.

Einzig der Umstand, dass die Star2-Topologien durchgehend langsamer konvergierten als die Complete-Topologien, ist ungewöhnlich. Dies wurde jedoch schon in Kapitel 7.1 auf Seite 35 behandelt.

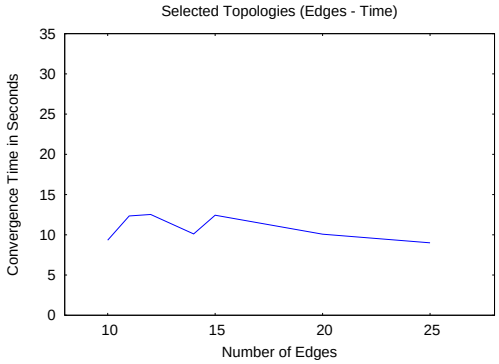

Abbildung 12: Konvergenzzeit bei konstantem Durchmesser und Knotenzahl bzgl. der Netzanzahl

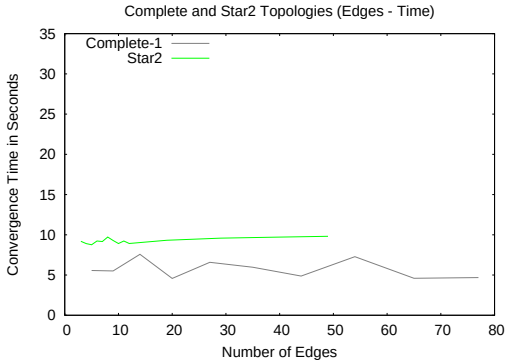

Abbildung 13: Konvergenzzeiten der Complete-1- und Star2-Topologien mit steigender Kantenzahl

Die Ergebnisse stützen die Annahme, dass die Anzahl der Netze keinen Einfluss auf die Konvergenzzeit hat.

Trafficvolumen

Das Trafficvolumen sollte durch die größeren Updatenachrichten mit zunehmender Anzahl der Netze steigen. Dies legen auch die Ergebnisse der durchgeführten Messungen nahe, mit denen die oben beschriebenen Topologien untersucht wurden.

Aus den Ergebnissen (Abbildungen 14 und 15) lässt sich schließen, dass das Trafficvolumen durch die Anzahl der Netze wesentlich beeinflusst wird. Daher hängt auch das durchschnittliche Trafficvolumen pro Netz von der Anzahl der Kanten ab.

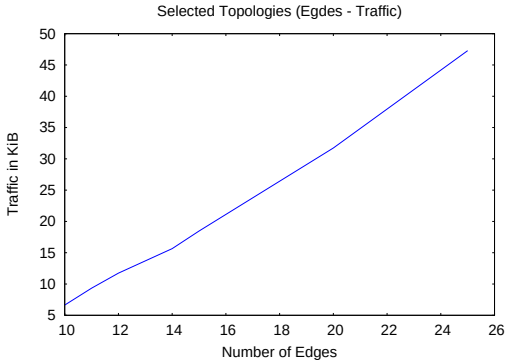

Abbildung 14: Trafficvolumen bei konstantem Durchmesser und Knotenzahl

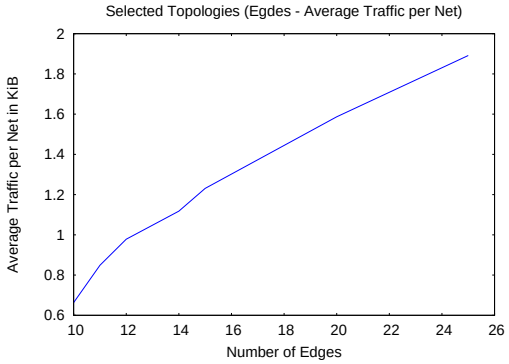

Abbildung 15: Durchschnittlicher Traffic pro Netz bei variabler Netzanzahl

Der Vergleich des Trafficvolumens von Star2- und Complete-1-Topologien (siehe Abbildungen 16 und 17) deutet darauf hin, dass auch die Anzahl der inneren Knoten einen Einfluss darauf haben müsste, denn sonst müsste das Trafficvolumen bei beiden Topologieklassen identisch sein. Da es sehr unwahrscheinlich ist, dass die Star2-Topologien mit wesentlich mehr Routern als die Complete-1-Topologien so viel weniger Traffic erzeugen, liegt es nahe, dass der Traffic durch die inneren Knoten erzeugt wird, von denen die Star2-Topologien immer nur einen haben, während sie bei Complete-1-Topologien mit wachsender Kantenzahl logarithmisch wachsen. Dies kann jedoch auch eine Ausnahme bei den gewählten Topologien sein, wie sie in Kapitel 7.1 „Durchmesser“ beschrieben wurde.

Eine genauere Analyse des Einflusses der Netzanzahl auf die Konvergenzeigenschaften wird durch statistische Auswertungen im Kapitel 8.2 auf Seite 47 erstellt.

Die Abbildungen 18 und 19 stellen noch einmal den Einfluss der Kantenzahl auf die Konvergenzeigenschaften dar.

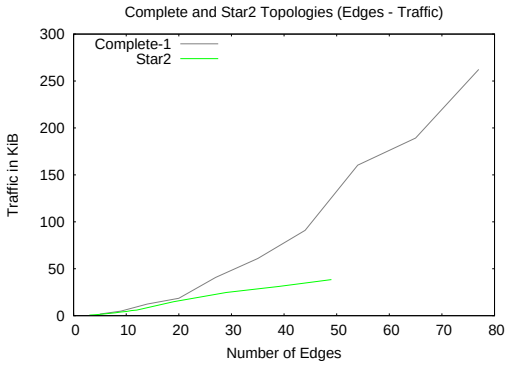

Abbildung 16: Vergleich des Trafficvolumens von Star2- und Complete-1-Topologien

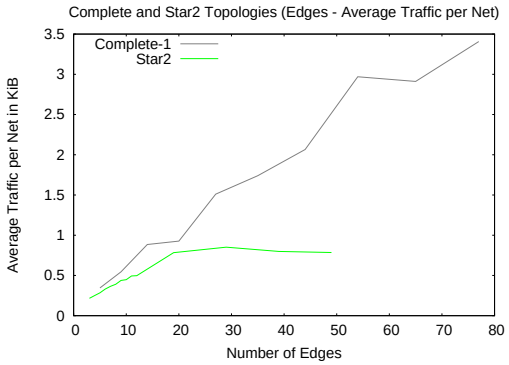

Abbildung 17: Durchschnittliches Trafficvolumen pro Netz von Star2 und Complete-1

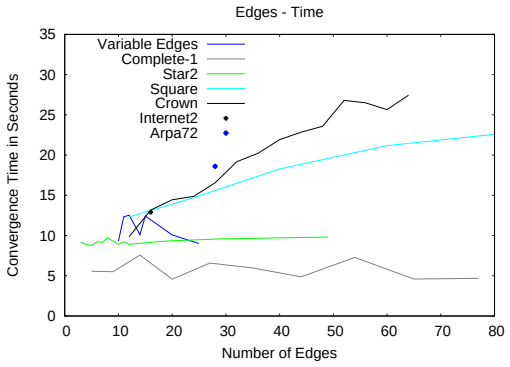

Abbildung 18: Konvergenzzeit der verschiedenen Topologien bei steigender Kantenzahl

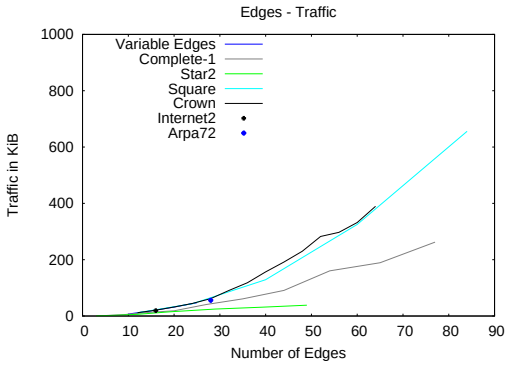

Abbildung 19: Trafficvolumen der verschiedenen Topologien bei steigender Kantenzahl

7.3 Knoten

In diesem Kapitel wird untersucht, welchen Einfluss die Anzahl der Router auf verschiedene Konvergenzeigenschaften hat. Dabei wird kein Unterschied zwischen Blättern und inneren Knoten gemacht.

Aufgrund der Tatsache, dass es bei konstanter Kantenzahl eine maximale Knotenzahl gibt (siehe Formel (3)), bei der ein Graph noch zusammenhängend ist und der maximale Durchmesser nach Formel (8) begrenzt ist (siehe auch die Definition des Durchmessers im Kapitel 1.1 auf Seite 11), sind Simulationen mit konstanter Kantenzahl, konstantem Durchmesser und variabler Knotenzahl nur begrenzt möglich. Daher wurden Simulationen mit zufällig generierten Topologien durchgeführt, bei denen die Kantenzahl sowie der Durchmesser gleich waren. Es wurden Graphen mit 15 Kanten und einem Durchmesser von 6 erstellt, die zwischen 10 und 16 Knoten besitzen.

Konvergenzzeit

Untersucht man die oben genannten Zufallstopologien mit variabler Knotenzahl auf ihre Konvergenzzeit (siehe Abbildung 20), so kann man keine eindeutige Tendenz erkennen. Dies deutet darauf hin, dass die Anzahl der Knoten keinen oder nur wenig Einfluss auf die Konvergenzzeit hat.

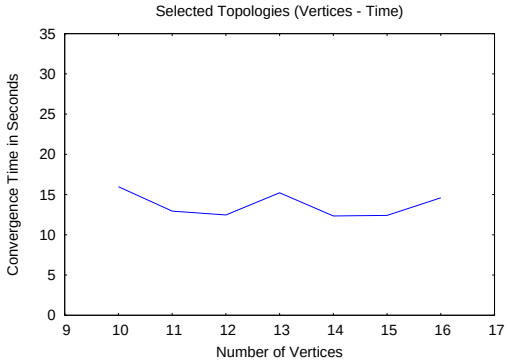

Abbildung 20: Konvergenzzeit bei variabler Knotenzahl

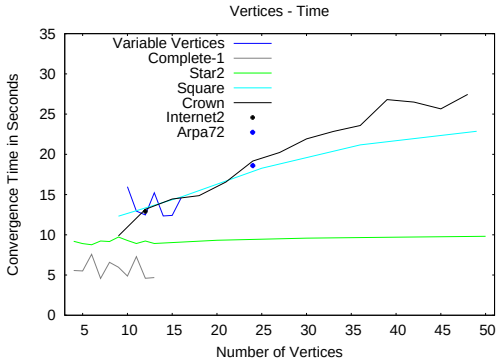

Abbildung 21: Konvergenzzeit der verschiedenen Topologien bei konstanter Knotenzahl

Auch die Star2- und Complete-Topologien benötigen eine konstante Zeit, um einen konvergenten Zustand zu erreichen. Dies kann man in den Abbildungen 9 und 10 auf Seite 37 sehen.

Trafficvolumen

Untersuchungen des Trafficvolumens der oben beschriebenen Topologien mit variabler Knotenzahl (siehe Abbildung 22), ergeben eine leicht fallende Tendenz mit zunehmender Knotenzahl.

Dieses Verhalten könnte daran liegen, dass die Anzahl der Blätter gestiegen ist und wird im Kapitel 7.4 untersucht.

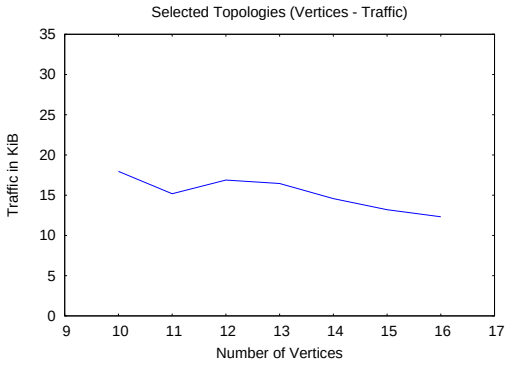

Abbildung 22: Trafficvolumen bei variabler Knotenzahl

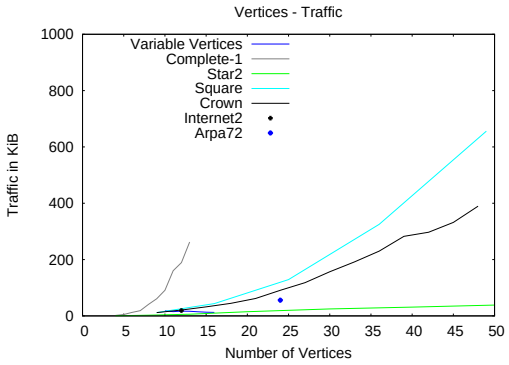

Abbildung 23: Trafficvolumen der verschiedenen Topologien bei steigender Knotenzahl

Der Traffic von Complete-1-Topologien steigt mit zunehmender Knotenzahl sehr stark an, während der Traffic von Star2-Topologien nur langsam mit zunehmender Knotenzahl wächst (siehe Abbildung 24). Dieses Ergebnis lässt drauf schließen, dass der Traffic nicht oder nur geringfügig von der Anzahl der Knoten abhängig ist.

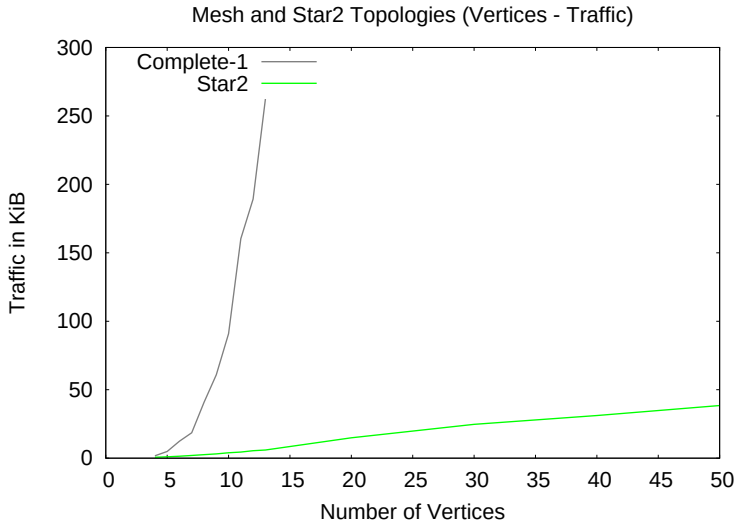

Abbildung 24: Trafficvolumen von Complete-1 und Star2

7.4 Blätter und innere Knoten

Für die Analyse der Konvergenzeigenschaften bezüglich der Blätter und inneren Knoten wurden Topologien erstellt, die elf Knoten und zehn Kanten besitzen. Von den Knoten waren zwei bis zehn Blätter und der Rest entsprechend innere Knoten. Der Durchmesser wurde bei diesen Topologien vernachlässigt und reduzierte sich um jeweils 1 pro zusätzlichem Blatt. Damit lag er zwischen 2 und 10. Da die Knotenzahl stets konstant ist und der Unterschied lediglich in der Verteilung von Blättern und inneren Knoten liegt, sind die Diagramme in ihrem Mittelpunkt gespiegelt, ansonsten jedoch identisch (siehe zum Beispiel Abbildungen 26 und 27). Aus diesem Grund werden nicht immer alle Diagramme abgebildet.

Konvergenzzeit

Die Konvergenzzeit sinkt mit zunehmender Anzahl von Blättern (siehe Abbildung 25) bzw. abnehmender Anzahl von Knoten. Dies hängt aber wahrscheinlich eher mit dem Durchmesser zusammen, der wie eingangs erwähnt mit zunehmender Anzahl von Blättern sinkt.

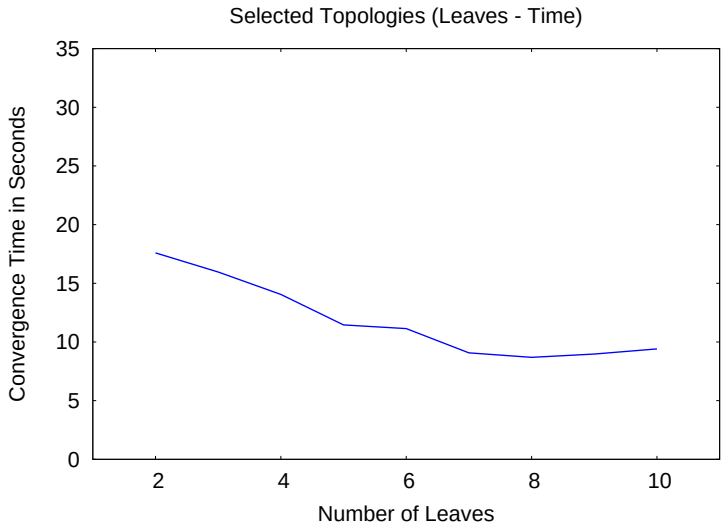

Abbildung 25: Konvergenzzeit bezüglich der Anzahl der Blätter

Somit sind aus diesen Messungen keine Erkenntnisse bezüglich des Einflusses von inneren Knoten oder Blättern auf die Konvergenzzeit möglich.

Da auch die Konvergenzzeiten der Star2- und Complete-1-Topologien trotz sehr unterschiedlicher Anzahl von Blättern und inneren Knoten konstant sind (siehe Abbildungen 9 und 10), ist ein Einfluss der Anzahl von Knoten insgesamt und der Anzahl von Blättern und inneren Knoten im Besonderen eher unwahrscheinlich.

Trafficvolumen

Bisherige Annahmen gingen davon aus, dass die Anzahl der Kanten ausschlaggebend für das Trafficvolumen ist (siehe Kapitel 5.5 und 7.2). Bei der Untersuchung des Einflusses von Knoten auf das Trafficvolumen (siehe Kapitel 7.3) wurde das sinkende Trafficvolumen bei steigender Knotenzahl auf die zunehmende Zahl von Blättern als Möglichkeit in Betracht gezogen. Dies wurde mit oben genannten Topologien genauer untersucht.

