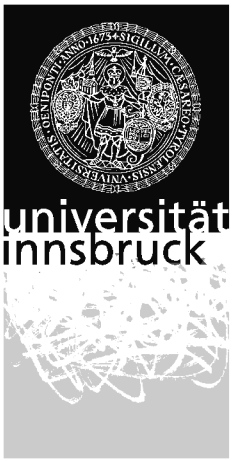

Leopold-Franzens-Universität
Innsbruck

Institut für Informatik

Evaluierung von Netzwerk - Testumgebungen

Bakkalaureatsarbeit

eingereicht bei Dr. Ing. Michael Welzl

Andreas Klotz
Markus Sigl

Innsbruck, 14. Juni 2004

Danksagung

Wir möchten uns bei jenen bedanken, die uns bei dieser Bakkalaureatsarbeit mit Rat und Tat zur Seite gestanden sind.

Besonders bedanken möchten wir uns natürlich bei unserem Betreuer Dr. Ing. Michael Welzl.

Weiters möchten wir natürlich auch bei den Mitarbeitern des Entwicklungsteams der einzelnen Emulatoren bedanken, die uns bei diversen Problemen zur Seite standen.

Ein Dank gilt natürlich auch dem Team der Informatik an der Universität Innsbruck, die es uns durch das Schaffen dieses Studienzweiges überhaupt erst ermöglichen, dass wir uns dieses Wissen aneignen können.

Zum Schluss möchten wir noch einen besonderen Dank an unseren Eltern richten, die uns dieses Studium ermöglicht haben und uns in allen Belangen unterstützen.

Vielen herzlichen Dank an alle.

Inhaltsverzeichnis

DANKSAGUNG **2**

INHALTSVERZEICHNIS **3**

KURZFASSUNG **6**

SUMMARY **7**

KAPITEL 1 ..... **8**

EINLEITUNG **8**

1.1. MOTIVATION DER ARBEIT 8
1.2. PROBLEMDEFINITION 9
1.3. ZWECK DER ARBEIT 9
1.4. DEFINITION „EMULATION“ 9
1.5. EINFÜHRUNG IN OTCL 10
1.5.1. Was versteht man unter OTcl 11
1.5.2. Installation von OTcl 11
1.5.3. Anwendungsmöglichkeiten von OTcl 12
1.5.4. Objektorientierung in OTcl 12
1.5.5. Neue Sprachelemte in OTcl 13
1.5.6. Vererbung in OTcl 15
1.5.7. Beispiel in OTcl 15

KAPITEL 2 ..... **17**

EMULAB **17**

2.1. EINFÜHRUNG 17
2.1.1. Was ist Emulab? 17
2.1.2. Wo befindet sich diese Einrichtung? 18
2.1.3. Wer kann ein Projekt starten, wie? 18
2.1.4. Begriffe 19
2.1.5. Arbeiten mit Emulab 20
2.2. ABLAUF 23
2.2.1. My Emulab 24
2.2.2. Begin an experiment 26

2.2.3. Create a PlanetLab Slice ... *30*
2.2.4. Experiment List ... *30*
2.2.5. Node Status ... *30*
2.2.6. List ImageIDs or OSIDs ... *31*
2.2.7. Start or Join a Project ... *31*
2.2.8. Was passiert mit dem gestarteten Experiment ... *31*
2.3. TOOLS ZUR AUSWERTUNG ... 34
2.3.1. portstats ... *34*
2.3.2. tcpdump ... *35*
2.4. WICHTIGE DETAILS AM RANDE ... 36
KAPITEL 3 ... 37
EMULATOR IN NS ... 37
3.1. ALLGEMEINES ... 37
3.1.1. Was ist Ns-2? ... *37*
3.1.2. Ns-2 und Emulation ... *38*
3.1.3. Installation von freeBSD ... *41*
3.1.4. Installation von ns-2 ... *42*
3.2. BESCHREIBUNG EINES NETZWERKES FÜR DEN NS-EMULATOR ... 49
3.2.1. Real-Time Scheduler ... *49*
3.2.2. Tab Agents ... *50*
3.2.3. Network Objects ... *50*
KAPITEL 4 ... 53
NISTNET ... 53
4.1. ALLGEMEINES ... 53
4.2. INSTALLATION VON NIST NET ... 54
1. Voraussetzungen: ... *55*
2. Software downloaden ... *55*
3. Software entpacken ... *55*
4. genauere Infos zu Ihrem Betriebssystem lesen ... *55*
5. eventuell Kernel patchen ... *56*
6. Makefile erstellen ... *56*
7. NistNet installieren ... *56*
4.3. ARBEITEN MIT NIST NET ... 56
4.3.1. Befehle von Nist Net ... *57*
4.3.2. cnistnet ... *58*
4.3.3. Hitbox ... *59*
4.3.4. xnistnet ... *59*

KAPITEL 5 ..... **63**

VERGLEICH DER EMULATOREN **63**

5.1. DEFINITION DER VERGLEICHSSITUATION 63

5.2. EMULAB 64

5.3. NSE 68

5.4. NISTNET 74

KAPITEL 6 ..... **80**

ABSCHLUSSBETRACHTUNG **80**

6.1. RESÜMEE 80

6.1.1. Was spricht für bzw. gegen Emulab? *80*

6.1.2. Was spricht für bzw. gegen nse? *81*

6.1.3. Was spricht für bzw. gegen NistNet? *81*

6.1.4. Welchen Emulator soll ich nun verwenden? *82*

6.2. PROBLEME 84

6.2.1. Emulab *84*

6.2.2. nse *85*

6.2.3. NistNet *86*

LITERATURVERZEICHNIS **88**

Kurzfassung

Dieses Dokument stellt die Bakkalaureatsarbeit zum Thema „Evaluierung von Netzwerk-Testumgebungen“ dar. Dabei geht es hauptsächlich darum, Netzwerk Emulatoren zu testen und dann zu evaluieren. Untersucht wurden drei der bekanntesten Emulatoren. Bei diesen Emulatoren handelt es sich um:

- Emulab
- Emulator in NS
- NistNet

Unter einem Emulator versteht man eine virtuelle Maschine, bei der die komplette Hardware eines anderen Systems virtuell und möglichst exakt nachgebildet wird. Bei Netzwerkemulatoren wird ein Netzwerk in einem Rechner simuliert. Um jedoch als Emulator zu gelten, muss Daten-Verkehr von außen in das simulierte Netzwerk eingeschleust und auch Verkehr vom simulierten Netzwerk in ein reales Netzwerk übergeben werden können.

Unsere Aufgaben bei dieser Bakkalaureatsarbeit waren:

- den Emulator installieren und zum Laufen bringen
- ein definiertes Szenario simulieren (welches den Vergleich zwischen den 3 Emulatoren ermöglichen soll)
- eine Anleitung für die Installation und Benützung der Emulatoren schreiben
- die Emulatoren miteinander vergleichen und Empfehlungen für die Verwendung abgeben

Die gesamte Bakkalaureatsarbeit hatte als Ziel, eine Anleitung zu geben, damit die untersuchten Emulatoren leicht installiert und verwendet werden können. Weiters soll sie als eine Entscheidungshilfe dienen, um den richtigen Emulator für die richtige Situation auszuwählen.

Summary

This document bears the title “Evaluation of network-emulators” and is meant as a bachelor thesis.

The work’s main objective is to test and evaluate network emulators, and in doing so three of the most common emulators have been analysed, namely

- Emulab
- emulator in NS
- NistNet

An emulator can be seen as a virtual machine which virtually reproduces the entire hardware of another system as accurately as possible. Network emulators simulate a network in a processor. To be called an emulator in that respect, the program has to allow for data-transfer to the simulated network from without, and for a data-transfer from the simulated network to a real one.

Our tasks regarding this bachelor thesis included

- installing and running the emulator
- simulating a defined scenario (which should enable us to compare three emulators)
- writing a guideline for the installation and use of the emulators
- comparing the emulators and giving hints for their application

The overall goal of this thesis is to provide a guideline to simplify the installation and use of the analysed emulators. Furthermore it should serve as a decision tool which would help to opt for the right emulator in the right situation.

Kapitel 1

Einleitung

1.1. Motivation der Arbeit

Um ein neues Netzwerk einzurichten, ist es regelmäßig notwendig vor der Realisierung den Entwurf zu überprüfen, um keine Überraschungen zu erleben. Oft reicht eine Simulation aber nicht aus, so z.B., wenn das Netzwerk sehr kritischen Anforderungen genügen muss. Das Netzwerk gleich hardware-mäßig zu testen wäre zu aufwändig und vor allem zu kostenintensiv. Mit Emulatoren ist es etwa möglich, kritische Teile eines Netzwerks real aufzubauen und mit dem Rest, bei welchem eine Simulation ausreicht, zu koppeln.

Da es aber mittlerweile schon sehr viele Emulatoren zur freien Verfügung gibt, ist es für Anwender nicht einfach, ein geeignetes Tool für dessen Anforderungen und vor allem auch Kenntnisse zu finden. Besonders die teilweise nur sehr schlechte Dokumentation erschwert eine Auswahl zusätzlich und führt wiederum zu einer sehr langen Einarbeitungszeit. Aus dieser Motivation heraus entstand unsere Arbeit. Sie soll dem Anwender den Einstieg in die Netzwerkemulation erleichtern und seine Einarbeitungszeit verkürzen. Desweiteren soll mit Hilfe unserer Arbeit die Installation, die Benützung der Emulatoren und die Auswahl des richtigen Emulators für die jeweilige Situation erleichtert werden.

1.2. Problemdefinition

Die Problemstellung bei dieser Bakkalaureatsarbeit war es, bestehende Netzwerk-Emulations-Tools zu testen. Die Hauptaufgabe lag darin, die Tools zu installieren und zum Laufen zu bringen. Dies war vor allem durch die teilweise sehr dürftig ausgefallenen Dokumentationen bedingt oft kein leichtes Unterfangen. Die Probleme bei der Installation und der Verwendung der Tools werden am Ende dieser Arbeit noch einmal detailliert aufgeführt.

1.3. Zweck der Arbeit

Da es im Internet teilweise nur sehr wenig Informationen über die Netzwerk-Emulatoren geschweige denn einen Vergleich beziehungsweise eine Empfehlung für die Verwendung eines bestimmten Tools gibt, war es unsere Aufgabe eine Dokumentation zu verfassen, die es nachfolgenden Studenten erleichtern soll, ihre Netzwerksoftware zu testen. Mit Hilfe unseres Dokumentes können sie sich zum einen mit dem jeweiligen Emulator vertraut machen und eben auch den passenden Emulator für ihre Situation auswählen. Zum anderen wird mit unserer Anleitung die Installation der Tools erleichtert und verständlich aufbereitet.

1.4. Definition „Emulation“

Die folgende Definition wurde aus einem Technik Lexikon übernommen:[1]

Ein Emulator ist eine virtuelle Maschine. Im Vordergrund steht, dass die komplette Hardware eines anderen Systems virtuell und möglichst exakt nachgebildet wird.

- Software, die für ältere Systeme entwickelt worden ist, kann auf modernen Systemen weiter laufen.

[1] vgl. http://www.net-lexikon.de/Emulator.html

    - Beispiel: Auf einem Linux-Rechner wird mittels VMware ein PC emuliert, auf dem Windows installiert werden kann. Alle bisher gekaufte Windowssoftware kann weiter eingesetzt werden.

- Es ist möglich, Software für andere Systeme zu entwickeln und zu testen.
    - Beispiel: Programme, die für Palm OS auf einem PC entwickelt werden, können mit dem Palm Emulator getestet werden.

- Alte Konsolspiele aus den frühen achtziger Jahren können dank geeigneter Emulatoren auf moderner Hardware laufen.

Um die bestmögliche Netzwerksoftware zu entwickeln sind zahlreiche Testläufe vonnöten. Dabei stehen den Netzwerkspezialisten zwei Möglichkeiten zur Verfügung:

- testen der Software in der tatsächlichen Umgebung, in der sie eingesetzt wird
- testen eines Modells der Software in einer künstlichen, simulierten Umgebung

Dabei ist die Simulation viel billiger und vor allem auch schneller. Ein Nachteil dabei ist jedoch, dass die Software zum Einsatz in der dann tatsächlichen Umgebung oft noch geändert werden muss. Es kommt auch vor, dass Anwendungen, die in der simulierten Umgebung problemlos gelaufen sind, in der realen Umgebung nicht fehlerfrei funktionieren. Mit Hilfe von Emulatoren kann man im realen Netzwerk eine kleine Testumgebung schaffen.

1.5. Einführung in OTcl

Für die Verwendung der Emulatoren benötigt man, außer man benützt die grafische Oberfläche von emulab oder nisnet, Tcl und oder OTcl. Für Tcl wurde bereits im Rahmen einer Bakkalaureatsarbeit eine Dokumentation erarbeitet[2]. Deshalb möchten wir im Rahmen unserer Arbeit nur kurz auf die Unterschiede beziehungsweise die Erweiterungen von OTcl im Vergleich zu Tcl eingehen.

[2] Vgl. http://www.welzl.at/research/tools/ns/index.html

1.5.1. Was versteht man unter OTcl

OTcl steht für MIT (Massachusetts Institute of Technology) Object-Tcl und ist eine Erweiterung von Tcl/Tk zur objektorientierten Programmierung. Entwickelt wurde OTcl von David Wetherall (E-Mail an: djq@lcs.mit.edu) am MIT Laboratorium für Computer-Wissenschaften. Zurzeit ist die 1995 veröffentlichte Version 0.96 die aktuellste.

OTcl ist unter der MIT-Lizenz frei nutzbar und sogar beliebig abänderbar.

Über den ftp-Server ftp://ftp.tns.lcs.mit.edu/pub/otcl kann man sich die neueste Version sowie auch frühere Versionen (bis 0.92) herunterladen. Außerdem findet man html-Dokumente zur Erklärung und zur Beantwortung der am häufigsten gestellten Fragen. Als Zusatz findet man Erläuterungen zu den OTcl Klassen, und es sind auch alle Änderungen bei den verschiedenen Versionen dokumentiert.

1.5.2. Installation von OTcl

Zur Installation von OTcl in der eigenständigen Form benötigt man eine der Tcl/Tk Versionen 7.3/3.6, 7.4/4.0 oder 7.5/4.1. Die Tcl/Tk Distributionen müssen den Tcl-Sourcecode, sowie die Tcl und Tk Bibliotheken enthalten. Eine weitere Möglichkeit ist die Integration von OTcl in bereits vorhandene Pakete (z.B.: Paket „wave“ von Neumann und Nusser).

Man erhält die OTcl Distribution (und auch die Tcl Distribution) als „*.tar.gz“-Datei. Als erstes muss man also die Dateien entpacken. Dann liegt im Verzeichnis die Installationsdatei von Otcl.

Mit `./configure` wird das Konfigurationsprogramm gestartet. Dieses verlangt nach den Verzeichnissen der Tcl- und Tk-Distributionen und nach den Bibliotheken. Es wird hierauf dann vom Konfigurationsprogramm ein Makefile erzeugt, welches mit `make` kompiliert wird. Der GNU C-Compiler erzeugt dann 4 Programme:

- otclsh – Otcl Shell
- owish – Otcl Shell mit Einbindung von Tk
- libotcl.a – Otcl Bibliothek
- libotcl.so – Otcl Bibliothek, die dynamisch in laufende Tcl Anwendungen eingebunden werden kann (`load libotcl.so OTcl`)

Mit der `otclsh` kann man nun Skripte laden und ablaufen lassen. Mit `source skriptname` wird es dann gestartet.

In der Distribution von OTcl enthält die Datei readme.html eine detaillierte Installationsanleitung.

Die neueste Distribution kann man unter folgendem Link kostenlos downloaden:
ftp://ftp.tns.lcs.mit.edu/pub/otcl/

1.5.3. Anwendungsmöglichkeiten von OTcl

Der Hauptgrund für eine Anwendung von OTcl ist Programmen einen objektorientierten Aufsatz zu geben. Einige Beispiele für Projekte, bei denen OTcl eingesetzt wurde, werden im Folgenden angeführt:

- VUSystem – Ursprung von OTcl[3]
- The View Station – Forschungsansatz am MIT zur Entwicklung eines Systems, welches Multimedia Anwendungen forciert
- Wafe – Projekt von Neumann und Nusser. Dabei handelt es sich um ein Front-End für die GUI Programmierung mit zahlreichen Erweiterungen
- The OTcl NeoWebScript™ Interface – Objektorientiertes Programmiersystem von Kunkee, das es erlaubt, einfache sowie komplexe Programme in html Dateien einzubinden (Neoscript)
- The CVL Library – Grafikbibliothek für Open-GL von Petr Krysl
- The Orpheus Preprocessor – Objektorientiertes Programmier- und Grafiksystem für Statik und Dynamik von Littwin.

1.5.4. Objektorientierung in OTcl

Immer mehr Softwareprojekte werden mit objektorientierten Programmiersprachen realisiert. Eine Verwendung von objektorientierten Programmiersprachen bedeutet nicht nur eine zeitliche Einsparung bei der Entwicklung, sondern auch eine erleichterte Erweiterung und/oder Wiederverwendung des Programms.

---

[3] Siehe http://www.tns.lcs.mit.edu/vs/vusystem.html

Was versteht man unter objektorientierten Programmiersprachen?
Es wird versucht die Trennung zwischen Daten und den darauf operierenden Funktionen aufzuheben. Objekte aus der Realität lassen sich auf Objekte im Programm abbilden.

Das Hauptziel bei der Entwicklung von OTcl war es, eine ähnliche Syntax, wie sie in Tcl zum Einsatz kommt, zu verwenden. Daher ist ein Umstieg von Tcl auf OTcl sehr leicht für uns Anwender.

Features von OTcl:

- Jedes Objekt kann in OTcl spezialisiert werden
- OTcl ist dynamisch. Darunter versteht man, dass Variablen, Prozeduren, Klassen, ... beliebig verändert und auch erzeugt werden können.
- Unterstützung von Prototyping (wird auch von Tcl unterstützt). Das erleichtert vor allem schnelle Änderungen.
- Method combining (geschachtelte Verarbeitung einer gleichnamigen Methode über mehrere Klassen). Es ist hier keine explizite Benennung der vererbenden Klassen oder Methoden wie in C++ notwendig, sondern es wird eine vordefinierte Variable benutzt. Dies ist ein Vorteil bei der Wiederverwendung des Codes.

1.5.5. Neue Sprachelemte in OTcl

Im Vergleich zu Tcl gibt es nur 2 neue Befehle:

- `object`
- `class`

Object

Objekte sind in OTcl immer Instanzen von Klassen. Alle Objekte und Klassen werden in OTcl als Befehle registriert.
Wichtige Befehle:

- `object obj_name`
  Hier wird ein Objekt mit dem Namen `obj_name` definiert.

- `obj_name set varname ?value?`

  Mit diesem Befehl werden für das Objekt `obj_name` Variableninhalte gesetzt. (z.B.: `Mitarbeiter27 set adresse ...`)
- `obj_name unset v1 ?v2...vn?`

  Hier werden die Variablen `v1` bis `vn` aus dem Interpreter gelöscht
- `obj_name set varname() ?value?`

  Es ist in OTcl auch möglich Array zu definieren. Wenn der Mitarbeiter zum Beispiel mehr als eine Adresse hat, können die entsprechneden Befehle lauten:

  `Mitarbeiter27 set adress(adresse1) …` und

  `Mitarbeiter27 set adress(adresse2) …`
- `obj_name proc name args body`

  Hier wird eine Methode für das Objekt `obj_name` deklariert. Dabei steht `name` für den Namen, `args` für die Argumente und `body` für den Rumpf.
- `obj_name destroy`

  Das Objekt `obj_name` wird aus dem Interpreter gelöscht.
- `obj_name info option args`

  Mit der Methode `info` bekommt man Informationen über ein Objekt. Dabei gibt es zahlreiche option-subbefehle, die wir hier aber nicht anführen möchten. Ein Beispiel für eine `info` Abfrage wäre zum Beispiel: `Mitarbeiter27 info procs *les*` gibt alle Methoden aus, in denen `*les*` vorkommt.

Class

In OTcl gibt es mehrere Möglichkeiten Klassen zu erzeugen. Die am öftesten verwendete Methode zeigt der erste der nun angeführten Befehle.

Wichtige Befehle:

- `Class Classname args`

  Hier wird eine Klasse mit dem Namen `Classname` und den Argumenten `args` erzeugt.
- `Class create Classname args`

  Dies ist eine weitere Möglichkeit zur Erzeugung einer Klasse und liefert das gleiche Ergebnis wie der oben angeführte Befehl.
- `ClassName instproc procname args body`

  Mit Hilfe der Instanzmethode `instproc` kann man einer spezialisierten Klasse neue Instanzmethoden hinzufügen.

- `ClassName info option args`

  Wie schon bei den Objekten kann man sich auch hier mit der Funktion `info` Informationen über die Klasse ausgeben lassen. Es gibt auch hier wieder zahlreiche Option-Subbefehle, die wir aber hier nicht anführen. Die Routinen funktionieren analog zu den Objekt-Methoden.

