

Net Simulator 2

Routing - Seminar

Andreas Mosig

17. Dezember 2004

Wintersemester 2004/05



Institut für Informatik
Universität Koblenz-Landau
Universitätsstraße 1, 56070 Koblenz
panic@uni-koblenz.de
<http://www.uni-koblenz.de/~panic>

Contents

1	Einleitung	3
1.1	Historie	3
1.2	Eigenschaften	3
2	Voraussetzungen	4
2.1	Windows	4
2.2	Unix	4
2.3	Komponenten	4
2.3.1	Tcl/Tk	5
2.3.2	OTcl	5
2.3.3	TclCL	5
2.3.4	ns-2	5
2.3.5	nam-1	5
2.3.6	xgraph	5
2.3.7	Perl	5
3	Funktionalität	6
3.1	Anwendungsschicht	6
3.2	Transportprotokolle	7
3.3	Routing	7
3.4	Router Mechanismen	7
3.5	Sicherungsschicht Mechanismen	8
4	Anwendung	8
4.1	Template	9
4.2	Nodes	9
4.3	Links	10
4.4	Agents	10
4.5	Traffic Generator	10
4.6	Connect Agents	10
4.7	Events	10
5	Zusammenfassung	11
	Bibliography	11

1 Einleitung

Der *Network Simulator - ns2* [Non] ist ein diskret ereignisgesteuerter Simulator für Netzwerktopologien. Er ist frei verfügbar und deshalb in der Forschung weit verbreitet [Rep03]. Ns unterstützt die Simulation von TCP, Routing und Multicast-Protokollen über verdrahtete und kabellose Verbindungen, wie WLAN oder Satellit.

1.1 Historie

Die Entwicklung von ns begann 1989 an der UC Berkeley als Variante des *REAL network simulator* und sollte ursprünglich zur Simulation von dynamischen Aspekten in paketorientierten Netzen wie Lastanalysen und Staukontrolle dienen. In den folgenden Jahren wurde ns ständig weiterentwickelt, bis im Jahr 1995 die erste Version ns-1 erschien. Durch die Unterstützung von der *DARPA* und einigen Instituten wie dem *Lawrence Berkeley National Laboratory (LBNL)*, *Xerox Palo Alto Research Center (PARC)*, *Information Science Institut at University of Southern California (USC/ISI)* und der *University of California, Berkeley (UCB)* entstand unter dem *VINT project ns-1*, mit dem man die Skalierbarkeit und Interaktion zwischen den Protokollen untersuchen konnte. Kurz darauf folgte 1996/97 die zweite Version ns-2, in der die gesamte Objektstruktur überarbeitet und weitere Simulationsmöglichkeiten wie Real Time Protocol (RTP), Scheduling-Algorithmen und mobile Hosts eingebunden wurden.

1.2 Eigenschaften

Ns wird ausschliesslich über Scriptdateien bedient und gestartet. Durch seine objektorientierte Struktur mittels C++ und OTcl ist es jedoch leicht, ein solches Script zu erstellen, da die Syntax einer modernen Programmiersprache ähnelt und leicht zu erlernen ist. Die eigentliche Simulation findet als interne Berechnung statt, deren Ergebnis eine Tracedatei ist, die man durch das Programm *Network Animator - nam* einlesen kann. Somit ist es möglich, die Simulation durch eine grafische Animation zu veranschaulichen.

Durch die eingebaute Funktion des Realtime-Schedulings ist ns auch fähig, als Emulator zu arbeiten. Damit ist es möglich, eine virtuelle Maschine nachzuahmen, in der echter Datenverkehr zu Echtzeitbedingungen durch eine Netzwerktopologie fließt. Diese Funktion ist jedoch zur Zeit noch in Entwicklung und nur bedingt einsatzfähig [KS00].

Der Network Simulator ist in C++ geschrieben und kann aufgrund seiner offenen Quelltexte und den vielen Dokumentationen [FV] und Tutorien [Gre] durch eigene Zusatzmodule erweitert oder einzelne Komponenten je nach Bedarf abgeändert werden.