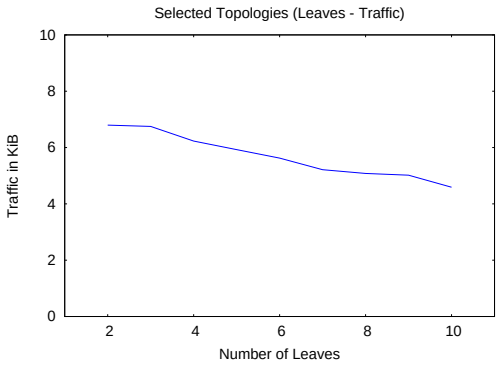

Abbildung 26: Trafficvolumen bezüglich der Anzahl der Blätter

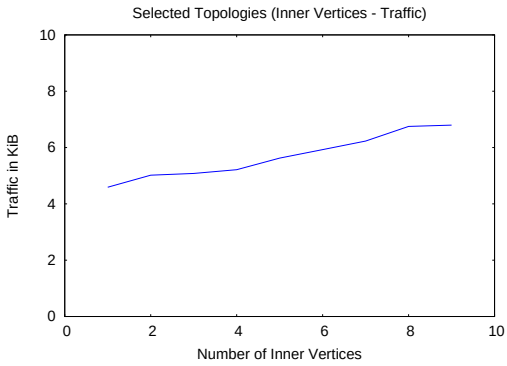

Abbildung 27: Trafficvolumen bezüglich der inneren Knoten

Wie die Ergebnisse aus den Abbildungen 26 und 27 zeigen, steigt das Trafficvolumen mit zunehmender Anzahl von inneren Knoten. Dies stützt die Annahme aus Kapitel 5.4, dass die Blätter vernachlässigt werden können.

Da alle Topologien genau 10 Kanten enthalten, ist das durchschnittliche Trafficvolumen pro Netz genau ein Zehntel des Gesamttraffics (siehe Abbildung 28).

Da keine der Vergleichstopologien Blätter besitzt, wird hier auf diesen Vergleich verzichtet.

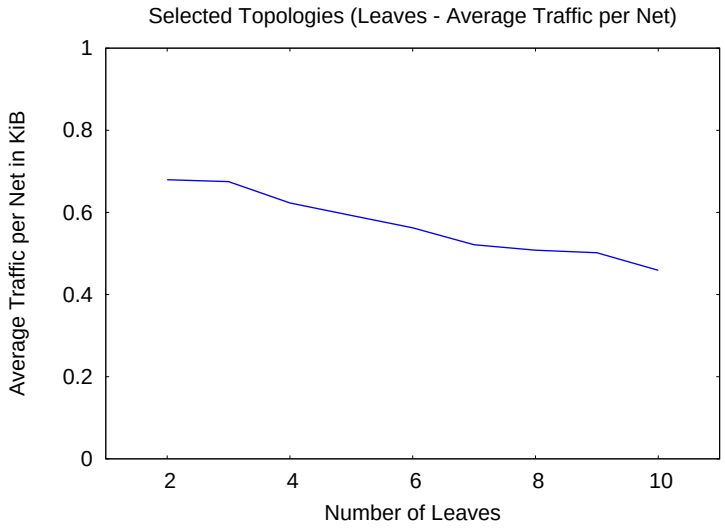

Abbildung 28: Durchschnittlicher Traffic pro Netz bezüglich der Blätter

8 Statistische Analysen

Da einige Topologieeigenschaften nicht oder nur schlecht mit gezielt ausgewählten Topologien getestet werden können und um die bisherigen Untersuchungen zu vertiefen, wird nun eine statistische Herangehensweise mit den in Kapitel 6 beschriebenen Messergebnissen gewählt.

Dabei wurde jeweils eine Kombination aus einer Topologieeigenschaft und einer Konvergenzeigenschaft untersucht. Für jede Untersuchung wurde ein Diagramm angefertigt, in dem die Topologieeigenschaft auf der x-Achse gegen die Konvergenzeigenschaft auf der y-Achse aufgetragen wurde. Anhand der Messergebnisse wurde mit der *Methode der kleinsten Quadrate* eine approximierende Funktion berechnet. Diese wurde jedoch nur ins Diagramm eingezeichnet, wenn ein Zusammenhang zwischen Topologie- und Konvergenzeigenschaft festgestellt werden konnte. Ansonsten wurde nur die Varianz berechnet.

In den Diagrammen dieses Kapitels stellt jedes rote Kreuz den Durchschnitt aus 15 Durchläufen einer Topologie dar. Die blaue Linie entspricht der approximierenden Funktion. Zum Vergleich wurden noch die beiden Vergleichstopologien Internet2 und Arpa72 eingezeichnet, deren Punkte ebenfalls aus dem Durchschnitt von 15 Messungen gebildet wurden.

8.1 Durchmesser

Aus den Annahmen aus Kapitel 5.2 sowie den vorangegangen Messungen in Kapitel 7.1 lässt sich schließen, dass der Durchmesser eine wichtige Rolle für die Konvergenzzeit spielt. Daher wurden zuerst die Ergebnisse aller Testläufe auf diesen Zusammenhang hin untersucht (siehe Abbildung

29). Die Methode der kleinsten Quadrate ergab dabei eine Steigung von 1.5 mit einer Varianz von 7.2, was einer Standardabweichung von 2.69 entspricht.

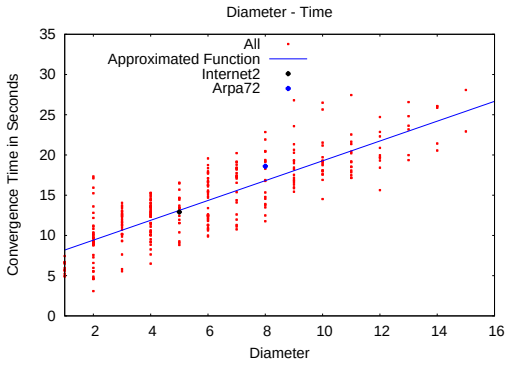

Abbildung 29: Konvergenzzeiten bezüglich des Durchmessers

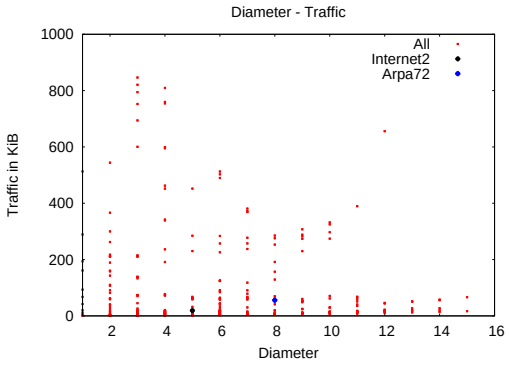

Abbildung 30: Trafficvolumen bezüglich des Durchmessers

Somit scheint der Durchmesser nicht nur in einfachen Topologien ausschlaggebend für die Konvergenzzeit zu sein. Man kann die Konvergenzzeit durch die Formel (14) für beliebige Topologien berechnen. Trotz der großen Varianz kann man alleine aus dem Durchmesser einer Topologie schon Aussagen über ihre Konvergenzzeit treffen.

$$t_{approx} = 1.5 \cdot d + 7 \pm 2.69[s] \tag{14}$$

Wie schon in Kapitel 7.1 festgestellt, hat der Durchmesser jedoch keinen Einfluss auf das Trafficvolumen. Dies spiegeln auch die Messungen, die in Abbildung 30 illustriert werden, wieder. Ein Diagramm des durchschnittlichen Traffics pro Netz mit steigendem Durchmesser wurde hier ausgelassen, da der Traffic pro Netz vom Gesamttraffic abhängt und somit auch kein Zusammenhang mit dem Durchmesser festzustellen war.

8.2 Kanten

Die Untersuchung des Trafficvolumens bezüglich der Kantenzahl konnte bei der statistischen Analyse noch deutlichere Ergebnisse erzielen als die Analyse mit ausgewählten Topologien in Kapitel 7.2. Wie die Abbildung 31 zeigt, wächst das Trafficvolumen mit steigender Kantenzahl polynomiell an. Dementsprechend wächst der durchschnittliche Traffic pro Netz linear (siehe Abbildung 32). Die approximierte Funktion für den Gesamttraffic entspricht Formel (15). Die Varianz beträgt 991.

$$T_{approx} = 0.08 * m^2 - 0.29 * m + 2.39 \pm 31.49[KiB] \tag{15}$$

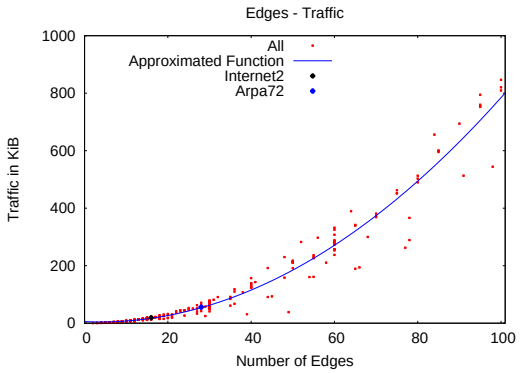

Abbildung 31: Trafficvolumen bei steigender Anzahl von Kanten

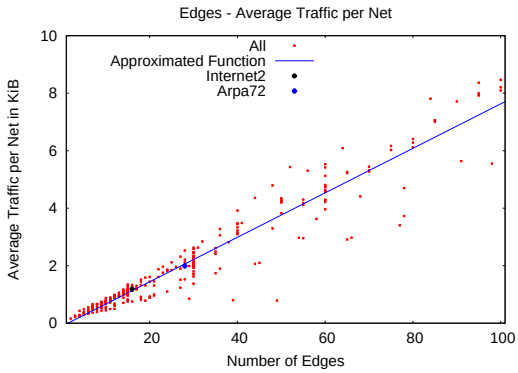

Abbildung 32: Durchschnittlicher Traffic pro Netz bei steigender Anzahl von Kanten

Wie erwartet (siehe Kapitel 5.5 auf Seite 31 sowie Kapitel 7.2) wurde bei der Auswertung der Messergebnisse kein Zusammenhang zwischen der Anzahl der Kanten und der Konvergenzzeit erkennbar. Dies kann gut man anhand von Abbildung 33 erkennen. Die approximierende Funktion wurde in das Diagramm nicht eingezeichnet, da diese mit einer Varianz von ca. 20 keine Aussagekraft hätte.

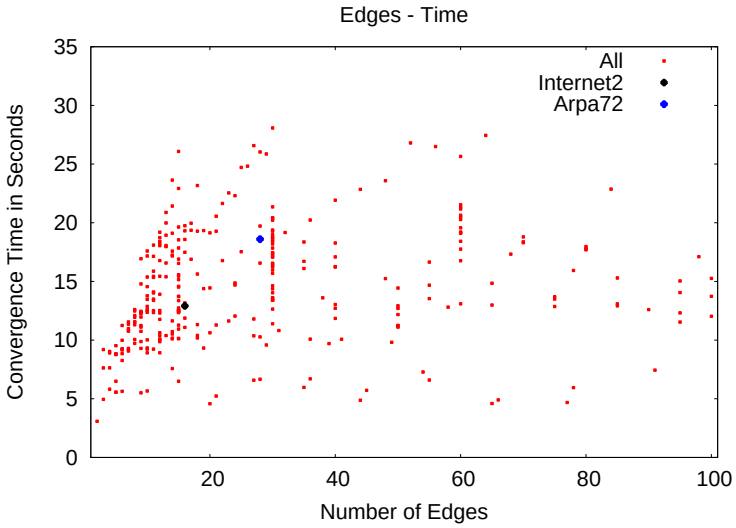

Abbildung 33: Konvergenzzeit bezüglich der Kantenzahl

8.3 Knoten

Den Annahmen aus Kapitel 5.3 zufolge sollte die Anzahl der Knoten zumindest eine geringe Auswirkung auf das Trafficvolumen haben. Aufgrund der Berechnungszeit und der Verzögerung durch die Timer des RIP-Protokolls wurde auch ein Anstieg der Konvergenzzeit nicht ausgeschlossen. Bei den ersten Untersuchungen aus Kapitel 7.3 konnte mit ausgewählten Topologien jedoch keinen Zusammenhang zwischen Konvergenzzeit und der Knotenzahl festgestellt werden. Auch das Trafficvolumen wurde nicht maßgeblich von der Anzahl der Knoten beeinflusst. Bei der statistischen Analyse zeigte die Anzahl der Knoten jedoch eine gewisse Auswirkung auf die Konvergenzzeit. Die Messpunkte sind zwar bei einer Varianz von 14.4 ziemlich weit gestreut, doch eine erhöhte Konvergenzzeit bei zunehmender Anzahl von Knoten ist erkennbar.

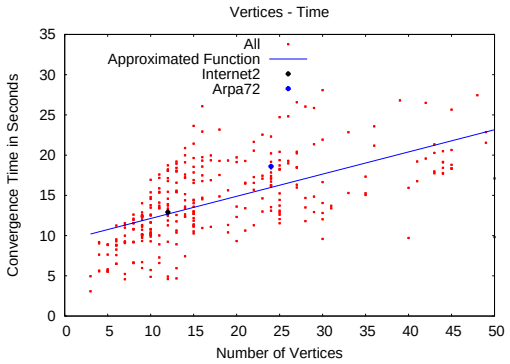

Abbildung 34: Konvergenzzeit bei steigender Anzahl von Knoten

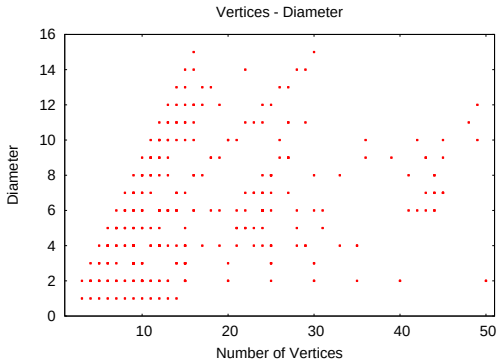

Abbildung 35: Zusammenhang zwischen Knotenzahl und Durchmesser

Um ausschließen zu können, dass in den getesteten Topologien der Durchmesser grundsätzlich mit steigender Anzahl von Knoten wächst, wurde in Abbildung 35 die Anzahl der Knoten gegen den Durchmesser der jeweiligen Topologie aufgetragen. Die großflächige Verteilung belegt, dass es in den getesteten Topologien keinen Zusammenhang zwischen Knotenzahl und Durchmesser gibt. Somit wurden die hier gemessenen Auswirkungen der Knotenzahl auf die Konvergenzzeit nicht indirekt durch den Durchmesser sondern wirklich durch die Knoten verursacht. Dies ist Anlass dazu, die Anzahl der Knoten zu einem gewissen Teil mit in die Berechnung der Konvergenzzeit in Kapitel 8.7.1 einfließen zu lassen.

Beim Trafficvolumen ist kein Zusammenhang zu der Anzahl der Kanten zu erkennen (siehe Abbildung 36). Die approximierende Funktion hat mit einer Varianz von 16301 (das entspricht einer Standardabweichung von ca. 127 KiB) keine Aussagekraft und wurde daher nicht eingezeichnet.

8.4 Innere Knoten und Blätter

Es wurde in Kapitel 5.4 angenommen, dass vor allem die inneren Knoten einen Einfluss auf die Konvergenzeigenschaften haben, während die Blätter wahrscheinlich zu vernachlässigen sind. Diese Annahme konnte auch bei den Untersuchungen in Kapitel 7.4 beobachtet werden. Die Diagramme der inneren Knoten (Abbildungen 37 und 38), welche die Ergebnisse der statistischen

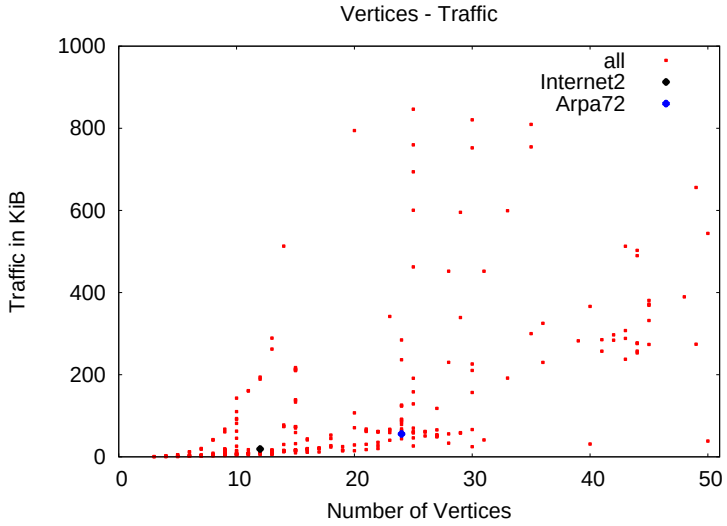

Abbildung 36: Traffic bei steigender Anzahl von Knoten

Analyse darstellen, ähneln den Diagrammen der Knoten (Abbildungen 34 und 36) sehr. Jedoch sind die Varianzen mit 13.2 für die Konvergenzzeit und 15074.9 für das Trafficvolumen geringfügig kleiner als bei den Knoten (dort sind es 14.4 und 16301). Dies ist ein weiterer Hinweis darauf, dass die Anzahl der Blätter vernachlässigbar für die Konvergenzeigenschaften ist.

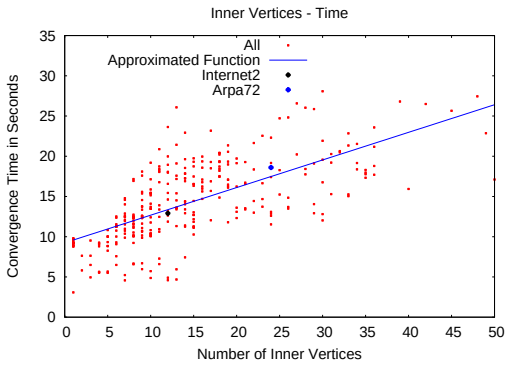

Abbildung 37: Konvergenzzeit bezüglich der inneren Knoten

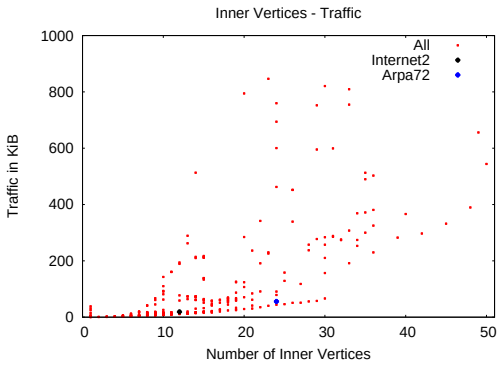

Abbildung 38: Trafficvolumen bezüglich der inneren Knoten

Bei der Analyse zur Anzahl der Blätter konnte kein Zusammenhang zu Konvergenzzeit oder Trafficvolumen festgestellt werden. Da sehr viele der getesteten Topologien keine Blätter besitzen, gibt es eine sehr hohe Konzentration an Messpunkten am Nullpunkt der x-Achse. Aufgrund der großen Varianzen von 23.6 für die Konvergenzzeit und 27819 für das Trafficvolumen wurden auch keine approximierenden Funktionen in den Diagrammen dargestellt.