1.5.6. Vererbung in OTcl

OTcl bietet wie auch andere Sprachen die Möglichkeit der Vererbungen an. Neben der einfachen Vererbung besteht auch die Möglichkeit der Mehrfachvererbung. Für unsere Arbeit hat die Vererbung nur einen sehr geringen Stellenwert. Deshalb wird hier nur auf die Dokumentation zu OTcl verwiesen.[4]

1.5.7. Beispiel in OTcl

Zum Abschluss der OTcl Einführung möchten wir noch ein kurzes Beispiel anführen, welches vor allem den grundlegenden Aufbau eines OTcl-Files erläutert:

```
Class bundesliga
bundesliga instproc sitz {} {
$self instvar sitzplz_
puts "$ sitzplz _ Wien Meiereistrasse 7"
}

Class fcwackerinsbruck – superclass bundesliga
fcwackerinnsbruck instproc sitzung {} {
$self instvar sitzplz_
puts "$ sitzplz _ Innsbruck Feldstraße 9"
}

set a [new bundesliga]
$a set sitzplz_ 1020
set b [new fcwackerinnsbruck]
$b set sitzplz _ 6020

$a sitz
$b sitz
```

kurze Erklärung des Files:

Zu Beginn wird eine Klasse `bundesliga` und eine Funktion `sitz` angelegt. Die Funktion definiert man mit Hilfe von `instproc`. Als nächstes wird mit Hilfe von `instvar` eine Variable `sitzplz` angelegt. `instvar` schaut, ob die Variable schon

[4] vgl. http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/

irgendwo im Programm deklariert wurde; wenn ja, wird auf die schon bestehende Variable referenziert, sonst wird die Variable neu deklariert.
Als nächster Schritt wird eine Sub-Klasse `fcwackerinnsbruck` mit der Super-Klasse `bundesliga deklariert`. In dieser Sub-Klasse wird, wie schon zuerst, zuerst eine Funktion `sitz` und dann eine Variable `sitzplz` definert.
Danach wird ein Objekt `bundesliga` und ein Objekt `wackerinnsbruck` erzeugt.
Zum Schluss wird die Funktion `sitz` aufgerufen und folgende Ausgabe generiert:

```
1020 Wien Meiereistrasse 7
6020 Innsbruck Feldstraße 9
```

Kapitel 2

Emulab

2.1. Einführung

2.1.1. Was ist Emulab?

- eine Plattform, um Netzwerke und verteilte System zu entwickeln und zu verstehen
- hunderte Computer mit konfigurierbaren und kontrollierbaren Verbindungen
- eine Einrichtung, um Zugriff auf Netzwerk-Knoten der ganzen Welt zu haben
- für jeden Benutzer frei zu verwenden
- einfach zu bedienen

Emulab ist eine Forschungseinrichtung, welche hunderte Computer vernetzt hat (Cluster), um darin Netzwerke zu simulieren - doch die eigentliche Idee hinter Emulab ist, dass man von außen Pakete an diese mit echten Rechnern vom Anwender konfigurierte Netzwerke schicken kann. Die Pakete laufen dann durch das inszenierte Netzwerk-Szenario.

2.1.2. Wo befindet sich diese Einrichtung?

Es gibt bereits mehrere Emulab-Cluster:
Georgia Tech, University of Kentucky, University of Utah, University of Winsconsin und The Aerospace Corporation, ISI.
Das Original-Emulab befindet sich an der University of Utah in den USA.

Dort stehen 168 Knoten auf folgenden Rechnern zur Verfügung:

- Pc600 (40 Knoten)
  600MHz Prozessor
  256 MB RAM
  13 GB DIE Platte
- Pc850 (128 Knoten)
  850 MHz Prozessor
  512 MB RAM
  40 GB DIE Platte

Bei den anderen Emulab Clustern stehen nicht so viele Knoten zur Verfügung. In Georgia und in Kentucky sind es zum Beispiel je 50 Knoten.

2.1.3. Wer kann ein Projekt starten, wie?

Ein Projekt wird für mehrere Personen angelegt, jedoch verlangt Emulab einen verantwortlichen Projektleiter. Auf der Homepage von Emulab[5] wird ausdrücklich darauf hingewiesen, dass Studenten kein Projekt anlegen können, außer sie bekommen durch das „Emulab Approval Committee“ eine Berechtigung. Jedoch kann ein beaufsichtigender Professor (=Projektleiter) ein Projekt für seine Studenten anlegen.
Dazu muss auf der Homepage unter „Interaction“ auf „Request Account“ geklickt werden. Es öffnet sich dann ein Fenster, das 2 Möglichkeiten zur Auswahl hat:

- Join an Existing Project
- Start a New Project

Indem man auf „Start a New Project” klickt, kann man einen neuen Account beantragen. Laut Homepage wird nach maximal einer Woche von den zuständigen

[5] Siehe http://www.emulab.net

Administatoren ein Projekt genehmigt oder abgelehnt. Die Antwort erhält man dann per e-Mail.
In unserem Fall hat die Anmeldung (wurde von Herrn Dr. Welzl vorgenommen) fast 2 Monate gedauert, da die zuständigen Administratoren unsere Anmeldungen vergessen hatten. Man kann sich also nicht hundertprozentig darauf verlassen, dass man nach einer Woche schon mit Emulab arbeiten kann.
„Start a New Project“ kann wie vorher besprochen nur von einer beaufsichtigenden Person (wie z.B.: Universitätsprofessor oder Projektleiter) durchgeführt werden, wobei man mit „Join an Existing Project“ diesem Projekt dann beitreten kann.
Personen, die an dem Projekt arbeiten (z.B. Studenten) müssen die Erlaubnis vom Projektleiter (z.B. Professor) zur Mitarbeit bekommen. Der Projektleiter schaltet dann die jeweiligen Personen frei und vergibt die Rechte. Die Rechte sagen aus, ob man selbst Experimente anlegen und starten kann, oder nur Einsicht in laufende Experimente hat.

2.1.4. Begriffe

Projekt

- Zentrale administrative Einheit
- Gestartet durch eine Person, die für die gesamte Arbeit am Projekt verantwortlich ist (wird zu Projektleiter) - Start via Web-Interface
- Nur der Projektleiter kann neue Mitarbeiter in sein Projekt aufnehmen
- Projekt erhält eigenen Disk-Space

Experiment

- zentrale ausführende Einheit
- Netzwerk-Konfigurationenen (Links, Knoten)
- Erzeugt durch ein NS-file (oder einfacher Java GUI)
- Gestartet durch Web-Interface
- E-Mail wird versendet, wenn das Setup fertig ausgeführt wurde

2.1.5. Arbeiten mit Emulab

Notwendige Schritte

1. www.netbed.org
2. start or join a project (siehe oben)
3. Ein Experiment erzeugen:

   Topologie / Konfiguration spezifiziert mit

   - einem NS File oder
   - Java GUI

4. Das Experiment starten und verwenden.

Erzeugen mit einem NS File

Emulab verwendet „NS“ (Network Simulator) Format, um Netzwerk-Topologien zu beschreiben. Dies ist dasselbe Tcl-basierte Format, das auch von ns-2 verwendet wird.
Da Emulab ein Emulator und kein Simulator ist, werden die Files etwas anders interpretiert als bei ns-2.
Ein paar ns-2 Funktionalitäten könnten anders ablaufen als erwartet bzw. gar nicht zum Laufen gebracht werden - dies wird aber durch eine Warnung folgender Form angezeigt:

```
***     WARNING: Unsupported NS Statement!
 Link type BAZ, using DropTail!
```

Es könnte aber auch der Fall auftreten, dass lauffähige Emulab-Files unter ns-2 nicht zum Laufen gebracht werden können, da bestimmte Emulab-Anweisungen stören. Dazu braucht nur das File www.emulab.net/tutorial/tb_compat.tcl in das Verzeichnis kopiert zu werden, in dem das NS-File liegt. Ns-2 wird dann die Emulab-spezifischen Befehle ignorieren. Dadurch kann dasselbe NS-File unter ns-2 und Emulab verwendet werden („most of the time“).

Erzeugen mit der GUI

Emulab bietet eine GUI (Graphical User Interface) an, um Experimente in graphischer Form zu erstellen. Damit können Knoten, Lans und Links platziert werden, Testbed-

spezifische Attribute wie Betriebssysteme, Hardware-Typen und Link/Lan-Charakteristika können zu jedem Objekt gesetzt werden.
Mit einem einfachen Klick kann die beschriebene (gezeichnete) Topologie des Netzwerkes auf Testbed als Experiment in einem Projekt instanziiert werden. Alternativ kann das automatisch generierte NS-file auch lokal gespeichert werden, um es ggf. zu bearbeiten oder später als Experiment zu starten.

Um die GUI zu nutzen, muss man sich nur einloggen und zur Seite „Begin Experiment“ wechseln. Beachte: Es wird ein Java-fähiger Browser benötigt!

Besonderheiten beim Aufbau eines OTcl-Scripts mit Emulab:

Bei Emulab gibt es einige Unterschiede beim Aufbau eines OTcl-Scripts im Vergleich zu ns. Da man aber großteils das ns Script bis auf ein paar Kleinigkeiten, die in der Folge erläutert werden, nutzen kann, verweisen wir an dieser Stelle auf die Emulab-Dokumentation.[6]. Falls bei der NS File Prüfung Fehler auftreten, empfehlen wir, sich dort über die Unterschiede zu informieren.
Folgende Zeile muss bei jedem File am Anfang des NS Files stehen (vor jeglichem testbed-Befehl):

```
source tb_compat.tcl
```

(um dasselbe NS-File unter ns-2 zu nutzen, kann das File tb_compat.tcl (www.emulab.net/tutorial/tb_compat.tcl) einfach in das Verzeichnis kopiert werden, in dem das NS-File liegt.)

Kann ich direkt zu einem Knoten verbinden?

Man kann mit Hilfe von „Tiptunnel“ vom eigenen System (Windows, FreeBSD und Linux) direkt auf den jeweiligen Knoten zugreifen.
„Tiptunnel“ kann man unter folgenden Adressen herunterladen:

- Windows: http://www.emulab.net/downloads/tiptunnel-win32.exe
- FreeBSD: http://www.emulab.net/downloads/tiptunnel-freebsd.tar.gz
- Linux: http://www.emulab.net/downloads/tiptunnel-linux.tar.gz

[6] Vgl. http://www.emulab.net/tutorial/docwrapper.php3?docname=nscommands.html

Die Verwendung von „Tiptunnel“ ist ganz einfach. Wenn ein Experiment gestartet wurde, klickt man einfach in den Knoten Optionen (man gelangt durch klicken auf den jeweiligen Knotennamen in die Optionen) auf „Connect to Serial Line“, oder man kann auch direkt über das Konsole-Symbol (siehe Abbildung 1) verbinden.

Reserved Nodes

Node ID	Name	Type	Default OSID	Node Status	Hours Idle[1]	Startup Status[2]	SSH	Console
pc23	nodeA	pc600	RHL73-STD	up	0	none		
pc28	nodeB	pc600	RHL73-STD	up	0	none		
pc30	tbsdelay0	pc600	FBSD47-STD	up	0	0		

Abbildung 1: Knoten Informationen

Es öffnet sich dann ein Downloadfenster für die Datei *.tbacl. Diese Datei speichert man in einen Ordner seiner Wahl. Durch doppelklicken auf das file kann man daraufhin ganz einfach und direkt zum Knoten verbinden. Details und Besonderheiten (besonders für FreeBSD und Linux) finden sich auf der Emulab Homepage unter FAQ.[7]

Wie lade ich meine eigene Software auf meine Knoten?

Man kann eigene Software wie „rpms“ oder „tarfiles“ automatisch laden lassen, wenn das Experiment konfiguriert wird. Mit Hilfe der ns Erweiterungen „tb-set-node-rpms“ und „tb-set-node-tarfiles“ kann man eine Liste der zu installierenden Software angeben. Dabei muss für jeden Knoten eine eigene Liste angegeben werden. Beim Start des Knotens wird die Software dann automatisch installiert.

- Installieren von rpms:

  Wenn man zum Beispiel die Software „trafshow.rpm“ installieren möchte, schreibt man einfach folgendes in sein ns file:

  ```
  tb-set-node-rpms $nodeA /proj/pid/rpms/trafshow.rpm
  ```

  Mit diesem Befehl wird für den Knoten A die Software `trafshow.rpm` installiert. Rpms werden als Root installiert. Das heißt, sie müssen entweder im `/proj` Verzeichnis oder im `/group` Verzeichnis abgelegt werden.

[7] Siehe: http://www.emulab.net/docwrapper.php3?docname=faq.html

- Installieren von tarfiles

  Im Prinzip funktioniert das Installieren von tarfiles gleich wie das installieren von rpms, nur muss man hier einen Ordner angeben, in den das tarfile entpackt werden soll.

  ```
  tb-set-node-tarfiles $nodeA /usr/site
  /proj/pid/tarfiles/silly.tar.gz
  ```

  Mit diesem Befehl wird das tarfile zuerst ins Verzeichnis `/usr/site` entpackt und dann anschließend auf den Knoten A installiert. Die trafiles müssen wieder im `/proj` oder `/group` Verzeichnis abgelegt sein.

Mit dem ns Befehl `tb-set-node-startcmd` kann man dann die installierten Applikationen automatisch starten, wenn der Knoten gestartet wird.
Zum Beispiel könnte so ein Befehl folgendermaßen aussehen:

```
tb-set-node-startcmd $nodeA "/proj/pid/runme.nodeA"
```

2.2. Ablauf

Wenn man ein neues Experiment starten möchte, muss man sich zuerst bei emulab.net einloggen (siehe Abbildung 2). Der Account muss vom Projektleiter zur Verfügung gestellt werden. Man meldet sich dann mit seinem Benutzernamen und Passwort an und kommt in den „Internen Bereich“.

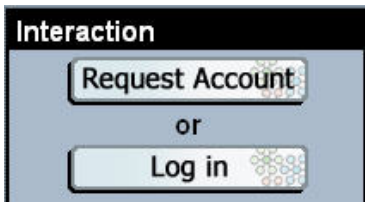

Abbildung 2: Login

Nachdem man sich angemeldet hat, wird der Interaktionsbereich um folgende Optionen erweitert (siehe Abbildung 3):

Abbildung 3: Interaktions-Optionen

Man sieht hier zum Beispiel, wieviele PCs (Knoten) noch zur Verfügung stehen. Außerdem gibt es 7 verschiedene Auswahlmöglichkeiten, auf die wir nun kurz eingehen werden.

2.2.1. My Emulab

Unter “My Emulab” sieht man, in welchen Projekten man mitarbeitet (siehe Abbildung 4), welche Experimente momentan noch laufen (siehe Abbildung 4), und man kann sein User Profil editieren. Im User Profil werden Daten wie Name, Adresse, E-Mail Adresse, Telefonnummer, usw. gespeichert.

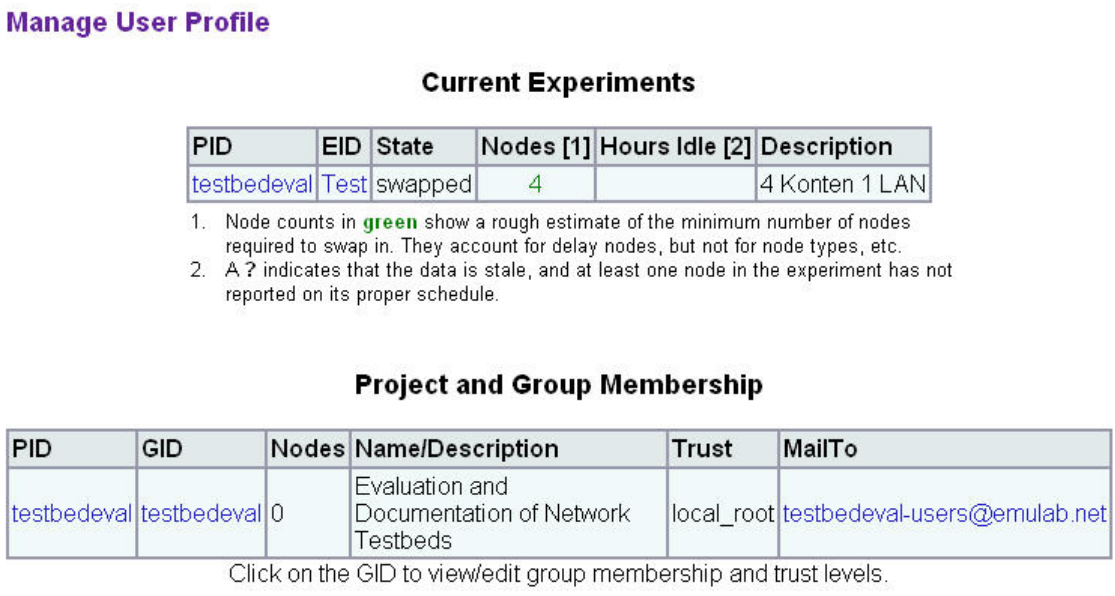

Abbildung 4: My Emulab

Die Projektdetails kann man sich durch klicken auf „PID“ anzeigen lasen. Dort hat man Einsicht, welche Experimente im Projekt laufen, wer am Projekt mitarbeitet, usw. .
Die Experimentdetails kann man sich durch klicken auf „EID“ des jeweiligen Experiments anzeigen lassen (siehe Abbildung 5).

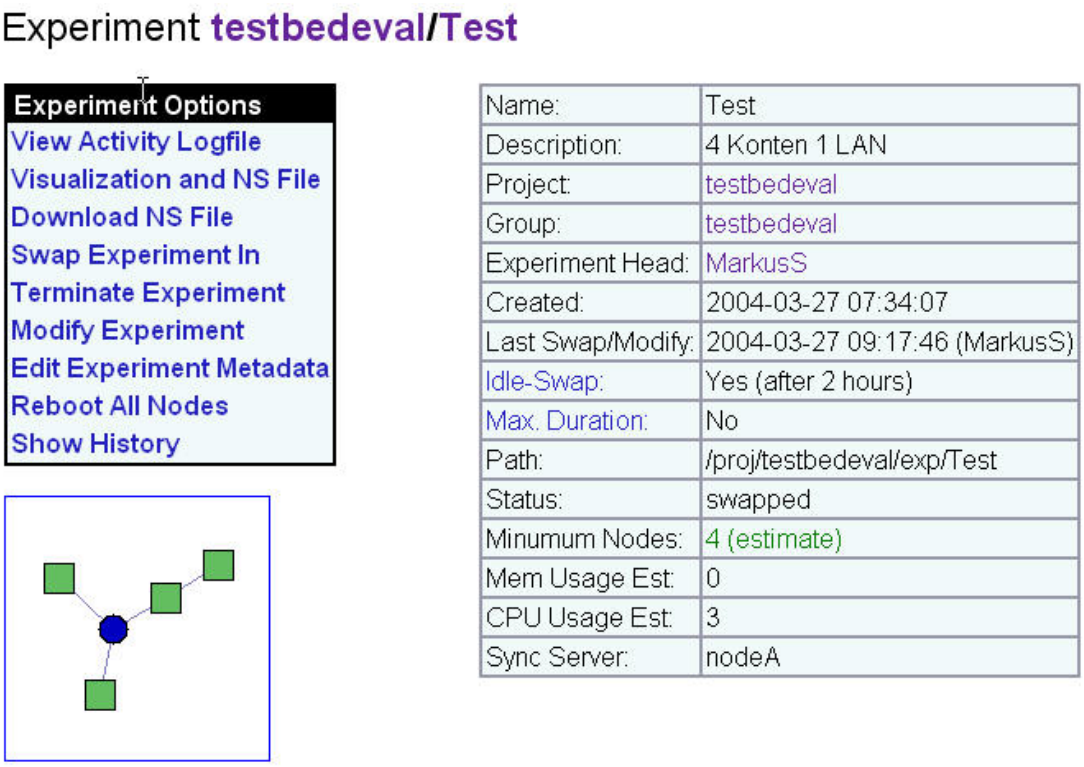

Abbildung 5: Experimentdetails

Der Screenshot zeigt die Experimentdetails anhand eines Beispiels. Man sieht also, wer das Projekt gestartet hat, wie viele Konten gebraucht werden, sowie den Status des Experiments (in diesem Fall „Swap“). Außerdem sieht man eine kleine Grafik, welche das Experiment beschreibt. Durch einen Klick auf die Grafik sieht man den detailierten Aufbau und das entsprechende NS File (gleich wie bei „Visualization and NS File“).

Weiters hat man wieder eine Vielzahl von Möglichkeiten, was man mit dem Experiment machen kann:

- View Activity Logfile

  Unter „View Activity Logfile“ kann man sich das Logfile von Emulab ansehen. Man sieht zum Beispiel, was alles passiert, wenn man ein Experiment „swapped“.

- Visualization and NS File

  Liefert den detaillierten Aufbau und das entsprechende ns-file.
- Download NS File

  Hier kann man sich das NS-File herunterladen (entweder das selbst geschriebene oder, falls man die GUI benutzt hat, das automatisch erstellte NS-File).
- Swap Experiment In

  Hier kann man das Experiment wieder aktivieren. Man kann dann ganz normal mit dem Experiment arbeiten.
- Terminate Experiment

  Es besteht auch die Möglichkeit das Experiment zu löschen. Dies geschieht mit „Terminate Experiment“. Bevor das Experiment gelöscht wird, kommt noch eine Abfrage, ob man sich auch wirklich sicher ist. So wird verhindert, dass ein Experiment unabsichtlich gelöscht wird.
- Modify Experiment

  Hier kann man das Experiment editieren. Dies geschieht mit Hilfe des NS Files. Man kann hier das NS File editieren, und mit einem Klick auf Modify werden die Änderungen dann übernommen.
- Edit Experiment Metadata

  Hier kann man die Metadaten des Experiments ändern. Siehe auch „Begin Experiment“.
- Reboot All Nodes

  Hier kann man alle Knoten „neu starten“. Dies benötigt man zum Beispiel, wenn man an einigen Knoten Änderungen vorgenommen hat, diese aber wieder rückgängig machen möchte. Auch hier wird man noch einmal gefragt, ob man wirklich alle Knoten neu starten möchte.
- Show History

  Hier kann man sich die Geschichte des Experiments anzeigen lassen. Man sieht, wann das Experiment gestartet bzw. beendet wurde.

2.2.2. Begin an experiment

Mit „Begin an Experiment“ kann man ein neues Experiment starten. Man hat dann 2 Möglichkeiten zur Auswahl. Entweder man verwendet ein schon geschriebenes NS-File oder man erstellt mit Hilfe der GUI ein neues Szenario.

NS File wird verwendet

Wenn man ein NS File verwenden möchte, sollte man es zuerst überprüfen lassen (siehe Abbildung 6). Dazu gibt man einfach an, wo sich sein NS File befindet und Emulab überprüft dann, wenn man auf „Check“ klickt, ob das File Emulab-konform ist. Wenn das File noch Fehler aufweist, werden die Fehlermeldungen auf der Folgeseite angezeigt.

