Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semesterarbeit

Enhancing a VPN Based Network Emulator

Daniel Hottinger

25. November 2005

Betreuer: Nicolas Burri
Prof. Dr. Roger Wattenhofer

Englisch

This document describes the semester thesis of Daniel Hottinger at the *Distributed Computing Group*[3] of the *Department of Information Technology and Electrical Engineering (D-ITET)*[1] at the *ETH Zurich*[4].

The goal of this semester thesis was to enhance the Virtual Private Network (VPN) based network emulator written by René Gallati so that more realistic network simulations should become possible. The following features have been added:

- Unidirectional traffic shaping
- Individual delay on the links
- Collection of statistical information for the analysis of the network traffic
- Automated distribution and execution of programs on clients

Additionally, the project was ported to a native Debian package.

Deutsch

Dieses Dokument beschreibt die Semesterarbeit von Daniel Hottinger bei der *Distributed Computing Group*[3] des *Departements Informationstechnologie und Elektrotechnik (D-ITET)*[1] an der *ETH Zürich*[4].

Das Ziel dieser Semesterarbeit war es, den von René Gallati geschriebenen, VPN-basierten Netzwerk-Emulator um Funktionen zu erweitern, die realistischere Netzwerk-Simulationen ermöglichen. Die folgenden Eigenschaften wurden neu hinzugefügt:

- Unidirektionale Bandbreitenbeschränkung
- Individueller Delay auf den Links
- Sammeln statistischer Informationen zur Analyse des Netzwerkverkehrs
- Automatisiertes Einrichten und Ausführen von Programmen auf Testsystemen

Zudem wurde das Projekt auf Debian portiert.

Inhaltsverzeichnis

1. Einleitung **4**

2. Migration des vorhandenen Frameworks **4**

3. Traffic Shaping **5**
3.1. Hierarchical Token Bucket 5
3.2. Network Emulator 7
3.3. Applet 8

4. Simulation **9**
4.1. User Mode Linux 9

5. Schlussfolgerungen **10**
5.1. Mögliche Erweiterungen 10
5.1.1. Bandbreite auf Knoten 10
5.1.2. Network Emulator 11
5.1.3. Bessere Automation 11
5.1.4. Weitere Anregungen 11

A. Verwendete Komponenten und Installation **11**
A.1. Installation 12

B. Nachrichtenformat **13**

C. Einrichten eines neuen User Mode Linux (UML) **13**

D. CD-ROM **15**
D.1. Die CD-ROM 15
D.2. Inhalt der CD-ROM 15

E. Abkürzungsverzeichnis **16**

Literatur **17**

Abbildungsverzeichnis

1. Die Klassen unterhalb der Hierarchical Token Bucket (HTB) Qdisc 6
2. Link-Eigenschaften 7
3. Darstellung der Netzwerk-Topologie im Java-Applet 8
4. Aufbau des UML Systems 10

1. Einleitung

Das von René Gallati im Rahmen einer Semesterarbeit erstellte, VPN-basierte Projekt zur Simulation von Netzwerken, ermöglicht es Clients, sich mittels OpenVPN mit einem Server zu verbinden. Dabei entsteht ein virtuelles Netzwerk, bei dem die Verbindungen zwischen den Clients mittels eines Java-Applets beliebig getrennt und wieder freigeschaltet werden können.

Für eine realistische Simulation sind jedoch weitere Einstellungen wünschenswert. So kommt es in Wireless Netzen zu Kollisionen, Pakete gehen verloren, kommen doppelt an, überholen einander oder werden durch Retransmissions auf Transmission Control Protocol (TCP)-Ebene verzögert. Je mehr Clients sich in einem Ad-Hoc Wireless Local Area Network (WLAN) befinden, desto öfter treten solche Störeffekte auf, was die verfügbare Bandbreite begrenzt. Die gesamte Bandbreite eines Knotens wird durch konstante Grössen wie der maximalen Senderate einer Netzwerkkarte oder der vertraglich vereinbarten Bandbreite zum Internet Service Provider (ISP) begrenzt.

Das Ziel dieser Semesterarbeit war es, die vorhandene Arbeit so zu erweitern, dass eine realistischere Simulation von Netzwerken möglich wird. Insbesondere waren folgende Eigenschaften gewünscht:

- Unidirektionale Bandbreitenbeschränkung
- Individueller Delay auf den Links
- Sammeln statistischer Informationen zur Analyse des Netzwerkverkehrs
- Automatisiertes Einrichten und Ausführen von Programmen auf Testsystemen

Für die ersten beiden Punkte wurden die im Linux-Kernel vorhandenen Mechanismen zum Traffic-Shaping benutzt. Die HTB-Qdisc ermöglicht das gezielte Beschränken der Bandbreite und die NetEm-Qdisc bietet vielfältige Möglichkeiten, Störungen auf dem Netzwerk zu simulieren. Zur Aufzeichnung des Netzwerkverkehrs wird das von Thomas Graf geschriebene Programm `bmon`[1] eingesetzt, das die Aufzeichnung auf der Konsole oder als HTML visualisiert, in eine Datenbank gespeichert oder mittels Multi- bzw. Unicast an einen anderen Rechner senden kann.