In dieser Ausarbeitung wird zusätzlich noch auf folgende Punkte eingegangen:

1. Voraussetzungen und Installationsanweisungen der Applikation und ihrer Komponenten in Kapitel 2.
2. Beschreibung der unterstützten Protokolle in Kapitel 3.
3. Anleitung zur Anwendung der Applikation und Beschreibung der Kommandos in Kapitel 4.

2 Voraussetzungen

Die einzigen Voraussetzungen für ns sind lediglich ein Computer und ein C++ Compiler. Ns ist nahezu auf allen POSIX-fähigen Betriebssystemen lauffähig. Schwierigkeiten beim Kompilieren sind jedoch bei exotischer Hardware und nicht-standard Betriebssystemen nicht ausgeschlossen.

2.1 Windows

Unter Windows klappt die Installation am besten unter einer Unixemulation wie Cygwin, mit der sich ns dann neu kompilieren lässt. Ansonsten sind bei der reinen Windowsinstallation einige Tricks zu beachten, um einen funktionsfähigen Simulator zu erhalten. Notfalls weicht man auf vorkompilierte Programmbinaries aus, die jedoch nicht auf jeder Windowsversion lauffähig sind.

2.2 Unix

Da ns ursprünglich auf FreeBSD entwickelt wurde, lässt es sich auf allen anderen Unixderivaten wie Linux, Solaris und SunOS recht einfach installieren. Wie unter Linux üblich werden die Programme mit `./configure && make && make install` kompiliert und installiert. Da nicht jedes Linuxsystem gleich ist, können kleinere Schwierigkeiten in Form von nicht gefundenen Bibliotheken und ähnlichem auftreten. Diese sind jedoch durch das Setzen von symbolische Links auf die betroffenen Stellen und anderen kleinen Eingriffen leicht zu beheben.

2.3 Komponenten

Ns besteht nicht aus einem einzigen Programm, sondern setzt sich aus vielen kleineren Komponenten zusammen. Die meisten sind in jeder Linuxdistribution zu finden, einige muss man aber zusätzlich herunterladen und einzeln kompilieren. Wenn man nicht weiss, welche Komponente auch tatsächlich auf dem System installiert ist, besteht auch die Möglichkeit ein all-in-one-package herunterzuladen. Dieses schlägt jedoch mit rund 250 Megabyte zu Buche.

Der Zusammenhang zwischen ns-2 und den einzelnen Hilfsprogrammen illustriert Abbildung 1.

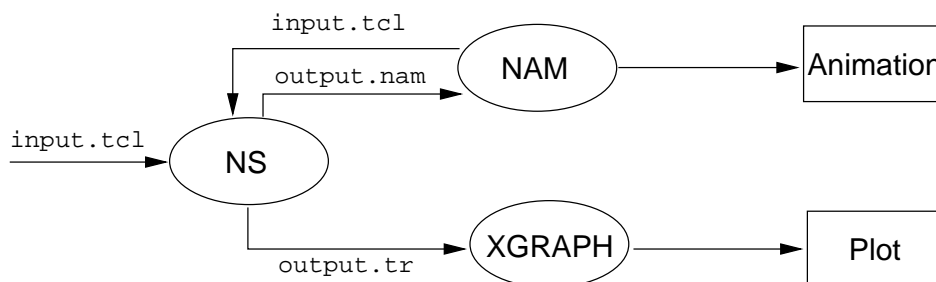


Figure 1: Datenfluss zwischen ns-2 und den Komponenten

Die einzelnen Komponenten sind:

2.3.1 Tcl/Tk

Die Tool Command Language mit dem zugehörigen Toolkit für grafische Benutzeroberflächen ist eine interpretierte Scriptsprache und das Fundament für ns. In dieser Sprache werden die Steuerscripte für den Simulator geschrieben. Tcl/Tk ist üblicherweise auf jedem Linuxsystem installiert.

2.3.2 OTcl

Mit Object Tcl wird die Tcl-Sprache um objekt-orientierte Fähigkeiten erweitert. Der Benutzer kann so mehrere Instanzen von Objekten erzeugen. Jede Funktion von ns kann als ein solches Objekt instanziiert und verwendet werden.

