Verbindung verschiedener Netzwerkmodelle für hochskalierbare Netzwerksimulationen

Bachelorarbeit

von

Patrick Szewior

geboren in
Langenfeld

eingereicht bei

Technik Sozialer Netzwerke
Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich-Heine-Universität Düsseldorf

März 2016

Betreuer:
Tobias Amft

Abstract

In dieser Arbeit geht es um die Verbindung verschiedener Netzwerkmodelle in Netzwerksimulationen. Im Speziellen wird dabei auf den Simulator *PeerfactSim.KOM* [Gra11] eingegangen, welcher unterschiedliche Modelle von Netzwerken anbietet. Diese lassen sich zwar in ihren Eigenschaften konfigurieren, aber nicht kombinieren. Durch die Verbindung von verschiedenen Modellen und Konfigurationen innerhalb einer Simulation kann man die Komplexität eines Netzwerks, auf dem beispielsweise ein Peer-to-Peer-Overlay aufsetzt, besser und realistischer abbilden. Es wird herausgearbeitet welche Anforderung eine solche Erweiterung des Simulators erfüllen muss und welche Schwierigkeiten sich dadurch ergeben. Anschließend wird diese als transparente Schicht in PeerfactSim.KOM implementiert und die Besonderheiten sowie Schwierigkeiten beim Design beleuchtet. Darauf hin wird die Implementierung auf ihre korrekte Funktionsweise, auch mit bereits vorhandenen Overlays, getestet und die Ergebnisse ausgewertet.

Danksagung

An dieser Stelle möchte ich mich zuerst bei meinem Betreuer, Tobias Amft, dafür bedanken, dass er mir dieses spannende Thema vorgeschlagen hat und mich während der letzten Monate mit seinen Ratschlägen und Anregungen unterstützt und motiviert hat. Außerdem danke ich Andreas Disterhöft, der seine Erfahrungen zum Determinismus von Simulationen mit mir geteilt hat, was mir dabei geholfen hat viele Probleme rechtzeitig zu erkennen und zu lösen. Ein großer Dank für das Korrekturlesen dieser Arbeit gilt Selina Nolden, Tobias Amft und meiner Freundin Vicky, die mich während des ganzen Studiums unterstützt und immer an mich geglaubt hat. Ganz besonders bedanken möchte ich mich bei meinen Eltern, die mich immer wieder motiviert und mich auf jede erdenkliche Weise unterstützt haben!

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In der heutigen Zeit gewinnen Computernetzwerke immer mehr an Bedeutung. Die Anzahl an vernetzten Geräten und angebotenen Diensten steigt rasant. Alles soll immer und überall verfügbar sein. Mögliche Ansätze diese Geräte zu vernetzen sind Cloud-Dienste oder Peer-to-Peer-Overlays. Letztere sind besonders dann interessant, wenn man ohne zentrale Serverstrukturen auskommen will. Dabei entstehen schnell sehr komplexe Protokolle, deren Funktion und Skalierbarkeit man unter verschiedenen Voraussetzungen testen möchte. Solche Tests sollten zum einen realistische, aber auch extreme Szenarien ermöglichen. Häufig sind Tests mit realen Geräten deshalb nicht möglich und man möchte auf einen Simulator zurückgreifen. So kann man das Verhalten eines Peer-to-Peer-Overlays in verschiedenen Situationen und Netzwerkmodellen testen. Mögliche Szenearien sind z.B. Netzwerke, die hohe Latenzen oder Paketverlustraten aufweisen. Auch unter solchen Umständen sollte ein Peer-to-Peer-Overlay oder eine Anwendung stabil laufen. In PeerfactSim.KOM [Gra11] können solche unterschiedlichen Modelle bereits getestet werden. Das Internet besteht allerdings aus vielen kleinen Teilnetzwerken, die ganz unterschiedliche Eigenschaften haben. Die Peers eines Peer-to-Peer-Overlays werden sich in der Regel in den äußersten Teilen des Netzwerks befinden. Gerade dort können die Peers aber sehr unterschiedlich angebunden sein.

Daher wäre es wünschenswert, genau diese Verteilung der Geräte simulieren zu können. Ein Teil der Peers sind vielleicht normale Desktop-PCs mit einer gewöhnlichen DSL-Verbindung, ein anderer Teil nutzt vielleicht einen WLAN-Hotspot oder LTE. Durch die Verwendung von verschiedenen Modellen in der selben Simulation, kann man komplexere Topologien simulieren, wodurch PeerfactSim.KOM durchaus auch für die Simulation von Client-Server Anwendungen interessant wird.

1.1 Gliederung der Arbeit

Kapitel 1 lieferte eine kurze Erklärung warum die Verbindung verschiedener Netzwerkmodelle in Simulationen wünschenswert sein kann. In Kapitel 2 wird unter anderem die interne Funktionsweise von PeerfactSim.KOM kurz erläutert sowie auf Protokolle eingegangen, welche in der Implementierung verwendet wurden. Das grundsätzliche Design zur Verbindung verschiedener Netzwerkmodelle in PeerfactSim.KOM wird in Kapitel 3 vorgestellt. Darauf folgt Kapitel 4, welches die Besonderheiten der Implementierung und ihrer Verwendung erklärt, auch unter dem Aspekt Determinismus von Simulationen. Anschließend sollen in Kapitel 5 Tests durchgeführt und ausgewertet werden. Zusätzlich werden die bei der Implementierung entstandenen Probleme analysiert. Danach folgen in Kapitel 6 eine Zusammenfassung der Ergebnisse dieser Arbeit sowie ein Ausblick auf mögliche weiterführende Arbeiten.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel wird zuerst kurz der Simulator PeerfactSim.KOM vorgestellt. Ferner wird dessen interner Aufbau, sowie die bisherigen Simulationsmöglichkeiten umrissen. Danach wird auf das Routing in Computernetzwerken eingegangen.

2.1 Simulator - PeerfactSim.KOM

Der Simulator PeerfactSim.KOM [Gra11] wurde ursprünglich von der TU Darmstadt entwickelt, später an der Universität Paderborn und der Heinrich-Heine-Universität Düsseldorf erweitert. PeerfactSim.KOM ist in Java geschrieben und arbeitet Event-basiert. Außerdem sind die Simulationen deterministisch, wodurch sich die Ergebnisse einer bestimmten Konfiguration beliebig reproduzieren lassen. Das Hauptaugenmerk lag dabei auf der Simulation von Peer-to-Peer-Netzwerken. Dabei wird von der Anwendungsschicht bis hin zur dritten Schicht des OSI-Modells, also der Vermittlungsschicht, simuliert. Da der Simulator auch intern nach dem Schichtenmodell aufgebaut ist, lassen sich prinzipiell alle Schichten beliebig austauschen und überwachen. So kann man also beispielsweise ein Peer-to-Peer-Overlay, um es auf grundlegende Probleme zu überprüfen, zuerst auf einem ganz simplen Netzwerk ohne Paketverlust und mit geringen Latenzen testen. Anschließend kann man weitere Tests unter schwierigeren Bedingungen simulieren, z.B. wenn das Peer-to-Peer-Overlay für den Einsatz in mobilen Adhoc-Netzwerken gedacht ist. Dabei benutzen jedoch alle Hosts in der Simulation stets dieselbe Art von Netzwerk. Diese lassen sich auch in geographische Gruppen unterteilen, um so beispielsweise unterschiedliche Latenzen simulieren zu können oder nur bestimmte Regionen vom Netz zu isolieren. Dafür lädt man bei der Erstellung der Netzwerkschicht verschiedene Module, die sich darum kümmern. Wenn eins dieser Module z.B. den Churn simuliert, also das verlassen und wieder beitreten zum Netzwerk, so wird das selbe Modul von allen Hosts verwendet. Man hat nur noch die Wahl dazwischen, ob es aktiviert ist oder nicht. Möchte man jedoch erreichen, dass verschiedene Re-

gionen unterschiedlich stark vom Churn betroffen sind, so muss man diese Möglichkeit bereits in das Churn-Modell einbauen. Das kann unter Umständen erfordern, dass man ein neues Modul erstellen muss, welches auf das gewünschte Szenario zugeschnitten ist.

2.2 Routing zwischen Netzwerken

In diesem Kapitel wird der Begriff und das Prinzip von Routing erklärt sowie auf mögliche Ansätze, Routing zwischen Netzwerken zu realisieren, eingegangen.

2.2.1 Prinzip von Routing und Forwarding

Als Routing bezeichnet man im Allgemeinen das Festlegen von Wegen für einzelne Datenpakete in Rechnernetzen. Durch Routing ist es möglich, dass Datenpakete aus einem Netz in ein anderes weitergeleitet werden, auch über die Grenzen eines Netzwerks hinweg. Knoten, die solche verschiedenen Netze miteinander verbinden, heißen Router. Wenn ein Router ein Datenpaket erhält, welches sein Ziel in einem anderen Netz hat, muss er entscheiden, über welchen Nachbarknoten er dieses weiterleitet. Diesen Prozess bezeichnet man als Forwarding. Für die Entscheidungsfindung benutzen Router unter anderem sogenannte Routing-Tabellen. In einer Routing-Tabelle kann der Router anhand der Ziel-Adresse nachschlagen, an welchen Nachbarknoten bzw. Router er das Paket weiterleiten muss. Welche Option am besten ist, wird anhand von Metriken entschieden, welche unter anderem viele verschiedene Informationen enthalten können:

- Bandbreite der Verbindung
- Auslastung der Verbindung
- Hop Count, also die Anzahl von Schritten zum Ziel
- MTU, also die maximal mögliche Paketgröße
- Latenz der Verbindung

2.2.2 Arten von Routing

Im wesentlichen kann man zwischen statischem und adaptivem Routing unterscheiden. Bei statischem Routing werden alle Routen fest vorgegeben. Diese Art von Routing kommt vor allem in kleineren Netzwerken zum Einsatz, wo die Anzahl der Router sehr klein ist. In Netzen mit deutlich komplexeren Topologien setzt man auf adaptives Routing, d.h. Router können selbständig Routing-Informationen untereinander austauschen und so ihre Routing-Tabellen aktualisieren. Außerdem kann mit adaptivem Routing schneller auf plötzliche Änderungen in der Netzwerktopologie reagiert werden.

2.2.3 Beispiel: Distanzvektoralgorithmus

Es gibt verschiedene Arten von adaptiven Routingprotokollen. Eine Klasse davon bildet der Distanzvektoralgorithmus. Als Metrik wird dabei der Hop Count verwendet. Anfangs sind einem Router nur die Netzwerke bekannt, an die er direkt angeschlossen ist. Zuerst wird seine Routing-Tabelle also mit diesen Informationen befüllt. Sind in seinem Netzwerk auch andere Router verfügbar, so wird er mit der Zeit auch von diesen eine Information erhalten, welche Netzwerke sie erreichen können und welchen Wert die dazugehörige Metrik hat. Diese Informationen kann der Router nun in seine Tabelle integrieren und diese auch an andere Router verteilen. Solche Informationen werden periodisch an die direkt erreichbaren Router verteilt und auch von diesen empfangen. Sie enthalten meist nur die Informationen über die beste Route in das entsprechende Netz. Ändert sich die Metrik einer solchen Route, so werden die Änderungen sofort an andere Router verteilt, ähnlich wie bei *Triggered Updates* in RIPv2 [Mal98].

Kapitel 3

Design einer Schicht zur Zusammenführung von verschiedenen Netzwerkmodellen

In diesem Kapitel wird das grundlegende Design des Moduls, welches die verschiedenen Netzwerkmodelle verbinden soll, beschrieben. Wir werden das Modul im weiteren Verlauf dieser Arbeit als *Network Switcher* bezeichnen. Der Network Switcher soll als eine neue transparente Schicht zwischen dem Transport- und dem Network Layer realisiert werden. Die bereits vorhandenen Module des Simulators sollen dabei in dieser Arbeit weitestgehend unberührt bleiben. Dies hat im Wesentlichen zwei Gründe: Zum einen soll der Network Switcher eine optionale Schicht sein. Da nicht ausgeschlossen werden kann, dass sich ein Bug erst im Laufe der Zeit bei einer ganz speziellen Konfiguration zeigt, sollte die Möglichkeit bestehen auch wie gewohnt ohne Network Switcher zu simulieren. Zum anderen sollte sichergestellt werden, dass Änderungen für die Unterstützung eines optionalen Network Switchers nicht die Simulationsergebnisse ohne eben diesen beeinflussen. Dieses Prinzip hat jedoch nicht nur Vorteile. Ein großer Nachteil ist, dass einige der bestehenden Network Layer sich noch nicht für eine Benutzung mit einem Network Switcher eignen. Teilweise werden in ihnen Funktionen und Objekttypen implementiert, die voraussetzen, dass auf allen Hosts, mit denen kommuniziert wird, die selben Network Layer vorhanden sind. Würde man versuchen alle diese Network Layer auf einen gemeinsamen Nenner zu bringen, müsste man jedes Modell individuell anpassen und erweitern. Dadurch wäre es wahrscheinlich, dass sich bisherige Simulationen nicht mehr eins-zu-eins reproduzieren lassen. Wenn man zusätzlich zur Ursprungsversion eine angepasste Version bereitstellen würde, dann könnte die Verwendung der Network Layer schnell unübersichtlich werden und für jedes Modell müssten wahrscheinlich zwei Versionen erstellt werden. Diese Lösung erscheint nicht sonderlich angemessen. Stattdessen bietet sich in diesem Fall an, den bereits vorhandenen `ModularNetLayer` sowie den einfacheren `SimpleNetLayer` zu verwenden. Der `ModularNetLayer` lässt sich in praktisch allen Parametern anpassen und durch einzelne (optionale) Module konfigurieren.

3.1 Grundlegender Aufbau

Wie bereits in diesem Kapitel erwähnt wurde, soll der Network Switcher als eine transparente und optionale Schicht innerhalb des Simulators implementiert werden (Abbildung 3.1). Der Network Switcher soll dabei aus drei einzelnen Modulen bestehen: dem eigentlichen Switcher-Modul (Kapitel 3.2), einem Router-Modul (Kapitel 3.3) und einem sog. Router-Connector (Kapitel 3.4). Im Folgenden sollen die Aufgaben der Module, sowie deren Schnittstellen untereinander erklärt werden.

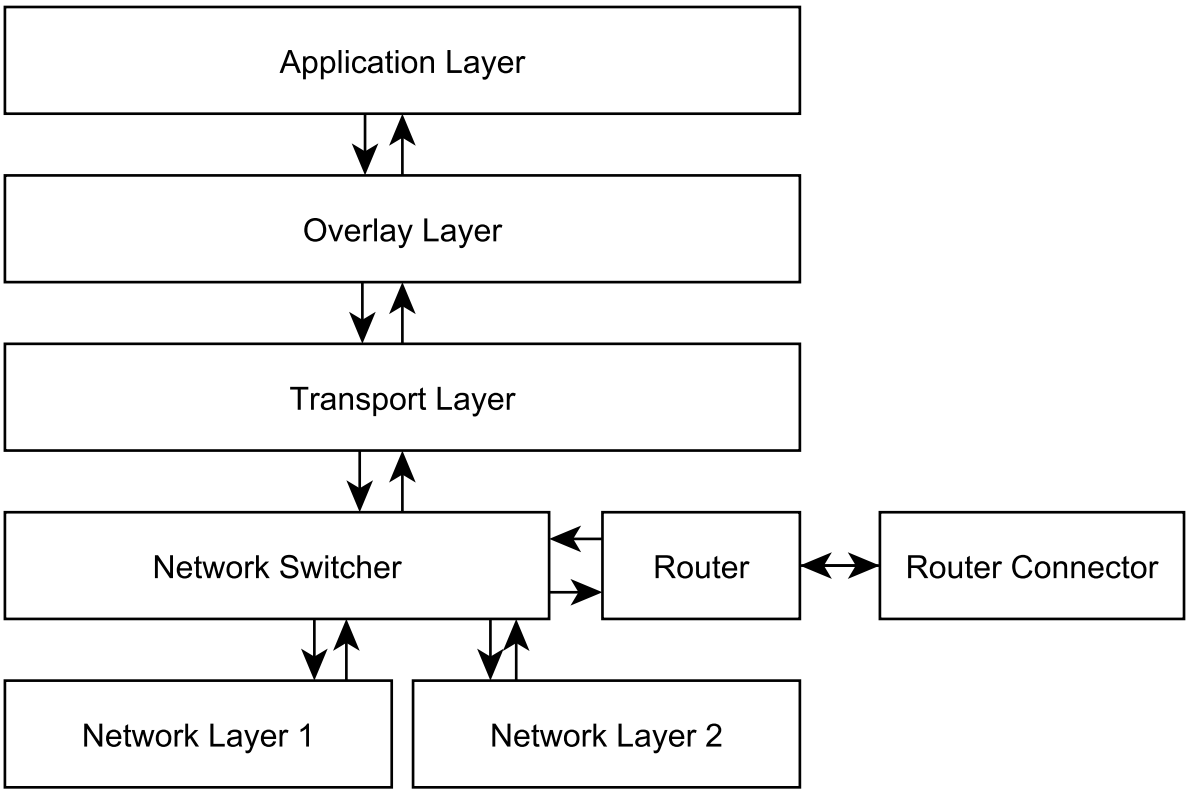


Abbildung 3.1: Grundlegende Integration des Network Switchers innerhalb des Schichtenmodells von PeerfactSim.KOM

3.2 Network Switcher - Modul

Das Network Switcher-Modul übernimmt, wie der Name bereits vermuten lässt, den Wechsel zwischen unterschiedlichen Netzwerken eines Hosts. Bisher kann ein Host nur mit genau einem Network

Layer arbeiten, auch wenn man in der Konfiguration beliebig viele einfügen kann und diese auch geladen werden. Es wird immer der Network Layer benutzt, welcher in der Konfiguration direkt über dem Transport Layer steht. Dennoch ist der Host dazu in der Lage, eine Liste mit allen erzeugten Network Layern zu liefern. Diese beiden Tatsachen werden wir uns für unseren Network Switcher zu Nutze machen. Da der Network Switcher transparent arbeiten soll, wird er dem Transport Layer alle Schnittstellen eines Network Layers anbieten. Gleichzeitig wird er allen darunterliegenden Network Layern sämtliche Schnittstellen eines Transport Layers anbieten. Weder der Transport Layer, noch einer der Network Layer werden wissen, dass dort ein Switcher arbeitet.