Da es für grössere Simulationen nicht sehr praktikabel ist, dutzende von physikalisch existierenden Rechnern mittels OpenVPN anzubinden, besteht die Möglichkeit, UMLs automatisiert einzurichten und zu starten. Dadurch können alle existierenden, unter Linux lauffähigen Programme getestet werden.

2. Migration des vorhandenen Frameworks

Das VPN Testbed entstand aus der Semesterarbeit von René Gallati. Es besteht aus zwei Java Programmen, einigen Scripten, und einem Applet. Zur Simulation des Netzwerks

[1] http://people.suug.ch/~tgr/bmon/

verwendet es vorhandene Komponenten wie IPTables und OpenVPN. Das ganze Testbed ist für den Einsatz unter SuSE Linux ausgelegt.

In dieser Arbeit wurde als erstes der vorhandene Code nach Debian portiert, wodurch er einfach auf den Neptun-Rechnern der Distributed Computing Group (DCG) eingesetzt werden kann. Dabei wurden diverse, fest eingebaute Konstanten konfigurierbar gemacht und die Scripte fast vollständig umgeschrieben. Nicht typesichere Teile der Java Programme wurden ausserdem auf Java 1.5 portiert.

Das so entwickelte Debian-Paket trägt den Namen `vpnemu`, was die Kurzform von "*VPN* based Network *Emu*lator" ist. Es setzt die Installation weiterer, teils selbst erstellter Pakete voraus, die in Debian nicht, in einer zu alten oder nicht funktionsfähigen[2] Version enthalten sind. Der Paketmanager sorgt bei der Installation dafür, dass alle nötigen Pakete in der richtigen Version vorhanden sind. Die Abhängigkeiten und die Vorgehensweise bei der Installation sind in Anhang A auf Seite 11 beschrieben.

3. Traffic Shaping

Unter Traffic Shaping versteht man das Begrenzen der Bandbreite des Upstreams auf einem Netzwerk Interface mittels einer Queueing Discipline (Qdisc). Man unterscheidet zwischen classful und classless Qdiscs. Classful Qdiscs wie die Class Based Queue (CBQ) oder der Hierarchical Token Bucket (HTB) erlauben es, den Traffic mittels Filter in Klassen einzuteilen, welche hierarchisch in einer baumartigen Struktur angeordnet sind. Die Klassen teilen den Traffic in einem einstellbaren Verhältnis untereinander auf und können – sofern sie sich im Baum nicht direkt unter der Qdisc befinden – nicht genutzte Bandbreite von anderen Klassen borgen. Classless Qdiscs sind nicht so flexibel konfigurierbar, können aber hoch spezialisierte Aufgaben übernehmen. Stochastic Fairness Queueing (SFQ) zum Beispiel verteilt die Verbindungen mittels einer Hash-Funktion auf eine gewisse Anzahl von Slots, welche der Reihe nach geleert werden. Dadurch wird erreicht, dass die Bandbreite unter allen Verbindungen "fair" aufgeteilt wird. Um "unfairen" Einteilungen durch eine schlechte Hashfunktion vorzubeugen, werden die Parameter der Hashfunktion jeweils nach wenigen Sekunden geändert (daher die "Stochastic Fairness").

Das Begrenzen der Bandbreite auf dem Downstream nennt sich Policing und ist viel weniger mächtig als das Shaping, da es nicht möglich ist zu bestimmen, was für Daten gesendet werden. Für eine vollständige Beschreibung des Traffic Shaping unter Linux sei auf das Linux Advanced Routing & Traffic Control (LARTC) HOWTO unter [5] verwiesen.

3.1. Hierarchical Token Bucket

Für diese Arbeit wird die HTB Qdisc eingesetzt, welche unter [2] ausführlich dokumentiert ist. In der Hierarchie unterhalb der Qdisc wird eine einzige Klasse angelegt, die al-

[2] OpenVPN $<$ 2.0 verwendete nicht initialisierte Werte für einen System Call. Debian erlaubt aber im Stable-Zweig nur Updates, die eine Sicherheitslücke beheben.

len Traffic aufnimmt. Diese Klasse verwaltet für jede eingehende Kante im Netzwerk-Graphen eine Unterklasse, wodurch es möglich wird, dass Verbindungen untereinander ungenutzte Bandbreite sharen können. Die Bandbreite eines Links wird also auf dem Ziel-Netzwerk-Interface begrenzt.

Hier ist es auch möglich, die gesamte ein- und ausgehende Bandbreite des Knotens zu begrenzen. Dies hat sich aber als nicht praktikabel herausgestellt, da dabei die Summe der Bandbreiten der Subklassen grösser werden kann als die eingestellte maximale Bandbreite der Parentklasse oder der Qdisc. In der FAQ kommentiert der HTB-Author diesen Fall mit "Then interesting things can happen.", was konkret bedeutet, dass die Partenklasse/Qdisc die eingestellte maximale Bandbreite überschreitet. Die Funktionalität zur Begrenzung der Bandbreite auf einem Knoten ist daher vorhanden, aber nicht aktiviert.