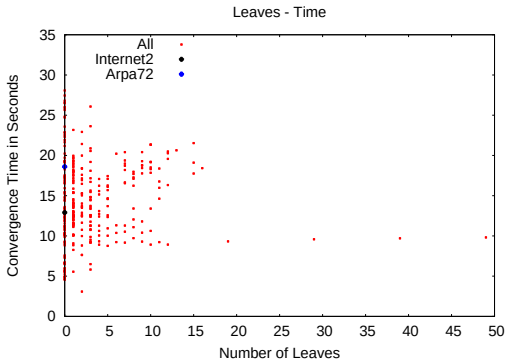

Abbildung 39: Konvergenzzeit bezüglich der Blätter

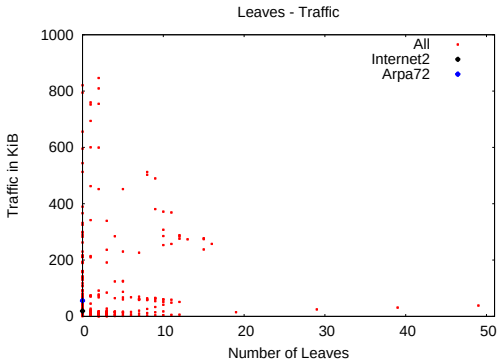

Abbildung 40: Trafficvolumen bezüglich der Blätter

8.5 Clusterkoeffizient

Die Untersuchungen zum Einfluss des Clusterkoeffizienten auf die Konvergenzzeit (siehe Abbildung 41) ergaben, dass mit zunehmendem Clusterkoeffizienten die Konvergenzzeit abnimmt. Formel (16) entspricht einer Funktion, die mit einer Varianz von 12.3 eine gute Annäherung darstellt.

$$t_{approx} = \frac{1}{0.822 \cdot cc} + 9.04 \pm 3.5[s] \tag{16}$$

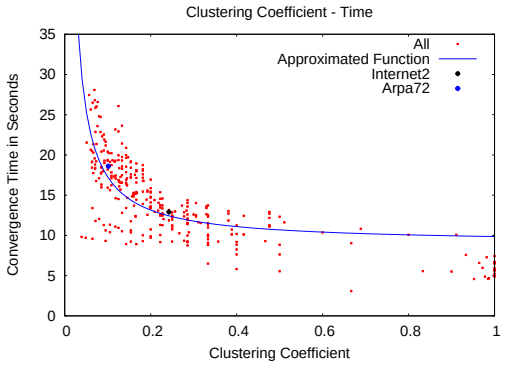

Abbildung 41: Konvergenzzeit bezüglich des Clusterkoeffizienten

Abbildung 42: Zusammenhang zwischen Clusterkoeffizient und Durchmesser

Dieser Zusammenhang zwischen Konvergenzzeit und Clusterkoeffizient kann jedoch auch mit dem abnehmenden Durchmesser bei steigendem Clusterkoeffizienten begründet werden (siehe auch Kapitel 5.6 auf Seite 31). Der Durchmesser sinkt zwar nicht zwangsläufig mit zunehmendem Clusterkoeffizient, aber eine Tendenz ist zumindest bei den in dieser Arbeit getesteten Topologien vorhanden. Wie man in Abbildung 42 sehen kann, sinkt der Durchmesser bei steigendem

Clusterkoeffizient in etwa mit der gleichen Geschwindigkeit wie die Konvergenzzeit. Daher ist es sehr wahrscheinlich, dass die Unterschiede in der Konvergenzzeit vor allem vom Durchmesser abhängen. Der Zusammenhang zwischen Clusterkoeffizient und Konvergenzzeit kommt somit nur indirekt durch den Zusammenhang zwischen Clusterkoeffizient und Durchmesser zustande.

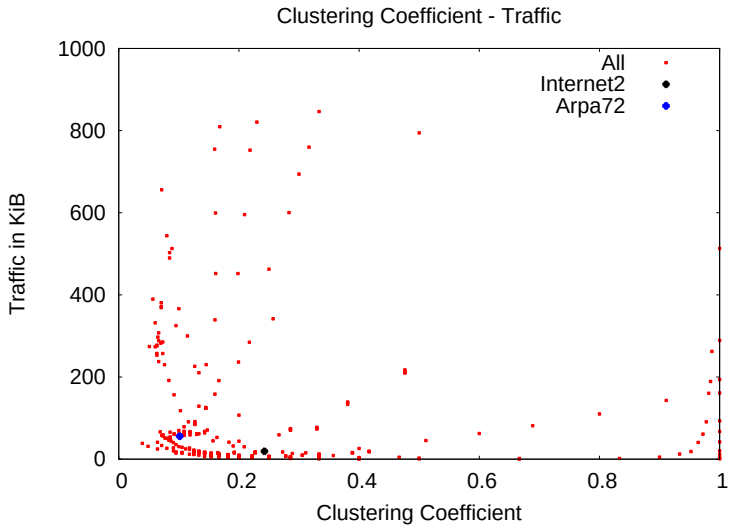

Abbildung 43: Konvergenzzeiten der Circle- und Row-Topologien mit zunehmender Knotenzahl

Obwohl die Kantenzahl pro Knoten mit steigendem Clusterkoeffizienten zunimmt, gilt dies nicht für die Anzahl der Kanten insgesamt. Daher ist ein indirekter Zusammenhang wie bei der Konvergenzzeit nicht gegeben. Es kann aus den vorliegenden Messergebnissen (siehe Abbildung 43) kein Zusammenhang zwischen dem Clusterkoeffizient und dem Trafficvolumen festgestellt werden. Aufgrund der hohen Varianz von 27054 wurde auf die Darstellung eines Funktionsgraphs verzichtet.

8.6 Kreise

Die Anzahl der Kreise bestimmt implizit die minimale Anzahl von Kanten. Aufgrund von Formel (4) ist die Anzahl der Kanten immer größer als die zyklomatische Zahl (also die Anzahl der Kreise). Dies wird auch in Abbildung 45 deutlich. Da das Trafficvolumen mit zunehmender Anzahl von Kanten wächst, gilt dies natürlich auch für die Anzahl der Kreise, wie es in Abbildung 44 ersichtlich ist. Allerdings sagt die zyklomatische Zahl nichts über die maximale oder durchschnittliche Anzahl der Kanten aus, wodurch die Varianz der approximierenden Funktion (siehe Formel (17)) mit ca. 6484 sehr hoch ist.

$$T_{approx} = 8.68 \cdot k + 3.7 \pm 80.53[KiB] \tag{17}$$

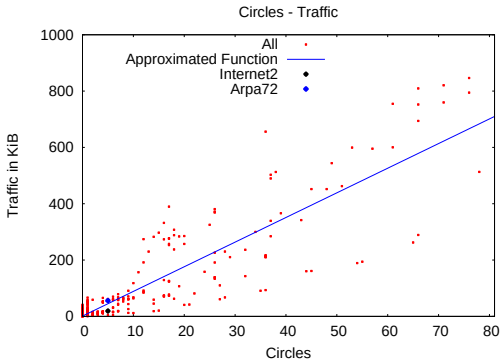

Abbildung 44: Trafficvolumen bezüglich der Kreiszahl

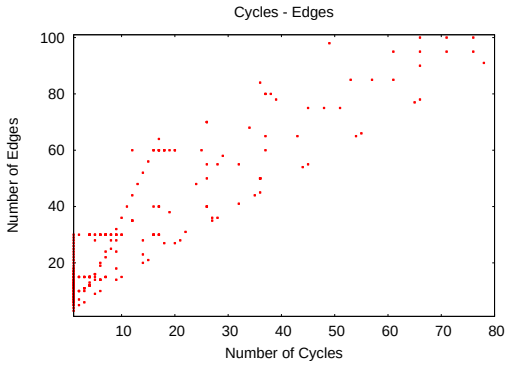

Abbildung 45: Anzahl der Kanten bezüglich der Kreiszahl

Wie in Abbildung 46 zu sehen ist, gibt es keinen Zusammenhang zwischen der zyklomatischen Zahl und der Konvergenzzeit. Eine approximierende Funktion hätte mit einer Varianz von 22.2 keinerlei Aussagekraft.

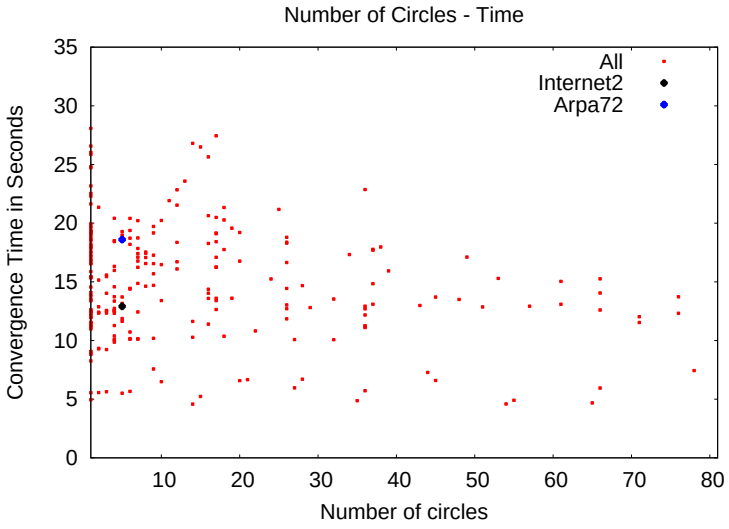

Abbildung 46: Konvergenzzeit bezüglich der Kreiszahl

8.7 Statistische Analysen mit multidimensionalen Funktionen

Nachdem im vorherigen Kapitel einzelne Zusammenhänge von jeweils einer Topologieeigenschaft mit einer Konvergenzeigenschaft untersucht wurden, sollen hier genauere Zusammenhänge durch Kombinationen von Topologieeigenschaften aufgedeckt werden. Mittels *Regressionsanalyse* sowie der Methode der kleinsten Quadrate werden im Folgenden multidimensionale Funktionen berechnet.

Für diese Analysen werden wie schon in Kapitel 8 die Datensätze aus Kapitel 6 genutzt. Dabei wurden die bisher erlangten Erkenntnisse verwendet und versucht, durch Veränderung der Grundformeln für die approximierenden Funktionen deren Varianzen zu minimieren.

8.7.1 Konvergenzzeit

Weitere Analysen ergaben, dass eine Berücksichtigung der Knotenzahl die Varianz von 7.2 auf 3.54 halbieren kann. Somit lässt sich mit Gleichung (18) die Konvergenzzeit t_{approx} einer beliebigen Topologie auf ca. 1.88 Sekunden genau berechnen (siehe Abbildung 47).

$$t_{approx} = 1.5 \cdot d + 0.16 \cdot n + 4.8 \pm 1.88[Sekunden] \tag{18}$$

t_{approx} = approximierte Konvergenzzeit
d = Durchmesser
n = Anzahl der Knoten

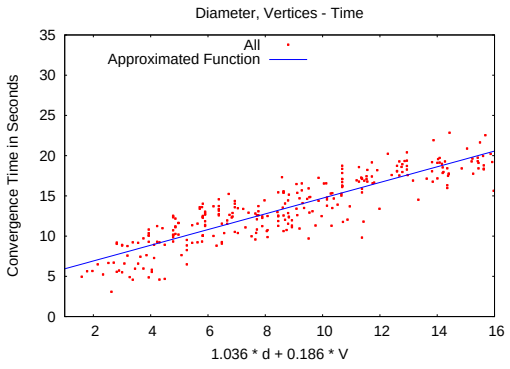

Abbildung 47: Approximation der Konvergenzzeit mittels Durchmesser und Knotenzahl

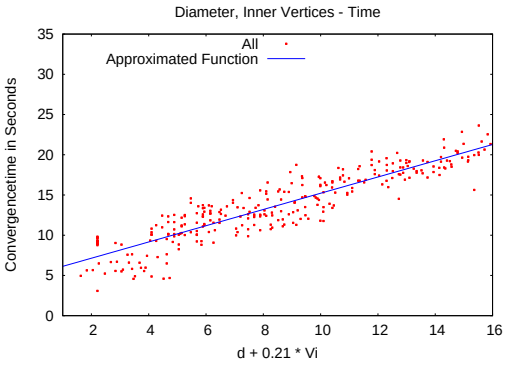

Abbildung 48: Konvergenzzeit bezüglich des Durchmessers und der Anzahl der inneren Knoten

Wie bereits in den Annahmen aus Kapitel 5.4 sowie in den Versuchen aus Kapitel 8.4 festgestellt, können die Blätter für die Berechnung der Konvergenzzeit vernachlässigt werden. Wendet man die Methode der kleinsten Quadrate nun nur auf die inneren Knoten und den Durchmesser an,

so hat die approximierende Funktion (siehe Formel (19)) eine Varianz von 3.47, was im Vergleich zu 3.54 für Formel (18) geringfügig genauer ist (siehe Abbildung 48). Somit kann man davon ausgehen, dass man bei der Vernachlässigung der Blätter geringfügig genauere Ergebnisse erzielt, als wenn man alle Knoten gleich gewichtet.

$$t_{approx} = d + 0.21 \cdot n_i + 3.5 \pm 1.86[Sekunden] \tag{19}$$

t_{approx} = approximierte Konvergenzzeit
d = Durchmesser
n_i = Anzahl der inneren Knoten

In Abbildung 49 wird der Zusammenhang zwischen Durchmesser, inneren Knoten und der Konvergenzzeit als dreidimensionales Diagramm dargestellt.

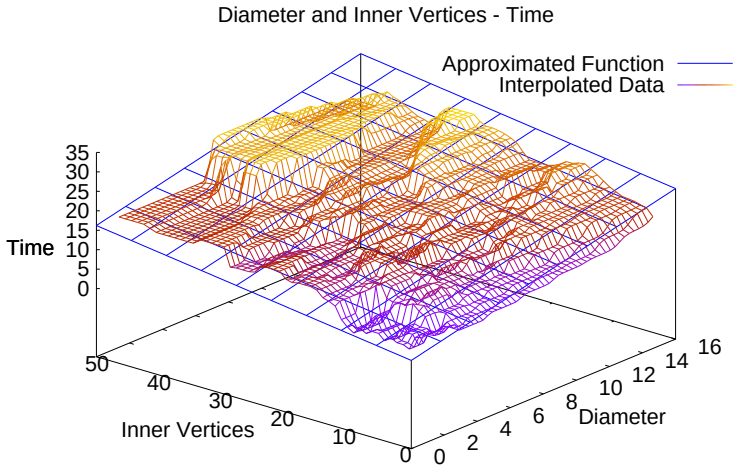

Abbildung 49: Konvergenzzeiten in Abhängigkeit von Durchmesser und inneren Knoten

8.7.2 Trafficvolumen

Die Untersuchungen bezüglich des Trafficvolumens ergaben, dass die Ergebnisse (siehe Abbildung 31) aus dem Kapitel 8.2 bereits ziemlich genau waren. Somit kann Formel (15) (Seite 47) bereits eine gute Annäherung liefern.

Die Varianz konnte unter der Berücksichtigung der Knoten von 991 auf 893 gesenkt werden. Abbildung 50 stellt den Zusammenhang zwischen Knoten, Kanten und dem Trafficvolumen als dreidimensionales Diagramm dar. Abbildung 51 zeigt einen linearisierten Funktionsgraph. Mit

Formel (20) kann die Standardabweichung für die Berechnung des Trafficvolumens von 31.48 auf 29.89 KiB verringert werden.

$$T_{approx} = 0.9 \cdot m^2 - 1.6 \cdot m - 0.025 \cdot n^2 + 2.6 \cdot n - 9.7 \pm 29.89[KiB] \tag{20}$$

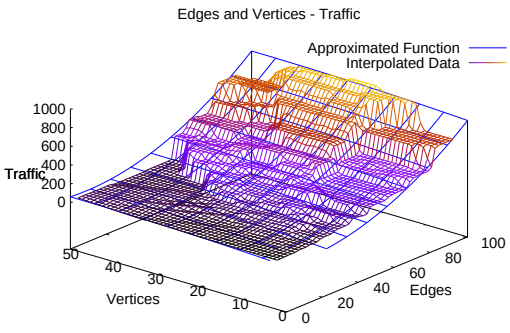

Abbildung 50: Approximation des Traffics mittels Anzahl von Knoten und Kanten

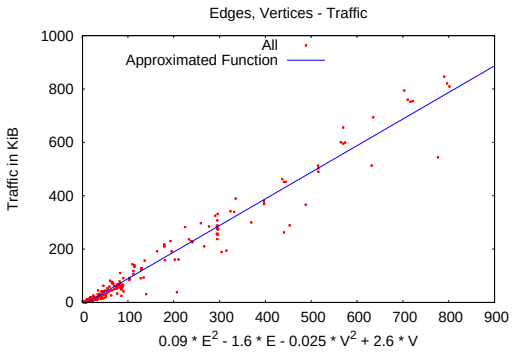

Abbildung 51: Linearisierter Traffic bezüglich Anzahl von Knoten und Kanten

Um ausschließen zu können, dass es bei den getesteten Topologien einen Zusammenhang zwischen Knoten und Kanten gibt, wurden die Ergebnisse daraufhin untersucht. Wie Abbildung 52 zeigt, wird die Anzahl der Kanten lediglich durch Formel (6) und (2) nach oben und unten beschränkt.

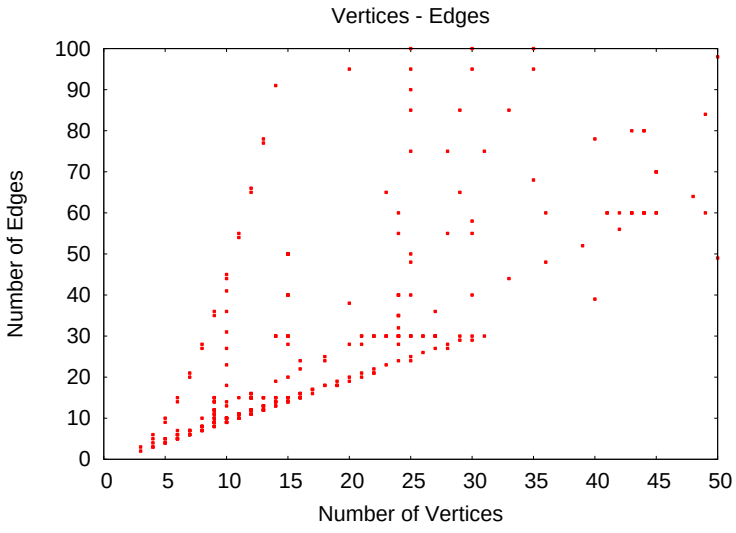

Abbildung 52: Anzahl der Kanten bezüglich der Knotenzahl

Die Berücksichtigung der Anzahl an inneren Knoten (statt der Anzahl der Knoten insgesamt) senkte die Varianz lediglich auf 948. Somit scheinen die inneren Knoten nur für die Konvergenzzeit eine relevante Größe zu sein.

Nutzt man die zyklomatische Zahl (anstatt der Knotenzahl) zur Approximation, verringert sich die Varianz auf ca. 900.