2.3.3 TclCL

Tcl with Classes stellt die Verbindung zwischen den in der Scriptdatei aufgerufenen und den in C++ realisierten Objekten her. Es ist damit der *glue* zwischen OTcl und C++.

2.3.4 ns-2

Dies ist das Herzstück des Simulators. Mit Hilfe der Steuerungscripte berechnet ns-2 die Simulation und schreibt die Ergebnisse in eine Tracedatei. Dieses Programm läuft ohne grafischen Aufsatz.

2.3.5 nam-1

Der Network Animator ist ein in Tcl/Tk geschriebenes Programm, das die von ns-2 erzeugten Tracedateien einliest und als grafische Repräsentation dem Benutzer vorführt (siehe Abbildung 2). Die Simulation lässt sich so animieren und steuern. Zusätzlich ist es mit einem eingebauten Editor (siehe Abbildung 3) möglich, eine Netzwerktopologie zu erstellen und diese in eine Scriptdatei zu speichern, die man ns-2 wieder übergeben kann.

2.3.6 xgraph

Ein einfacher Plotter zur Darstellung der Netzwerkauslastung als Graph (siehe Abbildung 4). Xgraph wird zusammen mit einer Tracedatei aufgerufen, die ns-2 erzeugt.

2.3.7 Perl

Perl ist eine mächtige interpretierte Scriptsprache, die auf jedem Linuxsystem installiert ist. Sie dient in Zusammenhang mit ns nur zu Testzwecken, kann aber auch zur automatischen Erzeugung von Scriptdateien für ns-2 verwendet werden.