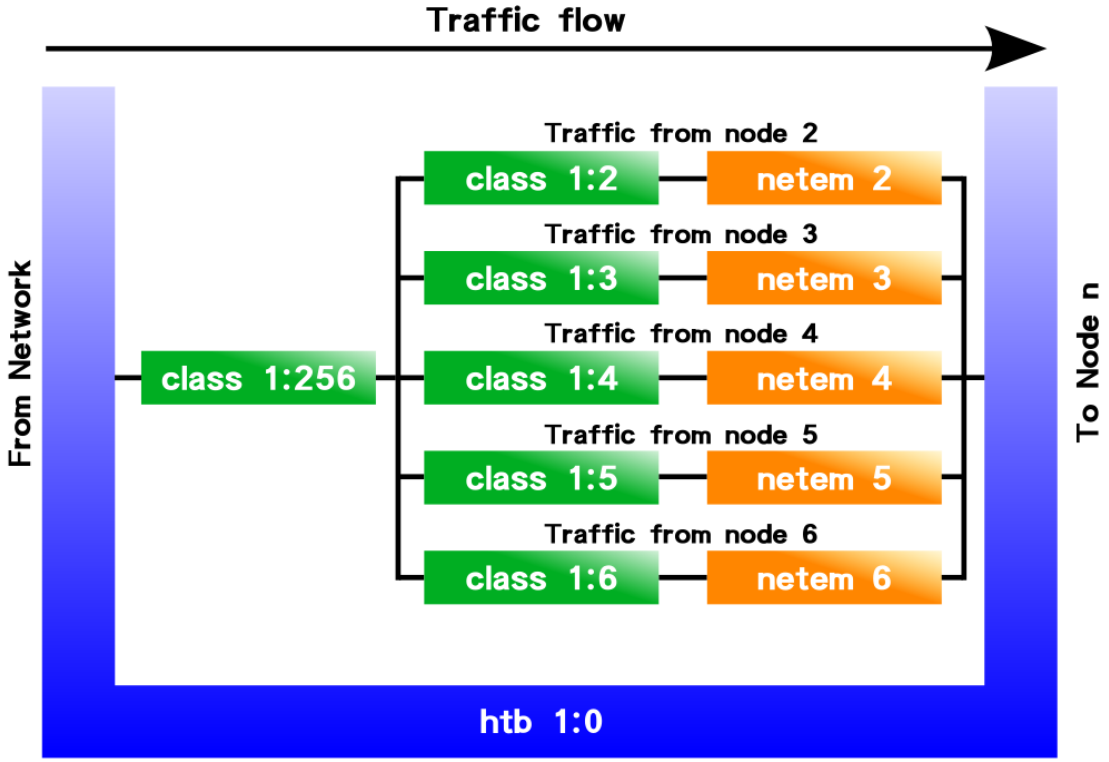


Abbildung 1: Die Klassen unterhalb der HTB Qdisc

Abbilung 1 auf Seite 6 zeigt die Konfiguration für einen Knoten n auf dem Interface `tap`n. Alle Klassen und Qdiscs werden durch ein eindeutiges Label der Form x:y identifiziert. Alle Pakete von einem Knoten y werden mittels eines Filters, welcher der HTB-Qdisc 1:0 zugeordnet ist, der Klasse 1:y zugeteilt. Die Klasse 1:256 könnte zur Begrenzung des Downstreams des Knotens n benutzt werden, dies wird aber leider von HTB nicht unterstützt.

Nachdem die Bandbreite begrenzt wurde, durchlaufen die Pakete die NetEm Qdisc

y bzw. y:0, wo diverse Netzwerk-Störungen simuliert werden.

3.2. Network Emulator

Um das Verhalten von Netzwerken besser simulieren zu können, wird unterhalb jeder Subklasse von HTB die Network Emulator (NetEm) Qdisc installiert. Der Name ist leicht missverständlich, da die Qdisk kein Netzwerk emuliert, wohl aber die typischen Effekte, die auf einem Link auftreten können. NetEm unterstützt folgende Parameter, welche, wie in Abbildung 2 auf Seite 7 zu sehen ist, auch alle im Applet einstellbar sind:

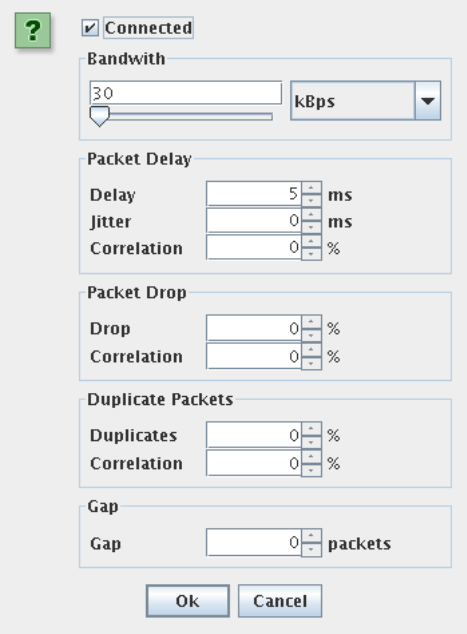


Abbildung 2: Link-Eigenschaften

delay Stellt den auftretenden Delay in Millisekunden ein. Da auf einem echten Netzwerk der Delay nicht konstant ist, kann ein Jitter-Wert eingestellt werden, welcher angibt, um wie viel der tatsächliche Delay vom eingestellten Wert abweichen darf. Mittels Correlation ist es zudem möglich anzugeben, wie stark der Delay zwischen zwei Paketen korreliert. Beispiel: Ein Delay von 100ms mit 10ms Jitter und 25% Correlation sagt aus, dass ein Paket durchschnittlich um 100ms $\pm$ 10ms verzögert, und die Verzögerung des nächsten Pakets zu $1/4$ aus dem aktuellen Delay abgeleitet wird.