Jeder der geladenen Network Layer erhält automatisch eine eigene `NetID`, welche hier als *IP-Adresse* interpretiert werden kann. Damit das Routing später übersichtlicher wird, bekommt auch der Network Switcher eine eigene IP-Adresse in Form einer NetID. Die API würde dies ohnehin verlangen. Dem Host bzw. der darauf laufenden Anwendung wird ausschließlich die IP des Network Switchers mitgeteilt. Bisher können Hosts noch nicht wissen, dass sie andere Hosts unter mehreren IP-Adressen erreichen können. Es wird nicht einfach nur irgendeine IP eines der Network Layer zugewiesen, da es das Routing unnötig verkomplizieren würde. Es wäre schwierig, zu unterscheiden, ob mit der Wahl der Ziel-IP gewünscht ist, den Host auch auf einem bestimmten Interface zu kontaktieren oder ob dies egal ist. Die IP-Adressen der unteren Network Layer werden wir in diesem Entwurf nur für die Adressierung innerhalb ihres implementierten *Subnets* benutzen. Sie werden also zu einer Art physikalischen oder link-lokalen Adressierung eingesetzt, während die IP-Adressen der Hosts für die logische Adressierung benutzt werden.

Der Network Switcher wird also, wie ein gewöhnlicher Network Layer, die Nachricht vom Transport Layer entgegen nehmen. Anschließend erfragt er beim Router anhand der Ziel-Adresse, an wen er die Nachricht tatsächlich senden soll. Der Router liefert ihm anschließend die Information, an welche Adresse die Nachricht gesendet werden soll und über welchen der verfügbaren Network Layer.

Nun kommt es zum ersten Problem. Wir haben nun einerseits die IP-Adresse des richtigen Empfängers und zusätzlich die IP-Adresse des Knotens, über den das Ziel erreicht wird. In der Nachricht, die gesendet werden soll, ist aber nur Platz für den unmittelbar nächsten Empfänger auf der Route, also den Router über den weitergeleitet wird. Somit würde beim Forwarden einer Nachricht, die Information über den tatsächlichen Empfänger verloren gehen. Um dieses Problem zu umgehen, erstellen wir einen neuen Nachrichtentyp. Darin wird einerseits die ursprüngliche Nachricht gekapselt, sowie die Information über den tatsächlichen Empfänger der Nachricht. Außerdem fügen wir einen Parameter hinzu, anhand dessen verschiedene Typen von Inhalten direkt unterschieden werde können, z.B. Routing-Nachrichten. Für den Network Layer wird die Nachricht trotzdem wie eine gewöhnliche Nachricht aussehen und er wird auch nur die Adresse des nächsten Knotens auf der Route als Empfänger erfragen können.

Erhält nun ein Network Layer eine solche Nachricht, wird er zuerst feststellen, dass diese an ihn adressiert ist. Er wird die Nachricht an den Network Switcher, welchen er für den Transport Layer hält, übergeben. Der Network Switcher kann nun prüfen um welche Art von Nachricht es sich handelt. Eine Routing-Nachricht würde er in unserem Fall nur von seinen direkten Nachbarn erhalten und kann diese direkt an den Router übergeben. Handelt es sich jedoch um eine gewöhnliche Nachricht, kann der Network Switcher nun prüfen, ob sein Host der tatsächliche Empfänger ist. Wenn das der Fall ist, wird die Nachricht entkapselt, in das Format gebracht, das der Transport Layer erwarten würde, und an ihn übergeben. Näheres dazu folgt in Kapitel 4.

Stellt der Network Switcher fest, dass er die Nachricht weiterleiten muss, so erfragt er, wie bei anderen Nachrichten auch, die nötigen Informationen beim Router-Modul und versendet wieder eine gekapselte Nachricht.

Der Network Switcher wird auch für das Router-Modul als Kommunikationsschnittstelle zur anderen Routern arbeiten. Dafür wird er die Routing-Nachrichten vom Router-Modul als Broadcast in die entsprechenden Netzwerke versenden.

Eigentlich gehören Broadcasts zum Aufgabenbereich von IP in der Vermittlungsschicht [Bak95], also des Network Layer. Diese Funktion ist dennoch nicht vorhanden. Wir müssen diese Aufgabe also dem Network Switcher übertragen. Vorerst wird der Network Switcher nur Routing-Nachrichten als Broadcasts versenden. Da keine Broadcast-Adresse vorhanden ist, muss der Network Switcher für jeden Empfänger eine separate Nachricht erstellen. Die Liste von Broadcast-Empfängern in einem Netzwerk kann der Network Switcher über sein Router-Modul beim Router-Connector-Modul erfragen.

Besonderheiten bei der Implementierung der Schnittstellen zum Transport- und zu den Network Layern hin, werden in Kapitel 4 behandelt.

3.3 Router - Modul

Die zentrale Aufgabe des Router-Moduls ist, zu bestimmen, wohin Nachrichten gesendet werden müssen, damit sie ihren Empfänger erreichen können. Dazu benutzt das Router-Modul sowohl eine eigene Routing-Tabelle, als auch weitere Informationen vom Router-Connector.

Sobald ein Router-Modul weiß, zu welchem Network Switcher es gehört, erfragt es bei diesem, welche Network-Layer vorhanden sind. Somit ist dann auch klar, welche Netze (im Simulator Subnets

genannt) direkt erreichbar sind und welche IP-Adressen man jeweils in diesen Netzen hat. Mit diesen Informationen meldet sich das Router-Modul beim Router-Connector, welcher für alle Router erreichbar ist. An diesen werden die Informationen übergeben, an welche Netzwerke der Host direkt angeschlossen ist und welche Adressen er dort hat. Aus den selben Informationen erstellt der Router seine ersten Einträge der Routing-Tabelle. Sobald diese fertiggestellt sind, erstellt er für jedes der Netzwerke eine Liste seiner besten Routen und übergibt diese dem Network Switcher, damit diese als Broadcast versendet werden. Um Routing-Schleifen zu vermeiden, wird darauf geachtet, dass die Liste für ein bestimmtes Subnet keine Routen enthält die direkt in dieses Subnet führen würden. Das entspricht im wesentlichen dem Mechanismus, den man als *Split-Horizon* [Mal98] bezeichnet.

Empfängt ein Router-Modul über seinen Switcher eine Routing-Nachricht, so werden die Informationen daraus in die eigene Routing-Tabelle integriert. In der Routing-Tabelle werden die Einträge anhand einer Metrik bewertet. In der Implementierung wird diese Metrik als Kosten bezeichnet. Sollte sich nach einem Update der Routing-Tabelle ergeben, dass sich die Kosten der besten Route zu einem Netz verändert haben, so werden sofort Updates an alle direkt erreichbaren Hosts als Broadcast versendet. Andernfalls wird dies periodisch alle 30 Sekunden erledigt.

Da das Versenden von Routing-Nachrichten verbindungslos über UDP erfolgt, muss das Router-Modul darauf achten, dass die Routen, die es von anderen Routern gelernt hat, weiterhin gültig sind. Da eine einzelne verloren gegangene Routing-Nachricht noch kein Grund ist die Routen direkt zu löschen, wird, ähnlich wie in RIPv2 [Mal98], 180 Sekunden auf Routing-Nachrichten gewartet bevor die Routen gelöscht werden. Anschließend werden sofort Updates versendet.

Der genaue Aufbau der Routing-Tabelle sowie ihrer Einträge wird in Kapitel 4.2.1 erklärt.

3.4 Router Connector - Modul

Der Router-Connector wird von den Routern benutzt um Informationen auszutauschen, die, wenn überhaupt, nur sehr schwer von den Network Layern zu bekommen sind. Es ist nicht ohne weiteres möglich, zu erfahren, welche weiteren Network Layer mit einem Subnet verbunden sind. Für einen Broadcast muss diese Information allerdings vorhanden sein. Beim Start teilt jeder Router diese Informationen dem Connector mit und dieser sammelt und verwaltet diese. Zum einen kann beim Connector abgefragt werden, welche Adressen noch in einem Netzwerk vorhanden sind (z.B. für Broadcasts) oder man kann abfragen, zu welchem Host eine IP-Adresse gehört. Ebenso können sämtliche Adressen eines Hosts erfragt werden, ebenso wie die Netze an die er angeschlossen ist. So können Router sicherstellen, auch wirklich aus allen möglichen Routen die beste auswählen zu können.

3.5 Sonstige Änderungen am Simulator

Da die IP-Adressen im Simulator nicht nach einem bestimmten Prinzip vergeben werden, kann nicht nach IP-Präfixen geroutet werden. Das würde bedeuten, dass jeder Host in seiner eigenen Routing-Tabelle Routen zu jedem einzelnen anderen Host speichern würde. Das würde nicht nur unnötig viel Speicher kosten und die Simulation somit schlecht skalieren lassen, sondern auch den Aufwand von Änderungsoperationen in den Routing-Tabellen drastisch erhöhen.

Glücklicherweise lassen sich in PeerfactSim.KOM die Netzwerke in Form von Subnets anhand des entsprechenden Objekts in Java sehr leicht unterscheiden. Es musste nur eine Möglichkeit geschaffen werden, von einem Network Layer zu erfahren, an welches Subnet er angebunden ist. Dafür wurde die entsprechende API der Network Layer nur um eine Methode `getSubnet` erweitert.

3.6 Ablauf einer Nachrichtenübertragung

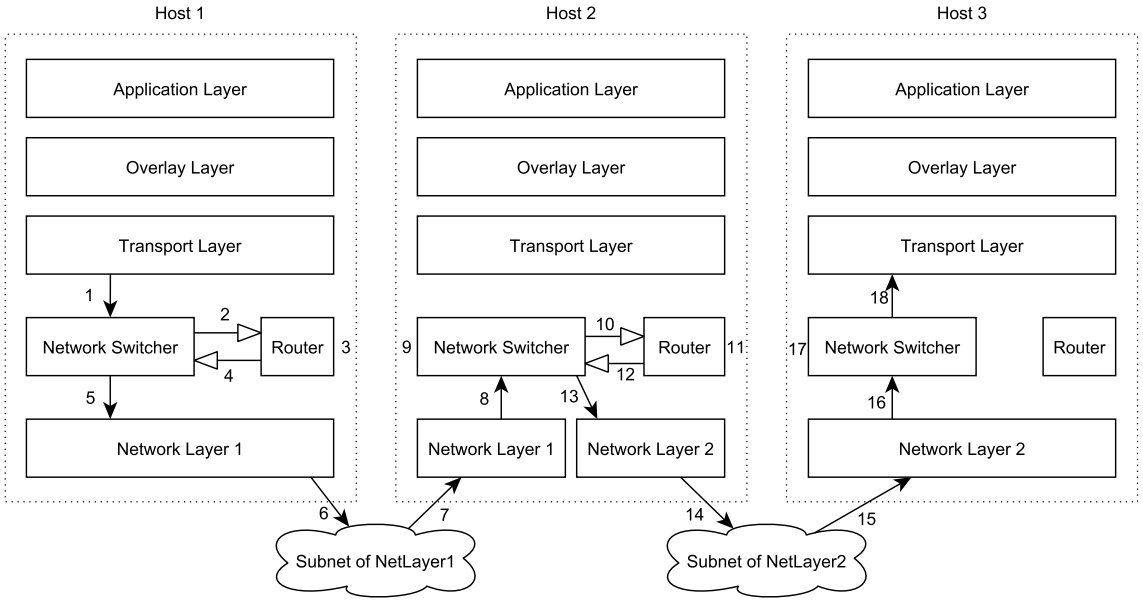


Abbildung 3.2: Visualisierung einer Nachrichtenübertragung mit dem Network Switcher

Schritt	Aktion
1	Der Transport Layer von Host 1 übergibt die an Host 3 adressierte Nachricht an den Network Switcher
2	Der Network Switcher von Host 1 fragt das Router-Modul, an wen die Nachricht gesendet werden soll
3	Das Router-Modul erfragt beim Router-Connector, welche Subnets Host 3 erreicht. Dann wird aus der Routing-Tabelle die beste Route ausgesucht
4	Das Router-Modul übergibt die Route zu Host 3 dem Network-Switcher
5	Der Network-Switcher übergibt die Nachricht an den Network Layer 1, um sie an Host 2 zu senden
6	Der Network Layer 1 von Host 1 überträgt die Nachricht an sein Subnet
7	Das Subnet liefert die Nachricht an den Network Layer 1 von Host 2 aus
8	Network Layer 1 übergibt die Nachricht an den Network Switcher von Host 2
9	Der Network Switcher entpackt die Nachricht und stellt fest, dass die Nachricht nicht für seinen Host bestimmt ist
10	Der Network Switcher von Host 2 fragt das Router-Modul, an wen die Nachricht gesendet werden soll
11	Das Router-Modul erfragt beim Router-Connector, welche Subnets Host 3 erreicht. Dann wird aus der Routing-Tabelle die beste Route ausgesucht
12	Das Router-Modul teilt dem Network Switcher mit, dass er die Nachricht direkt an Host 3 über den Network Layer 2 senden soll
13	Der Network Switcher kapselt die Nachricht wieder und übergibt sie an den Network Layer 2, um sie an Host 3 zu senden
14	Der Network Layer 2 von Host 2 übergibt die Nachricht an sein Subnet
15	Das Subnet liefert die Nachricht an den Network Layer 2 von Host 3 aus
16	Der Network Layer 2 übergibt die Nachricht an den Network Switcher von Host 3
17	Der Network Switcher von Host 3 entpackt die Nachricht aus und stellt fest, dass sie für seinen Host bestimmt ist.
18	Der Network Switcher übergibt die Nachricht an den Transport Layer

3.7 Zusammenfassung der bereitgestellten Funktionen

Durch den Network Switcher werden folgende Funktionen bereitgestellt

- Switcher-Modul

  Versand und Empfang von Nachrichten über verschiedene Network Layer

  Kommunikation über die Grenzen des eigenen Netzwerks hinweg

  Verbindung von mehreren Netzwerken

  Senden von Broadcasts

  Versand von Routing-Nachrichten

- Router-Modul

  Automatischer Aufbau von Routing-Tabellen

  Bereitstellung von optimalen Routen für Nachrichten anhand einer Metrik

  Zufälliger Wechsel zwischen gleichwertigen Routen zur gleichmäßigen Verteilung der Last

  Reaktion auf ausgefallene Routen

  Bekanntgabe der eigenen Routen an andere Hosts

- Connector-Modul

  Bereitstellung der nötigen Adressen für Broadcasts

  Umwandlung von lokal verwendeten Adressen in globale

  Bereitstellung der Information, welche Netzwerke ein Host direkt erreichen kann, damit alle möglichen Routen gefunden werden können

Kapitel 4

Implementierung und Verwendung

In diesem Kapitel wird neben der Implementierung des Network Switchers und seiner Module auch auf die zu beachtenden Besonderheiten eingegangen, welche auf das bestehende Design des Simulators und seiner Layer zurückzuführen sind. Zuerst wird das Switcher-Modul im Detail (Abschnitt 4.1) beschrieben, anschließend folgen Router-Modul (Abschnitt 4.2) und Router-Connector (Abschnitt 4.3).

4.1 Switcher-Modul

Wie bereits in Kapitel 3.2 beschrieben, übernimmt das Switcher-Modul den Wechsel zwischen verschiedenen Network Layern, sowie das Kapseln und Versenden von Nachrichten.