9 Fazit

Zusammenfassend kann festgehalten werden, dass der Durchmesser einen entscheidenden Einfluss auf die Konvergenzzeit hat, während die Anzahl der Kanten einer Topologie wesentlich den Traffic bestimmen, der benötigt wird, um das Netzwerk in einen konvergenten Zustand zu überführen. Mit den Formeln (19) und (20) können diese Topologieeigenschaften recht genau berechnet werden. Überraschenderweise führt die Vernachlässigung der Blätter entgegen der Annahmen aus Kapitel 5.4 bei der Berechnung des Trafficvolumens zu einer höheren Varianz als die Betrachtung aller Knoten. Dieses Verhalten lässt sich im Moment nicht erklären.

Die Durchschnittswerte der einzelnen Topologien hätten durch mehr Durchläufe noch verbessert werden können. Dies wurde jedoch zugunsten einer größeren Menge an getesteten Topologien unterlassen, da die Anzahl der Topologien wichtig für die Allgemeingültigkeit der Erkenntnisse ist.

Auch die Berücksichtigung von mehr als zwei Topologieeigenschaften hätte die Varianzen der approximierenden Funktionen verbessert. Die dadurch entstehenden Formeln wären jedoch nur besser auf die getesteten Topologien angepasst gewesen und hätten wahrscheinlich keinen allgemeingültigen Charakter mehr. Ansonsten hätten sich mehr Topologieeigenschaften in den statistischen Analysen aus Kapitel 8 hervorgehoben.

Da Triggered Updates durch anstehende periodische Updates verzögert werden können (siehe Kapitel 2), sind zwei Durchläufe der selben Simulation nie identisch. Die Konvergenzzeiten können um mehrere Sekunden schwanken. Dies ist auch ein Grund, warum die Varianzen teilweise recht hoch sind. Somit wäre bei noch mehr Durchläufen und weiteren getesteten Topologien eine genaue Vorhersage der Konvergenzeigenschaften nicht möglich, auch wenn sich dadurch die Varianzen verringern würden und ein besserer Durchschnitt erzielt würde.

Trotzdem lassen sich anhand der in dieser Arbeit erarbeiteten Formeln die Konvergenzeigenschaften eines Netzwerks allein durch die Kenntnis der Topologie zumindest näherungsweise bestimmen. Dies könnte beim Design von neuen Netzwerken zu Optimierungszwecken genutzt werden. Auch die Optimierung von bestehenden Netzwerken wäre aufgrund dieser Erkenntnisse möglich. Denkbar wäre auch, dass man bei einem bestehenden Netzwerk durch Einfügen von Netzen, die den Durchmesser stark senken auch die Ausfallsicherheit wesentlich effektiver erhöht als durch das Einfügen von Netzen, die den Durchmesser nicht oder nur minimal senken. Dafür müsste man jedoch eine quantitativ messbare Definition von „Ausfallsicherheit“ erarbeiteten und weitere Untersuchungen durchführen.

Eine mathematische Analyse der Konvergenzeigenschaften zum Beispiel durch die Berechnung von Flüssen in Graphen oder Higher Order Logic [Obr00] hätte zwar verifizierbare Aussagen geliefert, jedoch nur für den Best Case und den Worst Case. Diese sind für die Praxis jedoch nicht relevant, da sie fast nie auftreten und auch künstlich nur schwer zu provozieren sind.

Es wurden einige Anstrengungen unternommen, um möglichst realistische Ergebnisse zu erlangen und die Aussagen so gut wie möglich zu untermauern. Dazu gehört die Aufteilung der Durchläufe in eine Online- und eine Offline-Phase sowie die Verwendung von VNUML, also virtuellen Maschinen mit realen Protokollen. Des Weiteren wurden die erkannten Zusammenhänge auf indirekte Abhängigkeiten anderer Topologieeigenschaften hin untersucht. Dabei wurde verifiziert,

dass die für die Formeln herangezogenen Topologieeigenschaften unabhängig voneinander sind, was eine gute Auswahl der untersuchten Topologien bestätigt.

Trotzdem müsste man die Ergebnisse dieser Arbeit noch in anderen Simulationsumgebungen sowie realen Netzwerken und mit größeren Topologien verifizieren, um allgemeingültige Aussagen machen zu können. Für kleine bis mittelgroße Netzwerke (bis zu 50 Knoten und 100 Kanten) sollten die hier erarbeiteten Erkenntnisse jedoch zutreffen.

10 Anhang

A Bedienungsanleitung

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

A.1 Übersicht

Um Simulationen zu automatisieren, werden drei Konfigurationsdateien benötigt:

- Eine *Topologiebeschreibung* in Form einer ZVF-Datei (siehe auch Seite 66).
- Ein *Test-Script*, in dem die einzelnen Simulationsschritte definiert sind, als .cfg-Datei (siehe auch Seite 69).
- Eine *Ausführungsbeschreibung*, die beschreibt, welche Topologien mit welchen Test-Scripten wie oft simuliert werden sollen (siehe auch Seite 73).

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 bereits vorhanden ist.

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 jede Zeile einem Durchlauf 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 Topologiebeschreibung zusammengesetzt (ohne Endungen und mit Bindestrich getrennt).

Funktionen des Programms

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 Ausführungsbeschreibung aufgeführt 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] DUMP_FILE(S)` – Berechnet die Konvergenzeigenschaften für die angegebenen tcpdump-Dateien zu einer gegebenen Topologie.

`-a RESULT_FILE(S)` – Errechnet die durchschnittlichen Konvergenzzeiten von allen angegeben Ergebnisdateien.

`-z [-i] TOPOLOGY_FILE(S)` – 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 TOPOLOGY_FILE(S)` – Erzeugt VNUML XML-Dateien für alle angegebenen Topologiebeschreibungen.

`-g TYPE [-o OUTPUT_FILE] [-i] ARGUMENTS` – Generiert einen Graphen des Typs TYPE [17]. Es wird eine ZVF-Datei mit den Eigenschaften des Graphen erzeugt.

`-C TOPOLOGY_FILE(S)` – Überprüft die angegebene ZVF-Datei auf Sytaxfehler.

`-H FILE(S)` – 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 angegebene Datei (die Datei „-“ bezeichnet die Standardausgabe).

`-T TIME` – Zieht nur Pakete zur Konvergenzberechnung heran, die vor dem Zeitpunkt TIME versendet wurden (im Modus `-S` wird die Zeitangabe ignoriert, da diese aus der Ergebnisdatei entnommen werden wird).

`-F TIME` – Gibt den Startzeitpunkt für die Konvergenzberechnung an (im Modus `-S` wird die Zeitangabe ignoriert, da diese aus der Ergebnisdatei entnommen werden wird).

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

[17] Die möglichen Typen und ihre Argumente sind in Kapitel A.6 auf Seite 81 beschrieben.

A.2 Installation und Konfiguration

Im Folgenden werden die einzelnen Schritte der Installation und Konfiguration von VNUML und der Simulationsumgebung `zimulator.pl` auf Debian-basierten Linuxdistributionen beschrieben. Getestet wurde `zimulator.pl` unter Ubuntu 9.04, Ubuntu 9.10 und Fedora 11.

Installation von VNUML

Als erstes muss das benötigte Paket `bridge-utils` mit dem Befehl aus Quelltext 5 installiert werden.

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

Quelltext 5: Installation der bridge-utils

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

Als nächstes wird VNUML installiert. Dazu wird mit root-Rechten die Datei `/etc/apt/sources.list` editiert und folgende Zeile aus Quelltext 6 hinzugefügt:

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

Quelltext 6: VNUML Debian Repository

Die drei Befehle aus Quelltext 7 installieren dann VNUML und den UML-Kernel.

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

Quelltext 7: Installation von VNUML

Nun kann man von der Seite http://www.dit.upm.es/vnumlwiki/index.php/Download ein Dateisystem herunterladen. Im Abschnitt „Root filesystems“ werden zwei 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 stattdessen 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 Kommando in Quelltext 8 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
```

Quelltext 8: Kopieren des Dateisystems

Schließlich werden noch die Konfigurationsdateien für Zebra und RIP benötigt. Dafür wird ein Verzeichnis `conf` erstellt, in dem die beiden Dateien `ripd.conf` (siehe Quelltext 9) und `zebra.conf` (siehe Quelltext 10) liegen.

```
hostname zebra
password xxxx
enable password xxxx
```

Quelltext 9: Zebra-Konfigurationsdatei

```
hostname ripd
password xxxx
router rip
network 10.0.0.0/8
timers basic 10 30 20
```

Quelltext 10: RIP-Konfigurationsdatei

Installation und Konfiguration von zimulator.pl

Das Programm `zimulator.pl` nutzt einige CPAN-Bibliotheken sowie das Programm GraphViz[18]. Diese Pakete können mit dem Befehl aus Quelltext 11 installiert werden.

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

Quelltext 11: Installieren der Abhängigkeiten von zimulator.pl

Die Bibliothek `Graph` muss über CPAN installiert werden (siehe Quelltext 12), da der Dijkstra-Algorithmus der Ubuntu-Version (Paket libgraph-perl unter Ubuntu 9.04 und Ubuntu 9.10) nicht funktioniert. Ebenso muss die Bibliothek `Test::More` über CPAN installiert werden, wenn man die dem Programm beiliegenden Tests starten möchte.

[18] http://www.graphviz.org/

```
sudo cpan
install Graph
install Test::More
exit
```

Quelltext 12: CPAN-Installation der Abhängigkeiten von zimulator.pl

Beim ersten Start von CPAN werden einige Fragen gestellt. Die meisten Fragen kann man mit ENTER bestätigen. Man sollte jedoch den Kontinent, das Land und den zu verwendenden Server angeben. Es kann auch sein, dass alles automatisch eingerichtet wird.

Das Programm kann man unter http://github.com/zimon/zimulator beziehen (indem man auf den Link „Download Source“ klickt). Im während des Entpackens erstellten Ordner befindet sich dann das Programm mit einigen Beispieldateien.

Nun sollte man zuerst die wichtigsten Konfigurationsparameter des Programms einrichten. Dazu öffnet man in einem Editor die Konfigurationsdatei `.zimulatorrc` (sie befindet sich im gleichen Verzeichnis wie das Programm). Darin werden die wichtigsten Einstellungen festgelegt wie Pfade zu verschiedenen Programmen und Dateien. Die Konfigurationsdatei wird beim Start des Programms automatisch mit den Standardwerten erstellt, falls sie noch nicht existiert.

Eine gängige Konfiguration ist in Quelltext 13 aufgeführt (hier wird das „minimal root_fs“-Image genutzt, welches wie auf Seite 62 beschrieben „mini_fs“ genannt wurde). Der erste Abschnitt legt fest, wie aus ZVF-Dateien die Topologiebeschreibungen generiert werden. Im zweiten Abschnitt werden die Parameter für die VNUML-Aufrufe festgelegt. Der dritte Abschnitt definiert sonstige Variablen zur Steuerung des Programms. Alle Variablen sind mit Kommentaren versehen.

```
## Konfiguration fuer XML-Generierung

# Pfad zur DTD-Datei.
DTDPATH = "/usr/local/share/xml/vnuml/vnuml.dtd"

# Pfad zum oeffentlichen SSh Schluessel
SSH_KEY = "/root/.ssh/id_rsa.pub"

# Addresse des Management-Netzes
MANAGEMENT_NET = "192.168.0.0"

# Netzmaske des Management-Netzes
MANAGEMENT_NETMASK = "24"

# Offset, das auf jede IP aufaddiert wird
MANAGEMENT_NET_OFFSET = "100"

# Attribute des Tags "vm_defaults" (Standard: " exec_mode="mconsole"")
VM_DEFAULTS = "exec_mode="mconsole""

# Pfad zum VNUML-Dateisystem
FILESYSTEM = "/usr/local/share/vnuml/filesystems/mini_fs"
```

```
# Pfad zum VNUML-Kernel
KERNEL = "/usr/local/share/vnuml/kernels/linux"

# Typ der virtuellen Netze (Standard: "virtual_bridge")
NET_MODE = "virtual_bridge"

# Pfad zu Zebra im VNUML-Dateisystem (nur der Pfad ohne / am Ende)
ZEBRA_PATH = "/usr/lib/quagga"

# Pfad zu RIP im VNUML-Dateisystem (nur der Pfad ohne / am Ende)
RIPD_PATH = "/usr/lib/quagga"

# Pfad zu OSPF im VNUML-Dateisystem (nur der Pfad ohne / am Ende)
OSPF_PATH = "/usr/lib/quagga"


## Konfiguration fuer VNUML Steuerung

# Pfad zum VNUML-Programm (nur der Pfad ohne / am Ende)
VNUML_PATH = "/usr/local/bin"

# Parameter zum Starten von VNUML
VNUML_START_PARAMETERS = "-w 300 -Z -B -t"

# Parameter zum Ausfuehren von Tags
VNUML_EXEC_PARAMETERS = "-x"

# Parameter zum Beenden von VNUML
VNUML_STOP_PARAMETERS = "-P"


## Sonstige Konfigurationsvariablen

# Maximale Anzahl der Fehler bis Abbruch der Simulation,
# falls dies nicht in der Aufuehrungsbeschreibung angegeben ist
MAXFAIL_DEFAULT = "5"

# Maximale Anzahl der Durchlaeufe falls dies nicht in der
# Aufuehrungsbeschreibung angegeben ist
MAXRUN_DEFAULT = "15"

# Datei, in die VNUML-Ausgaben gespeichert werden
# "/dev/null/" um keine Logdatei zu verwenden
LOGFILE = "logfile.log"

# 1 um tcpdump-Dateien im RAW-Format zu speichern.
# 0 um sie im HEX-Format zu speichern
RAW_TCPDUMP = "0"

# 1 um die Namen der Netze in die PNG-Datei zu schreiben
VISUALIZE_NET_NAMES = "1"

# Einstellungen fuer Timeout Timer
```

```
TIMEOUT_TIMER = "180"

# Einstellungen fuer Garbagecollection Timer
GARBAGE_TIMER = "120"

# Genutzte Metrik fuer infinity
INFINITY_METRIC = "16"
```

Quelltext 13: Beispiel für die Konfigurationsdatei .zimulatorrc

Beim Aufruf des Programms kann man eine beliebige Datei als Konfigurationsdatei mit dem Parameter `-c` angeben. Alle darin enthaltenen Variablen überschreiben die Standardwerte. Wird keine Konfigurationsdatei explizit angegeben, so wird automatisch die `.zimulatorrc` genutzt, deren Inhalt ebenfalls die Standardwerte überschreiben. Das bedeutet, man braucht in die Konfigurationsdateien nur Variablen zu schreiben, die vom Standard abweichen.

A.3 Simulationen automatisieren

Im Folgenden werden die drei Dateien beschrieben, die für eine Simulation nötig sind.

Topologiebeschreibung mittels ZVF-Dateien

Die Topologie kann durch eine Datei beschrieben werden, die einfacher zu erstellen ist als VNUML-Konfigurationsdateien. Diese Topologiebeschreibung muss die Endung `.zvf` besitzen, um vom Programm erkannt zu werden.

Um aus einer ZVF-Datei eine VNUML-Konfigurationsdatei zu generieren, wird das Programm mit folgenden Parametern aufgerufen:

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

Es können auch mehrere Dateien gleichzeitig verarbeitet werden.

Das Beispiel aus Quelltext 14 generiert die VNUML-Konfigurationsdateien aus allen ZVF-Dateien im aktuellen Verzeichnis.

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

Quelltext 14: Beispiel für die Generierung einer VNUML-Konfigurationsdatei

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 fortlaufend nummeriert sein. Zum Beispiel *net1*,*net2*,... In den restlichen Zeilen wird jeweils der Name des Routers angegeben. Dieser muss mit *r* beginnen, zum Beispiel *r1* oder *r2*. Nach dem Router kommt ein Leerzeichen und dahinter eine 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 53 zu sehen ist, aus drei Knoten, welche mit drei Kanten verbunden sind. Quelltext 15 beschreibt diese Topologie in der ZVF Syntax:

```
net1,net2,net3

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

Quelltext 15: ZVF-Datei eines Dreiecks

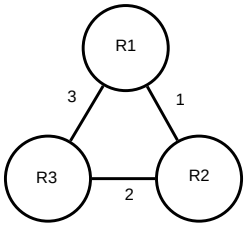

Abbildung 53: Graphische Darstellung der Dreiecks-Topologie

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 (`.zimulatorrc`, siehe auch Seite 64) gesteuert werden.

Zusätzlich kann man die Variablen `$global`, `$net` und `$router` in der Funktion `toXML()` der Datei `modules/Topologie.pm` anpassen, in der 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 insgesamt 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.

Im Folgenden sind noch einige Beispiele für Topologien aufgeführt.

Y-Topologie

Eine gängige Topologie ist die Y-Topologie (siehe Quelltext 16), welche aus vier bis fünf Routern besteht und die ungefähre Form eines Y besitzt. Die Konvergenzzeit beträgt etwa fünf bis zehn Sekunden. (Also sollte man nach dem Start der Routingdaemons mindestens 30 Sekunden warten, wenn man sicher gehen möchte, dass das Netzwerk konvergent ist.)

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

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

Quelltext 16: ZVF-Datei der Y-Topologie

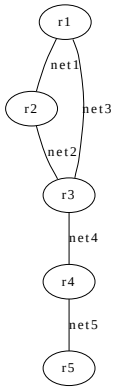

Abbildung 54: Visualisierung der Y-Topologie

Die zugehörige PNG-Datei zu dieser Topologie ist in Abbildung 54 dargestellt.

Square3 Topologie

Eine weitere, oft genutzte Topologie ist Square3 (siehe Quelltext 17). Sie besteht aus einem Quadrat von drei mal drei Routern. Die Konvergenzzeit beträgt etwa fünf bis zehn Sekunden.

```
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
```

Quelltext 17: ZVF-Datei der Topologie Square3

Abbildung 55 zeigt die Visualisierung der Square3-Topologie.

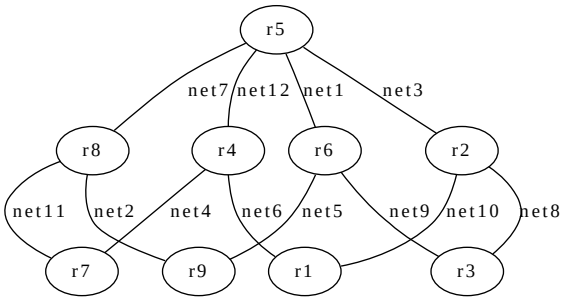

Abbildung 55: Visualisierung der Square3-Topologie

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.pm` interpretiert die Befehle nacheinander und ruft entsprechende Methoden der Klasse `modules/Scenario.pm` auf.