drop Gibt an, wieviele Prozent der Pakete verloren gehen. Auch hier kann eine Korrelation angegeben werden, wodurch sich "Bursts" besser simulieren lassen.

duplicate Gelegentlich werden Pakete doppelt gesendet, da der Empfang nicht bestätigt wurde oder sie irgendwo im Netz verdoppelt wurden. Die Anzahl der dop-

pelt versandten Pakete und ihre Korrelation kann ebenfalls in Prozent eingestellt werden.

gap Damit wird es möglich, dass nur jedes n-te Paket von den Netzwerkstörungen betroffen ist. Zusammen mit dem Delay kann so z.B. jedes 5. Paket um 200ms verzögert werden, was dazu führt, dass es out-of-order am Ziel eintrifft.

NetEm wurde vom Autor nicht in Verbindung mit HTB getestet, wodurch bei älteren Kerneln hohe, teils stetig wachsende Delays auftraten oder es sogar zum Stillstand des gesamten Systems kam. Der Fehler wurde erst im Laufe dieser Arbeit entdeckt und ist in den neuesten Kerneln ab 2.6.12 behoben.

Das von den Java-Komponenten verwendete Nachrichtenformat zur Kommunikation der Linkparameter ist in Anhang B auf Seite 13 beschrieben und liesse sich leicht erweitern, sodass weitere Qdiscs unterstützt werden könnten.

3.3. Applet

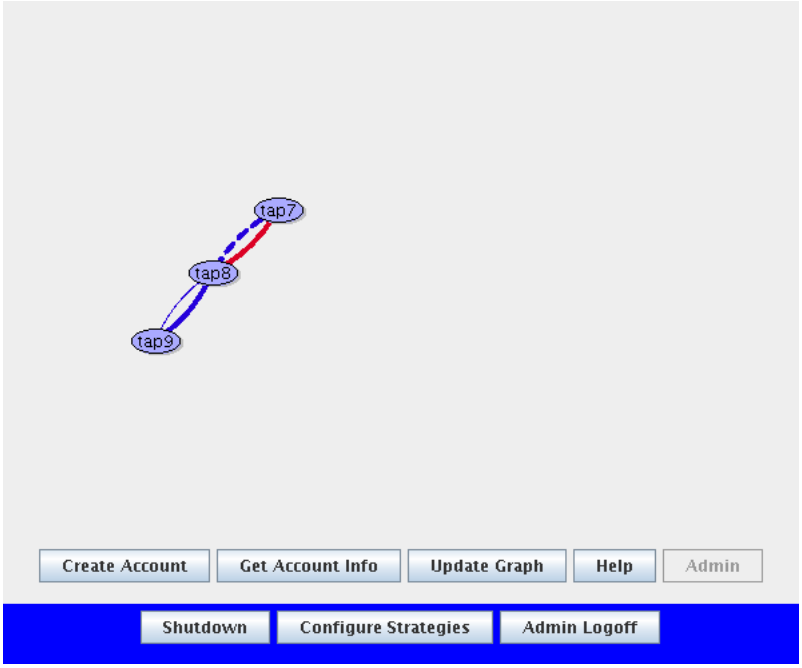


Abbildung 3: Darstellung der Netzwerk-Topologie im Java-Applet

Für die Visualisierung der Netzwerktopologie und das Ändern der Link-Parameter aus dem Browser heraus ist ein Java-Applet zuständig, welches unter `http://localhost/` erreichbar ist. Es wurde an die erweiterte Funktionalität angepasst: neu wird durch "Ziehen" von einem Knoten zu einem anderen der Dialog aus Abbildung 2 auf Seite 7 angezeigt.

Die Darstellung des Netzwerkes ist aus Abbildung 3 auf Seite 8 ersichtlich. Die Verbindungen sind nach Richtung getrennt worden und verlaufen immer rechts von der direkten Verbindungsgeraden zwischen zwei Knoten. Einige Eigenschaften der Verbindung sind direkt ersichtlich: Die blauen Verbindungen verfärben sich proportional zum Delay rot, die Bandbreite wird durch die Breite angedeutet und wenn Pakete verloren gehen, so wird die Verbindung zunehmend gestrichelt gezeichnet.

In der Abbildung besteht zwischen den Knoten tap8 und tap7 ein hoher Delay während in der Gegenrichtung einige Pakete verloren gehen. Zudem ist die Bandbreite von tap8 zu tap9 stark eingeschränkt.