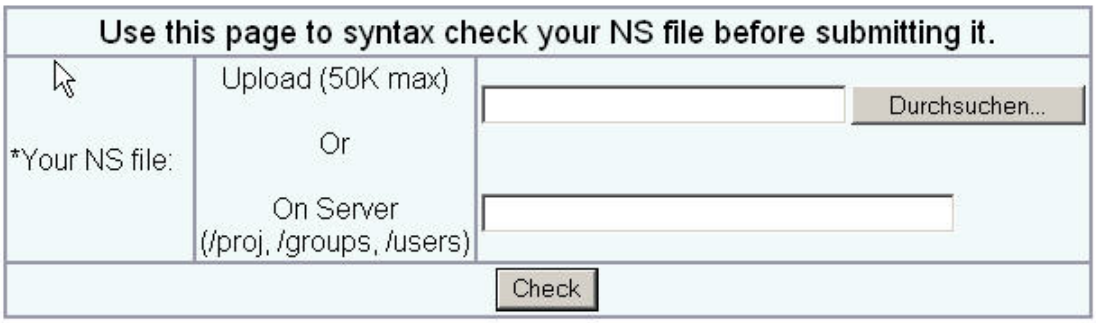

Abbildung 6: Syntax Check des NS Files

Wenn das NS File fehlerfrei ist, kann man das NS File uploaden und damit auch verwenden (siehe Abbildung 7). Weiters muss man dann noch wählen, für welches Projekt man das Experiment startet, einen Namen vergeben, dass Experiment kurz beschreiben und festlegen, nach wie vielen Stunden das Experiment geswapped wird. Geswapped wird es nur, wenn für die festgelegt Zeitspanne (hier zum Beispiel 4 Stunden) auf keinen der Knoten zugegriffen wurde.

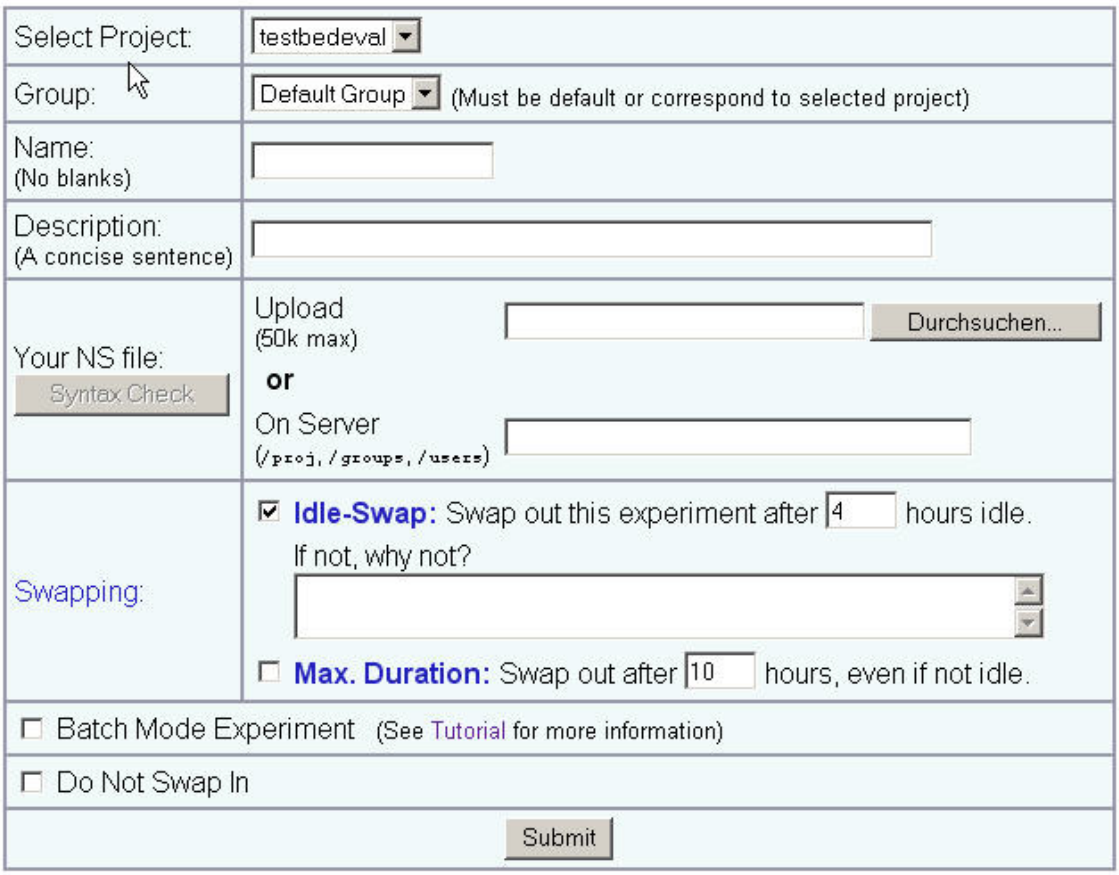

Abbildung 7: Neues Projekt starten

Mit „Submit“ startet man dann das Projekt. Der weitere Vorgang wird dann später unter „Was passiert mit dem gestarteten Experiment?“ beschrieben.

GUI wird verwendet

Als zweite Möglichkeit kann man die GUI von Emulab verwenden. Als erstes hat man einen leeren Bildschirm und 3 Funktionen:

- New Node
- New Lan
- Trash

Mit Drag and Drop zieht man die Knoten und LANs in die Arbeitsfläche. Wenn man Knoten und oder LANs miteinander verbinden möchte, klickt man auf den Knoten (LAN), den man verbinden möchte, hält „ctrl“ und klickt auf den Knoten (LAN), mit dem man den ersten verbinden möchte. Löschen kann man Knoten und LANs, indem man sie einfach in „trash“ zieht. Nachdem man dann die jeweilige Topologie aufgebaut

hat (siehe Abbildung 8), stellt man die Eigenschaften der LANs (siehe Abbildung 8) und der Knoten (siehe Abbildung 9) ein. Bei den LANs kann man den Namen, die Bandbreite, den Delay und die Verlustrate einstellen.

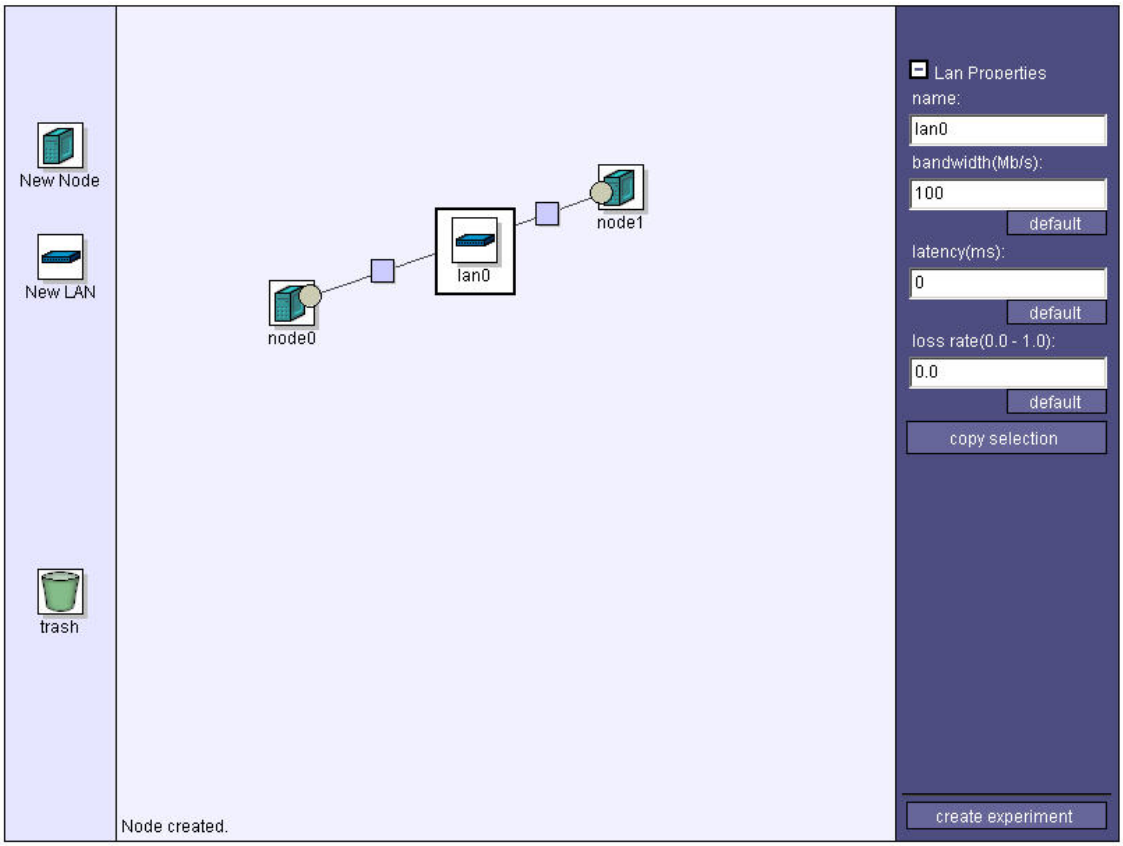

Abbildung 8: GUI

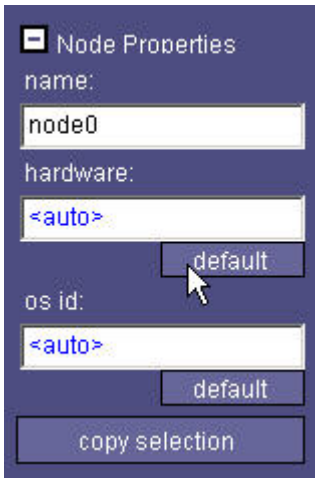

Abbildung 9: Eigenschaften der Knoten

Bei den Knoten kann man den Namen, die verwendete Hardware und das Betriebssystem einstellen. Wenn man alle Knoten und LANs gezeichnet und deren Eigenschaften eingestellt hat, kann man mit einem Klick auf „create experiment“ das Experiment starten. Der weitere Vorgang wird dann später unter „Was passiert mit dem gestarteten Experiment?“ beschrieben.

2.2.3. Create a PlanetLab Slice

Um „PlanetLab Slices“ zu erzeugen, benötigt man höhere Zugriffsrechte. Da wir diese Rechte nicht besitzen und diese Funktion hier auch nicht benötigt wird, werden wir dieses Feature nicht weiter behandeln.

2.2.4. Experiment List

Hier kann man sich eine Liste aller Experimente, die gerade im Projekt laufen, anzeigen lassen. Experimente, die ausgesetzt wurden sind hier nicht mehr zu finden, sondern nur mehr unter „My Emulab“.

2.2.5. Node Status

Unter Node Status kann man sich anzeigen lassen, wie viele Knoten noch zur Verfügung stehen (siehe Abbildung 10). Weiters kann man unter „pcs“ auch noch sehen, welche PCs verwendet werden und welche noch frei sind.


Show: summary, pcs, widearea.

Free Node Summary

Type	Free Nodes	Total Nodes
pc600	25	40
pc850	72	128

Abbildung 10: Node Status

2.2.6. List ImageIDs or OSIDs

Hier bekommt man eine Liste, welche Images bzw. Betriebssystem man auf die einzelnen Knoten laden kann. Zur Auswahl stehen mehrere Versionen von freeBSD, Redhat und andere Linux Images.

2.2.7. Start or Join a Project

Hier kann man wieder ein neues Projekt starten oder einem bestehenden Projekt beitreten. Ein neues Projekt starten kann man wie gehabt nur, wenn man zum Beispiel ein Universitätsprofessor ist oder in der Forschung arbeitet. Als Student kann man nur einem bestehenden Projekt beitreten, und das auch nur, wenn man vorher freigeschaltet worden ist.

2.2.8. Was passiert mit dem gestarteten Experiment

Bestätigungsmail und Setup Information des Experiments

Wenn man das Projekt gestartet hat, muss man noch auf das Bestätigungsmail von Emulab warten. Erst wenn man dieses Mail erhalten hat, sind die Knoten aktiv, und man kann auf diese zugreifen und mit dem Experiment arbeiten. Dieses Mail beinhaltet die Setup Informationen des Experiments:

```
Experiment: testbedeval/Test
State: active

Virtual Node Info:

ID          Type      OS       Qualified Name
----------- --------- -------- --------------------
nodeA       pc                 nodeA.Test.testbedeval.emulab.net
nodeB       pc                 nodeB.Test.testbedeval.emulab.net
nodeC       pc                 nodeC.Test.testbedeval.emulab.net
nodeD       pc                 nodeD.Test.testbedeval.emulab.net


Physical Node Mapping:

ID            Type      OS              Physical
------------- --------- --------------- ------------
nodeA         pc850     RHL73-STD       pc122
nodeB         pc850     RHL73-STD       pc121
nodeC         pc850     RHL73-STD       pc124
nodeD         pc850     RHL73-STD       pc123
tbsdelay0     pc850     FBSD47-STD      pc87
```

Virtual Lan/Link Info:

ID	Member	IP/Mask	Delay	BW (Kbs)	Loss Rate
lan0	nodeB:1	10.1.2.2	0.00	100000	0.000
		255.255.255.0	0.00	100000	0.000
lan0	nodeC:0	10.1.2.3	0.00	100000	0.000
		255.255.255.0	0.00	100000	0.000
lan0	nodeD:0	10.1.2.4	0.00	100000	0.000
		255.255.255.0	0.00	100000	0.000
link0	nodeA:0	10.1.1.2	25.00	30000	1.000
		255.255.255.0	25.00	30000	1.000
link0	nodeB:0	10.1.1.3	25.00	30000	1.000
		255.255.255.0	25.00	30000	1.000

Virtual Queue Info:

ID	Member	Q Limit	Type	weight/min_th/max_th/linterm
lan0	nodeB:1	50 slots	Tail	0/0/0/0
lan0	nodeC:0	50 slots	Tail	0/0/0/0
lan0	nodeD:0	50 slots	Tail	0/0/0/0
link0	nodeA:0	50 slots	Tail	0/0/0/0
link0	nodeB:0	50 slots	Tail	0/0/0/0

Physical Lan/Link Info:

ID	Member	Delay Node	Delay	BW (Kbs)	PLR	Pipe
link0	nodeA	tbsdelay0	50.00	30000	1.000	110
link0	nodeB	tbsdelay0	50.00	30000	1.000	120

Physical Queue Info:

ID	Member	Q Limit	Type	weight/min_th/max_th/linterm
link0	nodeA	50 slots	Tail	0/0/0/0
link0	nodeB	50 slots	Tail	0/0/0/0

Route List:

Node	Interface	Dest	Nexthop	Type	Cost
nodeA	10.1.1.2	10.1.2.0	10.1.1.3	net	0
nodeC	10.1.2.3	10.1.1.0	10.1.2.2	net	0
nodeD	10.1.2.4	10.1.1.0	10.1.2.2	net	0

Diese Informationen kann man sich auch in der shell anzeigen lassen. Man geht über ssh auf users.emulab.net (gleicher Username und Passwort wie bei dem normalen login) und dort kann man sich dann mit dem Befehl

```
tbreport -v pid eid
```

die Setup Informationen ausgeben lassen; dabei ist `pid` der Name des Projektes und `eid` der Name des Experiments.

Betrachten von Knoten

Mit Knoten kann entweder mit einem Konsolen-Programm oder mit einem speziellen Programm von Emulab interagiert werden.

Konsole:

1. `ssh users.emulab.net -l account-name` (gleicher Username und Passwort wie bei dem normalen login)
2. `console pc1` (um mit Knoten pc1 zu verbinden)

Wenn man dann mit dem Knoten verbunden ist, kann man ganz normale Befehle wie zum Beispiel einen Ping zu einem anderen Knoten ausführen.
Jede Konsolen-Ausgabe jedes einzelnen Knoten wird unter `/var/log/tiplogs/psXXX.run` gespeichert. In diesen Dateien werden jedoch nur Informationen über die Initialisierung des jeweiligen Knotens usw. gespeichert. Für die Auswertung bzw. Protokollierung des Experiments muss man zum Beispiel `tcpdump` verwenden (siehe 3.2). Achtung: sobald ein Experiment terminiert wird, werden diese Files gelöscht. Möchte man die Files sichern, so muss dies vor der Terminierung passieren.

Tiptunnel:

Mit Hilfe dieses Programmes ist es möglich .tbacl-files zu nutzen, um eine Verbindung zum im .tbacl-file beschriebenen Knoten zu erhalten. Das .tbacl-file erhält man, indem im Web-interface „Node View“ auf „Connect to serial line“ gegangen wird. Das angebotene File muss auf der Festplatte des Users gespeichert werden.
Unter Windows muss (nach Installation des tiptunnel-Programmes) das .tbacl-file nur mehr doppelt angeklickt werden, um eine Verbindung zum Knoten aufzubauen.

Das Programm und eine Installationsanleitung findet man unter folgender URL:
http://www.emulab.net/docwrapper.php3?docname=faq.html#UTT-TUNNEL
Das Installationsprogramm für Windows erhält man unter folgender URL:
http://www.emulab.net/downloads/tiptunnel-win32.exe

Beenden des Experiments

Sobald ein Experiment fertig ist und die reservierten Ressourcen nicht mehr benötigt werden, kann das Experiment beendet (terminiert) werden. Dies geschieht mittels Web-Interface: Mittels Klick auf „End An Experiment“ wird eine Liste mit allen Experimenten in allen Projekten angezeigt, zu denen der User das Recht hat, sie zu beenden. Um ein Experiment zu beenden muss auf den Button in der Spalte „Terminate“ geklickt werden (rechts daneben).
Nach der Bestätigung durch den User wird das System das Experiment beenden und dem User ein Email senden. Man kann nach der Terminierung das Experiment im Gegensatz zum „Swap out“ nicht mehr neu starten, sondern müsste das Experiment komplett neu anlegen.

2.3. Tools zur Auswertung

2.3.1. portstats

Portstats ist das einzige Tool, das von emulab zur Anzeige des Verkehrs zur Verfügung gestellt wird (außer den normalen Linux-tools wie tcpdump). Mit Hilfe von portstats hat man Zugriff auf ports vom switch. Um portstats verwenden zu können muss man sich mit Hilfe von ssh auf users.emulab.net einloggen; dann können mit dem folgenden Befehl die Statistiken der ports angezeigen werden:

```
portstats <pid> <eid>
```

`pid` ist dabei der Projektname und `eid` ist der Name des Experiments.

Mit dem Befehl

```
portstats -h <pid> <eid>
```

kann man sich eine Liste der Optionen, die man mit portstats hat, ausgeben lassen. Ein paar dieser Optionen wären zum Beispiel:

- `-a` zeigt alle Statistiken an

- `-e` zeigt nur die Anzahl der Fehler an
- `-q` unterdrückt die Ausgabe der Statistiken

Wenn man Informationen über alle Optionen erhalten möchte, sollte man sich entweder, wie schon gesagt, mit `-h` alle Optionen anzeigen lassen oder in der emulab Dokumentation nachlesen.
Bei portstats muss man auch noch beachten, dass der „Zähler" nicht neugestartet wird, sondern immer weiterzählt. Abbildung 11 zeigt die Ausgabe von portstats:

```
> portstats testbedeval test1

In        InUnicast  InNUnicast  Out   OutUnicast   OutNUcast
Port          Octets    Packets     Packets     Octets     Packets    Packets
-------------------------------------------------------------------------------
nodeA:3          660          8           1        626           3          5
nodeB:3          192          3           0        626           2          6
tbsdelay0:3      626          3           5        660           8          1
tbsdelay0:4      626          2           6        192           3          0
```

Abbildung 11: Portstats-Ausgabe

Diese Statistik ist aus der Sicht des Switches, das heißt, dass „In"-Pakete von den Knoten kommen und „Out"-Pakete zu den Knoten gehen.

2.3.2. tcpdump

Von emulab werden keine Tools für die Auswertung bzw. Protokollierung der Experimente zur Verfügung gestellt. Deshalb muss man sich Tools von Linux wie zum Beispiel tcpdump zuwenden. Mit Hilfe von tcpdump kann man den Datenfluss zwischen den Knoten mitprotokollieren. Auf die genaue Funktionsweise von tcpdump gehen wir hier nicht ein, sondern verweisen auf Dokumentationen zum Thema tcpdump im Internet.[8] [9]

[8] Siehe: http://t-marin.thunderhawk.de/modules.php?name=News&file=article&sid=18

[9] Siehe: http://www-iepm.slac.stanford.edu/monitoring/passive/tcpdump.html

Unter Linux erhält man mit dem Befehl `tcpdump -h` eine kurze Hilfestellung zu tcpdump.

Bevor man tcpdump nutzen kann, muss man Informationen über die verschiedenen Netzwerkkarten erhalten.

Mit dem Befehl:

```
ifconfig –a
```

erhält man die „Nummern“ der verschieden Ethernetkarten. Man sucht sich daraufhin den Knoten, den man mit Hilfe von tcpdump „abhören“ möchte, aus dieser Liste (zum Beispiel „eth4“) und verwendet dann die Ethernetkarte der Nummer, die man erhalten hatte.

Da man nun weiß, wie der gewünschte Knoten angesprochen werden kann, kann man tcpdump starten:

```
sudo tcpdump –i eth4
```

In diesem Beispiel hört tcpdump die Ethernetkarte 4 ab. Dabei gibt man mit `-i` das Interface, in unserem Fall `eth4` an. `sudo` benötigt man um root-Rechte für die Ausführung von tcpdump zu erhalten. Ohne `sudo` am Anfang des Befehls erhält man die Fehlermeldung „Permission Denied“.

2.4. Wichtige Details am Rande

Ein sehr wichtiger Punkt bei der Verwendung von Emulab ist das Ansprechen der Knoten. In dem von Emulab generierten Bestätigungsmail mit den Setup Informationen des gestarteten Experiments sieht man von den einzelnen Knoten nicht nur die Bezeichnung (z.B.: PC47), sondern auch die IP-Adresse. Wenn man mit den Knoten arbeiten will, darf man nicht die Bezeichnung angeben, sondern muss mit der IP-Adresse des jeweiligen Knotens arbeiten. Die Verwendung der Bezeichnung funktioniert zwar auch, nur finden die Tests dann am realen Netzwerk von Emulab statt und nicht auf dem erzeugten mit den gewünschten Parametern.