Beispiele für Test-Scripts befinden sich auf Seite 72.

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 den 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 einen Router oder ein 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 das der Topologie entsprechende VNUML-Szenario gestartet. Zwei 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 (`<Netzname>.dump` zum Beispiel `net1.dump`) um. Mit dem Parameter `protocol` wird der im globalen Teil definierte Routingdaemon gestartet.

Um alle drei Aktionen in der oben genannten Reihenfolge mit jeweils drei 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. (Der Routingdaemon wird zusammen mit dem Szenario 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 oder die Topologie geändert wird. Am besten probiert man aus, wie lange ein Netzwerk ungefähr braucht um den konvergenten Zustand zu erreichen, sodass die abgewartete 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. Dafür wird der entsprechende Befehl per ssh[19] an den entsprechenden Router gesendet. Als ersten Parameter erwartet `execute()` den Namen des Routers, auf dem der Befehl ausgeführt werden soll. Das zweite Argument gibt den Befehl selbst an. Dieser darf jedoch keine Kommata enthalten. 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 bestimmten Zeitpunkten die Forwardingtabelle eines Routers ausgeben zu lassen.

[19] genauer: OpenSSH – http://www.openssh.com

Das Beispiel in Quelltext 18 führt den Befehl `route -n` auf dem Router *r1* aus und hängt dessen Ausgabe an die Datei `route_r1.txt` an.

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

Quelltext 18: Beispiel für den execute()-Befehl

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, sodass man Aussagen über Konvergenzzeiten treffen kann, wenn ein beliebiger Router ausfällt (vorausgesetzt es wurden genügend Durchläufe ausgeführt).

Möchte man 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])`.

Wenn man einen Router deaktiviert, der im entsprechenden Graphen einer Artikulation entspricht, so kann es passieren, dass zwei Teilnetze entstehen, die parallel viel schneller konvergieren als das ursprüngliche große Netz. Da dies die Messergebnisse verfälschen würde, sollten solche Router ausgeschlossen werden (Artikulationen können durch die Analyse der Topologie ausfindig gemacht werden, siehe dazu Kapitel A.5).

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 sollten bei der zufälligen Auswahl des zu deaktivierenden Routers solche Router ausgeschlossen werden (Blätter können ebenfalls durch die Analyse der Topologie berechnet werden).

Um ein Device eines Routers zu deaktivieren, kann man die Nummer des Devices als zweiten Parameter 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])`
- `disable(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 ein VNUML-Szenario 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. Anschließend werden die tcpdump-Prozesse und das Routingprotokoll gestartet und 60 Sekunden abgewartet. Schließlich werden die tcpdump-Prozesse sowie der Durchlauf selbst beendet.

Quelltext 19 zeigt die zugehörige Datei `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)
```

Quelltext 19: Test-Script für eine einfache Konvergenzzeitbestimmung

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 fünf Minuten gewartet, bis das Netzwerk wieder konvergent ist und dann der Durchlauf beendet.

Die zugehörige Datei `routerausfall60\_300.cfg` ist in Quelltext 20 aufgeführt.

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

Quelltext 20: Test-Script für einen Routerausfall

Ausführungsbeschreibung

Die Ausführungsbeschreibung 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 Feld „Protokoll“ ist für zukünftige Zwecke eingeführt worden, um auch andere Protokolle als RIP testen zu können. Gleichzeitig entspricht es dem auszuführenden *execute-Tag* der VNUML-Konfigurationsdatei, welches das entsprechende Protokoll startet. 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 zum Beispiel, wenn ein Dienst nicht startet oder die errechnete Konvergenzzeit negativ ist.

Ruft man das Programm nun mit den Parametern aus Quelltext 21 auf (wobei `simulations.txt` die Ausführungsbeschreibung ist), so werden die Simulationen der Reihe nach abgearbeitet.

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

Quelltext 21: Beispiel für das Starten einer Simulation

Nach jedem Durchlauf wird die Datei erneut eingelesen und mit dekrementierter Anzahl von Durchläufen wieder zurückgeschrieben. 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) alle an der Simulation beteiligten Programme und Prozesse durch ein spezielles Script gestoppt werden und danach die mit dem abgebrochenen Durchlauf fortgesetzt werden können (siehe auch Kapitel A.3 auf Seite 74).

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

Die Ausführungsbeschreibung `simulations.txt` (siehe Quelltext 22) lässt jede Topologie in jeder Simulation zehn mal laufen. Wenn pro Simulation mehr als fünf Fehler auftreten, wird sie abgebrochen.

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

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

Quelltext 22: Beispiel einer Ausführungsbeschreibung

Überwachung und Neustart der Simulationen bei Problemen

Da es aus verschiedenen Gründen immer wieder vorkommen kann, dass VNUML hängen bleibt, wurde eine Reihe von Perl-Scripts geschrieben, welche die Simulationen überwachen und alle beteiligten Prozesse bei Problemen stoppen können. Danach werden die Simulationen genau an der abgebrochenen Stelle wieder aufgenommen.

Um diese Scripts zu benutzen, muss die Ausgabe von `zimulator.pl` in eine Datei umgeleitet werden (siehe Quelltext 23 für den entsprechenden Befehl), die regelmäßig geprüft wird.

```
./zimulator.pl -sv simulations.txt >> simulations.out
```

Quelltext 23: Starten von zimulator.pl mit Umleitung der Ausgabe

Danach startet man das Script `checkRunning.pl`, welches regelmäßig prüft, ob die Datei `simulations.out` größer wurde. Ist der Datei fünf Minuten lang nichts angehängt worden, so geht das Script von einem Problem aus und ruft das Script `restart.pl` auf. `restart.pl` ruft das Script `killsimulation.pl` auf, welches `zimulator.pl`, `vnumlparser.pl` sowie alle laufenden UML- und tcpdump-Prozesse beendet. Die LOCK-Datei im VNUML-Verzeichnis des Users wird gelöscht, damit VNUML wieder starten kann. Daraufhin wird der Ausführungsbeschreibung der Name des aktuellen Szenarios entnommen und dieses mit `vnumlparser.pl` gestoppt. Danach wird `zimulator.pl` durch `restart.pl` über den Aufruf in Quelltext 23 erneut gestartet. Es wird

die abgebrochene Simulation wiederholt und danach normal weiter gearbeitet. `checkRunning` läuft die ganze Zeit über weiter und überwacht weiterhin die Datei `simulations.out` auf Veränderungen.

A.4 Auswertung der Ergebnisse

Alle während einer Simulation generierten Dateien werden in einem eigenen Verzeichnis (dem Simulationsordner) gespeichert, welches wie folgt genannt wird: `simulationsname-topologiename`.

Dort wird auch die Ergebnisdatei abgelegt, die den Namen der Topologie 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. In Klammern steht der jeweilige Schlüssel, der im `resultHash` für den entsprechenden Wert verwendet wird (siehe Seite 88):

1. Durchmesser der getesteten Topologie
2. Anzahl der Knoten der Topologie
3. Anzahl der Blätter
4. Anzahl der inneren Knoten
5. Anzahl der Kanten
6. Clusterkoeffizient
7. Konvergenzzeit des Durchlaufs (TIMETOCONVERGENCE)
8. Anzahl der Routingpakete (TOTALPACKETCOUNT)
9. Trafficvolumen der Routingpakete (TOTALTRAFFIC)
10. Anzahl der durchschnittlichen Pakete pro Netz (AVERAGEPACKETCOUNT)
11. Durchschnittlicher Updatetraffic pro Netz (AVERAGETRAFFIC)
12. Anzahl der Pakete im Netz, über das die wenigsten Pakete versendet wurden (LEASTPACKETSNETCOUNT)
13. Nummer des Netzes, über das die wenigsten Pakete versendet wurden (LEASTPACKETSNET)
14. Anzahl der Pakete im Netz, über welches die meisten Pakete versendet wurden (MOSTPACKETSNETCOUNT)
15. Nummer des Netzes, über das die meisten Pakete versendet wurden (MOSTPACKETSNET)
16. Traffic des Netzes mit dem geringsten Trafficaufkommen (MINTRAFFICNET)
17. Nummer des Netzes mit dem geringsten Trafficaufkommen (MINTRAFFIC)

18. Traffic des Netzes mit dem höchsten Trafficaufkommen (MAXTRAFFICNET)
19. Nummer des Netzes mit dem höchsten Trafficaufkommen (MAXTRAFFIC)
20. Router oder Device, das ausgefallen ist – 0 wenn es keinen Ausfall gab (FAIL)
21. Zeit des Router- oder Deviceausfalls (FAILURETIME)
22. Name der getesteten Topologiedatei (TOPOLOGYNAME)
23. Name des getesteten Protokolls (PROTOCOL)
24. Nummer des Durchlaufs (INTERNRUNCOUNT)
25. Erster Zeitstempel (FIRSTTIMESTAMP)
26. Letzter Zeitstempel (LASTTIMESTAMP)

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. Auch hier wurde in Klammern der jeweilige Schlüssel des `avgResult`-Hashes angegeben.

1. Durchmesser der getesteten Topologie (DIAMETER)
2. Anzahl der Knoten der Topologie (VERTICES)
3. Anzahl der Blätter (LEAVES)
4. Anzahl der inneren Knoten (INNERVERTICES)
5. Anzahl der Kanten (EDGES)
6. Clusterkoeffizient (CC)
7. Längste Konvergenzzeit eines Durchlaufs (LONGESTTIME)
8. Kürzeste Konvergenzzeit eines Durchlaufs (SHORTESTTIME)
9. Durchschnittliche Konvergenzzeit aller Durchläufe (AVERAGETIME)
10. Anzahl der Pakete im Durchlauf mit den meisten Paketen (Maximum von Feld TOTALPACKETCOUNT der Ergebnisdatei) (MAXTOTALPACKETS)
11. Anzahl der Pakete im Durchlauf mit den wenigsten Paketen (Minimum von Feld TOTALPACKETCOUNT der Ergebnisdatei) (MINTOTALPACKETS)
12. Durchschnittliche Anzahl der gesendeten Pakete pro Durchlauf (AVERAGETOTALPACKETS)
13. Durchschnitt der maximalen Paketanzahlen pro Netz (Durchschnitt von Feld MOSTPACKETSNETCOUNT der Ergebnisdatei) (MAXAVERAGEPACKETS)
14. Durchschnitt der minimalen Paketanzahlen pro Netz (Durchschnitt von Feld LEASTPACKETSNETCOUNT der Ergebnisdatei) (MINAVERAGEPACKETS)
15. Durchschnitt der Pakete pro Netz (AVGAVERAGEPACKETS)
16. Höchster Gesamttraffic (MAXTOTALTRAFFIC)
17. Geringster Gesamttraffic (MINTOTALTRAFFIC)
18. Durchschnittlicher Gesamttraffic (AVERAGETOTALTRAFFIC)
19. Durchschnittlicher Traffic pro Netz (AVGAVERAGETRAFFIC)
20. Name der getesteten Topologie (TOPOLOGYNAME)

Es besteht weiterhin die Möglichkeit, tcpdump-Dateien im Nachhinein erneut auszuwerten. Dafür gibt es die Option `-r`. Mit der Option `-T` kann ein Zeitpunkt angegeben werden, 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 Option `-S` bei der statt der tcpdump-Dateien die Topologiedatei übergeben werden. 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 -o newresultfile.txt beispiel.zvf` – Wertet die Dumps aller Durchläufe aller Simulationen der Topologie „beispiel“ aus und generiert die Ergebnisdatei „newresultfile.txt“.

`./zimulator -S -F 0 -o newresultfile.txt beispiel.zvf` – 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 -T 0 -o newresultfile.txt beispiel.zvf` – 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 und 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. Wird eine Simulation mit gleicher Topologie und gleichem Protokoll noch einmal mit der gleichen Durchlauf-Nummer ausgeführt, so wird eine fortlaufende Zahl angehängt.

Beispiel:

```
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 abgebrochen wird oder im Nachhinein weitere Durchläufe ausgeführt 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 `.zimulatorrc` (siehe auch Seite 64) 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 gespeichert. 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 [Pos81] und der UDP-Header von 8 Byte [Pos80] aufzuaddieren.

Statistische Analyse der Messergebnisse

Die Ergebnisse der Simulationen können mit dem Programm gnuplot[20] 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, Knotenzahl, Kantenzahl 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

[20] http://gnuplot.info

berücksichtigt, ist in Quelltext 24 dargestellt. Dadurch wird für jeden Messwert die erste Spalte auf der x-Achse und die fünfte Spalte auf der y-Achse aufgetragen.

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

Quelltext 24: Beispiel einer minimalen Plotdatei

Für dreidimensionale Graphen wird die Funktion splot verwendet. Im nächsten Beispiel (Quelltext 25) wird die Paketanzahl in Zusammenhang mit Knotenzahl und Kantenzahl gebracht.

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

Quelltext 25: Beispiel einer minimalen 3d-Plotdatei

Methode der kleinsten Quadrate

`gnuplot` beherrscht auch die Methode der kleinsten Quadrate. Jedoch funktioniert sie nur für den eindimensionalen Fall richtig. Um diese Funktion zu verwenden, definiert man zuerst eine Funktion, die die Messwerte wahrscheinlich am besten approximiert, und lässt `gnuplot` dann mit der Funktion `fit` die Variablen berechnen. Das Beispiel aus Quelltext 26 zeigt die Approximation des linearen Zusammenhangs von Durchmesser und Konvergenzzeit. Dort sind a und b die Variablen, die berechnet werden, während x der Durchmesser und $f(x)$ die Konvergenzzeit ist.

```
f(x)=a*x+b
fit f(x) 'resultfile.txt' using 1:5 via a,b

plot f(x)
```

Quelltext 26: Methode der kleinsten Quadrate mit gnuplot

Für die Berechnung von Funktionen mit mehreren unabhängigen Variablen (multidimensionale Funktionen), wie sie in Kapitel 8.7 genutzt wurden eignet sich das Programm maxima[21] besser. Es liefert jedoch nur die Werte der approximierten Variablen sowie die Varianz. Diese Werte können jedoch für die Funktion in gnuplot genutzt werden.

Hat man maxima gestartet, so kann man die Ergebnisdatei über den Befehl `read_matrix` einlesen und als Matrix speichern. Wenn die Datei viele Zeilen hat, so kann es vorteilhaft sein, vorher die nicht benötigten Spalten zu löschen sowie die Werte auf Ganzzahlen zu runden. Als nächstes lädt man das Paket `lsquares` mit dem Befehl `load(lsquares)$`, welches Funktionen für die Berechnung der kleinsten Quadrate zur Verfügung stellt. Der nächste Befehl `lsquares_estimates` erwartet als ersten Parameter das Array, in dem die Messwerte liegen. Der zweite Parameter definiert Variablen für die einzelnen Spalten der Matrix. Im dritten Parameter wird die Funktion festgelegt, welche der Approximation zugrunde gelegt werden soll, während der vierte und letzte

[21] http://maxima.sourceforge.net

Parameter die zu berechnenden Variablen nochmals definiert. Das Ergebnis wird im Beispiel in die Variable `a` geschrieben. Das Beispiel in Quelltext 27 approximiert das Trafficvolumen über die Anzahl der Knoten und Kanten mit einem zweidimensionalen Polynom 2. Grades. Es wird für jede Variable der errechnete Wert ausgegeben und gleichzeitig in der Variable `a` gespeichert.

```
M : read_matrix("resultfile.txt");
load(lsquares)$
a : lsquares_estimates ( M, [r,s,t,u,v,w,x,y,z],
  z = A*s^2 + B*s + C*v^2 + D*v + E, [A,B,C,D,E] );
```

Quelltext 27: Methode der kleinsten Quadrate mit maxima

Um die Varianz zu berechnen, wird die Funktion `lsquares_residual_mse` genutzt. Die ersten drei Parameter der Funktion sind identisch mit denen der Funktion `lsquares_estimates`. Als vierten Parameter gibt man `first(a)` an. Ein Beispiel ist in Quelltext 28 angegeben. Die Ausgabe ist die Varianz der approximierenden Funktion mit den in `lsquares_estimates` errechneten Variablenwerten für die in der Matrix gespeicherten Werte.

```
lsquares_residual_mse ( M, [r,s,t,u,v,w,x,y,z],
  z = A*s^2 + B*s + C*v^2 + D*v + E, first(a) );
```

Quelltext 28: Berechnung der Varianz mit maxima

Quellen: [max], [gnu]

Die in dieser Arbeit genutzten Plotdateien haben noch einige weitere Einstellungen, um zum Beispiel die Achsen zu beschriften oder die Legende anders zu positionieren. Sie sind auf der beiliegenden CD enthalten oder online unter http://github.com/zimon/zimulator zu finden.

A.5 Graphentheoretische Analyse von Topologien

Mit Hilfe der CPAN-Bibliothek „Graph“ ist das Programm 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 Bibliothek `Graph` eingebunden ist).

Um eine ZVF-Datei zu analysieren ruft man das Programm mit folgender Syntax auf:

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

Dabei werden die folgenden graphentheoretischen Eigenschaften berechnet:

- Durchmesser d
- Anzahl der Knoten n
- Anzahl der inneren Knoten n_i
- Blätter
- Anzahl der Kanten m
- Artikulationen
- Clusterkoeffizient cc
- Anzahl der Kreise k (zyklomatische Zahl)

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

Die verwendeten Module haben einige Einschränkungen bei der Verwendung von Hypergraphen. Diese können zum Beispiel nicht mittels GraphViz dargestellt werden.

Hinweis: Das Programm wurde noch nicht mit Hyperkanten, deren Knotenzahl 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 das Programm GraphViz[22] berechnet. Es können die Namen der Netze an die Kanten geschrieben werden. Dies kann man in der Datei `modules/Constants.pm` festlegen.

A.6 Generierung von Topologien

Mit dem Programm können über die Option `-g` symmetrische und zufällige Topologien generiert werden. Dabei wird automatisch eine ZVF-Datei erstellt und beide Dateien unter einem automatisch generierten oder optional angegebenen Namen gespeichert. Eine graphische Darstellung der Topologie als PNG-Datei kann durch die zusätzliche Angabe des Parameters `-i` generiert werden.