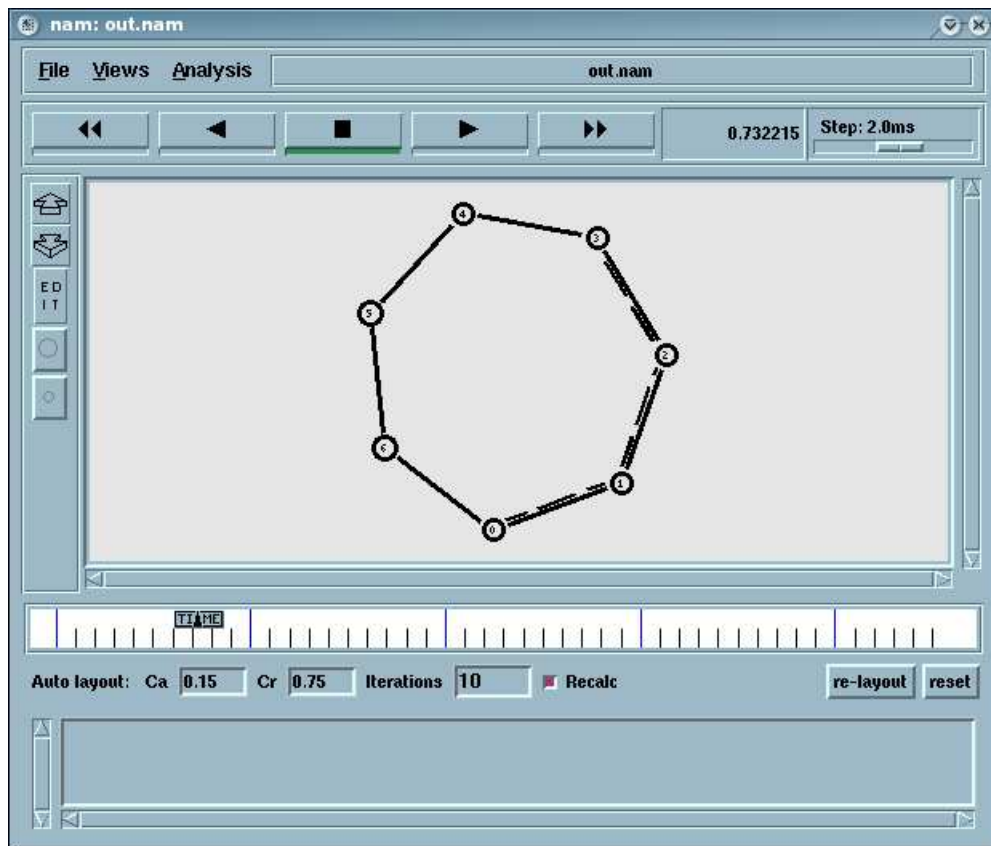


Figure 2: Der Network Animator mit einer laufenden Simulation

3 Funktionalität

Man kann pauschal sagen, dass ns fast alles simuliert. Schon das Basisprogramm selbst bietet eine Fülle an Funktionen und unterstützten Protokollen, die eine breite Anwendung in der Netzwerksimulation möglich macht. Wenn eine Funktion nicht vorhanden ist, dann lohnt sich eine Suche im Internet, wo viele Erweiterungen auf der Basis von ns zu finden sind [Tho04]. Mit diesen lässt sich ns dann einfach ausbauen. Auch ist es möglich, eigene Protokolle und Anwendungen zu implementieren und damit den Simulator höchst flexibel einzusetzen.

3.1 Anwendungsschicht

Für die Anwendungsschicht des OSI-Layer Modells ist das *Hyper Text Transfer Protocol (HTTP)*, *File Transfer Protocol (FTP)*, sowie diverser anderer TCP Applikationen, wie *Telnet* und Datenströme mit *Constant Bit Rate (CBR)* implementiert.