Kapitel 3

Emulator in NS

3.1. Allgemeines

3.1.1. Was ist Ns-2?

Ns-2 ist ein Netzwerk-Simulator, der die Funktionalität bietet, auch als Emulator verwendet zu werden. Der Simulator selbst läuft auf sämtlichen Plattformen. Ns-2 ist Freeware und kann unter www.isi.edu/nsnam herunter geladen werden.
Für Windows ist die Installation unter Cygwin möglich. Zum einfachen Verwenden gibt es aber auch ein ausführbares Programm als exe-File: www.isi.edu/nsnam/dist/binary/.
Wir verwendeten das Programm „ns-2.1b8a.exe“.

Das Simulator-Programm erwartet als Eingabe eine Datei, die eine Beschreibung des zu simulierenden Netzwerkes darstellt. Dieses Netzwerk wird in Tcl beschrieben. Man kann das Tcl-File so schreiben, dass das Programm eine Datei erzeugt, die mit NAM (Network AniMator) betrachtet werden kann (z.B. out.nam).

Mit NAM ist es möglich den Datenfluss graphisch darzustellen. Die Zeit kann gestoppt, zurückgesetzt oder „vorgespult“ werden. Ein Analysieren, was in der Simulation bei den einzelnen Knoten passierte, ist so leicht möglich. NAM bietet weiters die Möglichkeit einfache ns-Files mittels GUI zu erzeugen. Aus diesen könnte dann mit ns-2 ein out.nam erzeugt werden, welches unter NAM betrachtet werden kann.

NAM ist ebenfalls als ausführbare Datei für Windows erhältlich: www.isi.edu/nsnam/dist/binary/. Wir verwendeten das Programm „nam-1.0a11a-win32.exe“.

Um diese Programme zu nutzen, verwendet man die Konsole - z.B.:

1. ins Verzeichnis mit den Programmen wechseln
2. `ns <Pfad des NS-Files in Tcl>`
   bzw. `nam <Pfad des durch ns generierten NAM-Files>`

3.1.2. Ns-2 und Emulation

Mit dem inkludierten Emulator ist es möglich, den Simulator in ein wirkliches Netzwerk einzubinden. Spezielle Objekte des Simulators machen es möglich, live traffic in den Simulator einzuspeisen und ebenfalls vom Simulator traffic in ein reales Netzwerk zu speisen. So kann man ein simuliertes und ein reales Netzwerk miteinander koppeln.

Mittels NAM ist es dann möglich den Datenfluss graphisch darzustellen. Die Netzwerkkarten des Rechners, auf dem der Emulator ausgeführt wird, stellen Knoten dar, welche auf besondere Art beschrieben werden müssen, um sie ins simulierte Netzwerk einzubinden.

Es gibt 2 Modi von Emulationsverfahren:

- Opaque Mode
- Protocol Mode

Opaque Mode

Der Simulator wirkt nach außen als Router:

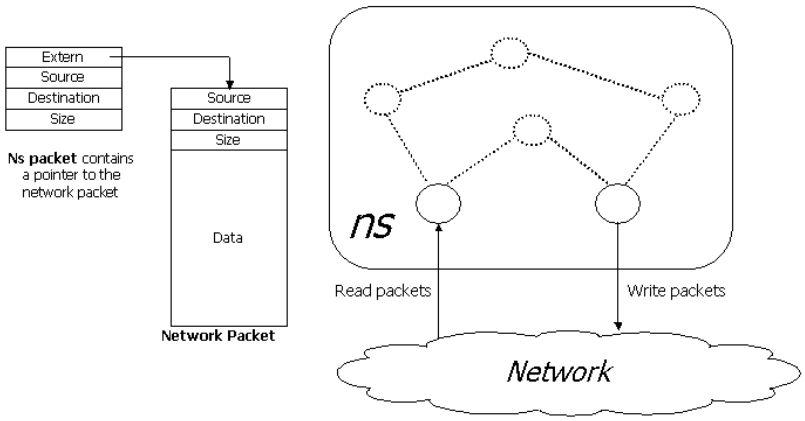

Abbildung 12: Opaque-Mode

Live Traffic läuft durch den Simulator. Der Simulator kann mit den erhaltenen Paketen arbeiten. Er kann analysieren, wie das simulierte Netzwerk mit den erhaltenen Paketen umgeht oder wie diese durch andere erhaltene Pakete beeinflusst werden. Live-data Pakete könnten verloren gehen, verzögert, umgeordnet, vervielfältigt, usw. werden.

Protocol Mode

Der Simulator wirkt nach außen als eine end-station.

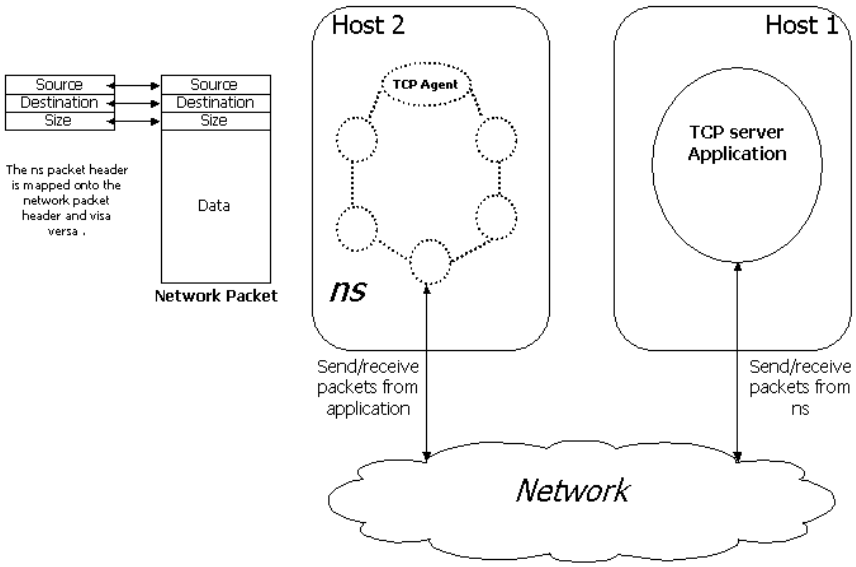

Abbildung 13: Protocol-Mode

Der Simulator stellt nach außen eine Quelle oder eine Senke für live-traffic dar, welche mit realen Netzwerken kommunizieren.

Die aktuelle Dokumentation in englischer Sprache ist unter www.isi.edu/nsnam/ns/ zu finden. Zu beachten ist, dass die angebotene HTML-Dokumentation älter ist als die PDF-Dokumentation. Laut dieser Dokumentation ist bislang nur der Opaque Modus implementiert worden (Dokumentation ist seit März 1998 unverändert).

Die Emulator-Funktionalität unter ns-2 heißt nse. Die aktuelle Dokumentation besagt, dass der Emulator unter dem Betriebssystem freeBSD 2.2.5 entwickelt und auch nur dort getestet wurde. Versuche, freeBSD mit Version 2.2.5 aufzutreiben, scheiterten; auf der Homepage von freeBSD[10] werden nur mehr Versionen ab 4.8 angeboten. Mit der Suchmaschine Google[11] konnten wir auch keine andere Quelle für die Version 2.2.5

[10] vgl. http://www.freeBSD.org

[11] vgl. http://www.google.at

finden. Man kann Versionen von freeBSD 4.8 bis 5.2 von diversen Anbietern erwerben (Installations-CDs kaufen) oder die Images direkt von freeBSD herunterladen: ftp://ftp.FreeBSD.org/pub/FreeBSD/
Es stellte sich heraus, dass die Emulator-Funktionalität für freeBSD geschrieben wurde. Versuche, dieses Betriebssystem zu installieren, erwiesen sich als äußerst. So schwenkten wir um und versuchten ns-2 auf einem bereits installierten Redhat-Linux zu installieren, was ohne Probleme funktionierte. Dennoch möchten wir im Zuge dieser Bakkalaureats-Arbeit einige Erkenntnisse wiedergeben, welche uns beim Versuch, freeBSD zu installieren, aufgefallen sind, da ja freeBSD das eigentliche Betriebssystem für ns-2 und dessen Emulator-Funktionalität ist.

Wenn man aktuellere ns-Versionen betrachtet, so gibt es dort ein Unterverzeichnis „emulate“, welches darauf schließen lässt, dass diese Versionen die Emulator-Funktionalität unterstützen. Auf der Homepage von ns gibt es aber keinerlei Dokumentation über diese Funktionalität bzw. darüber, welche Version von ns-2 diese Funktionalität unterstützt. So ist es extrem mühsam den Emulator zum Laufen zu bringen und dann zu verwenden.

3.1.3. Installation von freeBSD

Wir haben es nur geschafft, freeBSD 4.9 von ftp://ftp.FreeBSD.org/pub/FreeBSD/ und der Datei 4.9-i386-disc1.iso ohne funktionierendes X-Window-System (graphische Oberfläche für Linux) zu installieren. Um jedoch Resultate mittels NAM betrachten zu können, ist die graphische Oberfläche notwendig. Zusätzlich konnten wir unter diesem Betriebssystem keine Netzwerkkarte ansprechen, da wir die notwendigen Treiber dafür nicht hatten bzw. diese für freeBSD nicht angeboten werden. Das Image 4.9-i386-disc2.iso wurde bei der Installation nicht benötigt.

Die Installations-Anleitung auf http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/install.html ist sehr sinnvoll. Vor allem wichtig für die spätere Nutzung mit NS-2 ist der Abschnitt „2.9 Post-Installation“. Dieser gibt Aufschlüsse darüber, wie das Betriebssystem auf die richtige Zusammenarbeit mit dem Netzwerk konfiguriert wird. Wir installierten das System auf einer leeren Festplatte mit der vorgeschlagenen Partitionierung. Wir wählten die Installationsart „X-User“, was nur die Binaries, nicht aber den Source-Code installiert. Bei der Installation sollte das Lan-

Kabel an den Router angeschlossen sein, da das Installationsprogramm so gleich notwendige Konfigurations-Einstellungen treffen kann. Nach der Installation sollte noch manuell das Paket „tcl83“ installiert werden. Dieses wird dann von NS-2 benötigt. Das X-Window-System wird benötigt, um NAM nutzen zu können.

Weiters ist zu beachten, dass man nach erfolgter Eingabe oft nicht sofort zurückkommt. Daher sollte jede Eingabe wohl überlegt werden, da bei Fehl-Eingaben etliche Schritte eingegeben werden müssen, bevor man, wenn überhaupt, wieder vom Anfang beginnen kann. Die oben erwähnte Online-Dokumentation kann als Entscheidungshilfe herangezogen werden.

Fehlende Pakete o. ä. können im Nachhinein unter „sysinstall Main Menu“ – „Configure“ nachinstalliert werden.

Wie gesagt, konnten wir das X-Window-System nicht zum Laufen bringen. Man könnte jedoch freeBSD zum Produzieren der NAM-files verwenden. Da NAM jedoch graphisch arbeitet, muss in diesem Fall das NAM-file dann auf einem anderen Rechner (z.B. Windows mit installiertem NAM-binary nam-1.0a11a-win32.exe) betrachtet werden.

Der Versuch, die notwendigen Daten für die Installation von NS-2 von CD auf das System zu bekommen, scheiterte. Beim Befehl „mount /dev/acd0c“ bleibt das System hängen – es lässt sich weder die CD auswerfen, noch wird etwas gemountet, noch kann man die CD unmounten. Da weiters die Netzwerkkarte nicht angesprochen werden konnte, wandten wir uns vom Versuch, den ns-Emulator auf freeBSD zu installieren, ab und versuchten es auf Redhat 8.0, was sich als weniger mühsam herausstellte.

3.1.4. Installation von ns-2

Nse läuft zurzeit auf BSD-, Linux- und Mac-Betriebssystemen.[12] Es wird aber daran gearbeitet, es auf Cygwin zum Laufen zu bringen, damit es auch auf Windows-Plattformen verwendet werden kann.

[12] Vgl. http://www.sims.berkeley.edu/~christin/ns-cygwin.shtml

Um die Emulator-Funktionalität von ns nutzen zu können muss ns zuerst installiert werden. Ns besteht aus mehreren Paketen. Es werden jedoch auch all-in-one-Versionen von ns angeboten, welche alle Pakete enthalten. Die all-in-one-Version benötigt jedoch ca. 250MByte an Plattenplatz in installierter Version, daher ist es sinnvoll nur diejenigen Pakete zu installieren, die benötigt werden. Unter http://www.isi.edu/nsnam/ns/ns-build.html ist eine Installationsanleitung zu finden, mit dem auch die einzelnen Pakete installiert werden können.

Die nachfolgende Installationsanleitung bezieht sich auf das Betriebssystem Redhat 8.0 und die ns-Version ns-allinone-2.26.tar.gz.

Nach dem Download unter www.isi.nsnam/dist/ns-allinone-2.26.tar.gz und dem Entpacken der Datei braucht nur das install-Skript ausgeführt zu werden. Nach jedem Schritt muss man nur den leicht verständlichen Aufforderungen folgen. Zu beachten ist, dass bei uns die Installation ca. 40 Minuten und der Schritt „validate“ ca. 2,5 Stunden dauerten. Die Abschluss-Tests, welche mit „validate“ gemacht werden, führen im Normalfall bei Verwendung der „all in one“-Version zu keinerlei Probelmen ud können daher abgebrochen werden.

Nachdem ns installiert wurde, muss der Emulator kompiliert werden. Dazu muss im Verzeichnis `/ns-2.26` der Befehl `make nse` eingegeben werden. Diverse News-foren zeigen, dass es beim Ausführen von „make nse“ zu einer Fehlermeldung kommt, weil die Datei pcap.h nicht gefunden wird. Dies liegt daran, dass das benötigte Paket „lipcap“ oft nicht mit dem Betriebssystem mitgeliefert wurde. Dies ist z.B. bei Redhat 8.0 und Suse 8.0 der Fall.

Unter http://tcpdump.org/ kann man libpcap downloaden. Wir haben uns für lipcap-0.7.2.tar.gz entschieden, da diese Version am 27.2.2003 erschienen ist und die nächste Version am 29.12.2003 eventuell nicht verträglich ist mit der Version von ns, die vor Dezember 2003 erschienen ist. Die Installation von libcap ist einfach und wird daher hier nicht beschrieben. Nach Installation von libpcap könnte es (nach erneutem make nse) zu einer weiteren Fehlermeldung kommen, die wie folgt beginnt:

```
> emulate/net-ip.o emulate/net.o emulate/tap.o emulate/ether.o
> emulate/internet.o emulate/ping_responder.o emulate/arp.o
emulate/icmp.o
> emulate/net-pcap.o emulate/nat.o emulate/iptap.o emulate/tcptap.o
emulate/inet.o
> -L/home/anc/nsim/ns/ns-allinone-2.1b8a/tclcl-1.0b11 -ltclcl
> -L/home/anc/nsim/ns/ns-allinone-2.1b8a/otcl-1.0a7 -lotcl -
L/home/anc/nsim/ns/ns-allinone-2.1b8a/lib
> -ltk8.3 -L/home/anc/nsim/ns/ns-allinone-2.1b8a/lib -ltcl8.3 -
L/usr/X11R6/lib -lXext
> -lX11 -lnsl
> -ldl -lm
> emulate/net-pcap.o: In function `PcapNetwork::close(void)':
> emulate/net-pcap.o(.text+0x92): undefined reference to `pcap_close'
> emulate/net-pcap.o: In function `PcapNetwork::filter(char const *)':
> emulate/net-pcap.o(.text+0xdf): undefined reference to
`pcap_compile'
...
```

Sollte dies der Fall sein, wurde die libcap-library zwar gefunden, jedoch noch nicht in die Compilation inkludiert. Also muss man dies in das Makefile inkludieren, was folgendermaßen geschieht: In Makefile.in (nicht Makefile!) bei dem Statement

```
$NSE):      $(OBJ) ...
            &(LINK) $(LDFLAGS) $(LDOUT)$@ \
                  common/tclAppInit.o $(OBJ) \
                  &(OBJ_EMULATE_CC) $(OBJ_EMULATE_C) $(LIB)
```

müssen die Object-Files von libcap mitgelinkt werden, was zu folgender Änderung führt:

```
$NSE):      $(OBJ) ...
            &(LINK) $(LDFLAGS) $(LDOUT)$@ \
                  /usr/libcap-0.7.2/bpf_dump.o \
                  /usr/libcap-0.7.2/bpf_filter.o \
                  /usr/libcap-0.7.2/bpf_image.o \
                  /usr/libcap-0.7.2/etherent.o \
                  /usr/libcap-0.7.2/gencode.o \
                  /usr/libcap-0.7.2/... .o \
                  ...
                  common/tclAppInit.o $(OBJ) \
                  &(OBJ_EMULATE_CC) $(OBJ_EMULATE_C) $(LIB)
```

Da wir nicht genau wussten, welche .o-Files notwendig waren, banden wir einfach alle .o-Files, welche im Verzeichnis `libcap-0.7.2\` zu finden waren ein.

Wie aufgefordert, muss man anschließend `./configure` erneut ausführen, um dann auf das von Makefile.in generierte Makefile mit make nse das ausführbare Programm nse zu kompilieren. Nach Abschluss obiger Installation ist im Verzeichnis `ns-2.26/` die Datei „`ns`“ durch „`nse`“ ersetzt worden. Der NS-Emulator kann jetzt verwendet werden.

Nun sollte die richtige Funktionsweise des Emulators getestet werden. Dazu eignet sich das mit ns-allinone mitgelieferte File pingdemo.tcl, welches unter dem Verzeichnis emulate zu finden ist (ist auch im Internet mit der Suchmaschine google leicht aufzutreiben). Zu Beginn der Datei ist beschrieben, wie man sie verwendet. Es müssen einige Änderungen gemacht werden:

- `myname`: Wert ist egal
- `ifname`: Wie die Netzwerkkarte angesprochen wird; Den Namen erhält man z.B. unter Netzwerk-Konfiguration (GUI-Programm unter Redhat 8.0) und er lautet z.B. eth0, eth1 o.ä.
- `myaddr`: die IP-Adresse des Rechners, auf dem nse läuft. Diese erhält man z.B. mit dem Shell-Befehl `ifconfig`.

Nach den aufgeforderten Änderungen in pingdemo.tcl wird mittels Befehl in der Shell der Emulator gestartet:

```
./nse emulate/pingdemo.tcl
```

Bei unserem Test-Rechner ließ sich dieses File nicht ausführen. Folgende Gründe waren ausschlaggebend:

1.: Im File kommt

```
exec sysctl -w net-inet.ip.forwarding=0 net.inet.ipredirect=0
```

vor. Wenn man `sysctl -a | grep forward` in die Shell eingibt, so sieht man, wie `sysctl` aufgerufen werden kann. Bei unserem Test-Rechner wurde etwas anderes angeboten, so änderten wir die Zeile zu

```
exec sysctl -w net-ipv4.conf.all forwarding=0
net.ipv4.conf.all.accept_redirects=1
```

2.: `$ns: no such variable in newowdelay_cycle{}`

Aus irgendeinem Grund wird in dieser Funktion $ns nicht erkannt. Da jedoch direkt vor dem Statement mit $ns eine andere Funktion aufgerufen wird, die ebenfalls $ns verwendet und dabei keine Fehlermeldung erzeugt, kann man auch in dieser Funktion das Statement ausführen lassen. Dies funktioniert problemlos.