Die generierten Graphen werden automatisch analysiert und deren Eigenschaften als Kommentare in die ZVF-Datei geschrieben. Die generelle Syntax zum Generieren von Topologien ist:

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

In Tabelle 4 werden die möglichen Typen von Topologien mit ihren Argumenten aufgelistet:

[22] http://www.graphviz.org/

Name	Argumente	Beschreibung
row	N	Eine Reihe aus N Routern
circle	N	Ein Kreis aus N Routern
star2	N	Ein Router, an den N-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 (siehe Kapitel 6.1 auf Seite 6.1)
crown	N	Eine CrownN-Topologie (siehe auch Kapitel 6.1 auf Seite 6.1)
circlex_rowy	X Y	Ein Kreis aus X Routern an einer Reihe aus Y Routern
random	N M	Eine zufällige Topologie aus N Routern verbunden mit M Netzen

Tabelle 4: Tabelle der generierbaren Topologieklassen

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 ist:

```
./zimulator.pl -g random N M [NAME]
```

wobei N die Anzahl der Knoten angibt, M die Anzahl der Kanten und mit NAME der Name der zu erzeugenden Dateien angegeben wird. Der Befehl aus Quelltext 29 erstellt eine zufällige Topologie mit dem automatisch generierten Namen „random9_15“ bestehend aus neun Knoten und 15 Kanten. Durch diesen Aufruf könnten zum Beispiel die Dateien `random9_15.png` (siehe Abbildung 56) und `random9_15.zvf` (siehe Quelltext 30) erzeugt werden.

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

Quelltext 29: Beispiel zur Generierung einer zufälligen Topologie

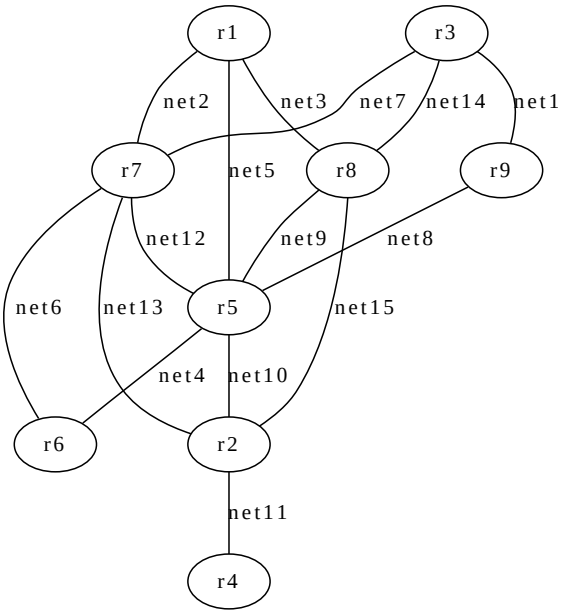

Abbildung 56: Visualisierung der erzeugten Zufallstopologie random9_15

```
# 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
```

Quelltext 30: ZVF-Datei der erzeugten Zufallstopologie

B Informationen für Entwickler

Das Programm `zimulator.pl` wurde objektorientiert in der Programmiersprache Perl geschrieben. Ein UML-Klassendiagramm mit einigen zusätzlichen Informationen ist in Abbildung 57 dargestellt. Im Folgenden werden die einzelnen Klassen und Module beschrieben.

Configuration

Die Klasse `Configuration` speichert alle wichtigen Konstanten, die auch über eine Konfigurationsdatei definiert werden können. Sie besitzt keine fest definierten Variablen. Stattdessen wird die gesamte Klasse als Hash genutzt, in dem beliebige Optionen mit den zugehörigen Werten gespeichert werden können. Dafür sind auch Getter- und Settermethoden implementiert. Ohne Angabe einer Konfigurationsdatei werden Standardwerte geladen. Wird beim Erzeugen eines Configuration-Objekts ein File-Objekt mit einer Konfigurationsdatei übergeben, so werden die Standardwerte mit den Werten der Konfigurationsdatei überschrieben. Dabei brauchen in der Konfigurationsdatei nur die Optionen definiert zu werden, die von den Standardwerten abweichen.

Die Klasse ist als *Singleton Pattern* implementiert, sodass es nur eine Instanz gibt. Diese Instanz kann von jedem Punkt im Programm über die statische Funktion `instance()` bezogen werden[23].

Utilities

Die Datei `Utilities.pm` ist ein Perl-Modul, welches statische Funktionen zur Verfügung stellt, die häufig gebraucht werden. Dies sind zum einen die Funktionen `makeTimeStamp()` und `makeTime()`, welche aus einem Zeitstring, wie ihn tcpdump liefert, einen Zeitstempel generieren und umgekehrt aus einem Zeitstempel den entsprechenden String erstellen. `getTime()` und `getTimeStamp()` geben die aktuelle Systemzeit zum Zeitpunkt des Aufrufs als Zeitstempel oder Zeitstring zurück.

Die Funktionen `getSimulationDirectory()` und `getResultFiles()` durchsuchen das aktuelle Verzeichnis nach Simulationsordnern und darin nach Ergebnisdateien. Sie liefern jeweils eine Liste der gefundenen Verzeichnisse bzw. Dateien zurück.

Die Funktion `remove()` bekommt zwei Listenreferenzen übergeben, und löscht dann jedes Vorkommen eines Elementes der zweiten Liste aus der ersten Liste. Diese Funktion wird vor allem genutzt, um Ausnahmen für zufällige Routerausfälle umzusetzen. (Die Liste der Router, die nicht ausfallen dürfen, wird von der Liste aller Router abgezogen.)

[23] Solange das Modul `Configuration` geladen wurde.

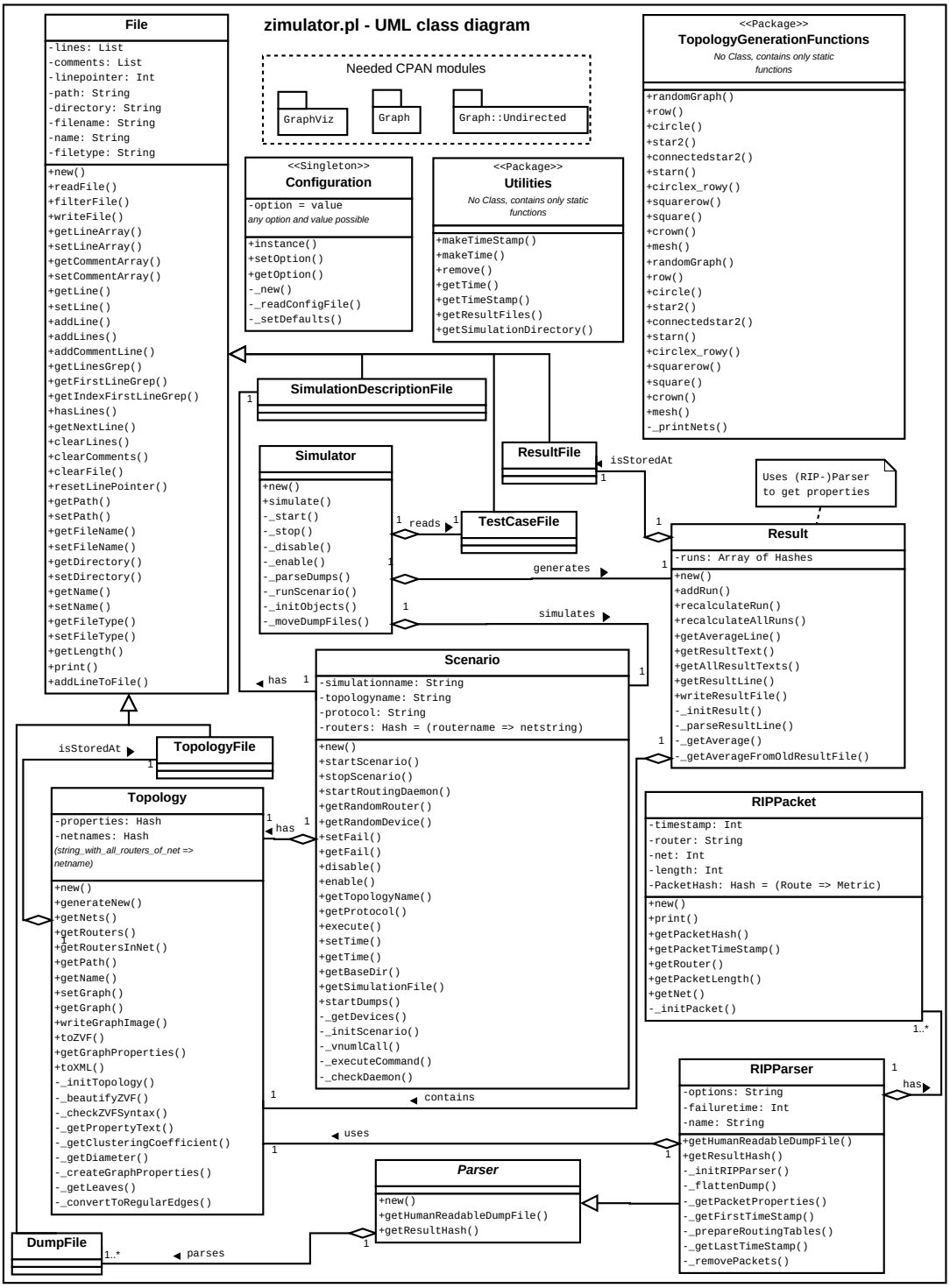

Abbildung 57: UML-Klassendiagramm

File

Die Klasse `File` repräsentiert eine Datei. Da bei allen im Programm genutzten Dateien Leerzeilen und Kommentarzeilen (welche mit einem `#` beginnen) ignoriert werden, werden zwei Listen mit Zeilen verwaltet. Die Liste `lines` beinhaltet die Nutzdaten, während in der Liste `comments` Kommentare und Leerzeilen gespeichert werden. Die Klasse `File` besitzt Methoden, um Zeilen zurückzugeben, anzuhängen und zu ändern. Es können auch reguläre Ausdrücke genutzt werden, um die erste Zeile, den Index der ersten Zeile oder alle Zeilen, die den regulären Ausdruck erfüllen, zurückzugeben. Die beiden Listen können auch komplett ausgegeben oder überschrieben werden.

Des Weiteren wurde ein *Iterator Pattern* implementiert. Dabei repräsentiert die Variable `linepointer` den Index der aktuellen Zeile. Die Methode `hasLines()` prüft, ob der `linepointer` bereits auf die letzte Zeile der Liste `lines` zeigt. Ist dies der Fall, so wird 0 zurückgegeben, ansonsten die Anzahl der restlichen Zeilen. Die Funktion `getNextLine()` gibt die aktuelle Zeile zurück und inkrementiert den `linepointer`, wenn er noch nicht auf der letzten Zeile steht. Mit der Methode `resetLinePointer()` wird der `linepointer` auf 0 gesetzt.

Der Pfad zur Datei wird in der Klasse in die Variablen `path` (kompletter Pfad mit Dateiname), `directory` (der Pfad ohne Dateiname), `filename` (der Dateiname mit Suffix), `name` (der Dateiname ohne Suffix) und `filetype` (nur das Suffix) aufgespalten. Wird einer der Werte geändert, so werden alle anderen automatisch angepasst.

Die statische Funktion `addLineToFile()` fügt eine Zeile zu der angegebenen Datei hinzu. Sie erwartet den Pfad zur Datei sowie den anzuhängenden String als Argumente. Diese Funktion kann auch ohne ein `File`-Objekt aufgerufen werden.

Von der Klasse `File` sind die Klassen `TopologyFile`, `ResultFile`, `TestCaseFile`, `ExecutionDescriptionFile` und `DumpFile` abgeleitet, welche einen Syntaxcheck für den jeweiligen Dateityp implementieren.

Topology

Die Klasse `Topology` repräsentiert eine Topologie (also einen Graphen), welcher aus einem `TopologyFile`-Objekt gelesen werden kann. Mit den Methoden `toZVF()` und `toXML()` kann der Graph wieder in ein `TopologyFile`-Objekt oder eine VNUML-XML-Datei überführt werden. Für die Datenstruktur des Graphen wird dabei das Perl-Modul `Graph` verwendet. Daneben besitzt die Klasse auch Methoden zur Berechnung von Grapheneigenschaften wie Durchmesser, Anzahl der Blätter oder Artikulationen. Für diese Berechnungen wird zum Teil auch auf Funktionen des Perl-Moduls `Graph` zurückgegriffen (siehe auch Kapitel 4.2 „Topologiebeschreibung und Analyse“ auf Seite 24). Des Weiteren existiert die private Funktion `_writeGraphImage()`, die mit dem Programm GraphViz und dem entsprechenden Perl-Modul `GraphViz`[24] den Graphen visualisieren und als PNG-Datei abspeichern kann. Diese Funktion wird von der Funktion `toZVF()` aufgerufen, wenn die Konfigurationsvariable `CREATEGRAPHIMAGE` gesetzt ist. Mit Hilfe des Moduls `TopologyGenerationFunctions` können auch verschiedene Topologien erstellt werden wie zum

[24] http://search.cpan.org/~lbrocard/GraphViz-2.04/lib/GraphViz.pm

Beispiel Row, Circle, Star, ... (siehe auch Abschnitt „TopologyGenerationFunctions“ auf Seite 87).

TopologyGenerationFunctions

Das Perl-Modul `TopologyGenerationFunctions` stellt statische Funktionen zur Verfügung, um verschiedene Topologien zu generieren. Neben vielen Topologieklassen können auch Zufallstopologien erstellt werden, bei denen entweder die Anzahl der Knoten und Kanten oder die Anzahl der Knoten sowie der Clusterkoeffizient vorgegeben sind. Eine Liste aller generierbaren Topologieklassen ist in der Bedienungsanleitung auf Seite 81 angegeben.

RIPPacket

Die Klasse `RIPPacket` entspricht einem RIP-Paket. Es werden nur die zur Berechnung der Konvergenzeigenschaften notwendigen Attribute gespeichert. Dies sind der sendende Router, der Zeitpunkt, an dem das Paket gesendet wurde, das Netz, über das das Paket gesendet wurde, die Gesamtlänge des Pakets (in Bytes) sowie ein Hash, welches jeder Route des Pakets die entsprechende Metrik zuordnet.

Die Klasse `RIPPacket` wird von der Klasse `RIPParser` benötigt. Sie kann bei der Implementierung anderer Protokollparser durch entsprechende Klassen oder Datentypen (wie zum Beispiel Hashes) ersetzt werden.

Parser

Die abstrakte Klasse `Parser` dient vor allem zur Berechnung der Konvergenzeigenschaften. Sie wird jedoch immer genutzt, wenn die gesendeten Pakete eines Routingprotokolls geparst werden müssen. Zur Erzeugung der Klasse müssen die `DumpFile`-Objekte des zu berechnenden Durchlaufs sowie die zugehörige Topologie in Form eines `Topology`-Objekts übergeben werden. Daneben wird noch der Zeitpunkt eines etwaigen Router- oder Deviceausfalls sowie eine Option zum Parsen übergeben. Je nach Parseroption wird die Konvergenzzeit bis zum oder ab dem Routerausfall berechnet. Um die Auswertung verschiedener Routingalgorithmen zu ermöglichen, muss jeweils eine Spezialisierung dieser Klasse für das entsprechende Protokoll implementiert werden. Für RIP wurde in dieser Arbeit die abgeleitete Klasse `RIPParser` implementiert.

Die beiden Methoden `getResultHash()` und `printHumanReadableDumpFile()` müssen bei jeder `Parser`-Spezialisierung implementiert sein und dienen der Berechnung der Konvergenzeigenschaften sowie der Erzeugung einer Datei in von Menschen lesbaren Format. `getResultHash()` gibt ein Hash zurück, welches alle Konvergenzeigenschaften des berechneten Durchlaufs enthält. Dieses `resultHash` wird im Abschnitt „Result“ auf Seite 88 beschrieben. Das `File`-Objekt, welches von `printHumanReadableDumpFile()` zurückgegeben wird, enthält alle wichtigen Informationen der einzelnen Pakete, die chronologisch sortiert sind.

RIPParser

Die von `Parser` abgeleitete Klasse `RIPParser` implementiert die Funktionen `getResultHash()` und `printHumanReadableDumpFile()` für das RIP-Protokoll.

Zum Generieren der `RIPPacket`-Objekte werden die `DumpFile`-Objekte mit den privaten Funktionen `_flattenDump()` und `_getPacketProperties()` genutzt. Aus den `DumpFile`-Objekten wird mittels dieser Funktionen eine Liste von `RIPPacket`-Objekten erstellt, die chronologisch sortiert sind.

Zur Berechnung der Konvergenzeigenschaften werden die privaten Funktionen `_getFirstTimeStamp()` und `_getLastTimeStamp()` implementiert, welche den Anfangs- und Endzeitpunkt der Konvergenzzeit berechnen. Danach werden mittels der privaten Funktion `_removePackets()` alle Pakete zurückgeliefert, welche innerhalb der beiden errechneten Zeitpunkte versendet wurden, um daraus die Anzahl der Pakete sowie den Updatetraffic zu berechnen.

Result

Die Klasse `Result` verwaltet die Ergebnisse einer Simulation. Ein solches Objekt kann entweder durch ein `Simulator`-Objekt erzeugt werden, um eine gerade beendete Simulation auszuwerten, oder vom Programm selbst, um die Konvergenzeigenschaften einer Simulation (oder eines Durchlaufs) neu zu berechnen. Zur Berechnung von Konvergenzeigenschaften wird kurzzeitig ein Parser-Objekt erzeugt, welches die Berechnungen für das jeweilige Protokoll durchführt.

Zu jedem Durchlauf wird ein `resultHash` gespeichert, welches von der entsprechenden `Parser`-Spezialisierung geliefert wird.

Die Funktion `_getAverage()` generiert aus mehreren Durchläufen ein `averageHash`, welches ähnlich wie das `resultHash` die Konvergenzeigenschaften einer Simulation enthält. Hier wird jedoch der Durchschnitt aus mehreren Durchläufen errechnet. Die Funktion kann auch mit verschiedenen Topologien innerhalb der dem `Result`-Objekt übergebenen `resultFile`-Objekt umgehen. In diesem Fall werden alle Topologien zusammengefasst. Die Funktion `getAverageLine()` gibt die Durchschnittswerte in je einer Zeile pro Topologie in einem von gnuplot lesbaren Format zurück.

Die Hashes werden in der Bedienungsanleitung auf Seite 75 genauer beschrieben.

Scenario

Die Klasse `Scenario` beinhaltet alle Informationen einer Messung und Methoden, um diese mittels VNUML durchzuführen. Dazu gehören ein `Topology`-Objekt und ein `SimulationDescriptionFile`-Objekt.

Ein `Scenario`-Objekt enthält alle Methoden, die während der Online-Phase eines Durchlaufs das VNUML-Netzwerk sowie die tcpdump-Prozesse steuern. Mit der privaten Methode `_getDevices()`

werden alle Devices des VNUML-Netzwerks mittels des UNIX-Befehls `ifconfig`[25] und regulären Ausdrücken herausgesucht, damit die Funktion `startDumps()` für jedes Device einen tcpdump-Prozess starten kann.

Des Weiteren steht für jeden Befehl, der in Test-Scripten genutzt werden kann, eine Funktion zur Verfügung, die diesen Befehl implementiert. Diese Funktionen werden durch ein `Simulator`-Objekt aufgerufen, welches das aktuelle Test-Script parst.