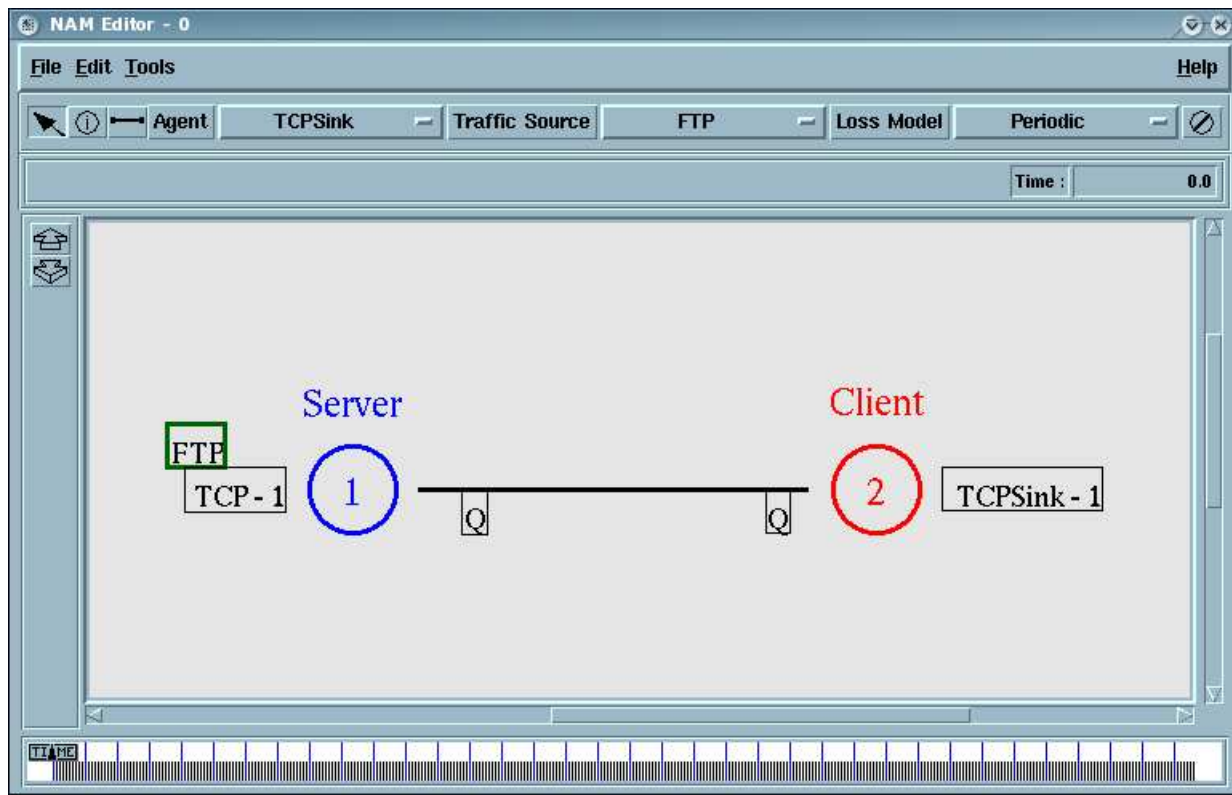


Figure 3: Der eingebaute Editor in nam zur Erstellung von Topologien

3.2 Transportprotokolle

Die unterst'utzten Transportprotokolle sind *UDP* und *TCP* mit allen Erweiterungen wie *Tahoe*, *Reno* und *TCP Vegas*. Auch das *Real Time Protocol (RTP)* wird unterst'utzt.

3.3 Routing

Als Routingtechniken sind das statische und das dynamische Routing zu nennen. Das dynamische Routing unterst'utzt dabei *distant vector* und *link state*. Damit ist das *Routing Information Protocol (RIP)* und *Open Shortest Path First (OSPF)* technisch m'oglich.

3.4 Router Mechanismen

Als Mechanismen zum Routing kommen alle Warteschlangenalgorithmen zur Geltung: Von der einfachen *FIFO/DropTail*, die alle Pakete verwirft, die nicht mehr in den Puffer passen, bis hin zu den fairen Routingmechanismen, wie *Fair Queueing (FQ)*, *Stochastic Fair Queueing (SFQ)*, *Deficit Round Robin (DRR)*, *Class Based Queueing (CBQ)* und *Random Early Discard (RED)*.

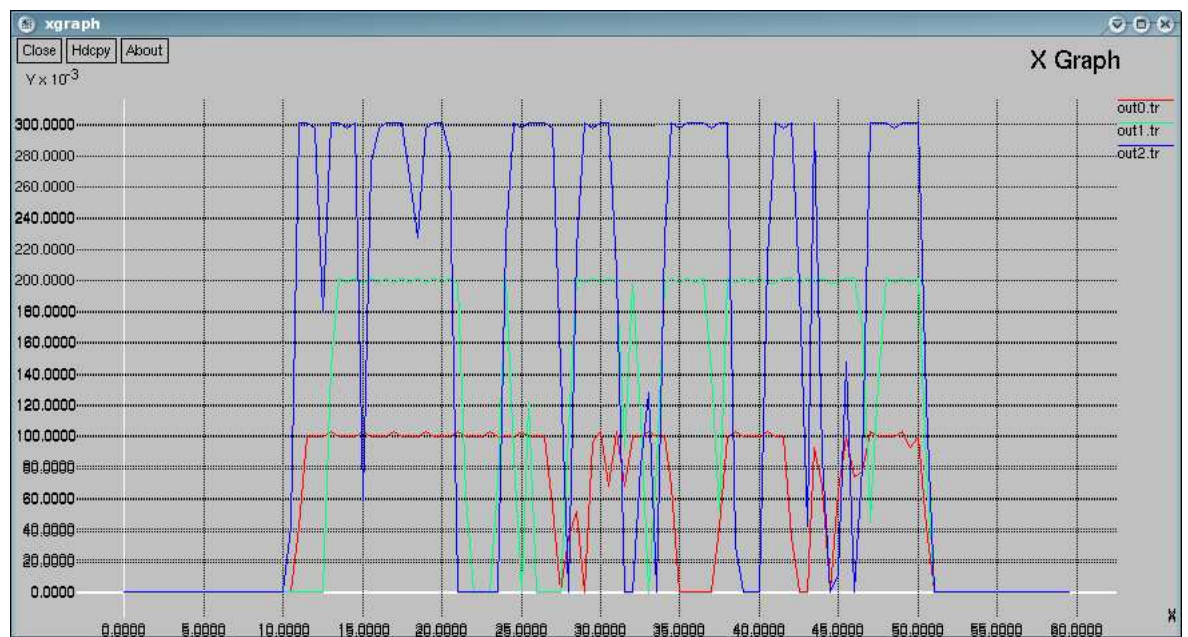


Figure 4: Darstellung des Netzverkehrs als Graph

3.5 Sicherungsschicht Mechanismen

Mit MAC Protokollen wie *Carrier Sense Multiple Access with Collision Detection (CSMA/CD)* kann auch ein Ethernet simuliert werden. Verwendet man statt dessen *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)*, simuliert man ein Funknetzwerk.