In PeerfactSim.KOM gibt es zu fast allen Komponenten, wie den Layern, eine Factory. Diese Factory wird ausgeführt, wenn eine Konfiguration eingelesen wird und erstellt anschließend die Komponente mit den gewünschten Parametern. Der Switcher wird also, wie andere Layer auch, von einer Factory erstellt. Dabei bekommt er als Parameter seinen Host, seine eigene NetID sowie sein Router-Modul übergeben. Danach baut der Switcher seine in Abschnitt 4.1.2 beschriebenen internen Datenstrukturen auf. Er erfragt zunächst beim Host eine Liste mit allen Network Layern. Diese und eine Liste von denen, die auch tatsächlich online sind, verwaltet er von da an intern. Danach muss der Switcher sich selbst als NetMsg- und Connectivity-Listener bei allen seinen Network Layern registrieren, um über eingehende Nachrichten, sowie Änderungen der Konnektivität informiert zu werden. Abschließend teilt der Switcher seinem Router-Modul seine Instanz und seine verfügbaren Network Layer mit.

Im Folgenden soll nun besonders auf die Schnittstellen eingegangen werden, welche sich in ihrer Implementierung von der eines normalen Network Layers unterscheiden. Anschließend wird auf die

Datenstrukturen eingegangen, welche von den Modulen intern zur Verwaltung aller relevanten Informationen und Daten benötigt werden.

4.1.1 Schnittstellen

Da der Network Switcher transparent zwischen dem Transport Layer und den Network Layern arbeiten soll, implementiert er sowohl die API eines Network Layers, als auch die des Transport Layers. Jedoch ist dabei nur der Teil der API interessant, der überhaupt von dem darunter- bzw. darüber liegenden Layer benutzt werden kann. Das bedeutet, dass von der Network Layer API nur die Schnittstellen gebraucht werden, die vom Transport Layer aufgerufen werden. Dazu gehören hauptsächlich:

- `send(Message msg, NetID receiver, NetProtocol protocol)`

  Schnittstelle zum Senden einer Nachricht

- `isOnline()`, `isOffline()`, `goOnline`, `goOffline`

  Schnittstellen zur Abfrage und Konfiguration des Online/Offline-Status

- `getCurrentBandwidth()`, `getMaxBandwidth()`

  Schnittstellen zur Abfrage der Bandbreiten

- `getNetID()`

  Schnittstelle zur Abfrage der NetID

- `addNetMsgListener(NetMessageListener listener)`,
  `addConnectivityListener(ConnectivityListener listener)`
  `addChurnListener(ChurnListener listener)`

  Schnittstellen zur Registrierung und Abfrage von diversen Listenern (Connectivity, Churn, Message)

Genauso verhält es sich mit der Transport-Layer-API. Davon benötigt man im Grunde nur die Schnittstelle `messageArrived(NetMsgEvent)`, welche Nachrichten von Network Layern entgegen-

nimmt. Der Empfang von Nachrichten wird jedoch erst im Abschnitt 4.1.3 genauer erklärt, da dort auch direkt die ganze Logik zum Entkapseln und Forwarding der Nachricht beschrieben wird.

Darüber hinaus enthält das Switcher-Modul noch weitere Schnittstellen, u.a. für die Kommunikation mit dem Router-Modul. Dazu zählen

- Schnittstelle zur Abfrage aller Network Layer
- Schnittstellen, mit denen sich der Status o. Bandbreite einzelner Network Layer abfragen lässt

Nun sollen die Implementierung der Schnittstellen betrachtet werden.

Senden einer Nachricht

Diese Schnittstelle wird ausschließlich vom Transport Layer benutzt. Es werden die zu sendende Nachricht, die Adresse des Empfängers in Form einer NetID, sowie das zu verwendende Protokoll (z.B. IPv4) übergeben. Ein normaler Network Layer würde diese Nachricht dann an sein Subnet übergeben. Der Network Switcher prüft zuerst, ob die übergebene Nachricht bereits eine geeignete Paketnummer bzw. ID hat. Falls nicht, dann wird eine neue generiert. Ohne diese Abfrage verhielten sich während der ersten Tests manche Anwendungen nicht korrekt. Anschließend wird mit Hilfe des Router-Connectors geprüft, ob es sich bei der angegebenen NetID um eine global bekannte handelt, oder ob dies eine lokale Adresse innerhalb eines Subnets ist. Diese wird dann automatisch durch die globale logische Adresse ersetzt. Dadurch vereinfacht sich zum einen das Routing, zum anderen ergeben sich damit auch mehr mögliche Routen zum Ziel, wenn man jeden der am Ziel vorhandenen Network Layer kontaktieren kann. Ein weiterer Vorteil liegt darin, dass sowohl Overlays als auch Anwendungen bisher davon ausgehen konnten, dass ein Host exakt eine NetID besitzt. Es bestünde die Möglichkeit, dass diese Anwendungen aufgrund mehrerer NetIDs auch mehr Hosts erwarten als tatsächlich vorhanden sind oder sie versuchen einen Host immer über die selbe NetID zu kontaktieren wodurch seine anderen Network Layer unbenutzt blieben. Im Regelfall sollte den Layern über dem Switcher ohnehin nur die globale Adresse bekannt sein. Anschließend wird die Nachricht innerhalb des Switchers an die Funktion übermittelt, die dann den korrekten Network Layer auswählt und die Nachricht kapselt (Abschnitt 4.1.3). Dabei werden auch weitere Parameter, wie die Adresse des Senders und der Typ der Nachricht, übergeben. Optional kann auch der zu benutzende Network Layer direkt bestimmt und ein optionales Objekt übergeben werden. Das optionale Objekt kann nützlich sein für spätere Erweiterungen oder kann einfach zum Debugging benutzt werden.

Broadcasts

Für das Versenden von Routing-Nachrichten in die eigenen Netzwerke, benötigt man eine Art Broadcast. Diese Möglichkeit ist in den bisherigen Network Layern aber nicht enthalten und wurde somit als Workaround in den Switcher integriert. Als Parameter werden dabei folgende Werte übergeben

- Nachricht
- Subnet in das der Broadcast gesendet werden soll
- der dabei zu benutzende Network Layer
- Protokoll
- Port von Sender und Empfänger

Da Broadcasts nicht von den Network Layern unterstützt werden, muss an jeden Host in dem spezifizierten Subnet eine eigene Nachricht gesendet werden. Dazu wird über den Router-Connector eine Liste aller Adressen innerhalb eines Subnets geholt. Dann wird für jeden Host eine eigene Nachricht erstellt, welche die Routing-Nachricht enthält. Diese wird dann intern an die Funktion zum Senden der Nachricht übergeben.

Dieser Vorgang ist natürlich bei genauerer Betrachtung nicht sonderlich effizient. Zum einen wird ausschließlich wegen den Broadcasts im Router-Connector für jedes Subnet eine Liste mit allen Adressen der Hosts bereit gehalten. Diese Daten sind redundant, da jedes Subnet diese intern sowieso in irgendeiner Form verwaltet. Zum anderen muss für jeden Host ein eigenes Paket erstellt werden. Das bedeutet, dass bei n Hosts auch n Pakete erstellt und gesendet werden müssen. Dies ließe sich vermeiden, wenn Network Layer einen Broadcast nativ unterstützen würden. Somit müsste nur exakt ein Paket gesendet werden, welches aber vom Subnet einfach an alle daran angeschlossenen Network Layer verteilt wird. Es müsste nicht einmal der Inhalt der Nachricht angepasst werden, da man beim Empfang eines Broadcasts die Empfangsadresse im Switcher ignorieren könnte. Somit würde nur noch der Aufwand für das Empfangen und Verarbeiten der Nachricht anfallen. Der Rechen- und Speicheraufwand für das Senden eines Broadcasts wäre für den sendenden Host somit konstant. Das würde sich aufgrund der wegfallenden Events zum Senden sowohl auf die Simulationsgeschwindigkeit, als auch auf die Ergebnisse auswirken, da dadurch insgesamt weniger Pakete gesendet werden müssen, wodurch der Traffic reduziert wird und sich somit auch der Verlauf der Simulation ändert.

Auf Lange Sicht erscheint die Unterstützung von Broadcasts innerhalb der Network Layer unumgänglich, auch wenn diese nicht ganz trivial erscheint, weil es keinerlei Vorgaben gibt, wie ein Network Layer sein eigenes Subnet realisiert und verwaltet. Daher müsste in jedem Network Layer eine individuelle Lösung implementiert werden. Da dies für jeden Network Layer eine lange Einarbeitungs- und anschließende Testphase voraussetzt, wurde sich stattdessen für den oben beschriebenen Workaround entschieden. Dies ist auch im Hinblick auf die Performance vertretbar, da Broadcasts ohnehin bisher ausschließlich für den Austausch von Routing-Nachrichten verwendet werden. Da diese in der bestehenden Implementierung standardmäßig nur alle 30 Sekunden versendet werden, stellen sie während der Simulation einer komplexen Anwendung oder eines Overlays nur einen kleinen Teil aller gesendeten Nachrichten dar.

Abfrage des Online-Offline-Status

Ursprünglich wird über diese Schnittstelle abgefragt ob ein Network Layer online bzw. offline ist. Da der Switcher nun aber mehrere Network Layer verwaltet, die einen unterschiedlichen Status haben können, müssen diese Informationen zusammengefasst werden. Zuerst wurden dem Switcher weitere Schnittstellen zur Abfrage des Status einzelner Network Layer bereitgestellt. Diese rufen dann selbst einfach die entsprechende Schnittstellen des spezifizierten Network Layers auf und geben das Ergebnis dann weiter. Intern nutzt der Switcher diese Funktion auch, um den Gesamtstatus zu ermitteln. Dafür wird einfach über eine Liste aller Network Layer des Hosts iteriert und deren Status abgefragt. Sobald mindestens ein Network Layer online ist, ist somit auch der Switcher und damit auch der Host online. Denn solange auch nur ein Network Layer online ist, kann der Switcher alle Nachrichten über diesen versenden, was aber nicht automatisch bedeutet, dass alle Netzwerke darüber erreichbar sind. Umgekehrt müssen für den Status offline auch alle Network Layer zwingend offline sein.

Abfrage der Bandbreiten

Diese Schnittstellen werden bei Network Layern benutzt, um deren aktuelle oder maximale Bandbreite zu erhalten. In Verbindung mit dem Switcher gestaltet sich dies schwierig. Man hat unter Umständen eine Vielzahl von Network Layern mit unterschiedlichen Bandbreiten zur Verfügung. Das führt dazu, dass einer Anwendung eventuell eine aktuelle Bandbreite mitgeteilt wird, die vollkommen von der Bandbreite des Network Layers abweicht, welcher für die Pakete dieser Anwendung benutzt werden muss. Noch schwieriger wird dies vor dem Hintergrund, dass die Bandbreite aus den beiden Werten für Upload und Download besteht, die unterschiedlich hoch sein können. Im Rahmen dieser Arbeit wurde sich für eine sehr pessimistische Bandbreitenangabe entschieden. Für die Bestimmung der aktuellen Bandbreite wird von allen verfügbaren Network Layern die jeweils geringste Upload-

und Download-Bandbreite herausgesucht. In den meisten Fällen wird diese Bandbreite also geringer sein, als die tatsächlich verfügbare. Dies verhindert allerdings, dass eine Anwendung von einer besseren Verbindung, als tatsächlich vorhanden, ausgeht und diese somit überlastet. Damit riskiert man eventuelle Einbußen bei der Leistung, vorausgesetzt eine Anwendung oder ein Overlay bedient sich überhaupt dieser Schnittstelle. Für die maximale Bandbreite gestaltet sich die Entscheidung einfacher. Dort kann man das jeweilige Maximum aller Network Layer verwenden.

Diese Art von Schnittstellen gehört zu dem Teil des Simulators, der für die Verbindung verschiedener Netzwerkmodelle eventuell überdacht werden muss. Je nach Anwendungsfall möchte man durchaus nur die maximale Bandbreite erhalten, die ein einzelner Network Layer bereitstellen kann. In anderen Fällen ist vielleicht aber die insgesamt zur Verfügung stehende Bandbreite von Interesse. Bisher können Anwendungen das nur realisieren, indem sie eine Liste aller verfügbaren Network Layer anfordern und diese Informationen selbst abfragen und verarbeiten. Dies erscheint für komplexere Anwendungen und Protokolle wohl auch am sinnvollsten, da man unmöglich alle Anwendungsfälle vorhersehen kann. Aus diesem Grund stellt der Switcher vorsorglich bereits die Möglichkeit zur Verfügung, die Bandbreiten einzelner Network Layer abzufragen.

Abfrage der NetID

In Kapitel 3.2 wurde bereits erklärt, wieso eine eigene NetID für den Switcher sinnvoll ist und wieso auch diese dem Host als NetID zurückgegeben wird. Die NetID des Switchers wurde bereits während seiner Erstellung festgelegt. Hier muss sie also einfach nur zurückgegeben werden. Da die Möglichkeit besteht eine Liste mit allen Network Layern zu erfragen, ist es also auch für eine Anwendung problemlos möglich, falls nötig, alle NetIDs zu abzufragen.

Registrierung und Abfrage von Listenern

Da der Simulator Event-basiert arbeitet, spielt die Verwendung von Listenern eine wichtige Rolle. Ein Listener wartet auf ein Ereignis, z.B. eine eingehende Nachricht. Er registriert sich dann z.B. bei einem Network Layer und wird bei einer eingetroffenen Nachricht dann von diesem benachrichtigt, oder der Network Layer ruft eine bestimmte Funktion des Listeners auf, die die Nachricht dann entgegennimmt. In unserem Fall müssen sich also andere Anwendungen oder Layer als Listener am Switcher anmelden können. Der Switcher muss mit folgenden Arten von Listenern umgehen können:

- `NetMsgListener`, also jemand der auf eingehende Nachrichten wartet, hier typischerweise

der Transport Layer

- `ConnectivityListener`, also jemand der über Änderungen des Online-/Offline-Status' informiert werden soll

- `ChurnListener`, also jemand der über Churn informiert werden soll

Tritt ein Event eines bestimmten Typs ein, so werden alle Listener aus der passenden Liste benachrichtigt.

4.1.2 Interne Datenstrukturen

Durch den modularen Aufbau muss das Switcher-Modul selbst intern nur wenige Daten verwalten. Das sind im Einzelnen

- Referenz auf den Host

- Referenz auf das eigene Router-Modul

- Die eigene NetID

- Eine Liste der zu verwaltenden Network Layer

- Eine Liste der Network Layer, die online sind

- Jeweils eine Liste für jede Art von Listener

- Einen Nachrichtentyp zur Kapselung der originalen Nachricht

- Einen Nachrichtentyp zur Übergabe einer empfangenen Nachricht an den Transport Layer

Bei den Referenzen und der NetID verwendet man jeweils ein einzelnes Objekt des entsprechenden Typs. Die Listen werden als verkettete Listen realisiert. Mehr Daten müssen zur Laufzeit nicht vom Switcher verwaltet werden.

Für die Kapselung von Nachrichten benötigt der Switcher einen eigenen Nachrichtentyp. Die Network Layer erwarten eine Nachricht vom Typ `TransMessage`. Also wurde ein davon abstammender Nachrichtentyp erstellt, die `DefaultNetSwitcherTransMsg`. Eine Nachricht von diesem Typ enthält folgende Daten

- Ursprüngliche Nachricht
- Sender der (ursprünglichen) Nachricht
- Empfänger
- *Next Hop*, also der auf der Route nächste Empfänger der Nachricht
- Protokoll, z.B. IPv4
- Typ der Nachricht, momentan 0 für normale Nachrichten, 1 für Routing-Nachrichten
- Nachrichten ID, benutzt vom Transport Layer
- ein beliebiges Objekt, welches mit übertragen werden soll

Außerdem wird für die Übergabe einer empfangenen Nachricht an den Transport Layer ein weiterer Nachrichtentyp benötigt, da der Transport Layer zwar Nachrichten vom Typ *TransMessage* versendet, aber beim Empfang eine Nachricht vom Typ *NetMessage* erwartet. Das Problem wird in Abschnitt 4.1.3 genauer beschrieben.

In der momentanen Implementierung stellt eine *DefaultNetSwitcherNetMsg* keine weiteren Funktionen oder Daten bereit, verglichen mit einer normalen *NetMessage*. Da später eine Erweiterung der Schnittstellen zwischen Network Switcher und Transport Layer wünschenswert sein könnte, ließe sich der Nachrichtentyp leicht erweitern. Der bestehenden Mechanismus zur Übergabe der Nachricht an den Transport Layer müsste dann lediglich um das Setzen der zusätzlichen Daten erweitert und nicht grundlegend verändert werden.

4.1.3 Interne Abläufe

Wie sich bereits bei der Beschreibung der Schnittstellen abgezeichnet hat, enthält das Switcher-Modul wichtige Funktionen, welche nur intern benutzt werden. Diese werden nun genauer beschrieben.