Simulator

Eine Simulation wird durch die Klasse `Simulator` repräsentiert.

Diese bekommt ein `ExecutionDescription`-Objekt übergeben, in dem aufgeführt ist, welche Topologien mit welchen Test-Scripten wie oft simuliert werden sollen. Es wird ein `Scenario`-Objekt mit der entsprechenden Topologie und dem Test-Script in Form eines `TestCaseFile`-Objekts erzeugt, um einen Durchlauf zu starten. Das Test-Script wird Zeile für Zeile gelesen und die jeweiligen Befehle durch Methodenaufrufe des `Scenario`-Objekts ausgeführt. Nach beendeter Simulation wird ein `Result`-Objekt erzeugt. Darin werden mit Hilfe eines `RIPParser`-Objektes die tcpdump-Dateien (in Form von `DumpFile`-Objekten) geparst und in `RIPPacket`-Objekte überführt. Die Konvergenzzeit wird schließlich im `RIPParser`-Objekt anhand einer chronologisch sortierten Liste der `RIPPacket`-Ojekte ermittelt.

Die Klasse `Simulator` ist so angelegt, dass Simulation und Auswertung später auch durch Threads parallelisiert werden können. Für diese Arbeit war eine Parallelisierung jedoch nicht erforderlich, sodass sie auch noch nicht implementiert wurde.

[25] Manual für ifconfig: http://linux.die.net/man/8/ifconfig

C Wissenschaftliche Publikation

Auf der Grundlage dieser Arbeit und mit der in Kapitel 4 vorgestellten Testumgebung wurde ein Artikel für die 9. International Conference on Networking[26] (ICN) 2010, Les Menuires, Frankreich verfasst, der hier angehängt ist.

Darin werden die Konvergenzeigenschaften von RIP mit einer an der Universität Koblenz entwickelten Erweiterung von RIP namens RMTI verglichen. Besonderes Augenmerk liegt dabei auf dem *Counting-to-Infinity*-Problem, welches ein bekanntes Problem von Distanzvektoralgorithmen ist und im Artikel näher erläutert wird.

Die in dem Artikel abweichenden Werte für die Konvergenzzeit bei einem Coldstart sind recht schwer nachzuvollziehen, da einerseits die Steigung der Funktion in dieser Arbeit geringer ist, während jedoch das Offset der Y-Achse wesentlich größer ist. Diese Werte könnten zum Teil darauf zurückgeführt werden, dass für den Artikel die Simulationen mit einem Update Timer von 30 Sekunden durchgeführt wurden, während er für diese Diplomarbeit auf 10 Sekunden eingestellt war (siehe Kapitel 2). Da Triggered Updates zurückgehalten werden, wenn ein Periodisches Update bevor steht, wurden bei einem Update Timer von 10 Sekunden mehr Updates zurück gehalten als bei der Standardeinstellung von 30 Sekunden. Somit brauchten die Netzwerke in dieser Arbeit länger zum Konvergieren als die Netzwerke im Artikel. Auf der anderen Seite wurde festgestellt, dass bei einem Update Timer von 10 Sekunden einige Netzwerke die letzte Route, die zur Konvergenz benötigt wurde, erst mit dem ersten Periodischen Update versendeten[27]. Wenn man davon ausgeht, dass es noch länger gedauert hätte, bis diese Route ohne ein Periodisches Update nach 10 Sekunden versendet worden wäre, könnte dies eine höhere Konvergenzzeit zur Folge haben. Ob diese Effekte die abweichenden Konvergenzeigenschaften ausreichend erklären können, müsste in weiteren Arbeiten noch geklärt werden. Hier würde eine solche Untersuchung zu weit führen.

[26] IARIA ICN Conference 2010, http://www.iaria.org/conferences2010/ICN10.html (Online; Stand 31. März 2010)

[27] Dies könnte ein Fehler in der Implementation des Quagga RIP Daemon sein.

Statistical Convergence Analysis of Routing Algorithms

Frank Bohdanowicz, Marcel Jakobs, Christoph Steigner

University of Koblenz
Germany
{bohdan,zimon,steigner}@uni-koblenz.de

***Abstract*—Whenever new and better routing algorithms are developed a comprehensive convergence analysis can show the real achievements of new algorithms. This paper presents a new approach for a convergence analysis together with a new distance vector routing algorithm, which is no longer affected by the well-known Counting-to-Infinity (CTI) problem. The RMTI algorithm uses event-triggered updates instead of time-periodic updates in order to speed up convergence time and to reduce update traffic. Convergence properties of RMTI are compared with RIPv2 under the impeded condition of provoked CTI-situations caused by link failures in order to show the difference of RMTI to common RIPv2 algorithms. The major focus of this paper is the description of a test environment and the approaches used to measure the convergence time of the routing algorithms. Special effort is directed at minimizing measurement perturbations by separation of measurement and evaluation tasks into online data capturing and offline statistical analysis. The results show that this convergence measurement method is a universally valid method and that RMTI is a newly competitive intra domain routing algorithm which can - in contrast to others - execute filtering policies.**

***Keywords*-Convergence Analysis, Performance Comparison, Routing Algorithms, RMTI**

I. Introduction

The newly developed *Routing with Metric-based Topology Investigation (RMTI)* algorithm [1] can recognize routing loops and omit the Counting-to-Infinity (CTI) problem. RMTI accelerates the convergence time and reduces the update traffic by not using time-periodic updates but only event-triggered updates and neighbor alive notifications.

A test environment has been designed for routing algorithms and a new technique has been developed to measure the convergence time in order to show the improved abilities of RMTI. The test environment consists of the following components:

- A computer network based on universally applicable virtual linux machines.
- An online data capturing facility to collect characteristic data.
- A script-based event generator capable of triggering failure events.
- A topology generator able to generate many different but regularly structured topology classes and random topologies with pre-defined constraints.
- An offline statistical analysis tool to visualize the results using plots and graphs.

With these tools network failures and CTIs can be triggered. The exchanged update traffic can be recorded and the sequence of events starting with the first network failure up to the end of a convergence process can be pursued. A major concern of this research was to include the CTI-problem of common RIP algorithms into the convergence analysis.

A convergent state in a distributed routing system is given when all forwarding tables in the routers contain the optimal next hop entries for a given network topology. The convergent state cannot be reached if the forwarding tables do not offer the optimal next hop entries due to a network failure.

This paper is structured as follows: In Section 2, we mention other approaches of convergence tests. In Section 3, we introduce the test environment. In Section 4, we present an overview about the topologies and routing algorithms tested so far. In Section 5, we describe the convergence measurement technique and discuss the results of the convergence analysis. We finish with a conclusion in Section 6.

II. Related Work

Newly developed routing protocols or enhancements to existing routing protocols are usually tested in simulation test environments. These simulation test environments only analyze a certain part of the respective protocol in order to reveal its benefit. The authors of [5] and [13] analyze their protocol enhancements by simulation techniques. The test environment described in this paper is not based on simulation, but on emulation technique. It can analyze routing software which is fully implemented and ready to be used, e.g., in enterprise networks. The developers of *Virtual Network User Mode Linux (VNUML)*[9] Counting-to-Infinity want their test environment to be used in this way [12]. However, the bare VNUML test environment cannot be managed automatically by scripts in order to generate a lot of network topologies, to collect large amounts of data, and to provide useful statistics. Currently, there seems to be hardly any research about convergence analysis in deployed networks. Therefore, a new test environment is presented in the next section that provides this refined approach.

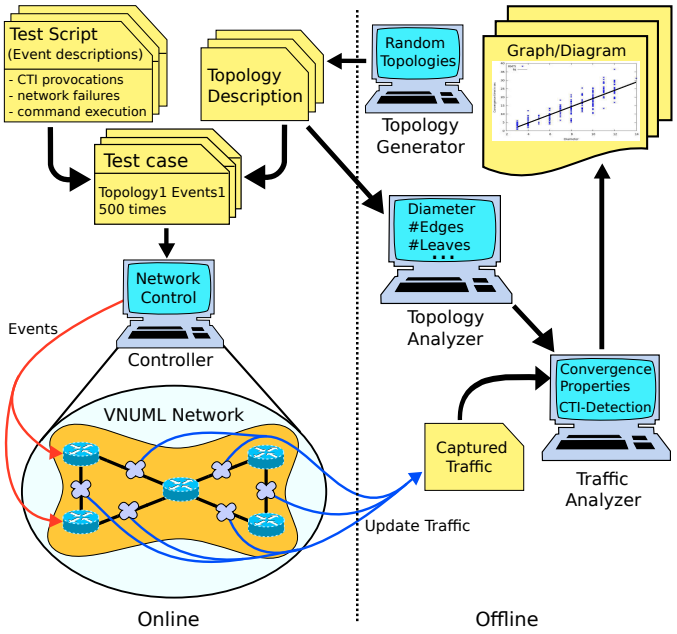

Fig. 1. Test Environment

III. Test Environment

To analyze convergence properties of routing algorithms, a test environment is needed in order to provide measurement results that are as close to reality as possible.

This new test environment is based on virtual machines using the well-known *user mode linux (UML)* [3] virtualization software. UML allows to test usable routing protocol implementations within a fully applicable linux network. The tested routing protocol implementations are part of the widely used Quagga Routing Software Suite [7]. The RMTI algorithm is implemented as a re-engineered version of the Quagga RIP router.

VNUML [9] is used to manage the UML test networks within the test environment. The test environment can carry out up to 1000 and more test runs where each run can be fully controlled and analyzed. For provoking certain topology changes, such as network failures, an additional test script is used to trigger pre-determined events.

During a test run all characteristic actions and changes are saved with a corresponding timestamp for later synchronization. When the test phase is finished the captured data is evaluated by statistic analyses. However, separating the capturing of data during an online phase and the evaluation afterwards in an offline phase guarantees that the evaluation process does not affect the test runs and the results become more accurate.

The test environment is developed for the purpose of evaluating complex and comprehensive test runs without any human interaction. The environment has to be configurable with many pre-defined test runs which cover many well-known problematic network situations. Each step of a test run can be pre-determined precisely by the test script. All kinds of test cases including random or previously defined device and router failures can be performed. All sorts of UNIX or router commands can be sent to each router at defined points in time. The output of these commands can be saved together with a timestamp in a logfile for later analysis. To be able to test various failure scenarios that could occur in networks, the test environment allows for the opportunity to cause packet losses or transmission delays for single or multiple devices of one or more routers. Being able to execute any UNIX command on each router or host, any kind of UNIX tool for traffic generation can be used. In order to expose the distance vector routing algorithms to the critical conditions of Counting-to-Infinity situations the test environment can cause these situations by retaining certain update messages on certain router ports. To be even more flexible, the network topologies to be tested are saved separately from the test scripts in topology description files. A test case now consists of a test script along with a topology description and a further specification as to how often this test case should be run. The test and analysis process can be performed without any human interaction and in an arbitrarily reproduceable manner.

The high degree of automation of the test environment allows to gather a great deal of result data that can be statistically analyzed. Together with the network environment for the test cases, very detailed insight into the behavior of common routing algorithms can be obtained. The overall results of the test runs can be visualized as diagrams. With this approach the relationship of the various test parameters as well as a performance comparison between the different routing algorithms can be revealed. The evaluation of these results enables the user to depict these relationships as mathematical functions (Fig. 3). These functions are approximated using regression analysis. Additionally, it supports the representation of frequency distributions as diagrams such as in Fig. 5. The test environment can average the results before analyzing them with statistical methods. It can create a network graph of the network topology even if the topology is randomly generated.

Before the test starts, topology properties of the network like diameter, articulations, leaves and number of nodes, edges and circles [4] are analyzed offline and saved in the topology description (Fig. 1). During the runtime of the test, all characteristic events are saved as routing update packets with a timestamp for the offline analysis after the run. After the runtime the convergence properties, such as convergence time or traffic amount, are automatically calculated with the saved update traffic and information about the topology tested.

As the test environment is modularly designed it can be used for the comparison and testing of other intra domain routing algorithms as well. For this purpose, only the specifications for the offline analysis module have to be modified[1].

A. Detailed description of test and measurement procedure

To measure the convergence time of a certain network, the network has to be changed from a convergent state to an in-

[1]This can be done by identifying the update packet that leads to the convergence of the network

convergent state by changing the status of certain links or routers in order to cause a link or router failure.

In order to calculate convergence properties of the network after a test run, the protocol analyzer *tcpdump* [10] is used to capture the existing network traffic. To get proper results the routing daemons in the routers have to reach a convergent state before the status of certain links or routers is changed to cause a failure. Then the sequence of state changes is recorded starting with the failure state and ending with the next convergent state. When the network has reached its convergent state, the run is stopped automatically and the next run follows.

After the online capturing of the router states is completed, the offline evaluation of the states can be executed. To evaluate the measurement, the captured update traffic has to be treated just in the same way as the routing algorithms do in order to calculate the forwarding table entries. That means that the offline evaluation module repeats the processing steps of the routing algorithm.

The timestamped update packets of all subnets are sorted chronologically for evaluation. To identify the starting point of the convergence time t_{conv} the first packet with the information about an unreachable subnet is selected and its timestamp saved for later use (as first timestamp t_f). As well, the timestamp of the first update packet can be taken as first timestamp to analyze the coldstart behaviors of a routing algorithm.

In order to find the timestamp of the update packet leading to convergence (the last timestamp t_l), all update packets are re-processed from the start.

Whenever a routing table is updated, the timestamp of the appropriate update packet is saved. As the network must be in a convergent state when no routing table is updated any more, the last timestamp t_l must stem from the update packet that led to this state. This timestamp is chosen for convergence time calculation.

The convergence time t_{conv} is now defined as:

$$t_{conv} = t_l - t_f \text{ with:} \tag{1}$$

t_{conv} = convergence time,
t_l = last timestamp,
t_f = first timestamp

Adding up the packet sizes of the whole traffic sent between the first timestamp t_f and the last timestamp t_l shows the traffic volume produced by routing updates during the convergence procedure.

IV. Routing Algorithms and Topologies

Within computer networks, routers are special nodes which provide the whole network with information about the location of subnets. In IP networks the internet protocol (IP) is used to identify the interfaces of the source and destination nodes. The IP protocol uses packet forwarding to deliver data packets from source via node to node until the destination is reached. Every router has to know which adjacent router is closer to the subnet of the destination node. A router maintains a forwarding table listing the subnets in connection with the corresponding next hop router and the metric which is a distance. In intra-domain routing the best route to a destination subnet is usually the one with the shortest distance. Routing is the process of completing and maintaining an accurate forwarding table. Routing is a distributed network-wide process provided by a routing protocol which offers some favorable properties such as dynamic adaptation to network changes, scalability and system stability. At any time, a forwarding table of a router must correctly describe the location of subnets in order to prevent misguided data packets. There is always a small time gap beginning with a network failure, i.e. a subnet disappears due to a link or router failure, and ending with the detection of the failure by the router. Keeping the network free of invalid routing information during this time is a challenge every routing protocol has to cope with. It is possible that routing updates between routers become invalid on their way through the network and forwarding loops can occur. In practice, all common routing protocols are affected by transient forwarding loops [11]. While a forwarding loop corresponds to a loop in the forwarding process of data packets, a routing loop corresponds to a loop in the routing process caused by circulating routing update packets. A routing loop is in close connection with a forwarding loop because the routing process has a direct impact on the forwarding process.

A. RMTI optimizations

In distance vector routing the Counting-to-Infinity problem (CTI) reflects a routing loop. The Routing Information Protocol (RIP) as a classical representative of the distance vector protocol family may get into an unstable state as soon as a CTI occurs. The protocol design of RIP is restricted in some ways to address the CTI problem, i.e. the diameter of a RIP network is limited to a metric of 15 which means there can be only 15 RIP routers in a row. Metric 16 is called *infinity* and marks a route as unreachable. The RMTI protocol avoids CTIs, routing loops and forwarding loops. RMTI is not limited by the design specifications of RIP although it can be compatible with RIP. The RMTI protocol and its enhancements towards RIP are described in [1], [2], [8]. The RMTI algorithm is an optimization that affects convergence time and update traffic. The update traffic is reduced by sending small neighbor-alive-checks instead of periodic routing updates. An incoming neighbor-alive-check confirms all routes received from this neighbor which are confirmed at once. Changes in the network will be advertised by event-triggered updates which are already part of the standard RIPv2 specification [6]. The neighbor alive check has an invariably small size in contrast to the ordinary periodic routing update whose size depends on the amount of subnets in the network. Moreover, due to the smaller neighbor alive checks either

- the entire routing update traffic can be reduced or
- the sending interval and thus the convergence time of the network can be reduced.

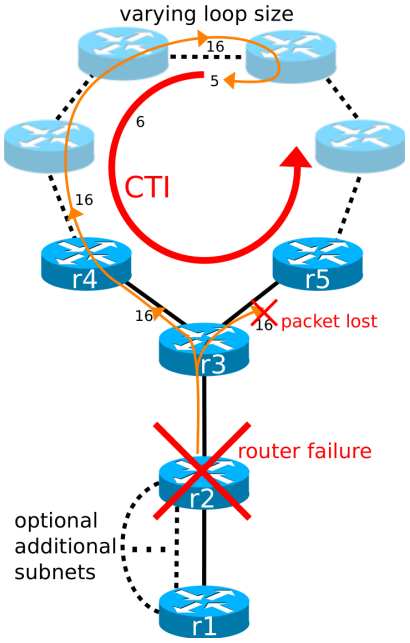

Fig. 2. Y-Topology class

There are two possibilities to optimize RMTI either to have less update traffic or to reduce the convergence time.

B. Tested Topologies

A simple network topology in which the CTI problem can easily occur is shown in Fig. 2. This topology consists of a topology loop and a row formed of routers $r1$, $r2$ and $r3$. Router $r3$ is the node connecting the topology loop with the row. The respective subnets are peripherally located outside the topology loop connected to router $r1$ and $r2$. Routing information about these subnets can flow into a CTI sequence within the topology loop after a link failure between router $r2$ and $r3$.