Nach den Änderungen erhielten wir folgenden Code für pingdemo.tcl (der Original-Code ist ebenfalls noch enthalten, jedoch auskommentiert):

```
Class PingDemo

PingDemo instproc init {} {
      $self next
      $self instvar myname myaddr dotrace stoptime owdelay ifname
delay_list next_delay_list
      $self instvar traceallfile tracenamfile

# original {
#     switch [exec hostname] {
#     pollytto* {
#           set myname "dash-w.isi.edu"
#           set ifname cnw0
#           set dotrace 0
#     }
#     kfalls-host* {
#           set myname "bit.ee.lbl.gov"
#           set ifname fxp0
#           set dotrace 1
#     }
#     default {
#           puts "error: running on an unknown host; edit
PingDemo::init"
#           exit 1
#     }
#     }

#     set myaddr [exec host $myname | sed "s/.*address\ //"]
#} ende original

# geändert zu {:
      set myname "desktop"
      set ifname eth0
      set dotrace 0

      set myaddr "192.168.1.101"
#} ende geändert zu

      set stoptime 200.0
      set owdelay 1000ms
      set delay_list {0.5 0.1 0.01 0.001}
      set next_delay_list ""

      set traceallfile em-all.tr
      set tracenamfile em.nam
      puts ""
}

PingDemo instproc syssetup {} {
      puts "turning off IP forwarding and ICMP redirect generation."
#original:
#     exec sysctl -w net.inet.ip.forwarding=0 net.inet.ip.redirect=0
#geändert zu:
      exec sysctl -w net.ipv4.conf.all.forwarding=0
net.ipv4.conf.all.accept_redirects=1
}

PingDemo instproc emsetup {} {
      $self instvar myname myaddr dotrace stoptime owdelay ifname
      $self instvar traceallfile tracenamfile ns

      puts "I am $myname with IP address $myaddr."
```

```
set ns [new Simulator]

if { $dotrace } {
      exec rm -f $traceallfile $tracenamfile
      set allchan [open $traceallfile w]
      $ns trace-all $allchan
      set namchan [open $tracenamfile w]
      $ns namtrace-all $namchan
}

$ns use-scheduler RealTime

set bpf2 [new Network/Pcap/Live]; # used to read IP info
$bpf2 set promisc_ true
set dev2 [$bpf2 open readonly $ifname]

set ipnet [new Network/IP];    #    used to write IP pkts
$ipnet open writeonly

set arpagent [new ArpAgent]
$arpagent config $ifname
set myether [$arpagent set myether_]
puts "Arranging to proxy ARP for $myaddr on my ethernet
$myether."
$arpagent insert $myaddr $myether publish

#
# try to filter out unwanted stuff like netbios pkts, dns, etc
#

$bpf2 filter "icmp and dst $myaddr"

set pfa1 [new Agent/Tap]
set pfa2 [new Agent/Tap]
set ipa [new Agent/Tap]
set echoagent [new Agent/PingResponder]

$pfa2 set fid_ 0
$ipa set fid_ 1

$pfa2 network $bpf2
$ipa network $ipnet

#
# topology creation
#

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]

$ns simplex-link $node0 $node1 1Mb $owdelay DropTail
$ns simplex-link $node1 $node0 1Mb $owdelay DropTail
$ns simplex-link $node0 $node2 1Mb $owdelay DropTail
$ns simplex-link $node2 $node0 1Mb $owdelay DropTail
$ns simplex-link $node1 $node2 1Mb $owdelay DropTail
$ns simplex-link $node2 $node1 1Mb $owdelay DropTail

$self instvar explink1 explink2; # link to experiment with
set explink1 [$ns link $node0 $node2]
set explink2 [$ns link $node2 $node1]

#
```

```
	# attach-agent winds up calling $node attach $agent which does
	# these things:
	#	append agent to list of agents in the node
	#	sets target in the agent to the entry of the node
	#	sets the node_ field of the agent to the node
	#	if not yet created,
	#		create port demuxer in node (Addr classifier),
	#		put in dmux_
	#		call "$self add-route $id_ $dmux_"
	#			installs id<->dmux mapping in classifier_
	#	allocate a port
	#	set agent's port id and address
	#	install port-agent mapping in dmux_
	#
	#
	$ns attach-agent $node0 $pfa2; #   packet filter agent
	$ns attach-agent $node1 $ipa; # ip agent (for sending)
	$ns attach-agent $node2 $echoagent

	$ns simplex-connect $pfa2 $echoagent
	$ns simplex-connect $echoagent $ipa
}

PingDemo instproc newowdelay delay {
	$self instvar explink1 explink2 ns owdelay
	set owdelay $delay
	set lnk [$explink1 link]
	puts "[$ns now]: change 1-way delay from [$lnk set delay_] to
$delay sec."
	$lnk set delay_ $delay
	set lnk [$explink2 link]
	$lnk set delay_ $delay
#original { leer }
#geändert zu {
	$ns at [expr [$ns now] + 10] "$self newowdelay_cycle"
#} ende geändert zu
}

# eternally cycle through the delays in delay_list
PingDemo instproc newowdelay_cycle {} {
	$self instvar delay_list next_delay_list
	if { "$next_delay_list" == "" } {
		set next_delay_list $delay_list
	}
	set next_delay [lindex $next_delay_list 0]
	set next_delay_list [lrange $next_delay_list 1 1]
	$self newowdelay $next_delay
#original {
#	$ns at [expr [$ns now] + 10] "$self newowdelay_cycle"
# } ende original
#geändert zu { leer }
}

PingDemo instproc run {} {

	$self instvar ns myaddr owdelay ifname explink
	$self syssetup
	$self emsetup

	puts "listening for pings on addr $myaddr, 1-way link delay:
$owdelay\n"

	$ns at 10.5 "$self newowdelay_cycle"
```

```
      $ns run
}

PingDemo thisdemo
thisdemo run
```

3.2. Beschreibung eines Netzwerkes für den ns-Emulator

Folgender Abschnitt wurde aus der englischsprachigen Dokumentation auf der Webpage von NS ins Deutsche übersetzt und vereinfacht. [13]

Die Schnittstelle zwischen dem Simulator und einem live network wird durch eine Sammlung von Objekten (tap agents und network objects) dargestellt. Tap agents verpacken Daten eines realen Netzwerks in simulierte Pakete bzw. bereiten simulierte Pakete so vor, damit diese an reale Netzwerke weitergegeben werden können.
Network Objects sind in tap agents installiert und bilden die Schnittstelle nach außen für das Senden / Empfangen von realen Daten. Beide Objekte werden im Folgenden beschrieben.

Wenn man den Emulations-Modus wählt, wird eine spezielle Version des System-Schedulers benötigt: der RealTime-scheduler. Dieser Scheduler verwendet dieselbe unterliegende Struktur wie der Standard-(calendar-queue based)-Scheduler, jedoch führt dieser Ereignisse in Echtzeit aus.

3.2.1. Real-Time Scheduler

Der Real-Time Scheduler implementiert einen einfachen Echtzeit Scheduler, welcher das Behandeln von Ereignissen innerhalb des Simulators an Echtzeit knüpft. Sollte nicht der Echtzeit Scheduler verwendet werden, könnte der Simulator schneller als in Echtzeit laufen. In solchen Fällen würde z.B. Paket c, welches durch das simulierte Netzwerk läuft, anders verzögert als von einem realen Netzwerk erwartet.

[13] Siehe: http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

Um den Real-Time Scheduler zu verwenden, muss folgende Spezifikation zu Beginn des Files stehen:

```
set ns [new Simulator]
$ns use-scheduler RealTime
```

3.2.2. Tab Agents

Diese Klasse ist von der Basisklasse „Agent“ abgeleitet. Mit Tab Agents ist es möglich, simulierte Pakete zu generieren, welche willkürlich zugewiesene Werte im NS-common-header enthalten. Der Tab Agent behandelt die Einstellungen im common-header der Felder „packet size“ und „type“. Es verwendet den Paket-Typ PT_LIVE für Pakete, die in den Simulator gesendet werden. Jedem Tab Agent wird genau ein simulierter Knoten zugewiesen, jedoch können einem simulierten Knoten mehrere Tab Agents zugewiesen werden.

Tap Agents sind in der Lage, Pakete vom verknüpften Netzwerk-Objekt zu empfangen und dahin zu senden.

Beispiel:

```
set a0 [new Agent/Tap]
$a0 network $netobj
$a0 set fid_ 26
$a0 set prio_ 2
$ns connect $a0 $a1
```

Flow-Id und Priority werden von der Agent-Basis-Klasse behandelt. Der Grund, warum Flow-Id hier gesetzt wird, ist, damit live-data-Pakete spezielle Labels erhalten.
Die Methode “connect” veranlasst, dass Agent $a0 live-traffic an den Agent $a1 sendet.

3.2.3. Network Objects

Diese bieten Zugriff auf live-Netzwerke. Es gibt verschiedene Arten von Netzwerk-Objekten, abhängig davon, welche Protokolle diese verwenden, um das darunterliegende Netzwerk anzusprechen. Generell bieten Netzwerk-Objekte einen Eingangspunkt in ein live-Netzwerk mit einem eigenen Protokoll (link, row IP, UDP, ...) und einer eigenen Zugriffs-Methode (read-only, write-only, read-write). Drei Netzwerk-Objekte werden bislang unterstützt: pcap/bpf, raw IP und UDP/IP.

Pcap/BPF Netzwerk-Objekte

Diese Objekte bieten eine erweiterte Schnittstelle zu LBNL-packet-capture-library (libpcap). Diese Bibliothek bietet die Möglichkeit link-layer-frames vom Netzwerk-Interface-Treiber entgegenzunehmen. Weiters bietet diese die Möglichkeit trace-files der Pakete im „tcpdump"-Format zu lesen / schreiben. Die erweiterte Schnittstelle erlaubt es, Frames zum Netzwerk-Interface-Treiber zu schreiben.

Pcap-Netzwerk-Objekte werden so konfiguriert, dass sie entweder mit live-networks oder mit trace-files verkehren.

Beispiel:

```
set me [exec hostname]
set pf1 [new Network/Pcap/Live]
$pf1 set promisc_ true
set intf [$pf1 open readonly]
puts "pf1 configured on interface $intf"
set filt "(ip src host foobar) and (not ether broadcast)"
set nbytes [$pf1 filter $filt]
puts "filter compiled to $nbytes bytes"
puts "drops: [$pf1 pdrops], pkts: [$pf1 pkts]"
```

Dieses Beispiel ermittelt zuerst den Namen des lokalen Systems, welches zur Erzeugung eines BPF/libpcap-Filter-Prädikat gebraucht wird.
Der Aufruf von new Network/Pcap/Live erzeugt eine Instanz des pcap-Netzwerk-Objektes zur Kopplung an live traffic.
Das open-Statement aktiviert den Paket-Filter und wird mit `readonly`, `writeonly`, oder `readwrite` angegeben. Es gibt den Namen des Netzwerk-Interfaces zurück, mit dem der Filter assoziiert wird. Diesem Befehl kann ein zusätzlicher Parameter übergeben werden, welcher den Namen der Netzwerkkarte angibt, falls mehrere Netzwerkkarten auf einem Rechner installiert sind: set intf[$bf1 open readonly eth1].

IP Netzwerk-Objekte

Diese Objekte bieten Zugriff auf das IP-Protokoll und ermöglichen komplette Spezifikation von IP-Paketen (samt Header).

Beispiel wie ein IP-Objekt konfiguriert wird:

```
set ipnet [new Network/IP]
$ipnet open writeonly
...
$ipnet close
```

Die IP-Netzwerk-Objekte unterstützen nur die Methoden „open“ und „close“.

IP / UDP Netzwerk-Objekte

Diese Objekte bieten Zugriff auf die systemeigene UDP-Implementation, mit Unterstützung für „IP multicast group membership“-Operationen.
Laut aktueller Dokumentation, welche seit März 1998 unverändert im Netz steht: „IN PROGRESS“. Genaueres findet sich in der aktuellen Dokumentation in englischer Sprache. Dort findet sich auch ein veranschaulichendes Beispiel.[14]

[14] Siehe: http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

Kapitel 4

NistNet

Mit Hilfe von Nist Net, das von Mark Carson am North American National Institute of Standards and Technology (Nist) entwickelt wurde, kann man im realen Netzwerk eine kleine Testumgebung schaffen. Der Vorteil dabei ist, dass die Testumgebung direkt am Netzwerk arbeitet. Nist Net arbeitet auf der IP-Ebene und klinkt sich direkt in den Kernel des Linux-Systems ein.

Nist Net wird auf einem Linux-System im Netzwerk installiert, welches als Router arbeitet. Es greift in den IP-forward-Mechanismus des Linux-Kernels ein. Dadurch kann man das Verhalten der Verbindungen zwischen den angeschlossenen Rechnern beeinflussen.

4.1. Allgemeines

Die offizielle Seite von NistNet ist http://snad.ncsl.nist.gov/itg/nistnet/. Dort finden sich Versionen zum Downloaden und Hilfen zum Installieren / Bedienen. Man kann sich auch in eine Mailingliste eintragen, um Fragen an andere Personen zu richten, die ebenfalls NistNet verwenden.

NistNet wird auf dem Betriebssystem Linux installiert und besteht aus zwei Teilen:

1. Kernel-Modul: Dieses wird in den normalen Linux-Kernel geladen. Es beeinflusst so den „networking“ und „real-time clock“ code von Linux.
2. Ein Bündel von User-Interfaces, mit welchen der Emulator gesteuert und konfiguriert werden kann. Angeboten werden eine graphische und eine kommandozeilen-basierte Schnittstelle.

Wie schon erwähnt, wird NistNet auf einem Linux-Rechner installiert, der als Router arbeitet. Somit ist es erforderlich, dass mehrere Netzwerkkarten in diesen Rechner eingebaut werden müssen. Um NistNet verwenden zu können muss IP-Forwarding eingeschalten sein. Dieser Mechanismus wird nach dem Starten von NistNet beeinflusst. Bei ausgeschaltenem Emulator arbeitet der Rechner somit als Router. Bei eingeschaltetem Emulator werden dann die durch den User beschriebenen Verbindungen entsprechend der Einstellungen beeinflusst.
Der Emulator kann somit ein- und ausgeschalten, oder umgeändert werden, ohne eine aktive Verbindung zu trennen. Es werden bei eingeschaltetem Emulator die Verbindungen beeinflusst, aber keine hergestellt, die bei ausgeschaltetem Emulator nicht vorhanden wären.

Die Funktionalitäten, die NistNet zur Beeinflussung des Netzwerks zur Verfügung stehen, sind folgende:

- packet delay: fixe und variable (jitter)
- packet recording
- Paketverlust: zufällig und Stau-abhängig
- Paket-Vervielfältigung
- Bandbreiten-Beschränkung

4.2. Installation von Nist Net

Im Folgenden geben wir eine Installationsanleitung für Nist Net an. Diese Installationsanleitung ist für die aktuelle Nist Net Version 2.0.12 gedacht. Die Version 2.0.12 wurde im Mai 2002 veröffentlicht.

Das NistNet-Software-Paket ist für jeden frei erhältlich. Der Sourcecode kann von der NistNet-Website heruntergeladen werden.[15]
Die Software ist einfach zu kompilieren. Keine speziellen Bilbliotheken sind erforderlich.

1. Voraussetzungen:

Nist Net 2.0.12 sollte auf den meisten Linux Versionen funktionieren. Getestet wurden die Versionen 2.0.27-2.0.39, 2.2.5-2.2.18 und 2.4.0-2.4.18. Nach unseren Untersuchungen ist es nicht möglich Nist Net auch unter Windows zu verwenden.

2. Software downloaden

Man lädt sich unter http://snad.ncsl.nist.gov/itg/nistnet/requestform.html die Nist Net Version 2.0.12 herunter. Die Datei bekommt man dann als *.tar.gz File.

3. Software entpacken

Mit dem Befehl *tar* `-xvzf nistnet.2.0.12.tar.gz` kann man die gezippte Datei entpacken. Man erhält dann ein Verzeichnis `/nistnet` in dem die Dateien in unkomprimierter Form vorliegen.

4. genauere Infos zu Ihrem Betriebssystem lesen

Gute Informationen zum jeweiligen Betriebssystem findet man im entpackten Verzeichnis z.B. in der Datei README.RedHat. Zu beachten ist, dass NistNet erfordert, dass der Linux-Kernel im Verzeichnis `/usr/src/linux` zu finden ist. Sollte dies nicht der Fall sein, so kann man einen Verweis erzeugen, der auf das Verzeichnis zeigt, in dem Ihr Kernel liegt. Genaueres steht in dem entsprechenden Dateien README.*.

[15] Siehe: http://snad.ncsl.nist.gov/itg/nistnet/

5. eventuell Kernel patchen

Für diese Nist Net Version benötigt man keinerlei Kernel patches. Falls aber bereits eine alte Version installiert ist und den Kernel wiederhergestellt werden soll, kann dies mit dem Skript `./Unpatch.Kernel` (liegt im NistNet Verzeichnis) geschehen.

Mit den Befehlen `cd /usr/src/linux` und dann `make menuconfig` läßt sich das Konfigurationsmenü des Kernels starten. Am Ende müssen die Änderungen noch gespeichern und das Konfigurationsmenü verlassen werden. Dies ist aber im Normalfall nicht notwendig, sondern man muss nur darauf zurückgreifen wenn man komplexere Experimente durchführen möchte.

6. Makefile erstellen

Starten der Nist Net Konfiguration und erstellen des Makefiles mit:

```
cd nistnet
./configure
```

Es muss nur den Vorgaben gefolgt zu werden: 1. „yes“, 2. „no“. Nach Eingabe der Optionen wird dann das Makefile automatisch erstellt.

7. NistNet installieren

Als nächsten Schritt installiert man NistNet, die API library und das user interface.

```
make install
```

Nach diesen Schritten sollten das Kernel-Modul und beide User-Interfaces (kommandozeilen-basierte und GUI-Version) kompiliert sein.

4.3. Arbeiten mit Nist Net

Normalerweise wird NistNet ja auf einem Linux-basierten Router installiert. Dieser Router sollte mehrere Netzwerkkarten (mindestens 2) installiert haben, von denen jede einzelne die Schnittstelle zu einem anderen IP-Subnetz bietet.

Zu Beginn muss NistNet in den Kernel des Linux-Systems eingebunden werden. Dies geschieht mittels des Befehls

```
modprobe nistnet
```
bzw.
```
Load.nistnet
```

Um anschließend mit NistNet arbeiten zu können, ist es erforderlich, als ROOT angemeldet zu sein.

Es gibt drei Möglichkeiten mit Nist Net zu arbeiten. Neben den zwei Kommandozeilen-Interpretern gibt es auch noch eine graphische Oberfläche von Nist Net.

- cnistnet: neuer Kommandozeilen-Interpreter
- Hitbox: alter Kommandozeilen-Interpreter
- xnistnet: GUI-version

Die Kommandozeilen-Interpreter dienen großteils dazu, Skripte zu laden und zu verarbeiten. Im Folgenden werden die 3 Varianten kurz vorgestellt; uuvor jedoch noch einige Befehle zum Arbeiten mit NistNet:

4.3.1. Befehle von Nist Net

Starten von NistNet

`Load.nistnet`	Lädt den Nist Net Emulator in den Kernel
`xnistnet`	Startet das user Interface
`cnistnet –h`	Benutzer Information für das command-line interface
`insmod mungemod`	Lädt „mungebox“ (das ist eine Emulatorerweiterung)
`mungebox -u -a src det –S`	Verkehr zwischen src und dest mit mungebox überwachen
`insmod nistspy`	Lädt die Erweiterung „nistspy“

Beenden von Nist Net

`cnistnet –d`	Beendet den emulator

`rmmod mungemod`	Beendet die Erweiterung „mungebox“. Zum Beenden von „nistspy“ geht man gleich vor.
`rmmod nistnet`	Entfernt Nist Net aus dem Kernel

4.3.2. cnistnet

Die Befehle zu den Argumenten wurden direkt aus der Nist Net Dokumentation übernommen:[16]

`-u` up (on)

`-d` down (off)

`-a src[:port[.protocol]] dest[:port[.prot]] [cos]`
add new
[--delay delay [delsigma[/delcorr]]]
[--drop drop_percentage[/drop_correlation]]
[--dup dup_percentage[/dup_correlation]]
[--bandwidth bandwidth]
[--drd drdmin drdmax [drdcongest]]

`-r src[:port[.prot]] dest[:port[.prot]] [cos]`
remove

`-s src[:port[.prot]] dest[:port[.prot]] [cos]`
see stats

`-S src[:port[.prot]] dest[:port[.prot]] [cos]`
see stats continuously

`[-n] -R`
read current settings (-n numerical format)

`-D value`
debug on (value=0 none, 1 minimal,... 9 maximal)

`-U` debug off

`-G` global stats

`-K` kickstart the clock

`-F` flush the queues

`-h` this help message

[16] vgl. http://snad.ncsl.nist.gov/itg/nistnet/usage.html

4.3.3. Hitbox

Die Argumente zu cninstnet wurden direkt aus der Nist Net Dokumentation übernommen:[17]

`-u` up (turn emulator on)

`-d` down (turn emulator off - entries are retained)

`-a src dest delay delsigma bandwidth drop dup drdmin drdmax`
add new entry

`-r src dest`
remove entry

`-s src dest`
see stats for entry

`-S src dest`
see stats for entry continuously

`-R` read current list of entries in kernel

`-D` debug on

`-U` debug off

`-G` see global stats

4.3.4. xnistnet

Xnistnet ist die GUI Version von Nist Net. Es können direkt bestimmte Regeln für die Emulation eingestellt werden. In jeder Zeile wird eine neue Regel eingegeben. Es werden dann Quell- und Zielrechner festgelegt, und Bandbreite, Delay, usw. angegeben. Die eingegebenen Regeln sind allerdings nur unidirektional und müssen deshalb für beide Richtungen separat definiert werden. Mit den folgenden Grafiken wird die Oberfläche von NistNet kurz vorgestellt. Die Grafiken stammen von der NistNet Seite.[15]

[17] vgl. http://snad.ncsl.nist.gov/itg/nistnet/usage.html

Die erste Grafik (siehe Abbildung 14) zeigt einen Screenshot von Nist Net. Dabei werden die einzelnen Eingabeoptionen erläutert.

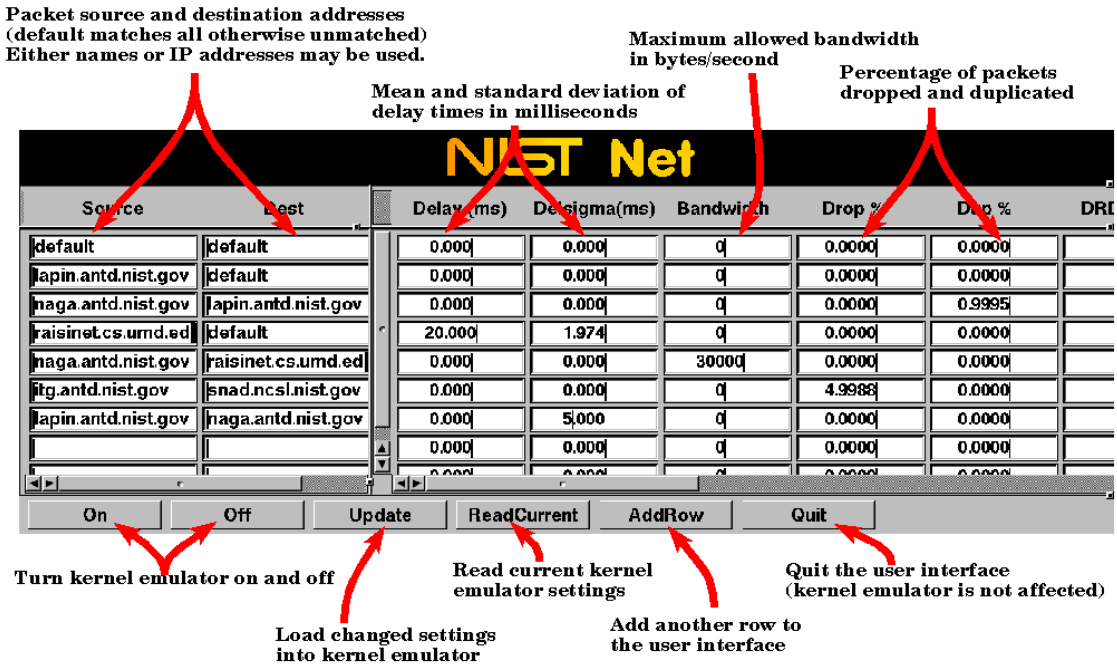

Abbildung 14: xnistnet - Beschreibung der graphischen Oberfläche

Jede Zeile des Fensters repräsentiert eine Emulations-Regel. Im linken Bereich können die Rechner angegeben werden, deren Verbindung während des IP-forwarding beeinflusst werden soll. Man beachte, dass die Regeln nur unidirektional gelten. Möchte man eine bidirektionale Regel einführen, so müssen dafür 2 Zeilen beschrieben werden.

Auf der rechten Seite können die Parameter der Verbindung eingestellt werden. Wenn man diesen Bereich noch weiter nach rechts scrollt, so kann man dort statistische Werte sehen, die während des Arbeitens als Emulator laufend aktualisiert werden.

Abbildung 15 zeigt die Bearbeitung der Verbindungs-Parametr.