Senden von Nachrichten

Diese Funktion erledigt das eigentliche Senden von jeglichen Nachrichten über einen der Network Layer. Wie zuvor bereits beschrieben wurde, erwartet diese Funktion als Parameter die originale Nachricht, das zu verwendende Protokoll, den Typ der Nachricht sowie Sender- und Empfängeradresse. Optional kann, zumindest bei Routing-Nachrichten, der zu benutzende Network Layer bestimmt werden. Außerdem kann ein beliebiges Objekt, welches mit übertragen werden soll, übergeben werden.

Der Network Switcher unterscheidet zwischen normalen Nachrichten, welche er für den Transport Layer senden soll, und Routing-Nachrichten. Diese werden anhand eines Integers unterschieden, wobei normale Nachrichten mit einer *0* bezeichnet werden und die Routing-Nachrichten mit einer *1*. Hier wurde sich bewusst für ein Integer statt einem Boolean entschieden, damit weitere Typen von Nachrichten sich später leichter ergänzen lassen und man nicht auf die bisherigen zwei Typen beschränkt ist. Ursprünglich wurde beabsichtigt, die Typ-Unterscheidung direkt anhand des Objekt-Typs zu vollziehen, da beispielsweise Routing-Nachrichten sowieso einen eigenen Nachrichtentyp benutzen. Jedoch ist es möglich, dass man später einen Nachrichtentyp für verschiedene Anwendungsfälle benutzen möchte, in denen die Nachricht anders behandelt wird. Spätestens dann wäre eine weitere Unterscheidungsmöglichkeit von Nöten. Dadurch hätte man eine Art Schachtelung innerhalb der Nachrichtentypen. Das bedeutet, dass die if-Anweisung dann nicht mehr nur von einer einzigen Variable abhängig und somit tendenziell langsamer ist, da mehrere Vergleiche von verschiedenen Typen nötig werden.

Da der Switcher Routing-Nachrichten nur an seine direkten Nachbarn senden soll, muss für eine solche Nachricht keine Route erfragt werden. Somit kann die Nachricht direkt gekapselt und, über den beim Funktionsaufruf gewählten Network Layer, gesendet werden.

Für alle anderen Nachrichtentypen wird beim Router-Modul anhand der Empfängeradresse eine passende Route angefordert. So eine Route liefert dem Switcher alle nötigen Informationen, wie z.B. an wen er die Nachricht senden und welchen seiner Network Layer er dafür verwenden soll. Der genaue Aufbau eines Routing-Eintrags wird noch in Abschnitt 4.2.1 beschrieben.

Sobald der Switcher also alle nötigen Informationen vom Router-Modul erhalten hat, kann er die Nachricht in eine *DefaultNetSwitcherTransMsg* kapseln. Diese Nachricht wird dann zum Senden an den richtigen Network Layer übergeben. Dabei wird der Next Hop aus dem Routing-Eintrag als Empfänger angegeben.

Um zu Testen, ob die Implementierung von *Split-Horizon* auch tatsächlich Routing-Schleifen verhindert, wurde das beliebige Objekt innerhalb einer Nachricht genutzt, um dort eine HashMap zu laden, in die sich jeder Host durch den eine Nachricht weitergeleitet wurde, einträgt. Sollte es dabei zu einer Hash-Kollision kommen, weil der Host bereits eingetragen ist, so ließe sich auf eine Routing-Schleife schließen.

Empfangen von Nachrichten

Der Network Switcher empfängt Nachrichten von den Network Layern wie ein Transport Layer. Aufgrund der bestehenden API im Simulator sind diese Nachrichten aber nicht mehr unmittelbar vom selben Typ, den man zuvor gesendet hatte. Es wird stattdessen ein *NetMsgEvent* empfangen. Dieses enthält dann erst die Nachricht vom Typ *DefaultNetSwitcherTransMsg*, sowie den Sender und das Protokoll. Der Sender, den das Event zurückgibt, ist jedoch lediglich der letzte Router auf dem Weg des Pakets. Das ist später zu beachten, falls wir der Empfänger der Nachricht sind.

Wird eine solche Nachricht in Form eines Events empfangen, entpacken wir zunächst die darin enthaltene Nachricht, welche vom Typ *DefaultNetSwitcherTransMsg* ist. Danach muss geprüft werden, um welchen Typ von Nachricht es sich handelt. Falls es eine Routing-Nachricht ist, werden die darin enthaltenen Informationen direkt an das Router-Modul übergeben und dort verarbeitet. Handelt es sich jedoch um eine gewöhnliche Nachricht, so muss nur noch geprüft werden, ob man selbst der Empfänger dieser Nachricht ist oder ob diese weitergeleitet werden muss. Im letzteren Fall wird die ursprüngliche Nachricht entkapselt und einfach an die interne Funktion zum Senden einer Nachricht übergeben. Sie durchläuft dann den selben Ablauf wie eine gewöhnliche Nachricht (siehe 4.1.3). Ist man selbst der Empfänger der Nachricht, so muss man diese an den Transport Layer übergeben. Dieser erwartet nun aber wieder ein NetMsgEvent, welches eine NetMessage enthält. Diese muss man erstellen. Dabei ist darauf zu achten, dass der Sender der Nachricht auch dem ursprünglichen Sender entspricht und nicht dem letzten Router. Anschließend wird das Event an alle registrierten *NetMessageListener*, also meistens Transport Layer, übergeben.

Zählen von Nachrichten

Der Simulator besitzt eine integrierte Funktion zur Zählung von Nachrichten. Diese liefert beim Einsatz des Network Switchers aber nur unzureichende Ergebnisse. Es werden automatisch alle Pakete gezählt, die an irgendeinen Network Layer übergeben werden. Dazu zählen dann auch automatisch alle Routing-Nachrichten. Besonders während der Tests war es aber wünschenswert nur die Pakete zu zählen, die von Schichten oberhalb des Network Switchers kamen. Daher benachrichtigt der Switcher bei jeder vom Transport Layer gesendeten oder empfangen Nachricht den Router-Connector, der diese dann zählt und am Ende der Simulation das Ergebnis ausgibt.

Dabei zeigte sich, dass Pakete, welche kurz vor Ende der Simulation gesendet werden und deren Ankunft beim Empfänger nach dem Ende der Simulation liegt, natürlich auch als verloren angesehen werden. Dies kann, besonders bei kurzen Simulationszeiten, aber die Ergebnisse bezüglich Paketverlusten stark verfälschen, da die Pakete nie hätten ankommen können. Auch die bereits vorhandene Zählung des Simulators schien dadurch verfälscht zu sein. Es wird daher bei allen Simulationen empfohlen, falls möglich, die Konfiguration der Anwendung so vorzunehmen, dass kurz vor Ende der Simulation keine Pakete mehr versendet werden, da diese entweder selbst, oder ihre Antworten verloren gehen würden. Dieser Mechanismus wurde auch in der verwendeten Test-Applikation in Kapitel 5 benutzt.

Die Anzahl der gesendeten, empfangenen, weitergeleiteten und verlorenen Pakete kann jederzeit zur Analyse beim Router-Connector abgerufen werden.

4.2 Router-Modul

Das Router-Modul bestimmt, wie und an wen das Switcher-Modul eine Nachricht versenden soll. Dazu muss zum einen bekannt sein, über welche Netzwerke der Empfänger zu erreichen ist und zum anderen, zu welchem dieser Netzwerke man die beste Route hat. Da ersteres allen Hosts und deren Router-Modulen zur Verfügung stehen muss, wurde dieser Teil in den globalen Router-Connector ausgelagert. Zuerst wird in Abschnitt 4.2.1 vorgestellt, wie Routing-Informationen in einer Routing-Tabelle verwaltet werden. Danach folgt in Abschnitt 4.2.2 eine Beschreibung der wichtigen Operationen, die auf einer Routing-Tabelle ausgeführt werden können. Zum Schluss erläutert Abschnitt 4.2.3 die Schnittstellen zu den anderen Modulen.

4.2.1 Struktur der Routing-Tabelle

In der Routing-Tabelle sollen Routen zu allen Netzwerken der Simulation gespeichert werden. Zuerst soll nun beschrieben werden, wie ein einzelner Eintrag in der Routing-Tabelle aufgebaut ist. Anschließend wird erklärt, wie aus den Routing-Einträgen eine Routing-Tabelle gebildet wird.

Routing-Eintrag

Da in dieser Implementierung der Distanzvektoralgorithmus zum Einsatz kommt, werden keine kompletten Routen gespeichert, sondern nur, an welchen Host eine Nachricht geleitet werden muss, damit sie in das gewünschte Ziel-Netzwerk zugestellt wird. Dieser Host wird im weiteren Verlauf als *Next Hop* bezeichnet. Da es unterschiedliche Routen geben kann, soll auch eine Metrik, hier der Hop-Count, gespeichert werden. In Kapitel 3.5 wurde bereits darauf hingewiesen, dass in PeerfactSim.KOM nicht anhand von IP-Präfixen geroutet werden kann. Stattdessen kann aber eine Referenz auf das entsprechende Objekt des Subnets benutzt werden. Für die Adressierung des Next Hop wird seine globale NetID verwendet, da dieses Schema bisher auch im Network Switcher benutzt wurde. Es kann jedoch sein, dass wir auch zum Next Hop verschiedene Verbindungen haben. Daher speichert der Routing-Eintrag auch, über welches Subnet die Routing-Informationen vom Next-Hop empfangen wurden und über welchen der eigenen Network Layer das geschehen ist.

In vielen Fällen gibt es mehrere Routen zum Ziel. Teilweise sind diese laut Metrik gleich gut. Werden solche Einträge sortiert, muss garantiert werden, dass das Ergebnis immer gleich und deterministisch ist. Da hier später nur nach der Metrik sortiert werden muss, ist, bei gleicher Metrik, die Reihenfolge der Einträge, die z.B. aus einer HashMap stammen, nicht mehr deterministisch, da die Hashes einzelner Objekte in unterschiedlichen Simulationen, trotz gleichem Seed, immer wieder verschieden sind. Daher wurden die Routing-Einträge um eine innerhalb der ganzen Simulation fortlaufende ID erweitert, was dazu führt, dass Einträge mit der selben Metrik auch immer in der selben Reihenfolge sortiert werden. Weiteres dazu folgt in Abschnitt 4.2.2.

Ein Routing-Eintrag enthält nun folgende Informationen

- Ziel-Netzwerk
- Next Hop
- Netzwerk, über welches der Next Hop kontaktiert werden soll

- Hop Count
- Network Layer, welcher zum Senden verwendet werden soll
- ID zur deterministischen Sortierung

Um eine Sortierung nur anhand der Metrik und der ID zu ermöglichen, musste die entsprechende Funktion *compareTo* überschrieben und verändert werden. Außerdem wurden die Funktionen *equals* und *hashCode* für Routing-Einträge so verändert, dass Einträge mit dem selben Ziel-Netzwerk, Next Hop und Netzwerk über das gesendet werden soll, als gleich angesehen werden. Das erleichtert später das Ersetzen alter Einträge durch neue.

Routing-Tabelle

Da der Zugriff auf eine Routing-Tabelle möglichst schnell sein soll, wurde die Tabelle mittels einer HashMap realisiert. Dabei wird über den *Schlüssel* der Hash gebildet und anhand diesem auf den *Wert* zugegriffen. Als Schlüssel wurde das Ziel-Netzwerk verwendet. Zu jedem Ziel-Netzwerk wird als Wert eine sortierte verkettete Liste von Routing-Einträgen gespeichert. Dadurch kann im Idealfall in nur zwei Schritten auf eine passende und optimale Route, nämlich die erste in der Liste, zugegriffen werden. In Abschnitt 4.2.2 wird erklärt, warum es dennoch sinnvoll ist, zufällig zwischen den besten Routing-Einträgen zu wählen, obwohl dies den Aufwand erhöht.

4.2.2 Lese- und Änderungsoperationen auf der Routing-Tabelle

Im Folgenden werden nun die Operationen beschrieben, die auf der Routing-Tabelle ausgeführt werden. Dazu gehört beispielsweise das Einfügen, Suchen und Ändern von Routing-Einträgen, sowie das Zusammenstellen von Routing-Nachrichten.

Einfügen und aktualisieren von Routing-Einträgen

Sowohl zu Beginn der Simulation, wenn alle Hosts ihre Routing-Tabellen erstellen, als auch im weiteren Verlauf müssen immer wieder neue Einträge in eine Routing-Tabelle eingefügt oder bestehende Einträge geändert werden, weil sich beispielsweise die Metrik verändert hat.

Wird die Routing-Tabelle zum ersten Mal mit Einträgen befüllt, so wird eine Liste der vom Switcher-Modul verwalteten Network Layer durchlaufen. Dabei wird für jeden Network Layer, falls nicht bereits vorhanden, ein neuer Eintrag in der HashMap mit dem Subnet des Network Layers angelegt. Als Wert wird dabei eine verkette Liste mit nur einem Routing-Eintrag übergeben. Dieser Routing-Eintrag enthält dann keinen Next Hop, da das Netzwerk direkt erreichbar ist. Das Netzwerk, über welches gesendet werden soll, sowie das Ziel-Netzwerk entsprechen dabei dem Subnet des Network Layers.

Während der Simulation wird ein Host periodisch Routing-Nachrichten erhalten. Diese Nachrichten enthalten im Wesentlichen eine Liste mit Routing-Einträgen. In diesen Routing-Einträgen ist der Sender als Next Hop gesetzt. Es wird zu jedem Subnet, welches der Next Hop erreichen kann, nur jeweils ein Eintrag gesendet, nämlich der mit der besten Metrik. Nach Erhalt einer Routing-Nachricht wird die Liste von Routing-Einträgen durchlaufen und für jeden Eintrag zuerst geprüft, ob bereits Routen zu dem Netzwerk vorhanden sind. Wenn dies nicht der Fall ist, dann wird, ähnlich wie bei einer neuen Routing-Tabelle, eine neue Liste mit diesem Eintrag und dem Subnet als Schlüssel in der HashMap hinterlegt. Sind jedoch schon andere Routen zu diesem Netzwerk vorhanden, so wird die vorhandene Liste geladen. Sollte bereits eine Route mit dem selben Next Hop, Ziel-Netzwerk und Netzwerk zur Übertragung vorhanden sein, so wird diese einfach gelöscht. Anschließend wird in jedem Fall der neue Eintrag an die Liste angehängt. In allen Fällen ist dabei zu beachten, dass der Hop Count inkrementiert wird. Abschließend muss die geänderte Liste noch sortiert werden. Dabei wird anhand der Metrik und ID sortiert, damit zum einen die besten Einträge oben stehen und zum anderen die Sortierung deterministisch ist. Sollte eine Änderung in der Routing-Tabelle dazu geführt haben, dass sich die minimale Metrik für ein Netzwerk geändert hat, so wird sofort eine Routing-Nachricht an alle benachbarten Router gesendet.

Da es sein kann, dass ein Router während der Simulation ausfällt oder sich seine Verbindungen so stark verschlechtern, dass keine Rounting-Nachrichten mehr von ihm empfangen werden, muss überwacht werden, ob die gespeicherten Routen noch aktuell sind. Dabei wurde sich an dem Protokoll RIPv2 orientiert. Dabei werden alle 30 Sekunden Routing-Nachrichten an andere Router geschickt. Erhält man für 180 Sekunden keine Updates mehr von diesem Router, so sollen diese Routen gelöscht werden. Der Prozess der Überwachung sowie der des Löschens werden im weiteren Verlauf diesen Kapitels genauer beschrieben.

Finden von Routing-Einträgen

Prinzipiell würde es genügen, wenn man aus der HashMap die zum Ziel-Netzwerk passende Liste abruft und dann den ersten Eintrag daraus lädt. Aufgrund der deterministischen Sortierung würde

das aber bedeuten, dass unabhängig davon, wie viele gleichwertige beste Routen man zur Verfügung hat, immer die selbe genommen werden würde. Jedoch ist wünschenswert, dass zufällig zwischen gleichwertigen Routen gewechselt wird, um auch die Last im Netzwerk besser zu verteilen. Daher gliedert sich die Ermittlung einer passenden Route in zwei Stufen.

In der ersten Stufe soll eine Liste aller Einträge mit der besten Metrik generiert werden. Dem Router-Modul ist der Empfänger bekannt. Vom Router-Connector wird eine Liste mit den Netzwerken geholt, zu denen der Empfänger direkt verbunden ist. Für jedes dieser Netzwerke können Routing-Einträge in der Routing-Tabelle vorhanden sein. Daher werden für jedes der Netzwerke alle Routing-Einträge mit der minimalen Metrik (für das entsprechende Netzwerk) geladen und in eine gemeinsame Liste kopiert. Anschließend wird diese Liste sortiert, wodurch klar ist, dass der erste Eintrag auch die geringste Metrik haben muss, unabhängig davon, ob alle dieselbe Metrik haben oder nicht. Da nun die minimale Metrik von allen möglichen Routen über alle möglichen Netzwerke bekannt ist, kann man alle übrigen Routing-Einträge mit einer schlechteren Metrik aus der gemeinsamen Liste entfernen.