4. Simulation

Durch Simulationen können Fehler in der Spezifikation von (verteilten) Protokollen leichter entdeckt und frühzeitig behoben werden. So könnte es wünschenswert sein, ein Peer-to-peer (P2P)-Protokoll auf die Resistenz gegen Ausfälle einzelner Knoten (z. B. durch gezielte Angriffe mittels Denial of Service (DoS) oder Anwälten) zu testen oder das Verhalten von Voice over IP (VoIP) Implementationen wie Skype[3] auf die Anfälligkeit auf Packetloss zu untersuchen.

Da die Implementationen solch (verteilter) Protokolle aber in der Regel nicht für einen Simulator geschrieben werden oder nicht im Quellcode verfügbar (kommerziell) sind, ist es von Vorteil, wenn sie in einer vollwertigen Betriebssystem-Umgebung ablaufen können. Linux wird aufgrund der hervorragenden Netzwerk-Eigenschaften von vielen Entwicklern (auch kommerzieller Programme) als Plattform geschätzt und unterstützt.

4.1. User Mode Linux

Da das Simulieren mit vielen, physikalisch exisitierenden Rechnern mühsam oder nicht praktikabel ist, besteht neu die Möglichkeit, auf dem Testrechner neben `vpnemu` mehrere Instanzen eines User Mode Linux (UML) zu booten. Das sind vollwertige Debian-Installationen, deren Kernel auf dem Host-System als normale Prozesse laufen. Die Konfiguration ist schematisch in Abbildung 4 auf Seite 10 zu sehen.

Um Konflikten mit den von OpenVPN verwendeten tap-Interfaces vorzubeugen, ist die Nummer der von UML benutzten tap-Interfaces jedoch um 256 höher als auf der Abbildung dargestellt. Für die Kommunikation der UML-Instanzen mit dem Host und dem Internet wird eine zweite Bridge verwendet. IP-Forwarding wurde aktiviert und die Firewall übersetzt die privaten Adressen mittels Network Address Translation (NAT).

Das Einrichten eines UML ist in Anhang C auf Seite 13 beschrieben.

[3] `http://skype.com/`

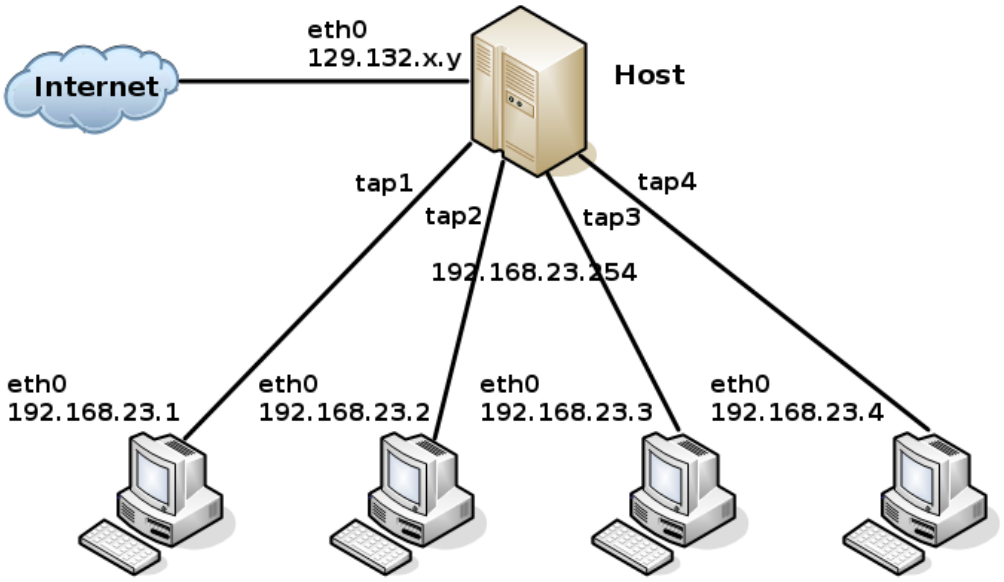


Abbildung 4: Aufbau des UML Systems

5. Schlussfolgerungen

Die Mechanismen zum Traffic Shaping sind unter Linux seit langer Zeit vorhanden, werden rege genutzt und haben sich für diese Semesterarbeit als sehr nützlich erwiesen. Allerdings lässt die Dokumentation manchmal zu Wünschen übrig, vorallem, wenn man nicht das typische Szenario eines Firmen- oder Heimnetzwerks, das ans Internet angebunden werden soll, anstrebt. Trotzdem war diese Arbeit sehr interessant und ich habe viel gelernt. Das Aufsetzen eines UML lässt sich unter Debian mit den richtigen Tools erstaunlich gut automatisieren und zeigt, wie flexibel und mächtig eine Linux-Installation inzwischen sein kann.

An dieser Stelle möchte ich mich ganz herzlich bei Attila Kinali, Thomas Graf und Franziska Meyer bedanken, welche mich mit Rat und Tat unterstützt haben.