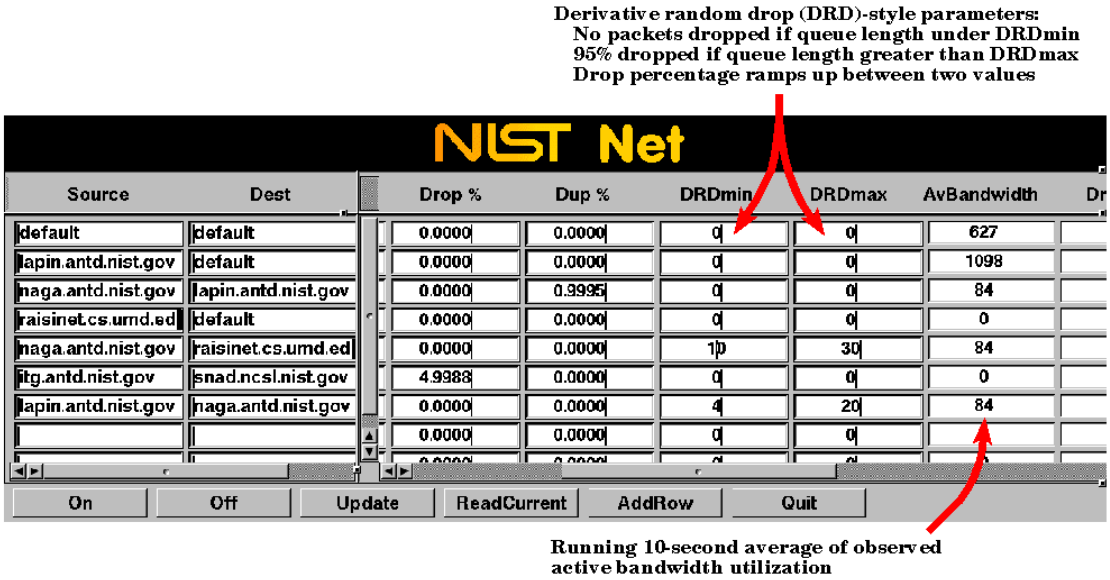

Abbildung 15: xnistnet - Parameter der Verbindung bearbeiten

Neben den prozentuell verlorenen und verdoppelten Paketen werden in Nist Net auch noch Parameter wie die aktive Bandbreite ausgegeben (siehe Abbildung 16

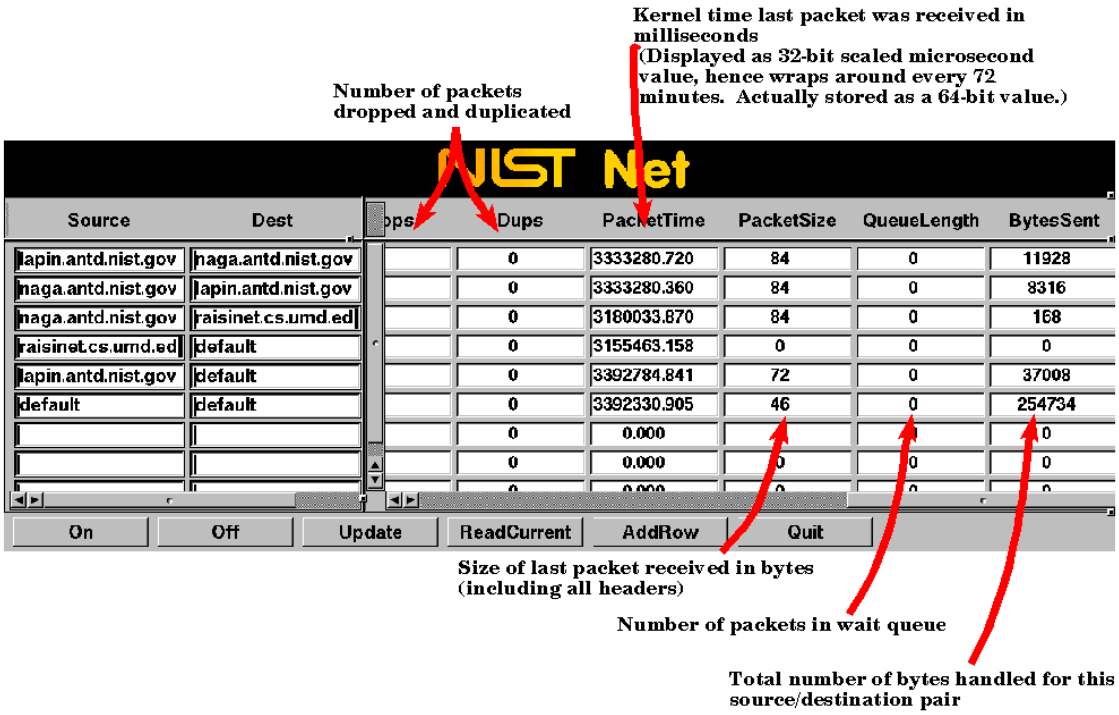

Abbildung 16: xnistnet – Ausgaben

Weiters werden die verlorenen und verdoppelten Pakete auch absolut ausgegeben. Die Paketzeit, Paketgröße, Warteschlangenlänge und Anzahl der gesendeten Bytes werden auch noch angeführt. Die Oberfläche sieht nicht in jedem NistNet-Paket gleich aus. Mit jedem Paket wird eine Hilfe mitgeliefert, in der jedes Feld und deren Funktion genauer erklärt wird.

Kapitel 5

Vergleich der Emulatoren

5.1. Definition der Vergleichssituation

Um die verschiedenen Emulatoren besser vergleichen zu können haben wir uns ein Referenzsystem definiert. Dieses System besteht aus 3 Rechnern bzw. Knoten, die jeweils miteinander verbunden sind (siehe Abbildung 17).

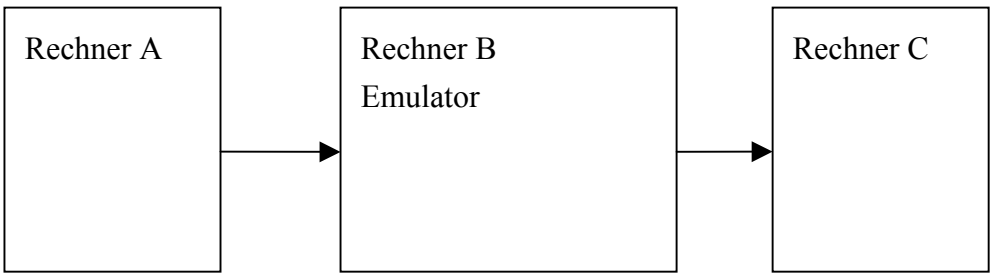

Abbildung 17: Aufbau des Vergleichsszenarios

Die Verbindungen zwischen A und B bzw. B und C haben folgende Eigenschaften:

- Bandbreite: 100 Mbit
- Latenzzeit: 10 ms
- Verlustrate: 0

Auf Rechner B wird immer der jeweilige Emulator installiert, und dann wird von A nach C ein Ping gesendet. Die Funktionsweise und produzierten Ergebnisse werden hierauf verglichen.

5.2. emulab

Wie schon erwähnt, verwenden wir als Experiment ein System mit 3 Knoten. In der folgenden Abbildung (Abbildung 18) wird dieses System dargestellt:

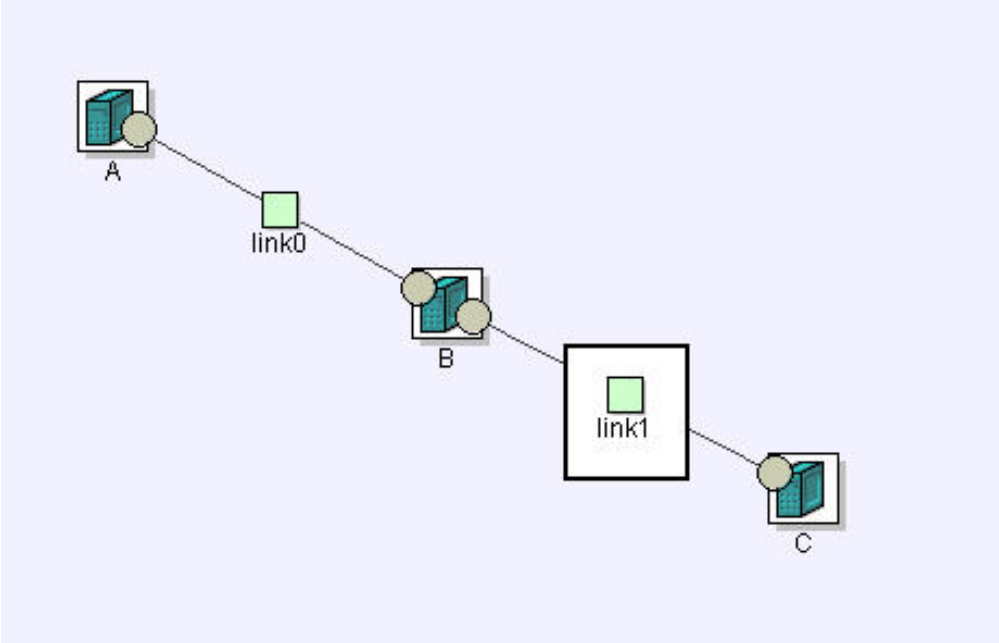

Abbildung 18: Referenzsystem

Das Experiment wurde mit Hilfe der GUI von emulab erstellt. Daher führen wir zum Vergleich mit nse bzw. NistNet das automatisch von emulab generierte ns-File an (Abbildung 19).

```
#generated by Netbuild 1.03
set ns [new Simulator]
source tb_compat.tcl

set A [$ns node]
set B [$ns node]
set C [$ns node]
set link0 [$ns duplex-link $A $B 100Mb 10ms DropTail]
set link1 [$ns duplex-link $B $C 100Mb 10ms DropTail]
$ns rtproto Static
$ns run
#netbuild-generated ns file ends.
```

Abbildung 19: automatisch erzeugtes ns-File

Dieses Experiment wird gestartet, und mit Hilfe von tcpdump wird der Datenfluss mitprotokolliert.

Als erstes möchten wir das Problem mit dem Ansprechen der Knoten (siehe Kapitel 5 Probleme) behandeln. Wenn man die Knoten nicht mit der IP sondern mit der Bezeichnung (z.B.: PC47) anspricht, erhält man bei einem Ping von Knoten A nach Knoten C (PC53) folgende Ausgabe (Abbildung 20):

```
[MarkusS@a ~]$ ping pc53
PING pc53.emulab.net (155.101.132.53) from 155.101.132.68 : 56(84) bytes of data.
64 bytes from pc53.emulab.net (155.101.132.53): icmp_seq=1 ttl=64 time=0.321 ms
64 bytes from pc53.emulab.net (155.101.132.53): icmp_seq=2 ttl=64 time=0.129 ms
64 bytes from pc53.emulab.net (155.101.132.53): icmp_seq=3 ttl=64 time=0.144 ms
64 bytes from pc53.emulab.net (155.101.132.53): icmp_seq=4 ttl=64 time=0.125 ms

--- pc53.emulab.net ping statistics ---
4 packets transmitted, 4 received, 0% loss, time 3006ms
rtt min/avg/max/mdev = 0.125/0.179/0.321/0.083 ms
```

Abbildung 20: Ausgabe bei Ping (falsches Ansprechen)

Wie in Abbildung 19 ersichtlich, haben wir als delay für jede der beiden Leitungen 10ms eingestellt. Bei der Ausgabe (Abbildung 20) sieht man allerdings, dass wir ein delay von ca. 0,135 haben. Es handelt sich daher nicht um unser Netzwerk, sondern um die Vernetzung der Knoten bei Emulab.

Sprechen wir allerdings die Knoten mit ihrer IP-Adresse (Abbildung 21) an, dann erhalten wir das gewünschte Ergebnis (Abbildung 22):

```
Virtual Lan/Link Info:
ID      Member/Proto   IP/Mask          Delay  BW (Kbs)   Loss Rate
--------------- --------------- --------------- --------- --------- ---------
link0   A:0            10.1.1.2         5.00   100000     0.000
        ethernet       255.255.255.0    5.00   100000     0.000
link0   B:0            10.1.1.3         5.00   100000     0.000
        ethernet       255.255.255.0    5.00   100000     0.000
link1   B:1            10.1.2.2         5.00   100000     0.000
        ethernet       255.255.255.0    5.00   100000     0.000
link1   C:0            10.1.2.3         5.00   100000     0.000
        ethernet       255.255.255.0    5.00   100000     0.000
```

Abbildung 21: IP-Adresse der Knoten

```
[MarkusS@a ~]$ ping 10.1.2.3
PING 10.1.2.3 (10.1.2.3) from 10.1.1.2 : 56(84) bytes of data.
64 bytes from 10.1.2.3: icmp_seq=1 ttl=63 time=41.1 ms
64 bytes from 10.1.2.3: icmp_seq=2 ttl=63 time=40.2 ms
64 bytes from 10.1.2.3: icmp_seq=3 ttl=63 time=40.2 ms
64 bytes from 10.1.2.3: icmp_seq=4 ttl=63 time=40.2 ms
64 bytes from 10.1.2.3: icmp_seq=5 ttl=63 time=40.2 ms

--- 10.1.2.3 ping statistics ---
5 packets transmitted, 5 received, 0% loss, time 4037ms
rtt min/avg/max/mdev = 40.200/40.416/41.170/0.398 ms
```

Abbildung 22: Ergebnis mit richtigem Ansprechen

Wenn man den Knoten C mit seiner IP-Adresse (10.1.2.3) anspricht, erhält man ein korrektes Ergebnis. Wir haben damit gezeigt, dass die im Emulator eingestellten Netzwerk-Daten einwandfrei übernommen werden. Zum Mitprotokollieren des Experiments haben wir tcpdump verwendet. Einen Teil der Ausgabe zeigt Abbildung 23:

```
02:58:46.339350 arp who-has 155.101.135.245 tell control-router.emulab.net

02:58:46.348949 pc53.emulab.net.32775 > boss.emulab.net.domain:  60290+ PTR?
245.135.101.155.in-addr.arpa. (46) (DF)

02:58:46.349383 boss.emulab.net.domain > pc53.emulab.net.32775:  60290
NXDomain* 0/1/0 (114)

02:58:46.349733 pc53.emulab.net.32775 > boss.emulab.net.domain:  60291+ PTR?
1.132.101.155.in-addr.arpa. (44) (DF)

02:58:46.350054 boss.emulab.net.domain > pc53.emulab.net.32775:  60291* 1/2/2
(150)
```

Abbildung 23: tcpdump Protokoll

Hier ist ersichtlich wie der Ping zwischen den Knoten verschickt wird.

5.3. nse

Rechner A soll ein Ping auf Rechner B absetzen. RecherC soll den Traffic aufzeichnen, sodass dieser dann mit NAM betrachtet werden kann.

Im File emulate/README_TCP_EMULATION ist eine Anleitung zu finden, wie das Szenario durchgeführt werden kann (die beschriebene Datei „thrutcp.tcl“ sollte im Verzeichnis `/emulate/` zu finden sein):

Diese Datei beschreibt folgenden Aufbau (siehe Abbildung 24):

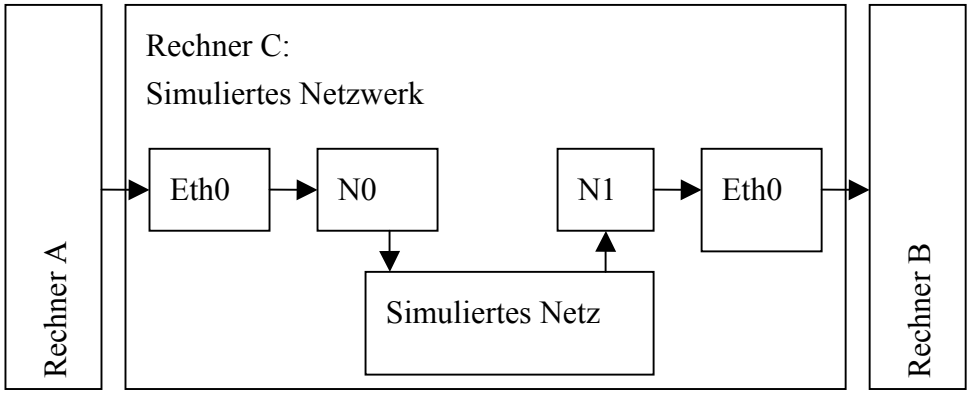

Abbildung 24: Aufbau des Vergleichsszenarios für thrutcp.tcl

Zum Ausführen dieser Datei mit nse werden 3 Rechner benötigt. Jedoch benötigt der Rechner, auf dem der Emulator läuft, nur eine Netzwerkkarte. Diese Karte kann als Ein- und Ausgang des simulierten Netzwerkes dienen. Wir haben die 3 Rechner mittels eines Routers zusammengeschlossen.
A möchte eine TCP-Verbindung mit B über C herstellen. Auf C läuft nse. Damit der Rechner A weiss, wo sich Rechner B befindet, muss dessen Routingtabelle verändert werden:

```
route add <IP-Adresse von B> <IP-Adresse von C>
```

Analog muss dieser Befehl auf Rechner B angewandt werden.

Im Detail:

- auf A und B:
    - `disable ip-redirects`
    - `disable ip-forwarding`

- auf A:
    - Hinzufügen der Route zu B über C:
        - `route add B_ip C_ip`
- auf B:
    - Hinzufügen der Route zu A via C:
        - `route add A_ip C_ip`
- auf C:
    - in thrutcp.tcl Zeilen bearbeiten:
        - Device richtig konfigurieren:
            - `Set dev [$bpf1 open readonly eth0]`
            - `Set dev [$bpf3 open readonly eth1]`
        - Filter richtig konfigurieren:
            - `$bpf1 filter „src A_ip B_ip“`
            - `$bpf3 filter „src B_ip A_ip“`
        - `run “nse thrutcp.tcl”`

- Auf A und B:
    - Eine TCP-Anwendung zw. A und B starten (z.B. ping oder FTP)

Die mit ns-allinone gelieferte Datei thrutcp.tcl wurde für freeBSD geschrieben und sollte dort auch funktionieren. Will man die Datei unter Linux ausführen, so muss man diese etwas umschreiben. Durch Zufall wurde nach etlichen gescheiterten Versuchen beim Suchen im Internet mithilfe der Suchmaschine google entdeckt, dass es unter Linux den voreingestellten Agent „Agent/IPTab“ nicht gibt. Problematisch ist, dass der Emulator einfach nicht funktioniert, wenn IPTab verwendet wird – es werden keine Fehlermeldungen ausgegeben, die darauf schließen lassen, dass IPTab nicht unterstützt wird. Um das Problem zu beheben, muss dieser Agent durch „Agent/Tab“ ersetzt werden. Danach sollte der Emulator die Datei unter Linux wie erwartet abarbeiten.

Zu Vergleichszwecken mit nun folgenden Ausgaben wurde ein ping mit ausgeschaltetem Emulator gemacht (siehe Abbildung 25). Dazu ist IP-forwarding auf dem Rechner, auf dem der Emulator läuft, einzuschalten (etwa sysctl –w net.ipv4.ip_forward=1).

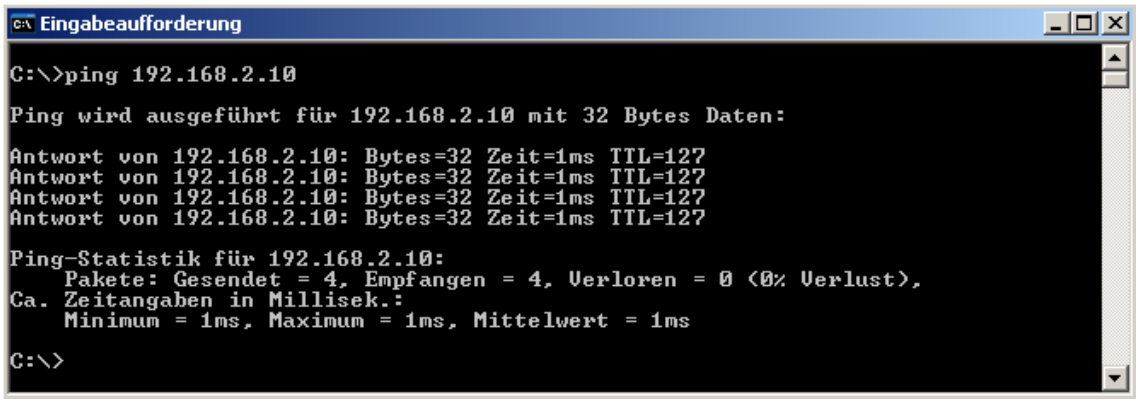

Abbildung 25: Ausgabe nach ping ohne Emulator

Beachte, dass für folgende Ausgaben IP-forwarding gleich 0 (ausgeschaltet) sein muss. Folgend wurden einige NS-Files getestet, welche dieses Szenario jeweils auf eine andere Art beschreiben. Die entsprechenden Quelldateien, die erzeugte Ausgabe auf dem Rechner mit dem Emulator und auf dem, von dem der Ping abgesetzt wurde, sind strukturiert auf der diesr Arbeit beiliegenden CD im Verzeichnis /ns/ping/ zu finden. Der Name des Unterverzeichnisses ist aussagekräftig und selbsterklärend. Ebenfalls ist das erzeugte out.nam in diesem Verzeichnis zu finden. Dieses NAM-file könnte mit dem Programm Network-Animator (NAM) betrachtet werden. (Die Windows-Version „nam-1.0a11a-win32.exe“ ist auf der CD unter ns/downloads zu finden).