In der zweiten Stufe wird, falls nicht die gesamte Liste benötigt wird, einer der besten Routing-Einträge zufällig ausgesucht. Dabei wird, wie im ganzen Simulator, Pseudo-Zufall verwendet, damit alle Simulationen deterministisch und reproduzierbar sind. Dieser Routing-Eintrag kann nun über die entsprechende Schnittstelle vom Switcher geladen werden.

Löschen von Routing-Einträgen

Routing-Einträge müssen dann gelöscht werden, wenn ein Router offensichtlich nicht mehr erreichbar ist. Dabei spielt keine Rolle, ob der Grund dafür Überlast ist oder ob der Router einfach offline ist. Wie zuvor bereits beschrieben, werden Routing-Updates alle 30 Sekunden versendet und nach 180 Sekunden ohne neue Updates sollen diese Routen wieder gelöscht werden. Dafür muss man die Verbindung zu einem Router überwachen. Zu diesem Zweck hat jedes Router-Modul eine zusätzliche HashMap, in der zu jedem bekannten Router die NetID im jeweiligen Netzwerk als Schlüssel und der Zeitpunkt der letzten Routing-Nachricht gespeichert wird. Erhält man nun eine Routing-Nachricht, wird zuerst geprüft, ob der Router bereits bekannt ist. Falls nicht, dann wird ein neuer Eintrag in der HashMap angelegt und ein Timer von 180 Sekunden gestartet, welcher am Ende prüft, ob in der Zwischenzeit neue Routing-Nachrichten von diesem Router empfangen wurden. Sollten bis dahin keine Nachrichten eintreffen, dann werden die Routen gelöscht, andernfalls wird wieder ein neuer Timer gestartet. Sollte der Router bereits überwacht werden, so wird nur der Zeitstempel in der HashMap aktualisiert, da im Hintergrund bereits ein Timer laufen muss.

Wenn also die Routen über einen Router gelöscht werden müssen, so wird aus der HashMap der

Routing-Tabelle die Liste für das entsprechende Netzwerk, über das man den Router nicht mehr erreicht, geladen. Aus dieser Liste müssen nun alle Einträge mit dem zu löschenden Router als Next Hop entfernt werden. Da man in Java nicht gleichzeitig über die Einträge einer verketteten Liste iterieren und Einträge aus ihr löschen kann, wird eine neue Liste erstellt, in die alle noch korrekten Einträge kopiert werden. Anschließend ersetzt man die alte Liste durch die neue. Zum Schluss wird der gelöschte Router auch aus der Überwachung entfernt und an alle benachbarten Router eine Routing-Nachricht gesendet. Die Erstellung dieser wird im folgenden Abschnitt erklärt.

Erstellen von Routing-Updates

Routing Updates werden in Form einer Liste von Routing-Einträgen mit Hilfe einer Routing-Nachricht versendet. Dies geschieht alle 30 Sekunden sowie bei einer relevanten Änderung der Metrik oder dem Wegfall einer Route in der Routing-Tabelle.

Die aktuellen Routing-Einträge sollen in alle Netzwerke gesendet werden, mit denen der Host direkt verbunden ist. Dabei soll aber auch sichergestellt werden, dass keine Routing-Schleifen entstehen. Aus diesem Grund wird für jedes Netzwerk eine eigene Routing-Nachricht erstellt. Zur Erstellung der Routing-Nachricht wird über die Routing-Tabelle bzw. HashMap iteriert. Dabei wird jedes mal die vollständige Liste an Routing-Einträgen für ein Subnet geladen. Über diese Liste wird nun auch iteriert, solange bis sich ein Eintrag findet, in dem weder Ziel-Netzwerk, noch das Netzwerk, über welches geroutet wird, dem Netzwerk gleichen, das die Routing-Nachricht erhalten soll. Sobald sich für ein Subnet ein solcher Eintrag gefunden hat, wird die Iteration abgebrochen und aus dem Eintrag ein neuer Routing-Eintrag für die Routing-Nachricht erstellt. Alle erstellten Routing-Einträge werden in einer gemeinsamen Liste gesammelt, die pro Ziel-Netzwerk maximal einen Eintrag enthält. Zum Schluss sendet das Router-Modul die so erstellte Routing-Nachricht über den Switcher als Broadcast in das entsprechende Netzwerk.

4.2.3 Schnittstellen

Das Router-Modul selbst besitzt nur sehr wenige Schnittstellen, die von außen benutzt werden. Dazu gehört zum einen die Schnittstelle, die eine Liste der vom Switcher verwalteten Network Layer entgegen nimmt und daraus die ersten Routing-Einträge erstellt. Zum anderen gibt es zwei Schnittstellen, um anhand einer NetID eine Route in Form eines einzelnen Routing-Eintrags oder eine Liste aller Routen mit minimaler Metrik zu erhalten.

Außerdem kann über das Router-Modul auf den Router-Connector zugegriffen werden, beispielsweise um Pakete zu zählen oder Network Layer wieder aus einer Broadcast-Liste (4.3.1) zu löschen.

4.3 Router-Connector

Der Router-Connector hat in der entwickelten Implementierung hauptsächlich vier Aufgaben. Zuerst soll er dem Switcher-Modul dabei helfen, Broadcasts zu realisieren. Außerdem soll es möglich sein, anhand einer NetID zu erfahren, zu welcher globalen NetID diese gehört, an welche Netzwerke ein Host angeschlossen ist und wie seine jeweilige NetID in diesem lautet. Diese Daten werden alle zu Beginn der Simulation gesammelt. Dies ist möglich, da das Router-Modul auf jedem Host eine Liste mit allen Network Layern hat und diese an den Connector übergeben kann. Dieser extrahiert dann alle gewünschten Daten. Darüber hinaus können mit Hilfe des Connectors gesendete, empfangene und weitergeleitete Pakete leicht gezählt werden. Der folgenden Abschnitt erklärt, wie diese Daten innerhalb des Connectors gespeichert werden.

4.3.1 Interne Datenstrukturen

Broadcast-Liste

Die Broadcast-Liste wurde durch eine HashMap realisiert, die als Schlüssel das Subnet benutzt und als Wert eine Liste mit allen lokalen NetIDs innerhalb des Subnets zurückgibt. Sobald sich ein Router-Modul beim Connector anmeldet, übergibt es eine Liste mit allen Network Layern und der globalen NetID. Für jeden der Network Layer wird geprüft, ob sein Subnet in der HashMap bereits vorhanden ist. Falls ja wird die NetID des Network Layers an die bestehende Liste angehängt, andernfalls wird eine neue Liste erzeugt und mit dem Subnet als Schlüssel in die HashMap eingetragen.

Globale NetID

Zur Bestimmung der globalen NetID anhand einer beliebigen NetID eines Hosts, wird auch eine HashMap benutzt. Als Schlüssel werden dort die NetIDs der Network Layer verwendet und als Wert die globale NetID. So kann die globale NetID in einem einzigen Schritt gefunden werden.

Netzwerke eines Hosts

Um die Netzwerke eines Hosts, sowie seine lokalen NetIDs anhand der globalen NetID ermitteln zu können, ist eine komplexere Struktur nötig. Sie besteht außen aus einer HashMap, welche als Schlüssel die globalen NetIDs der Hosts verwendet. Als Wert wird hier jeweils eine weitere HashMap zurückgegeben. Diese benutzt als Schlüssel nun die Subnets, mit welchen der Host verbunden ist, sowie als Wert eine verkettete Liste mit allen NetIDs, die er im jeweiligen Subnet hat.

Paketzähler

Die Zählung von Paketen wurde sehr simpel realisiert. Dafür gibt es für gesendete, weitergeleitete und empfangene Pakete jeweils ein einfaches Integer. Die Werte können durch globale Methoden vom Switcher inkrementiert oder abgefragt werden.

4.3.2 Schnittstellen

Der Router-Connector bietet den anderen Modulen Schnittstellen an, um folgende Daten zu erfragen

- `getGlobalID(NetID local)`

  Globale NetID eines Hosts anhand einer lokalen NetID

- `getSubnetsOfHost(NetID hostID)`

  Alle Netzwerke eines Hosts

- `getNetIDinSubnet(NetID globalID, Object Subnet)`

  Die lokale NetID eines Hosts innerhalb eines bestimmten Subnets

- `getHostsInSubnet(Object subnet)`

  Eine Liste von allen Hosts mit ihren lokalen NetIDs innerhalb eines Subnets

4.4 Verwendung

Die Verwendung des Network Switchers wurde so einfach wie möglich gehalten. Er wird genau wie alle anderen Layer der Simulation im selben XML-File konfiguriert. Um die Verwendung zu ermöglichen, muss folgende Zeile zum XML-File zu den definierten Layern hinzugefügt werden.

```
1 <Switcher class="org.peerfact.impl.networkswitcher.
    DefaultNetworkSwitcherFactory" />
```

Es können auch beliebig viele Network Layer geladen werden. Wichtig ist nur, dass der Switcher dann als Layer Zwischen den Network Layern und dem Transport Layer eingebunden wird. Das kann beispielsweise folgendermaßen aussehen:

```
1  <Host groupID="7" size="1">
2          <NetLayer7 />
3          <NetLayer5 />
4          <NetLayer6 />
5          <Switcher />
6          <TransLayer />
7          <Overlay />
8          <Application />
9          <Properties enableChurn="$churn" />
10         <Properties enableIsolation="false" />
11 </Host>
```

Da die Network Layer die tiefste Schicht darstellen, die konfiguriert werden kann, muss man sie in der Kombination mit dem Switcher auch gleichzeitig als direkte Netzwerke zwischen Hosts ansehen. In Abbildung 4.1 ist ein Beispiel eines Netzwerks zu sehen, welches aus drei verschiedenen Netzwerken besteht. Das folgende Listing zeigt, wie eine mögliche Konfiguration zu dem Netzwerk aus Abbildung 4.1 aussieht.

Beispielkonfiguration von Hosts mit dem Network Switcher

```
1  <Host groupID="1" size="1">
2          <NetLayer1 />
3          <NetLayer3 />
4          <Switcher />
5          <TransLayer />
6          <Overlay />
7          <Application />
8          <Properties enableChurn="$churn" />
9          <Properties enableIsolation="false" />
10 </Host>
11
12 <Host groupID="2" size="1">
13         <NetLayer1 />
14         <NetLayer2 />
15         <Switcher />
16         <TransLayer />
17         <Overlay />
18         <Application />
19         <Properties enableChurn="$churn" />
20         <Properties enableIsolation="false" />
21 </Host>
22
23 <Host groupID="3" size="1">
24         <NetLayer2 />
25         <NetLayer3 />
26         <Switcher />
27         <TransLayer />
28         <Overlay />
29         <Application />
30         <Properties enableChurn="$churn" />
31         <Properties enableIsolation="false" />
32 </Host>
```

Es ist zu sehen, dass es drei Hosts und drei Netzwerke gibt, jedoch jeder Host nur mit zwei davon direkt verbunden ist.

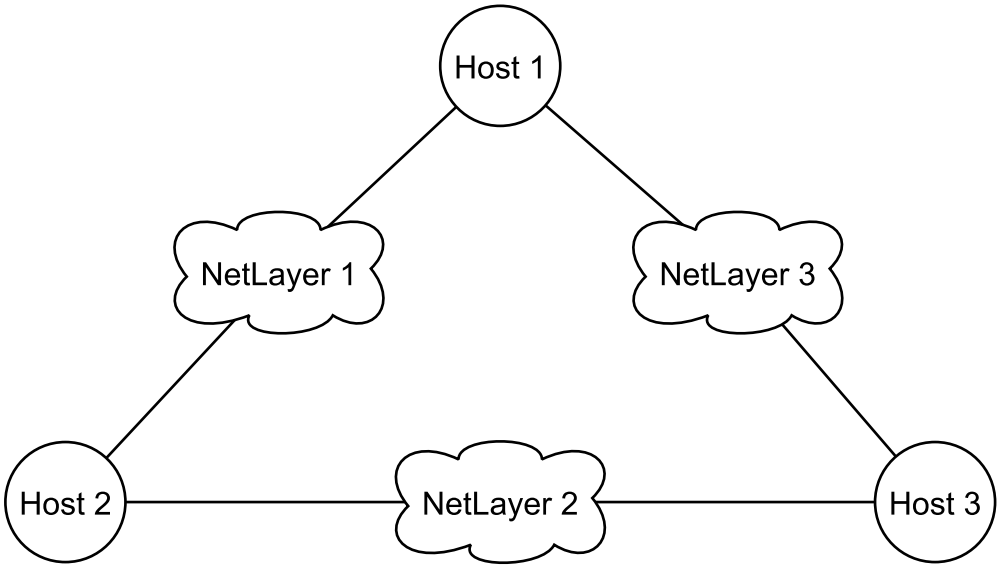


Abbildung 4.1: Ein Beispiel eines einfachen Netzwerks mit Verwendung des Network Switchers

Dabei können alle drei Network Layer in allen Aspekten vollkommen unterschiedlich konfiguriert werden. Es lassen sich auch *ModularNetLayer* und *SimpleNetLayer* miteinander kombinieren. An den Beispielkonfigurationen ist zu sehen, dass die Verwendung des implementierten Network Switchers nur wenig Einarbeitungszeit erfordert, da er sich automatisch konfiguriert und eigenständig alle nötigen Routen findet.

Kapitel 5

Analyse und Auswertung

In diesem Kapitel soll die Implementierung aus Kapitel 4 getestet und analysiert werden. Dazu werden in Abschnitt 5.1 zunächst einige Testszenarien konstruiert und simuliert. Zum Schluss werden in Abschnitt 5.2 zum einen die Simulationsergebnisse und zum anderen die Ergebnisse und Probleme der Implementierung ausgewertet.

5.1 Funktionstest des Network Switchers

Um die korrekte Übermittlung von Nachrichten bzw. Paketen durch den Network Switcher zu zeigen, wurde vor den Tests eine kleine Test-Applikation geschrieben, welche leicht im Simulator verwendet werden kann. Die Test-Applikation wird auf allen Hosts in der Simulation eingesetzt. Sie wird einmal pro Minute eine Ping-Nachricht per UDP an jeden Host der Simulation, der online sein sollte, senden. Die erste Ping-Nachricht wird dabei nach Ablauf der ersten Minute gesendet, die letzte zu Beginn der letzten Minute. Der andere Host wird mit einer Pong-Nachricht antworten. Sollte der sendende Host nach einer Minute keine Antwort erhalten haben, wiederholt er die Übertragung maximal drei Mal. Sollte der Ping-Pong-Versuch insgesamt vier Mal fehlschlagen, so wird eine entsprechende Fehlermeldung mit einem Zeitstempel ausgegeben. Zusätzlich zählt der Network Switcher selbständig alle vom Transport Layer versendeten und empfangenen Pakete und gibt das Ergebnis am Ende der Simulation aus.

Jedes der folgenden Testszenarien wurde, wenn nicht anders erwähnt, insgesamt zehn Mal mit jeweils unterschiedlichen Seeds simuliert. Für die Auswertung werden wir somit die gemittelten Werte aus jeweils zehn Durchläufen betrachten. Zu jedem Szenario werden außerdem die relevanten Parameter angegeben. Dazu gehören

- Simulationsdauer in Stunden
- Churn (aus/statisch/dynamisch)
- Packetloss (aus/statisch)
- TrafficQueue (infinite/bounded)
- Latenz (statisch)

Bei statischem Churn wechselt der Host im 15 Minuten Takt zwischen den Zuständen online und offline. Mit dynamischem Churn ist hier genauergesagt das KadChurnModel gemeint. Es erzeugt bei unterschiedlichen Seeds auch unterschiedlichen Churn bei einem Host.

Wenn statischer Packetloss aktiviert ist, gehen in dem entsprechenden Subnetz mit einer konstanten Wahrscheinlichkeit Pakete verloren.

Besonders wichtig ist die Wahl der TrafficQueue. Ist diese als `infinite` konfiguriert, werden auch bei einer Überlast keine Pakete verworfen. Dadurch kann aber die Latenz eines Paketes enorm ansteigen. Eine `bounded` TrafficQueue entspricht im Wesentlichen der realen Implementierung. Wird ein bestimmter Wert an gepufferten Paketen erreicht, müssen alle weiteren verworfen werden.

Für die folgenden Tests wird immer eine statische, also konstante, Latenz verwendet.

5.1.1 Szenario 1 - Forwarding von Nachrichten

Das erste Szenario in Abbildung 5.1 soll aus drei Hosts bestehen, wobei Host 1 nur über Host 2 Nachrichten an Host 3 versenden kann. Es gibt also zwei Subnetze, die nur durch Host 2 miteinander verbunden sind. Host 1 versendet pro Minute Ping-Nachrichten an die anderen beiden Hosts. Dadurch werden direkt drei Dinge getestet. Zum einen wird gezeigt, dass Nachrichten korrekt gekapselt und wieder entkapselt werden, egal ob sie irgendwo weitergeleitet wurden oder nicht. Des weiteren muss, damit Host 3 Nachrichten von Host 1 erhält und korrekt antworten kann, das Versenden und Verarbeiten von Routing-Nachrichten funktionieren, sowie das Forwarding der Nachrichten in Host 2.