5.1. Mögliche Erweiterungen

5.1.1. Bandbreite auf Knoten

Um die gesamte Bandbreite eines Knoten einzustellen, ist das Framework bereits erweitert worden. Das Begrenzen dieser hat jedoch nicht zufriedenstellend funktioniert, sodass die Funktionalität momentan deaktiviert ist. Das Problem war das spezielle Verhalten der HTB-Qdiscs wenn die Bandbreite der Parent-Klasse kleiner ist als die Summe der Bandbreiten der Kindklassen und dass das Matching auf der ingress Qdisc mit dem eingesetzten Kernel nicht funktioniert hat. Diese Probleme sollten mit genügend Zeit jedoch lösbar sein.

5.1.2. Network Emulator

NetEm nimmt für Prozentangaben eine Normalverteilung an, kann aber auch individuell einstellbare Verteilungen benutzen. Die Verteilungen sind Tabellen, welche unter `/usr/lib/tc` abgelegt werden. Es sollte möglich sein, neue Verteilungen selbst zu definieren und pro Link individuell einzustellen. Dazu muss das Nachrichtenformat erweitert und neue Nachrichten zum Erstellen von solchen Tabellen müssen eingeführt werden.

5.1.3. Bessere Automation

Auch das Einrichten von UMLs (siehe Anhang C auf Seite 13) liesse sich noch besser automatisieren. Insbesondere sollte die Konfiguration automatisch erfolgen. Die Informatik Support Groups (ISGs) der Departemente D-PHYS[4] und D-ITET[5] könnten hier sicher Auskunft geben. Denkbar wäre auch die vollständige Automation von Tests: Ein Programm richtet automatisch die UMLs ein, installiert die Testsoftware, welche als Debian-Paket vorliegt, startet die UMLs, verändert die Netzwerkverbindungen nach einem vorgegebenen "Drehbuch" und sammelt dabei mittels `bmon` statistische Informationen in einer Datenbank, welche dann ausgewertet werden kann.

5.1.4. Weitere Anregungen

Die Verschlüsselung mit OpenVPN ist rechenintensiv und begrenzt die erreichbare Bandbreite auf ca. 10 Mbit/s, je nach Leistungsfähigkeit der CPU. Für lokale Tests sollte diese deshalb abgeschaltet werden können.

UML ist beschränkt auf ein Betriebssystem: Linux. Es gibt mittlerweile weitere Virtualisierungslösungen wie Xen[6] oder VMware[7], welche möglicherweise eingesetzt werden könnten, um das Testen von mehr Programmen und/oder IP-Stacks zu ermöglichen.

Die Dokumentation auf [5] ist weder vollständig noch aktuell. Es wäre für alle Nutzer von Linux eine Bereicherung, wenn sie auf den aktuellen Stand gebracht würde.

A. Verwendete Komponenten und Installation

Das Debian-Paket `vpnemu` hat folgende Abhängigkeiten, die vom Installationstool (`apt-get`) grösstenteils automatisch aufgelöst werden können:

openvpn (>=2.0) Nimmt eingehende VPN-Verbindungen entgegen. Ältere Versionen hatten einen Programmierfehler, wodurch nicht initialisierte Werte in System Calls benutzt wurden.

[4] http://www.phys.ethz.ch/~franklin/Projects/index.html.en
[5] http://isg.ee.ethz.ch/
[6] http://xen.sf.net/
[7] http://www.vmware.com/

iptables Steuert die Firewall an und öffnet/schliesst Verbindungen auf den virtuellen Links.

iproute Ersetzt die veralteten Werkzeuge `ifconfig`, `route` und `arp`. Für das Ansteuern des Traffic Shaping wird das Programm `tc` unbedingt benötigt.

perl (>= 5.6.1) Die Scripts sind in Perl geschrieben und erwarten mindestens die Version 5.6.1.

j2re1.5 Da die Java-Programme Java 1.5 Konstrukte verwenden, wird eine neuere Java Virtual Machine (JVM) benötigt. Das Debian-Paket ist auf der CD enthalten, kann aber auch leicht selbst aus dem von SUN angebotenen Java Runtime Environment 1.5 mit Hilfe von `make-jpkg` aus dem Paket `java-package` erstellt werden.

adduser Wird zum Erstellen neuer Benutzer benötigt.

ntp Stellt sicher, dass die Zeiten in den Logfiles stimmen.

liblog4j1.2-java Wird von den Java-Programmen benötigt, um Fehlermeldungen zu loggen.

bridge-utils Sorgen dafür, dass die virtuellen Netzwerk-Interfaces untereinander verbunden werden können.

apache2 Wegen der modularen Konfiguration eignet sich Apache 2 besonders gut als Webserver, da eine parallel laufende Webseite nicht durch die Installation von `vpnemu` beeinträchtigt wird.

bmon (>= 2.1.1-pre1) Visualisiert den Netzwerkverkehr. Die auf der CD enthaltene Version ist neuer als die in Debian vorhandene und wurde für diese Arbeit leicht erweitert. Die Änderungen werden im kommenden Release übernommen werden.

kernel-image-2.6.12 Diese Abhängigkeit ist nicht explizit eingetragen, sollte aber unbedingt beachtet werden. Aufgrund eines Fehlers in der NetEm Qdisc kann es bei älteren Kerneln zu einem Delay von 1-2 Sekunden oder sogar zu einem Deadlock im Kernel kommen.