In thrutcp.tcl sieht man, dass ein Netzwerk mit 5 Knoten simuliert wird. Ein Knoten ist zentral und alle anderen sind sternförmig dazu angeordnet. 2 dienen als Empfänger, die restlichen 2 als Sender. So muss, um ein delay von 10ms für eine Richtung zu erhalten, für jede Verbindung zwischen den Knoten ein delay von 5ms gewählt werden.
Wenn der Emulator gestartet wird und ihn dieses File abarbeiten lässt, so erhält man folgende Ausgabe (siehe Abbildung 26):

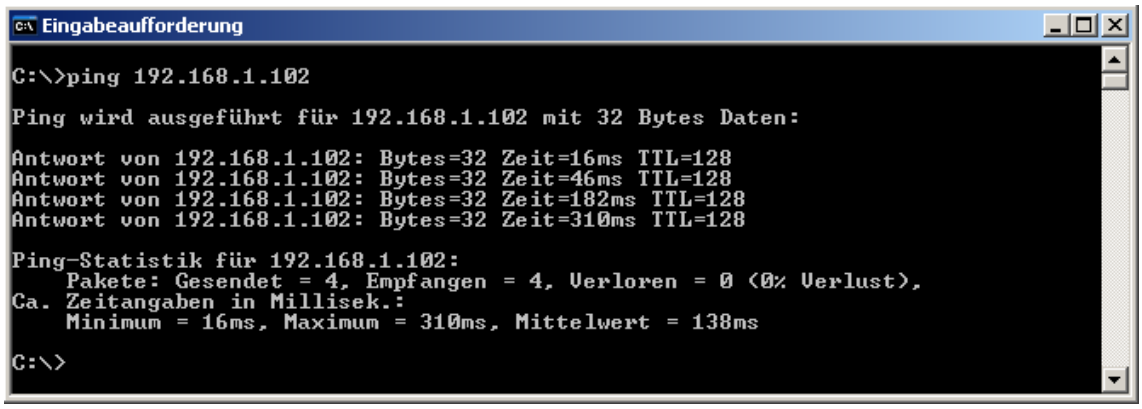

Abbildung 26: Ausgabe nach ping mit Router und 5 simulierten Knoten

Man sieht, dass der Ping zwar funktioniert, die delay-Zeiten schwanken jedoch bzw. immer größer werden. Das ns-Quell-File und das erzeugte „out.nam“ ist auf der CD unter `ns/ping/5nodes_router` zu finden. Seltsam ist, dass der Emulator immer noch Ausgaben auf die Shell produziert, auch wenn der Ping bereits abgeschlossen ist. Da diese Ausgabe ziemlich lang wurde, ist sie nicht auf der CD angeführt.

Um zu testen, ob die schwankenden Delay-Zeiten an der Art der Beschreibung liegen, haben wir dann versucht, ein Netzwerk zu beschreiben, welches nur 3 Knoten enthält. Als Delay wurden hier 5ms gewählt, um 10ms in eine Richtung zu erhalten. Die Ausgabe ist jedoch ähnlich (siehe Abbildung 27):

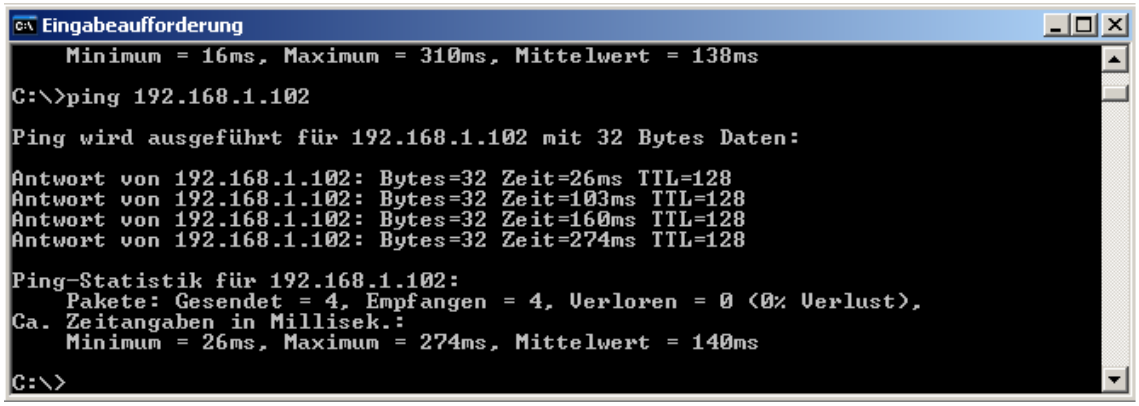

Abbildung 27: Ausgabe nach ping mit Router und 3 simulierten Knoten

Auch hier sieht man, dass der Ping zwar funktioniert, jedoch schwanken die delay-Zeiten ebenfalls.

Weiters haben wir getestet, was passiert, wenn man den Rechner, auf dem der Emulator läuft, mit 2 Netzwerkkarten versieht und mit den anderen beiden Rechnern direkt zusammenhängt. Zuerst wurde das File mit den 5 Knoten so abgeändert, dass die 2 installierten Netzwerkkarten verwendet werden (siehe Abbildung 28).

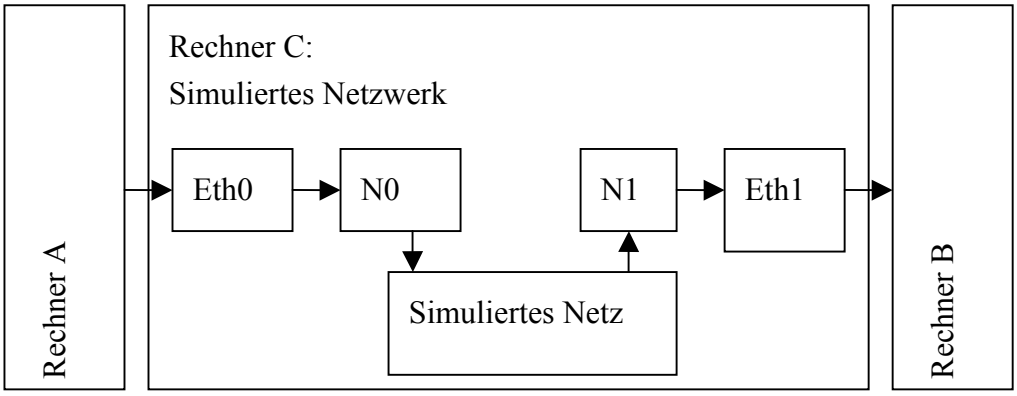

Abbildung 28: Aufbau des Vergleichsszenarios für das abgeänderte thrutcp.tcl

Dies führte zu folgender Ausgabe auf dem Rechner, auf dem der Ping abgesetzt wurde (siehe Abbildung 28):

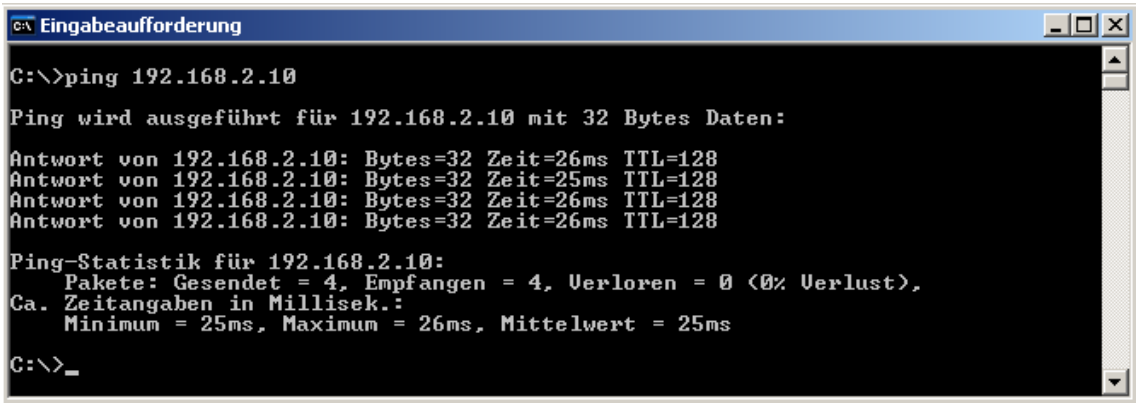

Abbildung 29: Ausgabe nach ping mit 2 Netzwerkkarten und 5 simulierten Knoten

Man sieht deutlich, dass die Delay-Zeiten jetzt konstant sind. Zu erwarten wäre, dass der Emulator ein delay von 20ms produziert. Dies ergibt sich aus 10ms in eine Richtung und 10ms in die andere Richtung für die Antwort auf das Ping. Mit 25-26ms liegen diese schon ziemlich nahe am erwarteten Wert. Fragwürdig ist jedoch, warum das produzierte NAM-File keinen Traffic mehr anzeigt. Was ebenfalls seltsam ist, ist, dass beim direkten Zusammenschließen der Emulator nur mehr etwas auf die Shell ausgibt, wenn ein Ping durchgeführt wird. Dieses Verhalten ist zwar wünschenswert, wurde aber

nicht erreicht, als die Rechner mittels Router zusammengeschlossen wurden (Ausgabe ist diesmal auch auf der CD zu finden).

Um den Test abzuschließen wurde noch ein NS-File getestet, welches 3 Knoten beschreibt und 2 Netzwerkkarten verwendet (siehe Abbildung 30).

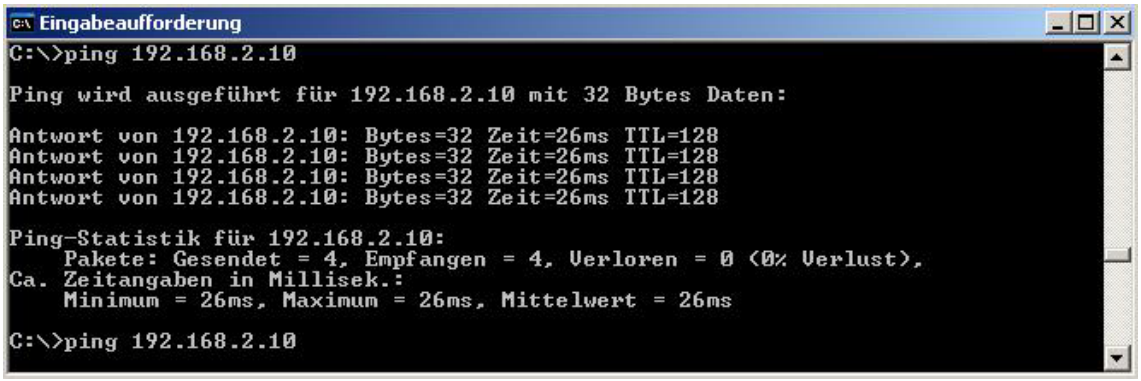

Abbildung 30: Ausgabe nach ping mit 2 Netzwerkkarten und 3 simulierten Knoten

Man sieht, dass hierbei die zu erwartende delay-Zeit von 20ms um denselben Betrag schwankt, wie bei den 5 Knoten, was jedoch akzeptabel ist. Im erzeugten NAM-file ist wiederum kein Traffic ersichtlich. Der Emulator produziert auch hier nur solange eine Ausgabe auf die Shell, solange ein Ping abgesetzt wird.

In der Dokumentation auf der Homepage von NS wird im Abschnitt „Emulate“ ein Beispiel-Skript angegeben, welches ebenfalls eins zu eins für dieses Szenario als Quelle dienen kann. Nach Anpassen der IP-Adressen und Beschreibung der Ethernetkarten lieferte nse nach Ausführen dieses Skripts folgende Ausgabe (siehe Abbildung 31):

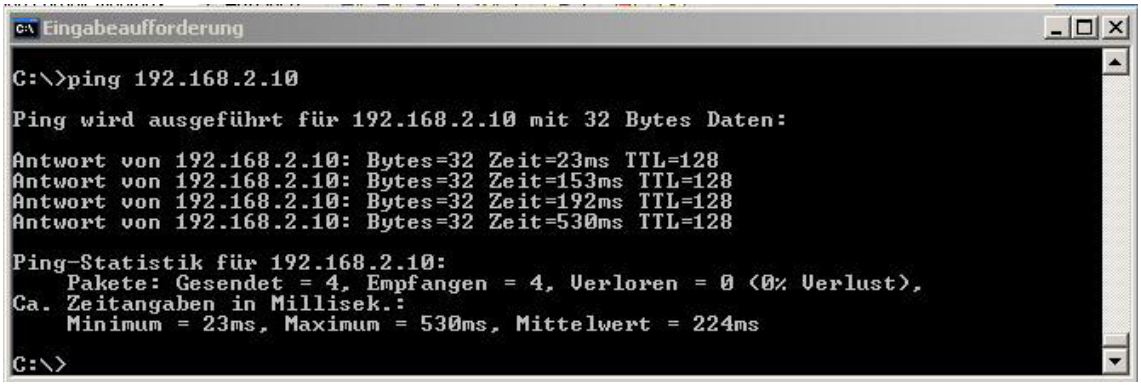

Abbildung 31: Ausgabe nach ping mit 2 Netzwerkkarten anhand Skript von Dokumentation

Auch hier ist zu sehen, dass das eingestellte delay von 10ms nicht zutrifft. Im produzierten „out.nam“ ist ebenfalls kein Traffic ersichtlich.

5.4. NistNet

Unter NistNet müssen die angehängten Rechner und somit auch IP-Adressen jeder einzelnen Netzwerkkarte einem anderen Subnet zugeordnet sein.

Dazu ist es eventuell erforderlich, dass statische IP-Adressen zugewiesen werden müssen. Unter Redhat 8.0 haben wir es nicht geschafft, mittels der angebotenen GUI den Netzwerkkarten eine statische IP-Adresse zuzuweisen. Die Einstellungen wurden zwar ohne Fehlermeldung akzeptiert, jedoch wurde mit dem Befehl `ifconfig` wurde ersichtlich, dass die Einstellungen nicht übernommen worden sind.

Wenn wir uns eine IP-Adresse per DHCP zuweisen ließen, so gibt xnistnet nach korrekt gemachten Einstellungen und „update“ folgende Meldung auf die Shell aus:

```
alarmgethostbyaddr( <ip, die in xnistnet eingegeben wurde> ): time out
```

Mit dem Befehl

```
ifconfig eth1 192.168.2.1 netmask 255.255.255.0
```

kann mittels der Shell der Netzwerkkarte `eth1` eine IP-Adresse und die Subnetmask zugewiesen werden. Dies funktionierte sofort und ohne Probleme. Auch wenn der Netzwerkkarte per DHCP dieselbe IP zugewiesen wird, welche manuell eingestellt werden kann, funktioniert dies in xnistnet und andernfalls nicht. Zu beachten ist jedoch, dass dieser Befehl vor dem Laden von NistNet in den Kernel angewandt wird.

Rechner A soll ein Ping auf Rechner B absetzen. Rechner C soll den Traffic beeinflussen (siehe Abbildung 32).

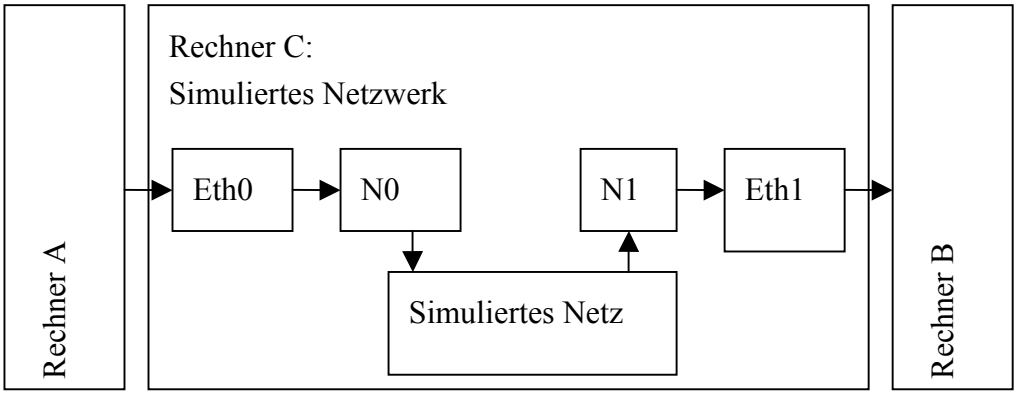

Abbildung 32: Aufbau des Szenarios für NistNet

Bei unserem Netzwerk lauteten die IP-Adressen der einzelnen Rechner folgendermaßen:

- Rechner A: 192.168.1.100
- Rechner C (eth0): 192.168.1.102
- Rechner C (eth1): 192.168.2.1
- Rechner B: 192.168.2.10

Damit ein Ping von Rechner A auf Rechner B funktionieren kann, muss zuerst Rechner A mitgeteilt werden, dass Rechner B hinter Rechner C zu finden ist bzw. analog für Rechner B – es müssen die Routing-Tabellen auf Rechner A und B geändert werden.
Dies geschieht mit den folgenden Befehlen:

- auf Rechner A (mit IP=192.168.1.100):
  `route add 192.168.2.10 192.168.1.1`
- auf Rechner B (mit IP=192.168.2.10):
  `route add 192.168.1.100 192.168.2.1`

Des Weiteren muss auf Rechner C IP-forwarding eingeschaltet werden, damit Pakete von der Netzwerkkarte eth0 zu Netzwerkkarte eth1 weitergeleitet werden:

```
sysctl -w net-inet.ip.forwarding=1
```

(Der genaue Befehl für ein Linux-System kann mit `sysctl -a | grep forward` herausfinden werden).

Nun kann NistNet auf Rechner C gestartet werden:

1. Das NistNet-Paket in den Kernel laden: `modprobe nistnet`
2. X-NistNet starten: `xnistnet`

In der angezeigten graphischen Oberfläche können nun die IP´s in xnistnet eingetragen werden:

Source	Destination
192.168.2.10	192.168.1.100
192.168.1.100	192.168.2.10

Um das Delay von 10ms für jede Verbindung einzustellen, muss dies im Feld „Delay“ eingetragen werden. Auch die Bandbreite von 100 muss noch angegeben werden.

Nach „UPDATE“ und einschalten des Emulators, ist NistNet aktiv.

Folgende Ansicht sollte nun zu sehen sein (siehe Abbildung 33):

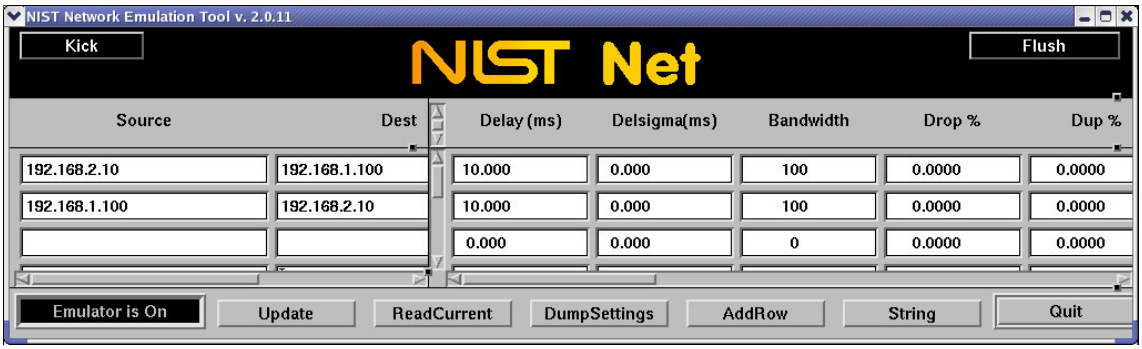

Abbildung 33: Eingaben in xnistent für das Vergleichsszenario

Wenn man nun auf Rechner A mit IP=192.168.1.100 ein

```
ping 192.168.2.10
```

macht, so erhält man folgende Ausgabe (siehe Abbildung 34):

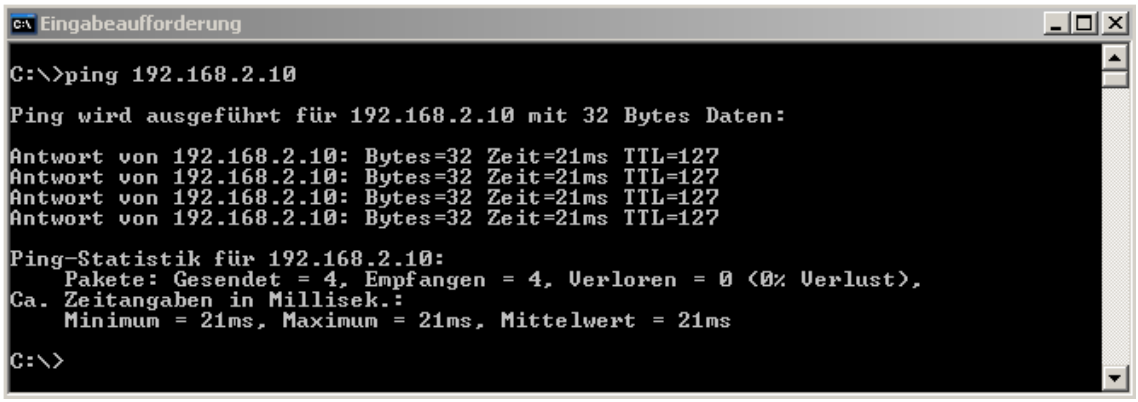

Abbildung 34: Ausgabe nach ping bei xnistnet

Zu Vergleichszwecken wurde nochmals ein ping mit ausgeschaltetem Emulator gemacht (siehe Abbildung 35):

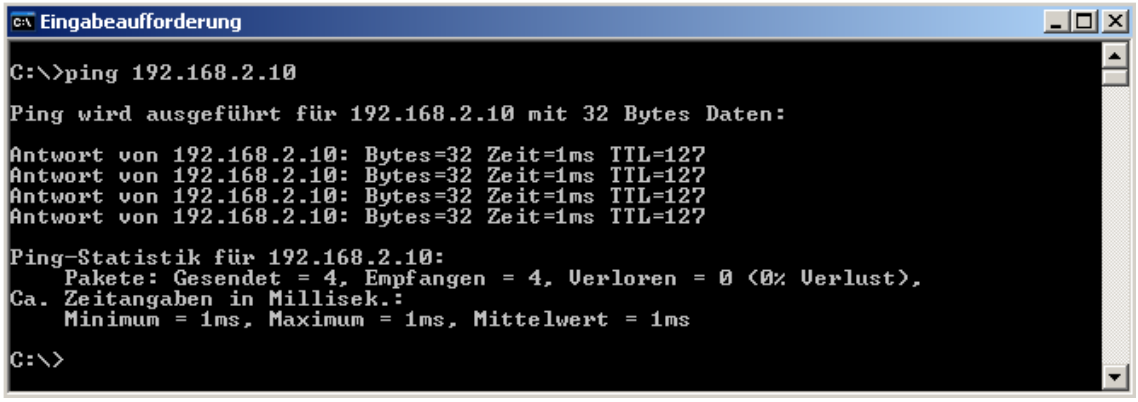

Abbildung 35: Ausgabe nach ping ohne Emulator

Man sieht deutlich, dass die Zeit sich gegenüber ausgeschaltetem Emulator auf 21ms erhöht, was der Einstellung delay=10.0ms für beide Richtungen entspricht und so den Datenverkehr wie erwartet beeinflusst. In xnistnet werden auch Informationen über die Pakete angegeben, die über den Emulator transportiert werden. Die Ausgabe nach dem Ping sieht folgendermaßen aus (siehe Abbildung 36):

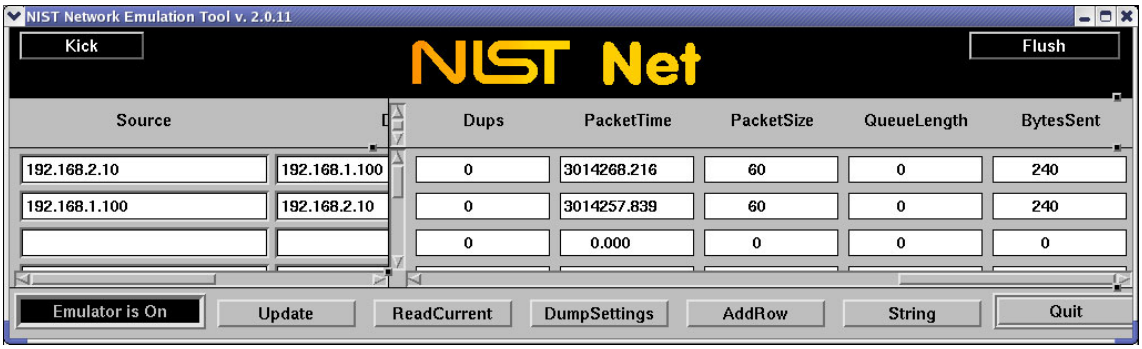

Abbildung 36: statistische Ausgabe in xnistnet nach durchgeführtem ping

Um den Datenverkehr mitzuprotokollieren, wird von NistNet das Tool „Mungebox“ mitgeliefert, zu dem wir allerdings keinerlei Dokumentation finden konnten.