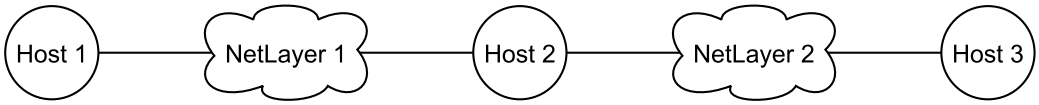


Abbildung 5.1: Aufbau eines Netzwerks zum Testen des Forwardings

Dazu wurde mit den Parametern aus Tabelle 5.1 simuliert.

Simulationsdauer	5000h
Churn	aus
Packetloss	aus
TrafficQueue	infinite
Latenz	statisch

Tabelle 5.1: Forwarding von Nachrichten - Parameter der Simulation

Bei allen zehn Simulationsdurchläufen ergaben sich folgende Werte

Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in %
1199996	1199996	0	0,000%

Die Anzahl an gesendeten Paketen entspricht genau dem zu erwartenden Wert. Pro Minute wird an zwei Hosts jeweils ein Ping versendet. Darauf folgt jeweils ein Pong. Das ergibt 4 Nachrichten pro Minute. Abzüglich der ersten Minute ergibt sich dann der Wert aus der Simulation. Dieser Wert bedeutet, dass also auch Pakete an Host 3 gesendet wurden. Da keinerlei Pakete verloren gegangen sind, kamen sowohl die Ping-Nachrichten und ihre Pong-Antworten korrekt an. Somit muss das Forwarding in Host 2 einwandfrei funktioniert haben. Da auf alle Ping-Nachrichten korrekt geantwortet werden konnte, muss auch die Kapselung der Nachrichten wie gewünscht funktioniert haben. Damit Host 2 überhaupt Nachrichten weiterleiten konnte, mussten Host 1 und 3 die Route über Host 2 kennen. Das bedeutet, dass Host 2 seine Routing-Nachrichten korrekt versendet hat und diese auch von Host 1 und 3 verarbeitet wurden. Sowohl der Network Switcher als auch die TestApp haben sich in diesem Szenario also wie gewünscht verhalten.

5.1.2 Szenario 2 - Unerreichbares Ziel

Das zweite Szenario soll dem Aufbau aus Abbildung 5.1 gleichen. Diesmal wird Host 2 nach der ersten Stunde von statischem Churn betroffen sein. Das heißt Host 2 wird abwechselnd 15 Minuten online und dann 15 Minuten offline sein. Somit wird es zu Offline-Zeiten keine Route von Host 1 zu Host 3 geben, es sollten also zu dieser Zeit auch keinerlei Pakete ankommen. Auch hier wird Host 1 wieder pro Minute eine Ping an alle Hosts senden, die anscheinend online sind.

Es wurde mit den Parametern aus Tabelle 5.2 simuliert.

Simulationsdauer	5000h
Churn	statisch (15 min) an Host 2 nach 1h
Packetloss	aus
TrafficQueue	infinite
Latenz	statisch

Tabelle 5.2: Unerreichbares Ziel - Parameter der Simulation

Bei allen zehn Simulationsdurchläufen ergaben sich folgende Werte

Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in %
1219985	640104	579881	$\approx$ 47,532%

Die Anzahl der empfangenen Pakete war so zu erwarten. In der ersten Stunde werden 59 Minuten lang erfolgreich Ping-Nachrichten gesendet. Erst in der zweiten Stunde beginnt Churn eine Rolle zu spielen. Theoretisch ist Host 2 somit nur 30 Minuten pro Stunde online. Das ergibt bereits 600116 empfangene Pakete. Zusätzlich muss bedacht werden, dass die TestApp auf Host 1 ständig versucht Pakete an Host 3 zu senden, da dieser prinzipiell online ist. Dadurch entstehen auch Übertragungswiederholungen. Solange Host 2 noch offline ist, kommen diese natürlich nicht an. In dem Moment wo Host 2 wieder online ist und somit auch die Route wieder verfügbar ist, versucht Host 1 nicht nur den neuen Ping an Host 3 zu senden, sondern auch die Übertragungswiederholungen der letzten drei Minuten. Das wären vier Nachrichten die Host 1 versucht gleichzeitig zu senden. Gesendet werden kann allerdings erst wieder, wenn alle Routen auch in der Routing-Tabelle vorhanden sind. Aufgrund der Latenz der Übertragung von Routing-Nachrichten, wird dies aber zu dem Zeitpunkt noch nicht

der Fall sein. Es kann also erst wieder in einer Minute übertragen werden. Da eine Übertragungswiederholung nun weg fällt aber eine neue hinzukommt, bleibt es bei vier Paketen, die gesendet werden sollen. Davon wurde der periodische (neue) Ping bereits gezählt, ebenso der Ping, dessen erste Übertragungswiederholung jetzt stattfinden soll. Somit bleiben noch zwei Pakete über, welche noch nicht mitgezählt wurden. Auch auf diese beiden Pakete wird Host 3 antworten. Somit ergeben sich insgesamt vier weitere empfangene Pakete pro Wechsel von offline zu online. Insgesamt gibt es $4999 * 2 - 1$ solcher Wechsel. Damit ergeben sich weitere 39988 Pakete. Das heißt, die Anzahl der empfangenen Pakete ist korrekt. Somit wurde gezeigt, dass Nachrichten ohne einen funktionierenden Router nicht von einem Netzwerk in ein anderes gelangen können. Gleichzeitig ist zu erkennen, dass der Network Switcher die Pakete wieder korrekt versenden kann, wenn eine Route nach einem Ausfall wieder verfügbar ist.

5.1.3 Szenario 3 - Zwei gleichwertige Routen und Ausfallerkennung

In diesem Szenario soll getestet werden, ob der Network Switcher sich korrekt verhält, wenn Routen plötzlich nicht mehr verfügbar sind, aber eine andere gleichwertige Route vorhanden ist. Dazu reicht das Netzwerk aus Abbildung 5.2. Es besteht aus insgesamt vier Hosts, wovon Host 2 und Host 3 als Router bzw. Gateways zwischen den beiden Subnetzen fungieren sollen. Host 2 wird, wie im zweiten Szenario, von statischem Churn betroffen sein. Es wird sich zeigen, ob der Network Switcher korrekt und zügig auf den Ausfall eines Routers reagieren kann.

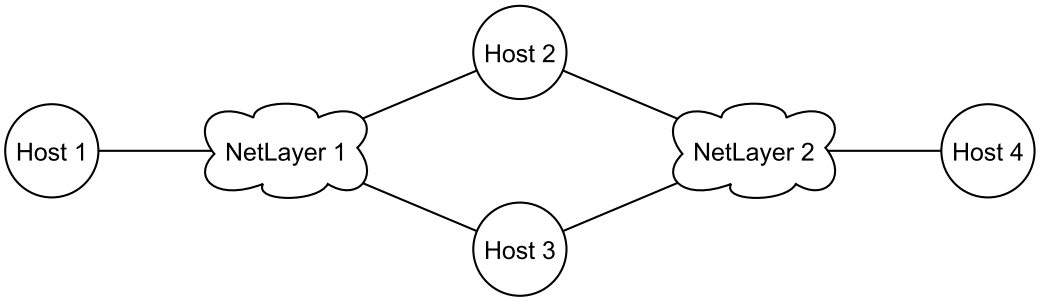


Abbildung 5.2: Aufbau eines Netzwerks mit gleichwertigen Routen

Zur Simulation wurden folgende Parameter aus Tabelle 5.3 verwendet.

Simulationsdauer	2h
Churn	statisch (15 min) an Host 2
Packetloss	aus
TrafficQueue	infinite
Latenz	statisch

Tabelle 5.3: Zwei gleichwertige Routen und Ausfallerkennung - Parameter der Simulation

Nach zehn Simulationen ergaben sich die folgenden Durchschnittswerte

Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in %
675,9	657	18,9	$\approx$ 2,796%

Da in dieser Simulation die Anzahl der verlorenen Pakete davon abhängt, welche Route von Host 1 bzw. Host 4 zum Zeitpunkt des Ausfalls von Host 2 gewählt wurde, lässt sich die genaue Anzahl nicht vorhersagen. Host 2 ist insgesamt ca. 25% der Zeit offline. Während dieser Zeit würde ohne eine Ausfallerkennung des Network Switchers eine 50% Wahrscheinlichkeit bestehen, dass Pakete von Host 1 zu Host 4 und umgekehrt verloren gehen. Diese stellen aber nur 50% aller zu dem Zeitpunkt gesendeten Pakete dar, da Host 3 noch Pings empfangen und beantworten kann. Insgesamt könnte man also dann mit Paketverlustraten von bis zu 12,5% rechnen. Gemessen wurden deutlich weniger. Das bedeutet also, dass die noch funktionierende Route bevorzugt genutzt wurde. Da aber zufällig zwischen zwei gleichwertigen funktionierenden Routen gewechselt wird, muss also erkannt worden sein, dass die Route über Host 2 ausgefallen ist.

Die wenigen, trotzdem vorhanden Paketverluste erklären sich dadurch, dass Host 1 und Host 4 ihre Routen über Host 2 erst nach drei Minuten ohne empfangene Updates löschen. In dieser Zeit können beide versuchen noch Pakete über Host 2 zu routen. Dazu kommen dann auch die nötigen Übertragungswiederholungen, die erneut über Host 2 geroutet werden könnten. Zusätzlich ist an den TIMEOUT-Meldungen der TestApp zu sehen, dass der Timeout maximal vier Minuten nach dem Churn auftritt. Ein Timeout nach vier Minuten bedeutet, dass die letzte Nachricht vor drei Minuten gesendet wurde, da noch eine weitere Minute auf ein Pong gewartet wird.

Insgesamt ist aber deutlich zu erkennen, dass der Wechsel zwischen Routen und die Ausfallerkennung funktioniert. Unterstrichen wird das durch die Ergebnisse von Szenario 4.

5.1.4 Szenario 4 - Wechsel zwischen gleichwertigen Routen und Einfluss auf Paketverluste

In diesem Szenario (Abbildung 5.3) wird eine ähnliche Netzwerk-Topologie wie in Szenario 3 vorausgesetzt. Lediglich Host 3 wird durch eine Gruppe von 19 Hosts ersetzt und die Simulationszeit auf 10 Stunden erhöht. Somit stehen Host 1 und Host 4 jeweils 20 gleichwertige Routen zur Verfügung. Da weiterhin nur Host 2 von Churn betroffen ist, sollte die Paketverlustrate im Vergleich zu Szenario 3 stark reduziert werden, obwohl die Ausfallzeit von Host 2 nun prozentual gesehen höher ist als in Szenario 3.

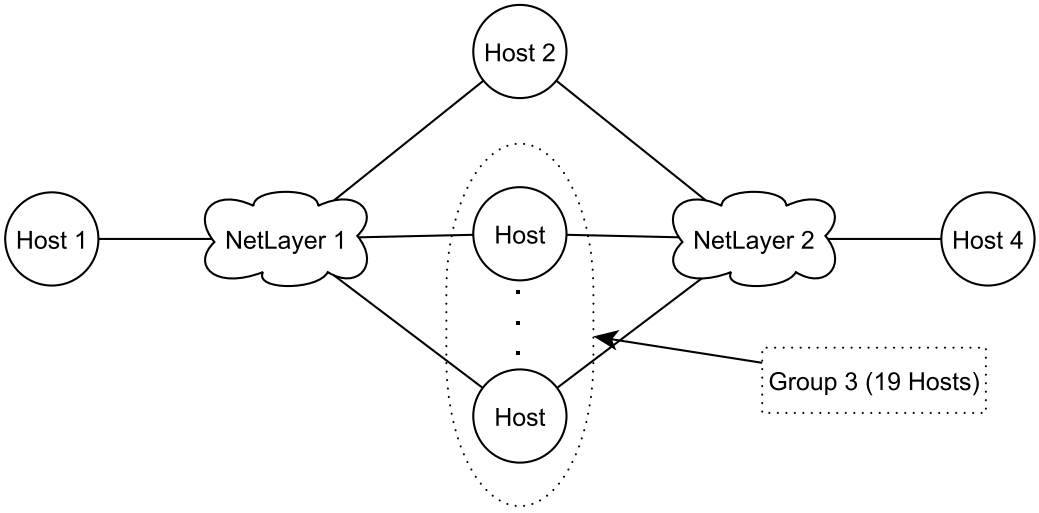


Abbildung 5.3: Aufbau eines größeren Netzwerks mit gleichwertigen Routen

Simuliert wurde mit den Parametern aus Tabelle 5.4

Simulationsdauer	10h
Churn	statisch (15 min) an Host 2
Packetloss	aus
TrafficQueue	infinite
Latenz	statisch

Tabelle 5.4: Viele gleichwertige Routen und Ausfallerkennung - Parameter der Simulation

Nach zehn Simulationen ergaben sich die folgenden Durchschnittswerte

Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in %
24652,2	24604,6	47,6	$\approx$ 0,193%

Wie zu erwarten war, ist die Paketverlustrate drastisch gesunken. Das bedeutet, dass auf jeden Fall korrekt zwischen gleichwertigen Routen gewechselt wird und somit weniger Pakete über Host 2 geroutet werden. Daher gehen dort pro Wechsel von online zu offline weniger Pakete verloren als vorher.

5.1.5 Szenario 5 - Ein komplexes Netzwerk mit möglichen Routing-Schleifen

Abschließend soll ein komplexes Netzwerk wie in Abbildung 5.4 getestet werden, in dem sowohl gleichwertige, als auch unterschiedlich bewertete Routen vorhanden sind. In dieser Simulation sendet sowohl die Gruppe 1 als auch der Host 6 periodische Ping-Nachrichten an alle anderen.

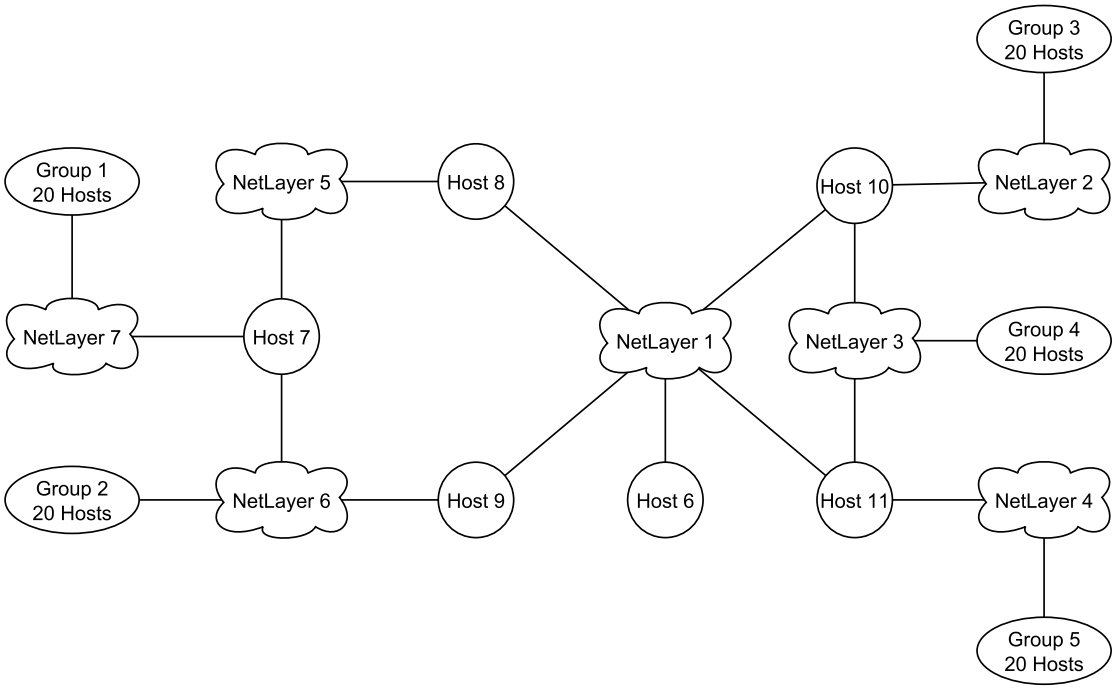


Abbildung 5.4: Ein größeres Netzwerk mit einer komplexen Topologie

Da nun mehr Hosts als Router vorhanden sind, werden die Router und Gateways zu den Engstellen im Netzwerk. Es wird interessant sein zu sehen, wie sich das auf die Simulationen auswirken wird. Dazu wird das Netzwerk mit vier verschiedenen Konfigurationen simuliert (Tabelle 5.5). Wir werden einmal komplett ohne Paketverlust simulieren, um zu zeigen, dass das Netzwerk einwandfrei funktioniert und auch keine Routing-Schleifen entstehen. Danach werden wir Paketverluste jeweils durch dynamischen Churn und durch statischen Packetloss simulieren.