As being depicted in Fig. 2, a CTI can occur if the information of the unreachable subnet will be lost or just simply delayed in one path of the topology loop. If the link between router $r2$ and $r3$ fails, the subnets located between $r2$ and $r1$ are unreachable for $r3$. Router $r3$ does not receive any more updates and the timeout timer for these subnets expires. The routes in $r3$'s forwarding table are marked as unreachable with metric infinity. Then, router $r3$ sends a triggered update with metric infinity to router $r4$ and $r5$ and advertises this new routing information. Router $r4$ forwards the information about the unreachable subnets to the next router. In Fig. 2 router $r5$ does not receive the routing information from $r3$ due to a transmission failure and the routers behind $r5$ in the topology loop do not receive the information about the unreachable subnets. At this point, the old invalid routing information prevails over the current routing one. The routers which have lost their route to the subnets accept the old routing information as seemingly new alternative routes. The well-known *Split Horizon Hack* cannot avoid the CTI situation. *Split Horizon* is an extension to RIP which prevents a router from

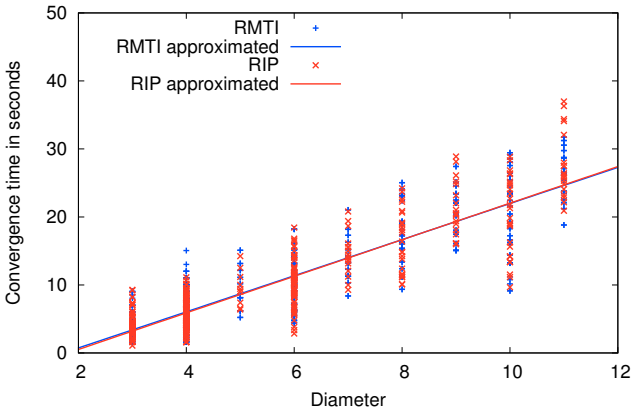

Fig. 3. Convergence time for RIP and RMTI at coldstart

advertising routing information back to the router from which it was received. Router $r4$ got the invalid routing information indirectly from router $r5$ and sends the information to $r3$. Router $r3$ accepts the invalid routing information from router $r4$ as seemingly new and valid alternative route to the subnets via $r4$. Then the CTI sequence starts. As long as the CTI occurs, a malicious routing loop is established.

V. Convergence Measurement

In this section, the measurement methods and the results of the convergence properties of the RIP and RMTI algorithms are presented.

A. General Convergence Properties

In order to compare different routing algorithms it is crucial to know how they behave in general. Coldstart tests are performed to measure convergence time and update traffic volume from the point in time where all router daemons are started until the network reaches the convergent state for the first time. For this test case no CTI is provoked as the CTI situation is a special case. This test scenario gives us the propagation speed of the routing updates and this is the worst case scenario for convergence time and traffic.

These measurements covered over 300 (mostly random generated) topologies showed that the convergence time for both algorithms mainly depends on the network topologies diameter d. As shown in Fig. 3 the convergence time increases in this test environment by a factor of 2.5 along with an increasing diameter d. However, the update traffic volume of both algorithms increases with polynomial growth with the number of subnets in the network. (Fig. 4)

Test cases show that RMTI and RIP reached their convergent state at the same time (Fig. 3). If the time-optimized RMTI (Section IV-A) is used, it does not affect the convergence time because the time benefits correspond to the detection of a topology change. With traffic improvement the RMTI needs 10 percent less traffic than RIP [2] (Fig. 4). The

[2]To achieve this result, the first 60 seconds from coldstart are measured instead of stopping the measurement at the convergent state.

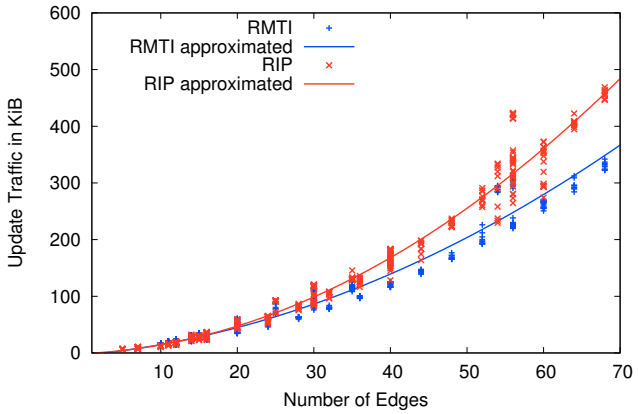

Fig. 4. Traffic comparison of optimized RMTI and RIP at coldstart

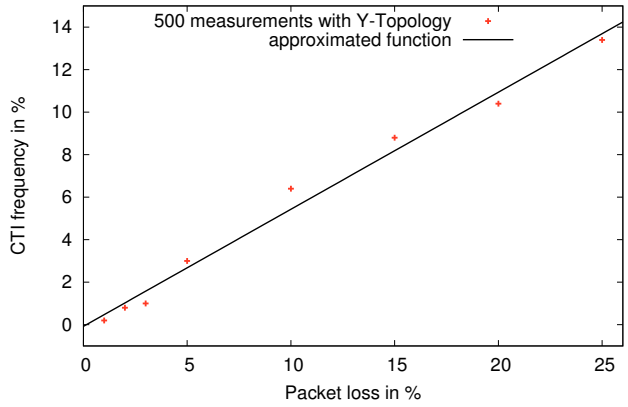

Fig. 5. CTI frequency depending on Packet loss

traffic volume for the time-optimized RMTI is the same as for RIP.

The analysis of the measurements show that an even better equation for the approximation of convergence time t_{approx} is achievable. Besides the network diameter d the number of nodes n in the network topology require to add the product $0.16 \cdot n$ to the convergence time. The variance is halved from 7.28 to 3.56. Now the approximated convergence time t_{approx} from coldstart can be calculated for every network with the following Equation (2):[3]

$$t_{approx} = 2.5 \cdot d + 0.16 \cdot n - 5 \pm 1.89 \text{ [seconds] with:} \quad (2)$$

t_{approx} = convergence time,
$d > 1$ = diameter of network topology ,
n = Number of nodes in the network topology

B. Importance of CTI avoidance

Despite the *Split Horizon* algorithm, the CTIs can occur under certain circumstances, when the network contains loops and links are flapping or routing updates are lost. To show the importance of CTI avoidance for distance vector algorithms, some test runs were performed that show the CTI frequency for links with different packet losses.

As CTIs can easily occur in Y-Topologies (Section IV-B) these were tested with a loop size of 3 where a failure was provoked on one subnet. Is the failure update of this subnet propagated to either upper router $r4$ or $r5$ (Fi. 2), a CTI occurs. The bigger the loop size, the more probable is the occurrence of a CTI. This test case is a best case scenario because there are more subnets to pass and more possibilities to lose the information about the failed subnet in bigger loops.

As the results illustrate in Fig. 5, the CTI frequency F_{CTI} increases with the routing update packet loss L_p (in percent) in a linear way.

The general rule is:

$$F_{CTI} = 0.55 \cdot L_p \pm 0.62 \text{ [\% (absolute)] with:} \quad (3)$$

[3]The values may differ a bit depending on the test environment

F_{CTI} = CTI frequency in percent,
L_p = packet loss in percent

That means with an update packet loss L_p of 0.01%, a CTI is provoked with a probability of 0.0055% or averaged at every 180st failing subnet.

C. CTI Duration

The impacts of CTI situations become apparent by examining their duration. As long as the CTI situation lasts, all payload traffic for the corrupted subnet is cycling in the loop. This is a burden to all routers and subnets located in the loop.

Due to the fact that RMTI can detect routing loops and avoid CTI occurrences it eliminates this burden. To quantify these benefits the convergence time of RIP and RMTI was measured in case of a CTI situation. Since CTIs only occur in loops several Y-Topologies with different loop sizes (Fig. 2) were tested.

To provoke a CTI router $r2$ (Fig. 2) is shut down. This leads to an unreachability of a certain subnet. Router $r3$ is forced to propagate the new infinity metric of the unreachable subnet only to one of its neighbors ($r4$ or $r5$) so that this kind of update transfer must result in a CTI.

For each topology tested the time was measured starting when a router detected a failing subnet and ending when the network was convergent again. Hence, these results are independent of the RMTI optimization (time or traffic - see Section IV-A), because time optimization only speeds up the topology change detection.

The result graph (Fig. 6) shows that the benefit of RMTI is between 30 and 120 seconds. The average CTI duration with increasing loop size increases with a factor of 18.6 for RMTI. That is nearly one fourth of RIP's growth factor which is 74. The average CTI duration of all tested topologies with RMTI only needs 30% of the average RIP CTI duration. The irregularities (Fig. 6) of the RIP results are generated when the update timer of one router expires and affects the CTI in reaching infinity (metric 16). This only happens for certain loopsizes together with specific timer settings. For these tests the standard RIP update timer of 30 seconds was used. With

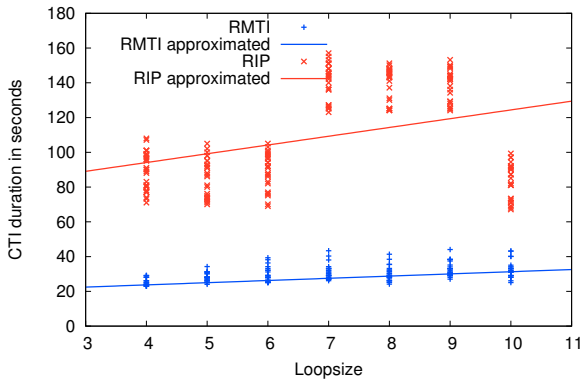

Fig. 6. CTI duration

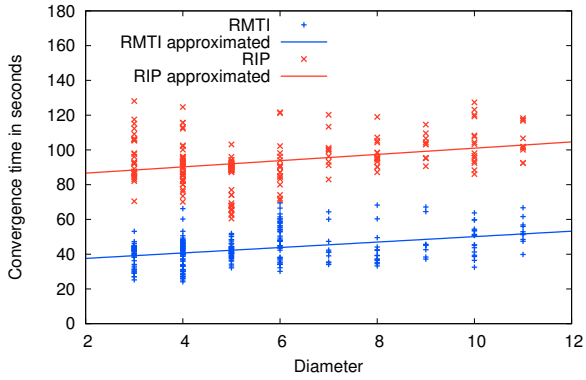

Fig. 7. Convergence time for random router failures

other timer settings this behavior occurs with other loop sizes. RMTI can converge so much faster because RMTI stops the CTI at router $r3$ (Fig. 2).

Starting the measurement with the topology change, RIP would converge even slower because of the timeout timer.

D. Common Convergence Behavior

As the most common topology changes do not provoke CTIs (Fig. 5), the convergence properties were also measured in case a router fails without CTI provocation. Therefore, a random router was shut down in each test run. As shown in Fig. 7 the RMTI algorithm with time optimization (Section IV-A) converges 50 seconds faster on average than the normal RIP from the time point where the topology change happens. With a traffic-optimized RMTI the convergence time is the same, but there is an update traffic reduction.

VI. Conclusion and Subsequent Work

To demonstrate the advantages of new routing algorithms in contrast to common intra domain routing algorithms, comprehensive tests have to be carried out. In practice, a variety of network topologies and a succession of critical events have to be automatically generated by random generators. All known critical test situations concerning the capability of a routing algorithm to reorganize itself after a topology change have to be covered.

With the newly developed test environment one can show the benefits of the new RMTI algorithm in comparison to RIP. The flexible design of the test environment allows us to test RIP, RMTI and other intra domain routing protocols. The RMTI algorithm shows two important advantages: the possibility to avoid CTI situations and to converge much faster than other distance vector algorithms in case of a topology change. The ability of RMTI to choose whether to optimize a fast topology change detection or a traffic reduction (or a mixture of both) makes the test environment adaptable to the specific needs of many different networks. With the VNUML script it is even possible to run larger networks on computer clusters which will support the results for those networks. The high degree of automating the test environment and the ability to gather large amounts of measurement data is a big advantage for analyzing all kinds of routing protocols. Another issue that we will deal with in our future research is the performance analysis of the RMTI algorithm with payload traffic and the comparison with other intra domain routing protocols.

We showed that we can obtain a variety of characteristic routing parameters and various relationships between them.

References

[1] F. Bohdanowicz, H. Dickel, and Ch. Steigner, *Routing with metric-based Topology Investigation*, International Journal on Advances in Internet Technology, 2009, vol 2 no 1

[2] F. Bohdanowicz, H. Dickel, and Ch. Steigner, *Metric-based Topology Investigation*, in Proceedings of the Eighth International Conference on Networking (ICN 2009), Cancun, Mexico, 2009.

[3] J. Dike, *User Mode Linux*, Prentice Hall, 2006.

[4] C. Berge, *Graphs and Hypergraphs*, North-Holland Publishing Company, 1973.

[5] K. Levchenko, G. M. Voelker, R. Paturi, and S. Savage , *XL: An Efficient Network Routing Algorithm*, Proc. Sigcomm 2008, August, 2008.

[6] G. Malkin, *RIP Version 2*, RFC 2453, 1998, URL: http://tools.ietf.org/html/rfc2453, 05.08.2009.

[7] Quagga home page, http://www.quagga.net/, 19.11.2009.

[8] A. Schmid and Ch. Steigner, *Avoiding Counting to Infinity in Distance Vector Routing*, Telecommunication Systems 19 (3-4): 497-514, March - April, 2002, Kluwer Academic Publishers.

[9] VNUML project home page, http://www.dit.upm.es/vnumlwiki, 19.11.2009, Technical University of Madrid (UPM).

[10] tcpdump project home page, http://www.tcpdump.org/, 19.11.2009.

[11] Z. Zhong, R. Keralapura, S. Nelakuditi, Y. Yu, J. Wang, C. Chuah and S. Lee, *Avoiding Transient Loops Through Interface-Specific Forwarding*, Transactions on Networking, IEEE/ACM, Volume: 15, Issue: 6, Dec. 2007 Transactions on Networking, Dec. 2007

[12] F. Galn, D. Fernndez, W. Fuertes, M. Gmez and J. Lpez de Vergara, "Scenario-based Virtual Network Infrastructure Management in Research and Educational Testbeds with VNUML: Application Cases and Current Challenges", Annals of Telecommunications, special issue on Virtualization: a path for the future Internet, vol. 64(5), pp. 305-323, May 2009

[13] B. Premore, An Experimental Analysis of BGP Convergence Time, ICNP '01: Proceedings of the Ninth International Conference on Network Protocols, 2001, IEEE Computer Society

D Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Quelltextverzeichnis

Quellenverzeichnis

[Bel57] Bellman, Richard E.: *Dynamic Programming*. Princeton University Press, 1957.

[Ber73] Berge, Claude: *Graphs and Hypergraphs*. Noth-Holland Publishing Company, 1973.

[Boh08] Bohdanowicz, Frank: *Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons*. Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, Mai 2008. http://kola.opus.hbz-nrw.de/volltexte/2008/305/pdf/Diplomarbeit_Frank_Bohdanowicz.pdf, [Online; Stand 09. Januar 2010].

[Deo04] Deo, Narsingh: *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall of India Pvt.Ltd, 2004.

[Die06] Diestel, Reinhard: *Graphentheorie*. Springer-Verlag Heidelberg, 2006.

[FF62] Ford, Lester R. und Delbert R. Fulkerson: *Flows in Networks*. Princeton University Press, 1962.

[FKK72] Frank, Howard, Robert E. Kahn und Leonard Kleinrock: *Computer communication network design - Experience with theory and practice*, 1972.

[GFR+04] Galán, Fermín, David Fernández, Javier Rúiz, Omar Walid und Tomás De Miguel: *A Virtualization Tool in Computer Network Laboratories*. In: *ITHET'04*, Mai 2004.

[gnu] *Gnuplot Manual - Fit Command*. http://www.gnuplot.info/docs/node82.html, [Online; Stand 08. Januar 2010].

[GS07] Gunes, Mehmet H. und Kamil Sarac: *Inferring Subnets in Router-level Topology Collection Studies*. In: *IMC'07*, Oktober 2007.

[Hä67] Hänsel, Horst: *Grundzüge der Fehlerrechnung*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1967.

[int09] *Internet 2 Fact Sheet*, Juli 2009. http://www.internet2.edu/resources/AboutInternet2.pdf, [Online; Stand 08. Januar 2010].

[Kni05] Knieschewski, Sebastian: *Das Routing Information Protocol im Network Simulator 2*, April 2005. Studienarbeit.

[Kra] Krasnyansky, Maxim: *Universal TUN/TAP device driver*. http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt, [Online; Stand 08. Januar 2010].

[LVPS08] Levchenko, Kirill, Geoffrey M. Voelker, Ramamohan Paturi und Stefan Savage: *XL: An Efficient Network Routing Algorithm*. In: *SIGCOMM'08*, August 2008.

[Mal98] Malkin, Gary S.: *RFC2453 - RIP Version 2*. IETF, November 1998. http://tools.ietf.org/html/rfc2453, [Online; Stand 07. Januar 2010].

[max] *Maxima Manual - lsquares Packet*. http://maxima.sourceforge.net/docs/manual/en/maxima_62.html, [Online; Stand 08. Januar 2010].

[Moy98] Moy, John: *RFC2328 - OSPF Version 2*. IETF, April 1998. http://tools.ietf.org/html/rfc2328, [Online; Stand 07. Januar 2010].

[ns2a] *Ns Manual*. http://www.isi.edu/nsnam/ns/doc/index.html, [Online; Stand 08. Januar 2010].

[ns2b] *Ns Manual - DV Routing*. http://www.isi.edu/nsnam/ns/doc/node319.html, [Online; Stand 08. Januar 2010].

[Obr00] Obradovic, Davor: *Formal Analysis of Convergence of Routing Protocols*. Dissertation, University of Pennsylvania, November 2000.

[Ora90] Oran, David R.: *RFC1142 - OSI IS IS Intra-domain Routing Protocol*. IETF, Februar 1990. http://tools.ietf.org/html/rfc1142, [Online; Stand 07. Januar 2010].

[PB94] Perkins, Charles E. und Pravin Bhagwat: *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*. In: *SIGCOMM'94*, August 1994.

[PBRD03] Perkins, Charles E., Elizabeth M. Belding-Royer und Samir R. Das: *RFC3561 - Ad hoc On-Demand Distance Vector (AODV) Routing*. IETF, Juli 2003. http://tools.ietf.org/html/rfc3561, [Online; Stand 07. Januar 2010].

[Pos80] Postel, Jonathan B.: *RFC768 - User Datagram Protocol (UDP)*. IETF, August 1980. http://tools.ietf.org/html/rfc768, [Online; Stand 07. Januar 2010].

[Pos81] Postel, Jonathan B.: *RFC791 - Internet Protocol*. IETF, September 1981. http://tools.ietf.org/html/rfc791, [Online; Stand 07. Januar 2010].

[Sch04] Schmitt, Frank: *Simulation von Routing-Szenarien mit dem Netzwerksimulator SSFNet*. Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, Dezember 2004.

[SL03] Solie, Karl und Leah Lynch: *CCIE Practical Studies: Configuring Route-Maps and Policy-based Routing*, November 2003. http://www.ciscopress.com/articles/article.asp?p=102092, [Online; Stand 09. Januar 2010].

[vir] *Ethernet Bridging*. http://openvpn.net/index.php/open-source/documentation/miscellaneous/76-ethernet-bridging.html, [Online; Stand 09. Januar 2010].