Mungebox wird mit

```
insmod mungemod
```

geladen, und mit

```
mungebox -u -a <src-IP> <dest-IP> -S
```

gestartet.

Die Ausgabe sieht folgendermaßen aus:

```
[root@localhost NistNet]# mungebox -u -a 192.168.1.100 192.168.2.10 -S
addmunge 192.168.1.100 to 192.168.2.10
ip: 192.168.1.100->192.168.2.10 length 5 icmp type 2048 code 0 Fri May
7 11:52:38 2004
[root@localhost NistNet]#
```

Diese Zeile ändert sich, sobald ein neues ping-Paket abgesetzt wird, auf die aktuelle Zeit. Man kann die Pakete ebenfalls mit tcpdump mitprotokollieren, was zu folgender Ausgabe führt:

```
root@localhost NistNet]# sudo tcpdump -i eth0
tcpdump: listening on eth0
11:52:16.887464 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:16.887464 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:16.908065 192.168.2.10 > 192.168.1.100: icmp: echo reply
11:52:17.888239 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:17.888239 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:17.908777 192.168.2.10 > 192.168.1.100: icmp: echo reply
11:52:18.890570 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:18.890570 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:18.911086 192.168.2.10 > 192.168.1.100: icmp: echo reply
11:52:19.892968 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:19.892968 192.168.1.100 > 192.168.2.10: icmp: echo request
11:52:19.913511 192.168.2.10 > 192.168.1.100: icmp: echo reply

12 packets received by filter
0 packets dropped by kernel
[root@localhost NistNet]#
```

Um nach dem Verwenden von NistNet den Linux-Rechner wieder ohne Emulatoreigenschaft verwenden zu können, müssen diese Module aus dem Kernel entfernt werden:

```
rmmod mungemod
rmmod nistnet
```

Kapitel 6

Abschlussbetrachtung

6.1. Resümee

In diesem Abschnitt möchten wir dem Leser eine abschließende Bewertung der Emulatoren geben, die vor allem die Auswahl für den jeweiligen Anwendungsfall erleichtert.

6.1.1. Was spricht für bzw. gegen Emulab?

Der große Vorteil von Emulab ist sicherlich die einfache Anwendung. Man benötigt keine Installationsroutinen, sondern kann sofort nach Anmeldung mit den Experimenten starten. An dieser Stelle müssen wir auch einen Nachteil erwähnen. Es kann sein, dass die Anmeldung etwas länger dauert (bei uns z.B. 2 Monate). Laut Emulab sollte das zwar nicht die Regel sein, aber ausschließen kann man es auf alle Fälle nicht.

Weiters kann man Emulab ohne Kenntnis von Tcl bzw. OTcl zu besitzen nützen, indem man die grafische Oberfläche verwendet und sich damit das ns file automatisch erzeugen lässt. Das ist natürlich für alle „Quereinsteiger“ ein großer Vorteil. Ein weiterer Vorteil ist die Flexibilität von Emulab. Man kann zum Beispiel andere Betriebssysteme auf den Knoten laden lassen ohne diese selbst installieren zu müssen.
Nicht jeder kann Emulab nützen, da man einen Projektleiter benötigt, der zum Beispiel an einer Universität arbeitet. Als Student ist eine selbstständige Verwendung nur schwer möglich.

6.1.2. Was spricht für bzw. gegen nse?

Die Emulator-Funktionalität von NS hat den Nachteil, dass diese Funktionalität nur wenig dokumentiert ist und man bei Problemen oft nur schwer Rat findet. Auf der Website von NS gibt es nur wenige Informationen über die Emulator-Funktionalität. Auch im Internet mit Hilfe der Suchmaschine google konnte nur wenig Nützliches gefunden werden. Die Mailinglist, zu der man sich auf der Website von NS anmelden kann, scheint die beste Informationsquelle zu sein.

Im Vergleich zu NistNet hat ns-e jedoch den Vorteil, dass die Netzwerkkarten nur speziell beschriebene Knoten eines simulierten Netzwerkes darstellen. Bei NistNet wird ja die IP-forwarding-Eigenschaft beeinflusst, sodass die Kommunikation zwischen Rechnern, die an dem Rechner mit NistNet hängen, beeinflusst wird. Bei ns-e können zusätzlich simulierte Knoten hinzugefügt werden. So können viel komplexere Situationen getestet werden.

Nachteilhaft ist, dass es nicht einfach zu verstehen ist, wie die Knoten beschrieben werden müssen um sie mit den Netzwerkkarten zu verbinden. Es werden jedoch einige Bespiel-Skripte mit der Installation mitgeliefert, welche man nach seinen Bedürfnissen anpassen kann.

Ein weiterer Pluspunkt für nse ist die Möglichkeit, sich den Datenverkehr aufzeichnen zu lassen und dann mittels dem Programm „Network Animator“ (NAM) zu analysieren. Wenn man NS ohne Emulator zur Erzeugung solcher nam-files verwendet, sieht man darin deutlich, wie der Datenverkehr stattfindet. Uns ist es jedoch nicht gelungen, diesen Verkehr zu betrachten, wenn nse solche Files generiert. Wir wissen nicht, ob es an einem Bedienungsfehler liegt, oder ob es sich um einen Bug im Emulator handelt.

6.1.3. Was spricht für bzw. gegen NistNet?

NistNet eignet sich hervorragend um zu testen, was passiert, wenn man die Verbindung zwischen 2 Rechnern verändert. Verlustraten, delay und Bandbreite können variiert werden.

Vorteilhaft ist auch, dass der Emulator immer online ist – er braucht nicht neu gestartet werden um bestimmte Eigenschaften zu verändern. Bei eingeschaltetem Emulator genügt ein Verändern der Parameter und ein Klick auf „update“ um die gewünschten Einstellungen sofort wirksam zu machen. Allerdings hat dies bei Verkleinerung der Werte nicht funktioniert: Wenn die Bandbreite von 10 auf 100Mbit/s vergrößert wurde, so wurden die Einstellungen übernommen. Wurde jedoch die Bandbreite von 100 auf 10 reduziert, so wurde nach Klick auf „update“ wieder der alte Wert angezeigt. Vielleicht zeigt sich dieses (fehlerhafte) Verhalten nur in der getesteten NistNet-Version. Normalerweise sollten die Einstellungen nach „update“ sofort übernommen werden.

NistNet hat den Vorteil, dass es sehr leicht zu installieren ist. Mit dem herunterladbaren Quellcode werden betriebssystembedingte Installationsanleitungen mitgeliefert, denen man ohne Probleme folgen kann. Unsere getestete NistNet-Version hat als komprimierte Datei zum Downloaden 1,41MByte und findet somit gerade noch auf einer Diskette Platz. Es kann z.B. auch mit einem 56k-Modem komfortabel heruntergeladen werden. Zum Vergleich nse: 51MByte. Auch die Bedienung ist übersichtlich gestaltet. Die Mailinglist, in die man sich auf der Website von NistNet eintragen kann, gibt oft Rat, wenn dennoch Probleme auftauchen sollten.

6.1.4. Welchen Emulator soll ich nun verwenden?

Diese Frage können wir nicht allgemein beantworten, da man differenzieren muss, welche Ansprüche man an den Emulator stellt. Deshalb können wir hier nur zusammenfassen, was für welchen Emulator spricht. Die Entscheidung, welchen Emulator man verwendet, liegt beim User.

Um einen Überblick zu bekommen haben wir in Abbildung 37 wichtige Entscheidungsvariablen, die bei der Auswahl des Emulators helfen sollen, aufgeführt.

	Tcl Kenntnisse erforderlich	eigenes Netzwerk erforderlich	Installation erforderlich	Anmeldung erforderlich	Anzahl der Netzwerkkarten für Ping	Möglichkeit von virtuelen Knoten	grafische Visualisierung der Ergebnisse	interne Netzwerksimulation möglich	Mailingliste zur Hilfe vorhanden	Umfangreiche Dokumentation
emulab				x	1	x		x	x	x
ns-e	x	x	x		2	x	x	x	x	x
NistNet	x	x	x		3				x	

Abbildung 37: Zusammenfassung wichtiger Entscheidungsvariablen

Daraus können wir schließen, dass emulab zB. besonders für „Einsteiger“ geeignet ist. Im Prinzip benötigt man keine Tcl Kenntnisse, wenn man die GUI von emulab verwendet, und man benötigt auch kein eigenes Netzwerk, sondern nur einen Computer, der mit dem Internet verbunden ist und einen ssh-Client. Außerdem kann es alle benötigten Funktionen vorweisen und bietet vor allem auch eine umfangreiche Dokumentation. Einziger Wehrmutstropfen ist eigentlich die notwendige Anmeldung und das Verlangen nach einem Mitglied einer Forschungsgruppe oder Universität als Projektleiter und das Fehlen einer grafischen Auswertung der Ergebnisse. Dafür verwendet man normalerweise tcpdump, welches aber „nur“ den Verkehr mitprotokolliert und keine grafische Aufbereitung unterstützt.

NistNet ist in der Bedienung und Verwendung relativ einfach, und man kann eine Vielzahl an Experimenten damit durchführen. Wie schon bei emulab gibt es hier keine grafische Auswertung der Ergebnisse, sondern nur die Ausgabe in der Kommandozeile und die Ausgabe von tcpdump. Nisnet eignet sich für den normalen Anwender sehr gut, da man nur eine sehr kurze Einlernphase benötigt und dann vor allem die Anwendung relativ einfach ist. Ein Problem bei NistNet ist die Dokumentation. Die Installation und teilweise die Verwendung von NistNet wird zwar erläutert, jedoch sucht man vergebens nach Beispielen oder einem Versuchsaufbau.

Ns-e hat einen sehr großen Vorteil vorzuweisen. Und zwar kann man bei ns-e mit Hilfe des Programms NAM seine Ergebnisse graphisch visualisieren. Diese Möglichkeit hat man bei den anderen Emulatoren, außer man installiert zusätzliche Visualisierungstools, nicht. Ein Problem bei ns-e ist die mangelhafte Dokumentation, die seit Jahren nicht mehr geändert wurde und ein gutes Wissen über Tcl voraussetzt. Das erschwert vor allem die Anwendung.

6.2. Probleme

6.2.1. Emulab

Zuerst hatten wir, oder besser gesagt unser Betreuer, das Problem, dass die Anmeldung des Projekts sehr lange dauerte. Wir starteten die Anmeldung Mitte Dezember und konnten aber erst Mitte Februar unser Projekt auch nutzen. Laut Emulab wurde die Anmeldung übersehen und dauerte deshalb so lange. Es bleibt also zu hoffen, dass das nicht der Standardfall war und sie normal etwas schneller die Projekte freischalten. Nachdem wir von unserem Betreuer die Zugriffsrechte bekamen, konnten wir mit den Tests beginnen.

Ein weiteres Problem bei Emulab war die Auswertung. Die Dokumentation führt zu leichten Konfusionen, da die `*run` files, die von Emulab automatisch erzeugt werden, besonders hervorgehoben werden, und der Eindruck entsteht, dass in diesen Files alle Informationen über das Experiment mitprotokolliert werden. Das stimmt jedoch nur begrenzt, da nur das Starten der Knoten und Informationen, die mit dem eigentlichen Experiment nichts zu tun haben, in diesen Dateien enthalten sind. Über den Verkehr erhält man dabei keinerlei Information.
Um ein Experiment auswerten zu können, stellt Emulab nur Portstats zur Verfügung, welches die Statistik an den ports anzeigt (Beschreibung auf Seite 35). Die Ergebnisse von portstats sind auch nur bedingt brauchbar, und so muss man auf die standardmäßigen Linux-Tools (tcpdump) umsteigen. Die Verwendung von tcpdump funktionierte dann ohne Probleme, und wir konnten die Arbeit an Emulab damit abschließen.

6.2.2. nse

Beim Netzwerkemulator von NS ist das größte Problem, dass es kaum Informationen gibt, wie man den Emulator bedient. Auch auf der Website von NS ist nur eine veraltete Dokumentation zu finden.

Größte Probleme hatten wir zu Beginn, als wir versuchten den Emulator aufzutreiben und herauszufinden, wo man diesen installieren muss. In der Dokumentation auf der Website von NS steht, dass der Emulator unter freeBSD entwickelt und auch nur dort getestet wurde. So versuchten wir, freeBSD aufzutreiben. Die Dokumentation scheint jedoch sehr alt zu sein, da freeBSD nur mehr ab Version 4.8 verfügbar war. In der Hoffnung, dass der Emulator auch dort läuft, versuchten wir dieses Betriebssystem zu installieren; unsere Versuche scheiterten jedoch.

Zum Schluss stellte sich durch Zufall heraus, dass der Emulator von NS auch unter sämtlichen Linux-Derivaten läuft. Die Funktionalität von NS auch als Emulator eingesetzt werden zu können, wird mit allen aktuelleren NS-allinone-Paketen mitgeliefert. Herausgefunden kann es werden, indem man nach einem Verzeichnis „emulate“ sucht. Dieses sollte im (nicht installierten) Paket im Verzeichnis ns-allinone-2.26/ns-2.26/ für die NS-allinone-Version mit Nummer 2.26 zu finden sein. Wird dieses Verzeichnis gefunden, so wird auch die Emulator-Funktionalität angeboten.

Während der Installation stößt man auf unerwartete Probleme. Des Öfteren kommen Fehlermeldungen, bei denen man anfangs nicht weiß, wie man sie beheben kann. Z.B. benötigt der Emulator von NS zusätzliche Software, die sich libpcap nennt. Diese Software wird jedoch von NS nicht mitgeliefert. Nach der Installation von libpcap ist man schon ein paar Schritte weiter. Beim Linken der kompilierten Dateien ergibt sich jedoch eine weitere Fehlermeldung. Die kompilierten Dateien (*.o) müssen noch ins Makefile eingetragen werden, um diese in die Kompilation von NS mitaufzunehmen.

Problematisch ist, dass es kaum Hilfe gibt, wenn solche Fehler auftreten. Die Mailinglist, in die man sich auf der Website von NS eintragen kann, scheint jedoch hilfreich zu sein.

Nachdem der Emulator endlich installiert ist, stellt sich das Arbeiten damit als sehr mühsam heraus. Die mit dem Paket mitgelieferten Beispielskripte (im Verzeichnis `/emulate` zu finden) funktionieren nicht auf Anhieb. Diese Skripte wurden für freeBSD geschrieben. Für Linux gibt es Abweichungen. In keiner Dokumentation ist zu finden, dass der verwendete `Agent/IPTap` in Linux nicht funktioniert. Dieser muss dort durch `Agent/Tap` ersetzt werden. Dies wäre nicht so schlimm, wenn es irgendwo dokumentiert wäre.

NS hat theoretisch den Vorteil, dass man Ausgaben produzieren lassen kann, welche dann graphisch betrachtet werden können. Der Datenverkehr kann analysiert werden - doch leider ist es uns nicht gelungen, brauchbare Details zu sehen. Oft sieht man nur die Knoten und keinerlei Verkehr, obwohl das Skript korrekt funktionierte.

6.2.3. NistNet

Zu NistNet ist vor allem zu sagen, dass es nicht auf allen Linux Rechnern funktioniert. Man benötigt eine bestimmt Kernelversion um nisnet problemlos verwenden zu können.[18] Probleme mit NistNet gibt es vor allem mit der Linux-Light Version „Knoppix“. Wir möchten hier nun kurz und allgemein die Probleme zusammenfassen, die man bei der Verwendung von Knoppix hat.

Als erstes mussten wir eine Kernelversion installieren, die von NistNet unterstützt wird. Nach der Installation dieser Version wollten wir NistNet starten, aber es kamen immer Fehlermeldungen und NistNet ließ sich nicht starten. Der Grund für die Probleme war eine Einstellung im Makefile von NistNet. Auf dem Computer, auf dem wir NistNet anfangs testeten, lief Knoppix (Festplatteninstallation). Knoppix unterstützte aber einige Funktionen nicht, welche im Makefile gefordert wurden. So mussten wir zuerst das Makefile etwas abändern. Doch auch nach erfolgreicher Installation lief NistNet nicht einwandfrei, was an den nicht installierten Funktionen lag. Wir können abschließend nur von einer Verwendung von Knoppix im Zusammenhang mit NistNet abraten.

Bei der Verwendung von NistNet ist uns aufgefallen, dass die IP-Adressen unbedingt mit dem Linux-Befehl `ifconfig` zugewiesen werden müssen. Wir konnten NistNet

[18] Siehe: http://snad.ncsl.nist.gov/itg/nistnet/

nicht verwenden, wenn die IP-Adresse per DHCP-Server zugewiesen wurde. Ob dies ein Problem von NistNet oder vom verwendeten Linux-Betriebssystem ist, wissen wir nicht. Sollten dieses Problem auftreten, ist zu beachten, dass nach dem Anwenden des Befehls `ifconfig` das NistNet-Kernel-Modul erneut in den Kernel geladen werden muss, damit NistNet dann funktioniert. Genaueres zum Beheben dieses Problems ist im Punkt „Vergleich der Emulatoren“ – „NistNet“ zu finden.

Da es zu Problemen führt, wenn man nicht weiß, wie NistNet arbeitet, möchten wir hiermit nochmals darauf hinweisen: NistNet nimmt Einfluss auf das IP-forwarding von Linux. IP-forwarding wird benötigt, wenn Pakete von einer Netzwerkkarte zu einer anderen weitergeleitet werden, wenn das Paket nicht den Rechner mit eingeschaltetem IP-forwarding als Ziel hat. Somit ist es erforderlich, dass mehrere Netzwerkkarten installiert werden. NistNet kommt damit nicht zurecht, wenn die Netzwerkkarten im selben Subnetz liegen. Um NistNet verwenden zu können, muss daher jede Netzwerkkarte einem anderen Subnetz zugeordnet sein.

Literaturverzeichnis

1. Cygwin-Homepage
   http://www.sims.berkeley.edu/~christin/ns-cygwin.shtml
2. Download von Otcl
   ftp://ftp.tns.lcs.mit.edu/pub/otcl/
3. Emlab-Homepage
   http://www.emulab.net/
4. Emulab-Dokumentation
   http://www.emulab.net/tutorial/docwrapper.php3?docname=nscommands.html
5. Emulab-Downloads
   http://www.emulab.net/downloads/
6. Emulab-FAQ
   http://www.emulab.net/docwrapper.php3?docname=faq.html
7. FreeBSD-ftp
   ftp://ftp.FreeBSD.org/pub/FreeBSD/
8. FreeBSD-Homepage
   http://www.freeBSD.org
9. FreeBSD-Installationsanleitung
   http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/install-post.html
10. Google-Homepage
    http://www.google.at
11. Libcap-Download
    http://tcpdump.org/

12. NistNet Mailingliste
nistnet-admin@antd.nist.gov

13. NistNet-Download
http://snad.ncsl.nist.gov/itg/nistnet/requestform.html

14. NistNet-Homepage
http://snad.ncsl.nist.gov/itg/nistnet/

15. NistNet-Usage
http://snad.ncsl.nist.gov/itg/nistnet/usage.html

16. Ns-2-Homepage
http://www.isi.edu/nsnam/ns/

17. Ns-Dokumentation
http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

18. Ns-Download
www.isi.nsnam/dist/ns-allinone-2.26.tar.gz

19. Ns-Installationsanleitung
http://www.isi.edu/nsnam/ns/ns-build.html

20. Ns-Simluator/Tcl Dokumentation
http://www.welzl.at/research/tools/ns/index.html

21. OTcl-Dokumentation
http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/

22. Tb_compat.tcl-Download
http://www.emulab.net/tutorial /tb_compat.tcl

23. Tcpdump-Dokumentation
http://t-marin.thunderhawk.de/modules.php?name=News&file=article&sid=18

24. Tcpdump-Dokumentation
http://www-iepm.slac.stanford.edu/monitoring/passive/tcpdump.html

25. Technik-Lexikon
http://www.net-lexikon.de/Emulator.html

26. Tiptunnel-Installationsanleitung
http://www.emulab.net/docwrapper.php3?docname=faq.html#UTT-TUNNEL

27. VU-System

http://www.tns.lcs.mit.edu/vs/vusystem.html