Parameter	Durchlauf 1	Durchlauf 2	Durchlauf 3
Simulationsdauer	10h	10h	10h
Churn	aus	aus	dynamisch an Host 8
Packetloss	aus	statisch	aus
TrafficQueue	infinite	bounded	bounded
Latenz	statisch	statisch	statisch

Tabelle 5.5: Komplexes Netzwerk und mögliche Routing-Schleifen - Parameter der Simulation

Es wurde jede der Konfigurationen jeweils zehn Mal simuliert. Daraus ergaben sich im Schnitt die folgenden Werte

	Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in Prozent
ohne Packetloss	2641590	2641590	0	0,000 %
mit Packetloss	2700634,9	2661079,8	39555,1	$\approx$ 1,465 %
mit Churn	2641644,2	2626520,5	15123,7	$\approx$ 0,573 %

Tabelle 5.6: Zusammengefasste Messergebnisse

Im ersten Durchlauf sind, wie zu erwarten, keinerlei Pakete verloren gegangen, da kein Churn oder Packetloss aktiviert war und die TrafficQueue unendlich eingestellt war. In den beiden anderen Durchläufen kommt es erwartungsgemäß zu Paketverlusten. Außerdem wurden in keiner der Simulationen Routing-Schleifen erkannt. Das ist ein Zeichen dafür, dass der Split-Horizon Mechanismus korrekt funktioniert. Tatsächlich wurden während der Entwicklung des Network Switchers periodisch die Routing-Tabellen von allen Hosts ausgegeben und analysiert. Auch dort bestätigte sich diese Annahme.

Da Host 8, sobald er wieder online ist, sowohl von der Gruppe 1, als auch von Host 6 alle Pings erhält und beantwortet, waren offensichtlich zu dem Zeitpunkt die Routing-Tabellen der anderen Hosts wieder erfolgreich aufgebaut.

5.1.6 Szenario 6 - Ein komplexes Netzwerk mit Überlast

Nun soll noch getestet werden, wie schnell in einem Netzwerk Überlast entstehen kann und wie diese zu Paketverlusten führt. Dafür werden in dem Netzwerk aus Abbildung 5.4 die Gruppen auf jeweils 100 Hosts erweitert. Außerdem senden einfach alle Hosts zum Zeitpunkt 1m einen Ping an alle anderen verfügbaren Hosts. Packetloss und Churn sind vollständig deaktiviert. Einzig die TrafficQueue wird als bounded konfiguriert.

Es wurde also mit folgenden Parametern simuliert

Simulationsdauer	2h
Churn	aus
Packetloss	aus
TrafficQueue	bounded
Latenz	statisch

Tabelle 5.7: Überlast in einem komplexen Netz - Parameter der Simulation

Es ergaben sich nach zehn Simulationen die folgenden durchschnittlichen Messwerte

Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in %
944123,4	254339,6	689783,8	$\approx$ 73,061%

Es ist zu sehen, dass die Anzahl der gesendeten Nachrichten nicht besonders hoch ist. Dennoch kam es zu enormen Paketverlusten, weil alle Pakete schlagartig zur selben Zeit versendet wurden und sich somit an den Hosts zwischen mehreren Netzen Engpässe gebildet hatten. Eine Überlastsituation kann also in einem ungünstigen Fall das ganze Netzwerk lahm legen, denn wahrscheinlich wird bei einer Paketverlustrate von knapp 73% keine Anwendung und kein Overlay mehr korrekt funktionieren. Die

wenigen Pakete, die trotzdem ihr Ziel finden, sind die Pakete die innerhalb der äußeren Netzwerke innerhalb der Gruppen versendet wurden. Es hat also eine Art Isolation stattgefunden. Das ist eine Eigenschaft, die vorher bewusst simuliert werden musste.

5.1.7 Szenario 7 - Ein Peer-to-Peer Overlay auf komplexen Netzwerk

Abschließend sollte gezeigt werden, dass ein Peer-to-Peer Overlay einwandfrei auf einem Netzwerk, welches den Network Switcher benutzt, funktioniert. Dazu nehmen wir das Netzwerk aus dem dritten Durchlauf von Szenario 6, welches nur Paketverluste durch Churn eines Routers erzeugte, und ersetzen die TestApp durch ein Chord-Overlay [SMK+01] und eine Applikation die Lookups in einer Distributed-Hash-Table generiert. Zusätzlich werden die Gruppen auf jeweils 100 Hosts erweitert. Somit wird an den Routern eine höhere Last zu erwarten sein. Sowohl das Overlay, als auch die Applikation werden nur auf den Hosts in den Gruppen 1-5 laufen. Diese werden zwischen den Minuten 1 und 26 dem Overlay beitreten. Ab Minute 40 werden dann von Gruppe 1 die Lookups generiert.

Dazu wurde mit den folgenden Parametern simuliert. Die Simulation wurde nur einmal durchgeführt.

Simulationsdauer	10h
Churn	dynamisch auf Host 8
Packetloss	aus
TrafficQueue	bounded
Latenz	statisch

Tabelle 5.8: Chord Overlay auf einem komplexen Netzwerk - Parameter der Simulation

Die Simulation erzeugte folgende Resultate auf Netzwerkebene

Gesendete Pakete	Empfangene Pakete	Verlorene Pakete	in %
11492200	10528936	963264	$\approx$ 8,382%

Es ist zu erkennen, dass es eine relativ hohe Paketverlustrate von über 8% gegeben hat. Das ist zu einem geringen Teil auf den Churn von Host 8 zurückzuführen, da dieser in der Simulation nur zweimal

offline ging. Ein Großteil der Paketverluste wird durch die hohe Last an den Routern verursacht.

Umso interessanter ist es zu sehen, wie sich das Overlay verhalten hat. Dazu kontrollieren wir zuerst, ob alle Hosts, wie gewünscht, dem Overlay beigetreten sind (Abbildung 5.5). Danach betrachten wir die gestarteten, abgeschlossenen und fehlgeschlagenen Lookups.

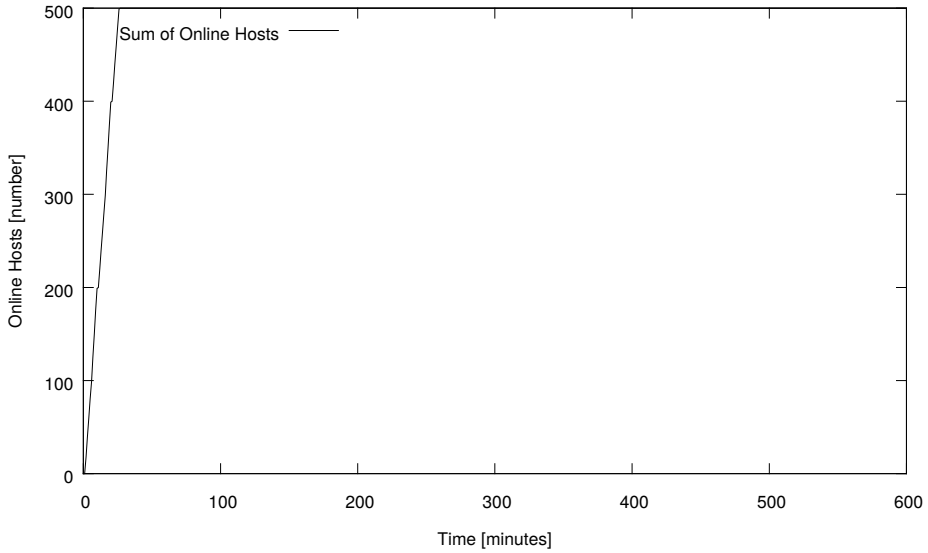


Abbildung 5.5: Anzahl der dem Overlay beigetretenen Hosts im zeitlichen Verlauf

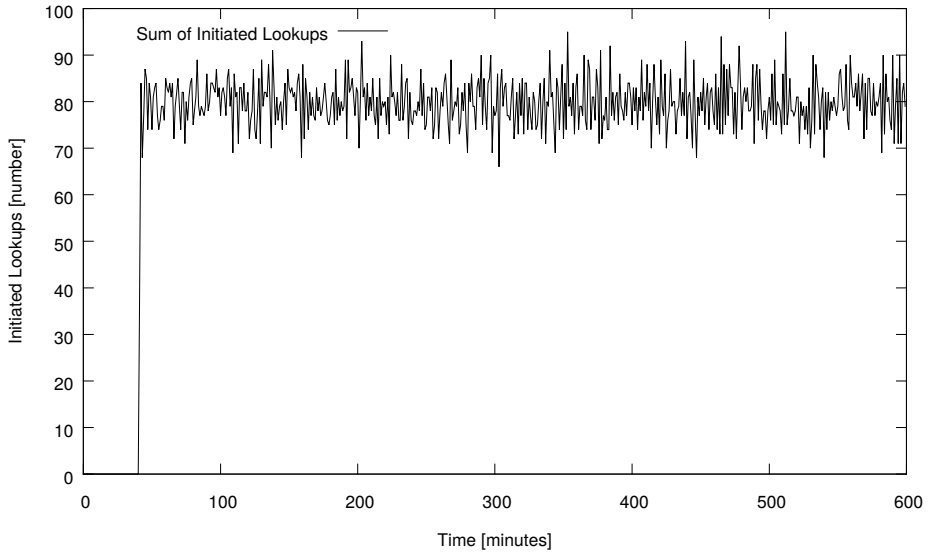


Abbildung 5.6: Anzahl der gestarteten Lookups im zeitlichen Verlauf

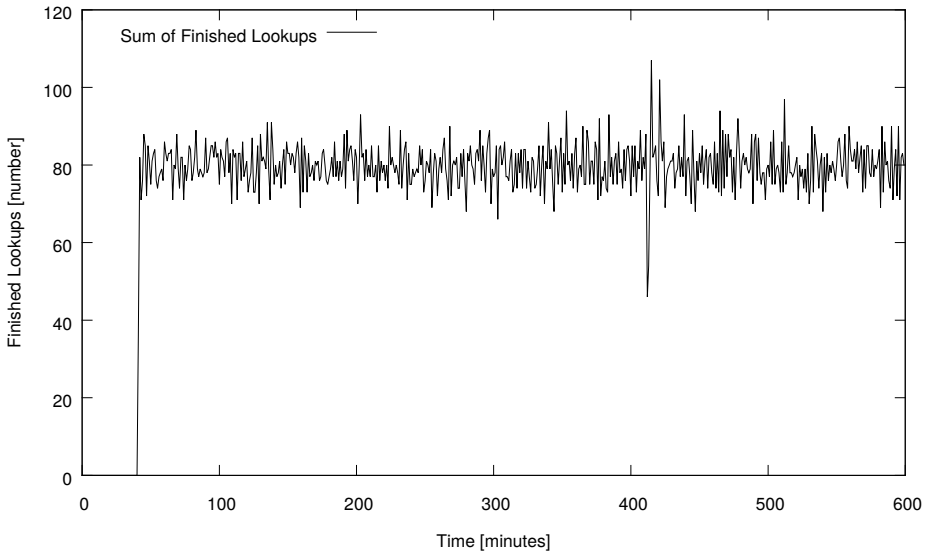


Abbildung 5.7: Anzahl der fertiggestellten Lookups im zeitlichen Verlauf

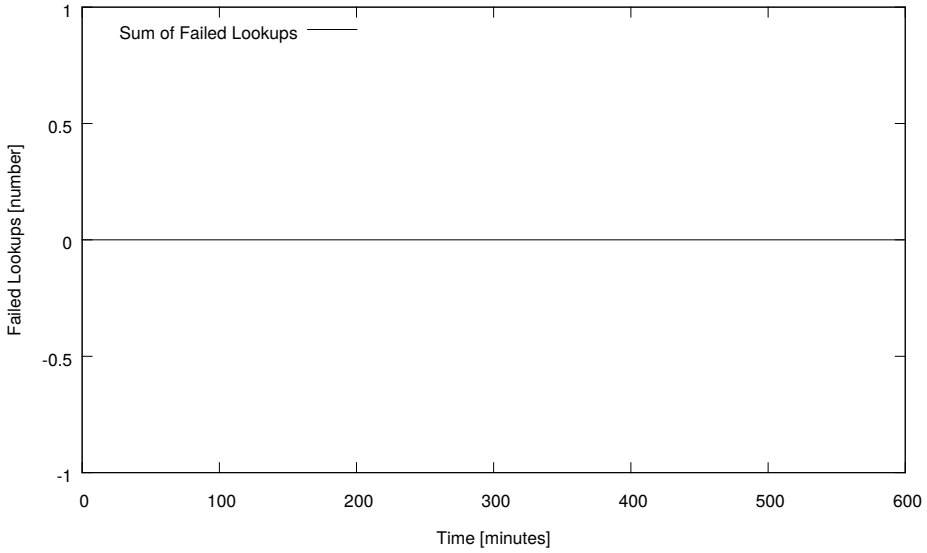


Abbildung 5.8: Anzahl der fehlgeschlagenen Lookups im zeitlichen Verlauf

In Abbildung 5.5 is erkennbar, dass alle 500 Hosts aus den Gruppen 1-5 dem Overlay zur gewünschten Zeit beigetreten sind. Das deutet schon auf eine prinzipiell funktionierende Verbindung zwischen den Netzwerken hin. Unterstrichen wird das von den Lookups. Zum einen ist in Abbildung 5.8 sofort zu erkennen, dass keinerlei Lookups fehlgeschlagen sind. Trotz relativ hoher Paketverluste von rund 8% funktionierte das Routing auf Netzwerkebene scheinbar immer noch gut genug, um einen stabilen Betrieb des Overlays zu ermöglichen. Vergleicht man nun noch die Verteilung von gestarteten und

fertiggestellten Lookups im zeitlichen Verlauf (Abbildung 5.6 und 5.7), so sieht man, dass die Verteilung, bis auf einen kleinen Ausreißer bei ca. 420 min, in beiden Graphen sehr ähnlich ist. Das deutet darauf hin, dass Pakete sehr zügig weitergeleitet werden und es auch nicht zu unnötigem Aufstauen von Paketen durch den Network Switcher kommt.

5.1.8 Zusammenfassung der Tests

Es wurden diverse Szenarien getestet, in denen der Network Switcher und sein Router-Modul auf verschiedene Ereignisse reagieren mussten. Es hat sich gezeigt, dass das Verhalten des Network Switchers in allen Fällen korrekt und nachvollziehbar war. In den Szenarien, in denen es zu keinen Paketverlusten kommen durfte, traten diese auch nicht ein. Waren hingegen Paketverluste zu erwarten oder bewusst provoziert worden, so traten diese in einem absolut nachvollziehbaren Umfang auf. Es zeigten sich während der Tests auch Effekte, wie wir sie aus realen Netzwerken, wie dem Internet, kennen. Durch hohe Lasten oder das Wegbrechen einer Verbindung kam es zu einer temporären Partitionierung des Netzwerkes, wobei einzelne Teile des großen Netzwerkes voneinander isoliert wurden. Dies kann natürlich auch durch die bewusste Abschaltung von *Border-Gateways* oder die Löschung ihrer Routen realisiert werden, wie es z.B. 2011 in Ägypten zu beobachten war [GA15].

Hier zeigt sich das hohe Potenzial der Verbindung verschiedener Netzwerkmodelle und Konfigurationen durch den Network Switcher innerhalb einer Simulation. Effekte wie die Isolation von Netzwerken lassen sich einerseits sehr leicht durch den Churn bestimmter Router erzeugen, andererseits können diese auch durch Überlasten auftreten. Die Entwicklung komplexer Modelle, die so etwas erledigen, entfällt. Dies kann besonders dann nützlich sein, wenn man ein möglichst realistisches Szenario zum Testen seiner Anwendung erstellen möchte, ohne zusätzlich neue Modelle für Paketverlust und Isolation entwickeln zu müssen. Durch den Network Switcher bietet sich die Möglichkeit, die ohnehin schon zahlreichen Konfigurationsmöglichkeiten des `ModularNetLayer` beliebig miteinander zu kombinieren und somit eine praktisch beliebig große, komplexe und detaillierte Simulationen zu ermöglichen.

Darüber hinaus werden durch den Network Switcher auch Simulationen und Analysen auf tieferen Schichten des Protokoll-Stapels interessant. So ließe sich beispielsweise Vergleichen, wie sich Änderungen in der Topologie eines Netzwerks auf Eigenschaften wie Paketverluste und Latenzen bei Last auswirken.

Durch den modularen Aufbau des Network Switchers, wäre prinzipiell auch die Erweiterung um weitere Routing Protokolle denkbar. Auch deren Verhalten und die Auswirkungen auf das Netzwerk und die darauf laufenden Anwendungen und Overlays ließen sich analysieren.

Abschließend kann man sagen, dass sich durch den Network Switcher viele neue Möglichkeiten und Anwendungsfälle von Simulationen in PeerfactSim.KOM ergeben.

5.2 Auswertung der Implementierung und Schwierigkeiten

Während der Implementierung zeigten sich immer wieder Probleme, die eine Kombination einiger Network Layer in ihrer jetzigen Form unterbinden. Diese Probleme sind wohl auf die Tatsache zurückzuführen, dass PeerfactSim.KOM ursprünglich nicht für die Verwendung unterschiedlicher Arten von Network Layern ausgelegt wurde. Diese Probleme werden nun genauer beleuchtet.