4 Anwendung

Die Scriptdatei stellt die Schnittstelle zur Anwendung von ns dar. In ihr ist die Netzwerktopologie, die verwendeten Protokolle und Routingmechanismen deklariert. Zusätzlich beschreibt sie die Ereignisse während der Simulation. Das kann z.B. der Ausfall einer Verbindung zwischen zwei Routern zu einem bestimmten Zeitpunkt sein. Das Script selbst wird in OTcl geschrieben und ns beim Programmaufruf mit `übergeben`.

Bei der Erstellung wird immer der selbe Ablauf beibehalten:

1. Zuerst werden alle *nodes* plaziert.
2. Je zwei Knoten werden mit *links* verbunden.
3. Jeder Knoten wird mit einem *agent* versehen.
4. Agenten wird *traffic* hinzugefügt.
5. Sender- und Empfängeragenten werden miteinander verbunden.
6. An bestimmten Zeitpunkten werden *events* gesetzt.

4.1 Template

Das Script hat immer den selben Aufbau. Dieses Template kann wieder verwendet und erweitert werden.

```
# erzeuge eine neue Instanz des Simulators
set ns [new Simulator]

# legt den Routingmechanismus fest
# default -> static routing
# DV      -> distant vector
# LS      -> link state
$ns rtproto DV

# öffnet das nam tracefile für schreibenden Zugriff
set nf [open output.nam w]
$ns namtrace-all $nf

# hier stehen die eigenen Kommandos zur Beschreibung
# der Topologie und der Agenten

# definiert die finish-Funktion, die nach Beenden
# der Simulation nam mit dem tracefile aufruft
proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the trace file
    close $nf
    #Execute nam on the trace file
    exec nam output.nam &
    exit 0
}

# ruft nach 5 Sekunden Simulationszeit die finish-Funktion auf
$ns at 5.0 "finish"

# startet die Simulation
$ns run
```

4.2 Nodes

Die Knoten repräsentieren einzelne Computer. Das können Server, Clients oder auch Router sein, die den Datenverkehr weiterleiten. Jedem Knoten kann auch ein Name, eine Farbe und eine Position in der Topologie zugeordnet werden.

```
set n1 [$ns node]
set n2 [$ns node]
```

4.3 Links

Zwischen zwei Knoten werden die Verbindungen aufgespannt. Das können verdrahtete oder auch kabellose Verbindungen sein. Es gibt simplex- oder duplex-Links, mit einer bestimmten Bandbreite, Ausbreitungsgeschwindigkeit und einer Warteschlange am Ende.

```
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
```

4.4 Agents

Nun wird jedem Knoten ein Agent zugeordnet. Dieser implementiert das zu unterstützende Protokoll. Dabei ist zu beachten, dass auf der Serverseite das Protokoll spezifiziert wird und auf der Clientseite ein entsprechender Abfluss für das jeweilige Protokoll. Für UDP wäre das der Null-Agent, der alle UDP-Pakete verwirft und für TCP ein spezieller TCPSink-Agent, der dem Server noch ein Acknowledge zukommen lässt, bevor er das Paket verwirft.

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n2 $null
```

4.5 Traffic Generator

Um Datenverkehr durch das Netzwerk durchzuleiten, muss dem Agenten auf der Serverseite ein Trafficgenerator zugeordnet werden, der Daten erzeugt, die zu dem Client übertragen werden. Dies kann ein Datenstrom mit einer konstanten Bitrate (CBR) oder auch mit einer exponentiell steigenden Bitrate sein. Auch FTP- oder Telnet-Traffic ist möglich, wobei diese Trafficgeneratoren nur auf TCP-Agenten laufen. Zusätzlich kann man die Paketgröße und das zeitliche Intervall zwischen zwei Paketen angeben.

```
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 500
$cbr set interval_ 0.005
$cbr attach-agent $udp
```

4.6 Connect Agents

Schliesslich werden die beiden Agenten noch miteinander verbunden, so dass der Datenquelle auch der Datenabfluss zugeordnet wird.