A.1. Installation

Die auf der CD enthaltenen Pakete von `bmon`, `libnl` und Java sollten entweder vor `vpnemu` mittels `dpkg -i datei.deb` installiert oder auf einen lokalen Debian-Mirror kopiert werden, sodass das Installationstool von Debian (`apt-get`) die Abhängigkeiten automatisch auflösen kann. Soll die vorhandene UML-Funktionalität genutzt werden (Siehe Abschnitt 4 auf Seite 10), so müssen zusätzlich die Pakete `user-mode-linux` und `rootstrap` installiert sein. Dies wird automatisch gemacht, sobald versucht wird, ein UML, wie in Anhang C auf Seite 13 beschrieben, zu erstellen.

Nach erfolgreicher Installation muss die Konfiguration in `/etc/vpnemu/global.conf` angepasst werden. Danach ist das Neustarten mittels `/etc/init.d/vpnemu restart`

notwendig. Nun ist alles betriebsbereit und das Applet zur Visualisierung des Netzwerkverkehrs kann unter `http://localhost/` erreicht werden.

B. Nachrichtenformat

Innerhalb der Java-Programme werden die Meldungen über die Linkparameter von einem Knoten X zu einem Knoten Y in folgendem Format (ohne Zeilenumbruch) ausgetauscht:

```
tapX:tapY,bandwidth,delay,jitter,correlation,drop,correlation,
  duplicates,correlation,gap
```

`tapX:tapY` ist der Link, alle übrigen Parameter sind Integerzahlen in den Einheiten Prozent oder Millisekunden.

Da das Format sehr einfach gehalten ist, wäre es z. B. ohne weiteres möglich, anstelle von NetEm eine andere Qdisc einzusetzen. Das Format müsste dann lediglich um den Namen der verwendeten Qdisc erweitert werden und die Parameter wären von der eingesetzten Qdisc abhängig.

Auch für Knoten können die Parameter für Up- und Downstream eingestellt werden. Das Format ist wie folgt definiert:

```
tapX,upstream,downstream
```

Da diese Funktionalität, wie bereits in Abschnitt 3.2 auf Seite 7 erwähnt, bei Tests nicht zufriedenstellend funktioniert hat, ist der entsprechende Dialog zur Zeit im Applet deaktiviert - die internen Meldungen sind aber bereits vollständig implementiert.

C. Einrichten eines neuen UML

Um ein neues UML einzurichten, muss das Init-Script `/etc/init.d/vpnemu` bereits gestartet worden sein. Dies geschieht im Normalfall automatisch beim Booten oder bei der Installation des Debian-Pakets. Nun müssen folgende Schritte ausgeführt werden:

- Im Applet muss ein neuer Benutzer erstellt werden. Seine interne Nummer n wird sowohl für das tap-Interface von OpenVPN, als auch für das des UML, sowie für den letzten Teil der IP-Adresse des UML verwendet.
- Auf der Konsole gibt man nun als *root* folgendes ein:
  `vpnemu-uml create` n
  Dies installiert das neue UML System. Wenn die Installation mit einer Kernel-Panic abbricht, so liegt das zumeist daran, dass keine Internet-Verbindung aufgebaut werden konnte und man sollte die Firewall und die Erreichbarkeit von `http://debian.ethz.ch/` überprüfen.
- Nach der Grundinstallation fährt das Linux ordnungsgemäss herunter und kann nun ein erstes Mal mittels
  `vpnemu-uml run` n

gestartet werden. Das Einloggen ist als *root* ohne Passwort möglich. Das System muss nun noch mittels:
`dpkg-reconfigure -a`
vollständig konfiguriert werden. Es können eigentlich alle Fragen mit `<Enter>` bestätigt werden ausser:

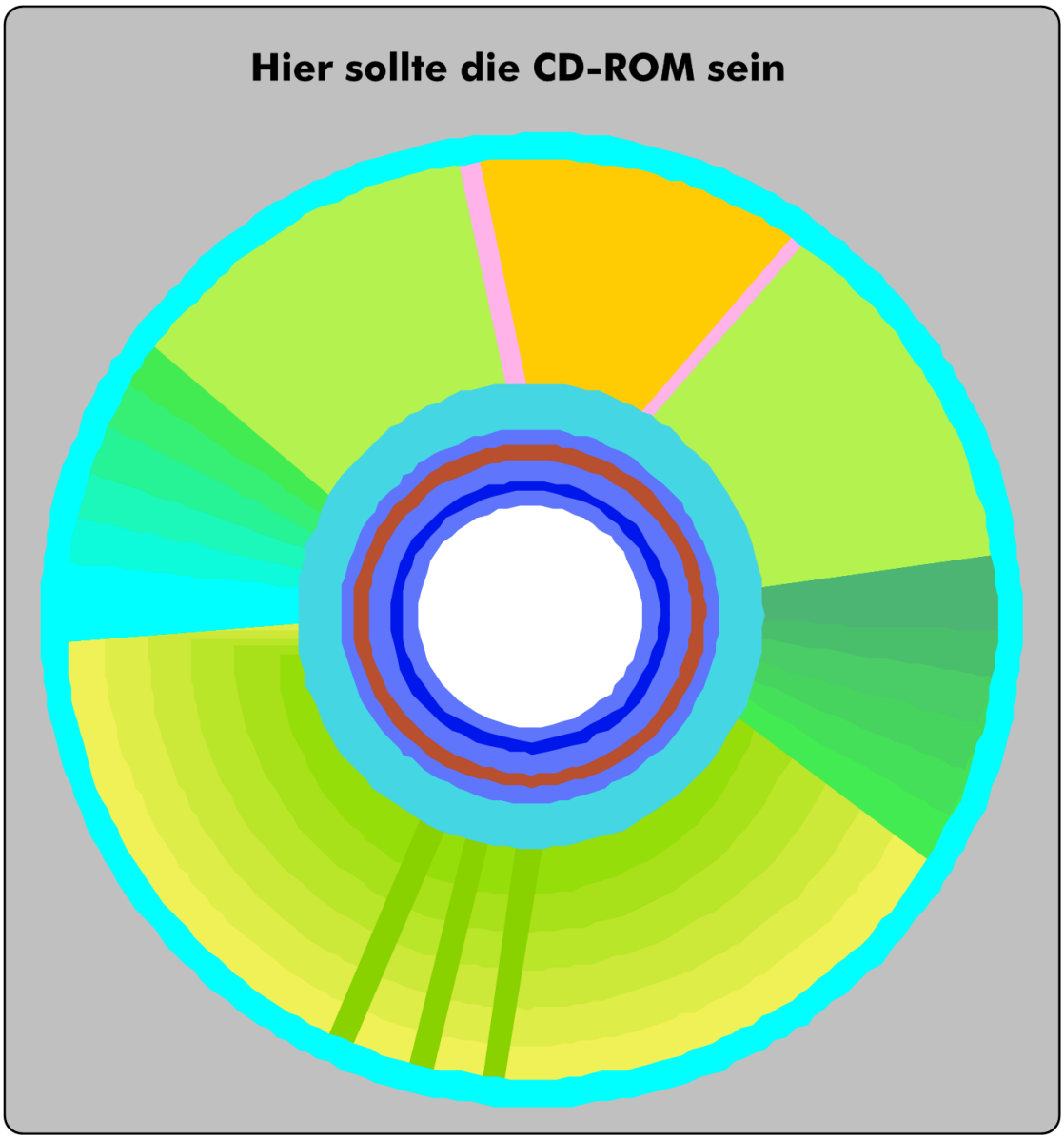
  - Das Root-Passwort muss gesetzt werden
  - Bei der Konfiguration von OpenVPN muss ein tap-Interface erstellt werden (der Vorschlag ist, es nicht zu erstellen)
  - Es muss kein zusätzlicher Benutzer angelegt werden

- Nach abgeschlossener Konfiguration kann das UML mittels `halt` heruntergefahren werden.
- Beim nächsten Start mittels `vpnemu-uml run` *n* wird automatisch eine VPN-Verbindung zum Host aufgebaut und der Knoten erscheint im Applet.

Um den Traffic über TCP-Verbindungen zu testen, wurde in `/etc/inetd.conf` ein leichtgewichtiger Service auf Port 4242 eingerichtet, welcher dauernd Zeilen mit dem Text "y" generiert. Er kann mittels `telnet <ip> 4242` oder `nc <ip> 4242` getestet werden. Die Verwendung von NetCat (`apt-get install netcat`) ist zu empfehlen, da das Programm leicht mittels Control-C abgebrochen werden kann.

D. CD-ROM

D.1. Die CD-ROM



D.2. Inhalt der CD-ROM

Die CD-ROM enthält folgende Verzeichnisse:

doc Der Bericht, die Präsentation und die Handouts.

packages Debian-Pakete und Quelltexte

tex LaTeX-Dateien für den Bericht und die Präsentation

E. Abkürzungsverzeichnis

CBQ Class Based Queue

CPU Central Processing Unit

DCG Distributed Computing Group

DoS Denial of Service

ETH Eidgenössische Technische Hochschule

FAQ Frequently Asked Questions

HTB Hierarchical Token Bucket

HTML HyperText Markup Language

IP Internet Protocol

ISG Informatik Support Group

ISP Internet Service Provider

JVM Java Virtual Machine

LARTC Linux Advanced Routing & Traffic Control

NAT Network Address Translation

NetEm Network Emulator

P2P Peer-to-peer

SFQ Stochastic Fairness Queueing

TCP Transmission Control Protocol

UML User Mode Linux

VoIP Voice over IP

VPN Virtual Private Network

WLAN Wireless Local Area Network

Literatur

[1] Departement Informationstechnologie und Elektrotechnik (D-ITET). http://www.ee.ethz.ch/.

[2] Martin Devera and Don Cohen. HTB Linux queuing discipline manual – user guide. http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm.

[3] Distributed Computing Group. http://dcg.ethz.ch/.

[4] Eidgenössische Technische Hochschule Zürich. http://www.ethz.ch/.

[5] Bert Hubert, Thomas Graf, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, and Pedro Larroy. Linux Advanced Routing & Traffic Control (LARTC) HOWTO. http://www.lartc.org/lartc.html.