5.2.1 Probleme der bestehenden API und fehlerhafte Verwendung

Die API von Layern in PeerfactSim.KOM definiert eine Menge von Schnittstellen, die bei allen Arten eines Layers verfügbar sein muss. Dadurch wird sichergestellt, dass Layer auch universell untereinander austauschbar sind, ohne dass die angrenzenden Layer ihre Arbeitsweise ändern müssen. Dieses Prinzip gilt für praktisch alle Objekte und Komponenten innerhalb des Simulators. Jedoch wird vor allem in den Network Layern die API oft nicht korrekt verwendet. Auch, wenn es möglich und teilweise notwendig ist, für eine neue Art von Network Layer beispielsweise eine neue Art von NetID zu erstellen, so muss bedacht werden, dass es nicht ausreicht, dass die neue NetID die alte API unterstützt. Sobald innerhalb der neuen NetID oder des neuen Network Layers Methoden entstehen, die auch mit NetIDs anderer Hosts arbeiten, beispielsweise ein Vergleich derer, so muss strikt die allgemeine API benutzt werden. Geht man also nicht vom allgemeinen Typ `NetID`, sondern von `Foo-NetID` aus, so verlangt man implizit, dass jeder Host bzw. Network Layer eine `Foo-NetID` hat. So etwas passiert unter anderem in der Implementierung des Mobility-Models aus einer früheren Arbeit [Kor15]. In Zeile 23 des folgenden Listings ist so ein Fall zu sehen.

Aus org.peerfact.network.wireless.impl.DefaultMobilityHost.java

```
1  public void move(long CurrentTime) {
2          // + " Speed is: " + this.getSpeed());
3          movements++;
4          this.movementStartTime = CurrentTime;
5          setSpeed(getSpeed() * acc.getAcceleration());
6
7          long ct = Simulator.getCurrentTime();
8          if (CurrentTime < ct) {
9
10                 ct = CurrentTime;
11
12         }
13
14         if (this.movementEndTime > ct) {
15
16         } else {
17                 if (battery > 0.01) {
18
19                         inMove = true;
20
21                         oldPosition = position.clone();
22                         PositionVectorN desti = new PositionVectorN
                               (
23                         move.getNew(oldPosition, (MobilityNetID)
                               this.getHost()
24                         .getNetLayer().getNetID()));
25                         if (!oldPosition.equals(desti)) {
26                                 battery -= 0.01;
27                                 long dis = getDist(desti,
                                       oldPosition);
28                                 this.movementEndTime += (long)
                                       (1000000 * dis / getSpeed());
29                                 this.position = desti;
30
31                                 for (int jk = 0; jk < this.
                                       getDimensions(); jk++) {
32                                         this.setEntry(jk, desti.
```

```
                                                    getEntry(jk));
33                                          }
34
35                                  } else {
36                                          this.movementEndTime = Simulator.
                                                getEndTime();
37                                  }
38                          } else {
39                                  this.turnOff();
40                          }
41                          inMove = false;
42                  }
43  }
```

Die bisherige API des Simulators setzt von sich aus voraus, dass ein Host nur eine einzelne NetID besitzt. Daher wurde durch den Network Switcher eine globale NetID eingeführt, welche zur logischen Adressierung verwendet wird. In Zeile 23 des Listings wird die NetID des Hosts geholt und als Parameter für eine Funktion benutzt, die in jedem Fall eine MobilityNetID erwartet. Das kann nicht mehr garantiert werden, sobald ein Host mehrere Network Layer haben kann. Ähnlich verhält es sich, wenn Network Layer intern ihre eigene NetID mit anderen vergleichen, ohne zu prüfen, ob diese vom selben Typ ist. Eine Lösung kann sein, einer Funktion eine allgemeine NetID als Parameter vorzugeben und dann intern zu überprüfen, ob damit überhaupt gearbeitet werden kann. Das ist jedoch nicht immer sinnvoll, besonders wenn es dazu führen würde, dass wichtige Funktionen innerhalb des Network Layers häufig nicht ausgeführt werden können. In diesen Fällen sollte man überlegen, ob eine Erweiterung der vorhandenen allgemeinen API sinnvoll ist. Das hat jedoch zur Folge, dass die API auch entsprechend in allen vorhandenen Network Layern implementiert werden muss. In diesem Fall muss entschieden werden, ob diese Funktion von allgemeinem Interesse ist und, ob sie tatsächlich einen direkten Bezug zum betrachteten Objekt hat. Dies führt direkt zu einem weiteren Problem, welches im folgenden Abschnitt erklärt wird.

Position und Bewegung

Ein weiteres Problem zeigt sich bei der Verwendung einer geographischen Position. Diese wird aktuell noch in den meisten Fällen vom Network Layer bestimmt. Das funktioniert, solange das benutzte Modell auf allen Hosts gleich ist und pro Host nur einmal verwendet wird. Es ist klar, dass ein Network Layer auf die geographische Position angewiesen ist, beispielsweise um Latenzen, Signalstärken, Pa-

ketverluste etc. in Abhängigkeit davon zu simulieren. Jedoch wird die geographische Position doch durch die Position des ganzen Hosts bestimmt. Das bedeutet, auch wenn ein Host mehrere Network Layer besitzt, so müssen diese auch mit ein und der selber geographischen Position arbeiten, da sich ein Host zu einem Zeitpunkt auch nur an einem einzigen Ort befinden kann, inklusive aller seiner Network Layer. Es ist also zu sehen, dass Network Layer zwar durchaus von einer geographischen Position abhängig sein können, aber diese keinesfalls vorgeben sollten. Es erscheint sinnvoll, den Host um eine entsprechende Schnittstelle zur erweitern und somit die geographische Position sowie das Bewegungsmodell dabei in ein separates Modul des Hosts auszulagern.

Bandbreiten

Als problematisch erwies sich auch die Implementierung mancher Teile der bestehenden API. In Kapitel 4.1.1 wurde bereits besonders auf das Problem der Bestimmung der aktuellen Bandbreite eingegangen. Es zeigte sich, dass bei der Verwendung mehrerer Network Layer auch die Bestimmung der Bandbreite für jeden Network Layer einzeln erfolgen sollte. Dies ist zwar bei der bestehenden API möglich, jedoch nur solange kein transparenter Network Switcher eingesetzt wird, da der Host und seine Layer keine Kenntnis über das Vorhandensein mehrerer Network Layer haben. Hier zeigt sich, dass für eine nahtlose Integration der Verbindung verschiedener Network Layer in PeerfactSim.KOM eine Erweiterung bzw. Veränderung bestehender API, auch in anderen Layern, nötig sein wird.

Individuelle Analyzer

Um den Verlauf und die Ergebnisse einer Simulation besser nachvollziehen zu können, bietet PeerfactSim.KOM die Möglichkeit, verschiedenste *Analyzer* einzusetzen. Diese können auch einzelne Schichten analysieren. Häufig ist es nötig, für eine neue Art von Layer auch einen passenden Analyzer zu entwickeln. Um verschiedene Netzwerkmodelle innerhalb einer Simulation zu ermöglichen, muss sichergestellt werden, dass der Analyzer entweder passiv auf Informationen vom Network Layer wartet, oder aber erkennen kann, ob es sich überhaupt um einen zu überwachenden Network Layer handelt. Ansonsten wird wieder das exklusive Vorhandensein eines bestimmten Typs von Layer vorausgesetzt.

5.2.2 Problem der fehlenden Schichten unterhalb der Netzwerkschicht

Wie bereits in Kapitel 2 beschrieben wurde, simuliert PeerfactSim.KOM bis zur Vermittlungsschicht des OSI-Modells, also dem Network Layer. Dabei handelt es sich aber noch um eine Schicht, die mit einer logischen Adressierung auch über verschiedene physikalische Netzwerke hinweg arbeitet. Somit hat die Art der physikalischen Übertragung nichts mit der Vermittlungsschicht zu tun. Als man auch die Auswirkungen verschiedener physikalischer Übertragungsmodelle mit in die Simulation einbeziehen wollte, wurde der Simulator jedoch nicht um weitere *tiefere* Schichten erweitert, sondern man implementierte dies direkt in die Vermittlungsschicht. Man fasste damit die Funktionen von bis zu drei Schichten in einer Schicht zusammen. Daher schien wohl auch eine Unterscheidung zwischen logischer und physikalischer Adressierung nicht notwendig. Es zeigte sich während der Implementierung des Network Switchers, dass eine solche Unterscheidung nützlich gewesen wäre. Die Vermittlungsschicht des Simulator enthält also einige Simulationsfunktionen, die eigentlich aus anderen Schichten stammen, lässt jedoch einige Funktionen eines Protokolls wie IP oder ICMP vermissen. Zusätzliche Schichten im Simualtor müssen nicht automatisch bedeuten, dass die Simulation aufwendiger und langsamer wird. Sie tragen tendenziell dazu bei, dass die Funktionen klarer voneinander getrennt werden. Das führt außerdem dazu, dass ein einzelner Layer weniger aufwändige Operationen ausführen muss und sich dadurch wahrscheinlich auch seine Datenstrukturen vereinfachen. Das kann wiederum zu einer Vereinfachung der Operationen auf den Daten führen, was als Vorteil angesehen werden kann. Eine Erweiterung des Simulators um weitere Schichten, besonders wenn es darum geht, bereits vorhandene Funktionen klarer voneinander zu trennen, bedeutet also nicht automatisch auch eine Erhöhung des Rechenaufwands.

Kapitel 6

Zusammenfassung

In dieser Arbeit wurde ein neuer Layer für den Simulator PeerfactSim.KOM entwickelt, welcher grundsätzlich die Möglichkeit zur Verbindung verschiedener Netzwerkmodelle bietet. Anhand dieser Entwicklung wurde herausgearbeitet, welche Anforderungen der neue Layer und auch der bestehende Simulator erfüllen müssen. Dabei zeigten sich teilweise gravierende Probleme, die nur bedingt im Rahmen dieser Arbeit lösbar waren. Aus diesem Grund wurde sich auf eine möglichst transparent arbeitende Lösung konzentriert, welche die Eingriffe in den Simulator möglichst gering hält. Dennoch konnte anhand des entwickelten *Network Switchers* recht deutlich gezeigt werden, welches Potenzial die Verbindung verschiedener Netzwerke innerhalb einer Simulation bieten kann. Dadurch werden auch Simulationen für Bereiche außerhalb der Peer-to-Peer-Netzwerke interessant. Nach der Implementierung des Network Switchers wurde die korrekte Funktionsweise getestet und anschließend die Probleme und deren Ursachen analysiert. Es zeigte sich, dass die Ursachen teilweise in einer fehlerhaften Verwendung der API, aber auch zu einem großen Teil im bestehenden Design des Simulators liegen, da einerseits eine Vermischung von Layern vorhanden ist, andererseits die vorhandene API nicht mehr den aktuellen Anforderungen gerecht wird. Dennoch konnte der Simulator durch den transparenten Network Switcher um eine sehr umfangreiche Funktion erweitert werden, die nun eine Vielzahl an neuen Simulationsmöglichkeiten für verschiedenste Anwendungsfälle bietet.

6.1 Zukünftige Arbeiten

Da sich während dieser Arbeit gezeigt hat, dass das bestehende Design des Simulators die Möglichkeit von neuen Erweiterungen und deren Nutzung teilweise sehr stark eingrenzt, könnten die Ursachen davon in weiteren Arbeiten genauer analysiert und Konzepte für eine Erweiterung oder Änderung des aktuellen Designs erarbeitet werden. Dazu würde unter anderem die Erweiterung der bestehenden API um weitere sinnvolle Schnittstellen gehören. Außerdem könnte genauer untersucht werden, welche

Funktionen aus den vorhandenen Network Layern in andere oder neue Schichten ausgelagert werden sollten. Ein möglicher Ansatz wäre, den in dieser Arbeit entwickelten Network Switcher so zu erweitern, dass dieser vollständig die Aufgaben einer Vermittlungsschicht übernimmt, sodass die vorhandenen Network Layer in ihrer Funktionalität auf die Simulation der unterschiedlichen physikalischen Übertragungswege reduziert werden können. Interessant und nützlich wäre auch die Implementierung von weiteren Protokollen der Vermittlungsschicht. Außerdem könnte der Network Switcher um weitere Routing-Protokolle erweitert werden, die sich auch innerhalb einer Simulation kombinieren lassen.

Literaturverzeichnis

[Bak95] BAKER, Fred: Requirements for IP Version 4 Routers. Version: June 1995. https://tools.ietf.org/html/rfc1812. 1995 (1812). RFC.

[GA15] GRAFFI, Kalman; AMFT, Tobias: *A Technological Point of View on the Transformation of Online Social Networks and Communication Infrastructures*. Manuscript submitted for publication, 2015.

[Gra11] GRAFFI, Kalman: PeerfactSim.KOM: A P2P System Simulator – Experiences and Lessons Learned. In: *IEEE P2P '11: Proc. of the Int. Conf. on Peer-to-Peer Computing*, 2011.

[Kor15] KORFMACHER, Tobias: *Implementation and Evaluation of a Mobility Model in a Peer-to-Peer Simulator* . Bachelor's thesis, Mai 2015.

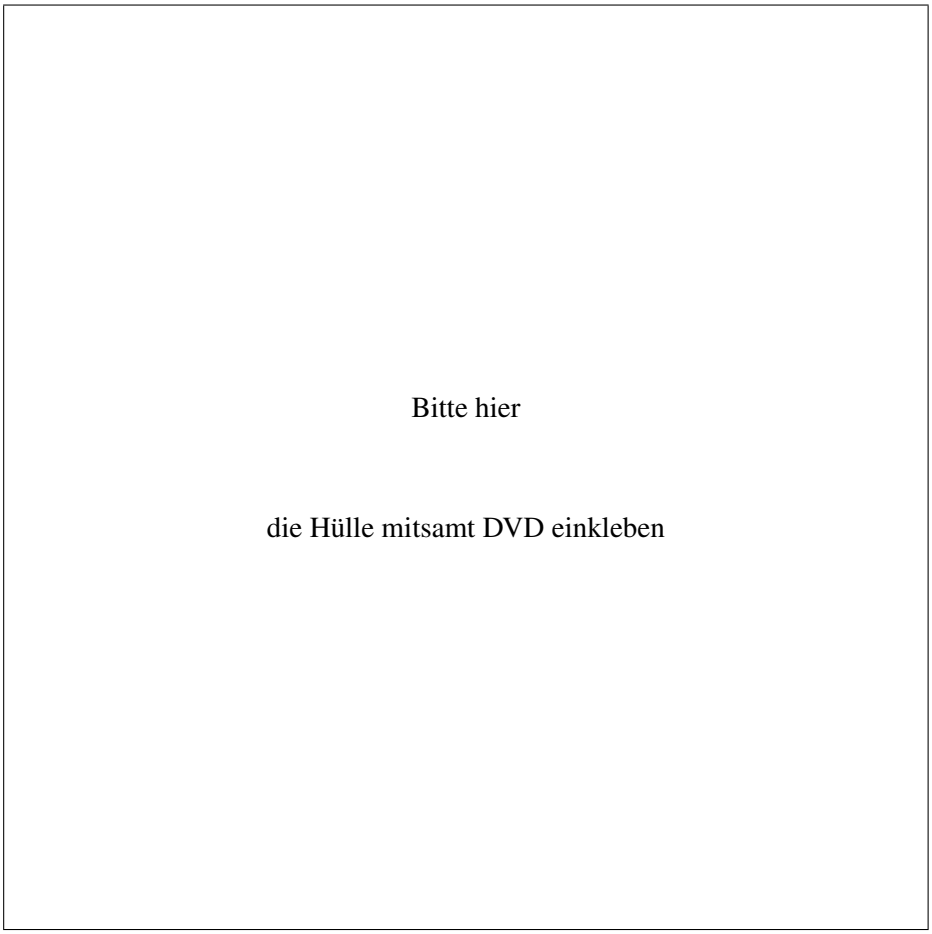
[Mal98] MALKIN, Gary: RIP Version 2. Version: November 1998. https://tools.ietf.org/html/rfc2453. 1998 (2453). RFC.

[SMK+01] STOICA, Ion; MORRIS, Robert; KARGER, David; KAASHOEK, M. F.; BALAKRISHNAN, Hari: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: *SIGCOMM '01: Proc. of the Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM, 2001. ISBN 1–58113–411–8.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 11.März 2016 Patrick Szewior



Diese DVD enthält:

- Eine *pdf* Version der Bachelorarbeit
- Alle LaTeXund Grafik Dateien mitsamt dazugehörigen Skripten, die verwendet wurden
- Der Quellcode, der während der Bachelorarbeit erarbeitet wurde
- Alle während der Evaluation angefallenen Messdaten
- Alle referenzierten Webseiten und wissenschaftlichen Arbeiten