```
$ns connect $udp $null
```

4.7 Events

Um die Simulation zu steuern, kann man Ereignisse festlegen, die zu einem bestimmten Zeitpunkt stattfinden. Dazu gehören z.B. das Starten der Trafficgeneratoren oder das Ausfallen einer Verbindungsleitung.

```
$ns at 0.5 "$cbr start"  
$ns rtmodel-at 1.0 down $n1 $n2  
$ns rtmodel-at 1.5 up $n1 $n2  
$ns at 2.0 "$cbr stop"
```

5 Zusammenfassung

Ns ist ein sehr leistungsfähiger Simulator. Die Unterstützung nahezu jeden Protokolls macht ihn zu einem mächtigen Werkzeug zur Evaluation von Netzwerken. Gerade die freie Verfügbarkeit hat ihn für die Forschung an Universitäten prädestiniert, was dazu führte, dass ns zum Fundament von vielen weiteren Protokollen und Routingmechanismen wurde. Durch die einfache Syntax von OTcl ist das Programmieren der Scriptdateien nicht weiter schwer. Auch die Entwicklung von eigenen Protokollen ist dank vieler Dokumentationen und nützlicher Tutorien aus dem Internet recht einfach möglich.

Da der Simulator selbst in C++ geschrieben ist, ist seine Ausführung schnell genug, um auch grosse Netzwerktopologien zu simulieren. Eine hohe Skalierbarkeit der Simulation ist also gewährleistet, vorausgesetzt man hat viel Hauptspeicher im Computer. Als interessant erweist sich die Echtzeitfähigkeit des eingebauten Schedulers: Mit ihm ist es möglich, ns als Emulator zu betreiben und so das tatsächliche Verhalten eines Netzwerkes als *black-box* in einem Computer zu testen. Leider befindet sich dieser Emulator zur Zeit in Entwicklung und läuft noch nicht stabil genug, um ihn ernsthaft einzusetzen. Aber da die Entwicklung an ns noch nicht abgeschlossen ist, wird auch dieses Problem bald behoben sein.

Die Konfiguration von ns über Scriptdateien hat aber auch den Nachteil, dass man erst die vielen Funktionen verwenden kann, wenn man die ganzen Schlüsselwörter kennt. Ein Nachschlagen in den Dokumentationen ist meistens unvermeidlich. Bei sehr grossen Topologien bedeutet das auch viel Tipparbeit, bis man das ganze Netzwerk in Textform beschrieben hat. Zwar bietet nam einen eingebauten Editor an, in dem man sich recht einfach und intuitiv eine Netzwerktopologie zusammenklicken kann, die dann als Scriptdatei für ns exportiert wird, aber es werden nur die Basisfunktionen und nicht alle Protokolle von ns unterstützt. Möchte man ein anderes Protokoll verwenden, dann muss die Scriptdatei nachträglich noch bearbeitet werden. Zusätzlich läuft nam nicht ganz stabil, so dass es vorkommen kann, dass der Editor abstürzt und die erstellte Topologie verloren ist.

Trotz der wenigen Nachteile ist ns ein hervorragendes Programm für den erfahrenen Benutzer, der damit auch grosse Netzwerke auf dem Trockendock austesten kann, zumal es auch unentgeltlich ist.

References

- [FV] Kevin Fall and Kannan Varadhan. *The ns Manual*.
- [Gre] Marc Greis. Marc greis' tutorial for the ucb/lbnl/vint network simulator ns". Tutorial für den ns Simulator.
- [KS00] Andreas Klotz and Markus Sigl. Evaluierung von netzwerk - testumgebungen. Master's thesis, Leopold-Franzens-Universität Innsbruck, 12 2000.
- [Non] one Non. The network simulator - ns2. Homepage des ns Simulators.

- [Rep03] Rainer Repp. Vergleich der verfahren simulation und emulation für die evaluation von protokollen. Master's thesis, Universität Stuttgart, 12 2003.
- [Tho04] David Thomi. Untersuchung eines bgp simulators. Master's thesis, Universität Koblenz-Landau, 10 2004.