Projektarbeit

IP − Netzsimulator

21.05.2001 − 06.07.2001

Dozent: Dipl. El−Ing. Eth Hans Weibel

Partnerfirma: Omnisec
Dipl. El−Ing. HTL Peter Fernandez

Gruppe: Dejan Miletic & Lukas Valle

1 Inhaltsverzeichnis

2 Zusammenfassung

Je länger je mehr wird die Sicherheit im Internet zu einem der wichtigsten Themen überhaupt. Für die Firma Omnisec ist dies umsomehr von zentraler Bedeutung, beschäftigt sie sich doch fast ausschliesslich mit dieser Thematik.

In einer Diplomarbeit [DA04] ist ein IP−Paketgenerator entstanden, welcher in der Lage ist, das Verkehrsaufkommen mehrerer kompletter Netze zu simulieren. Der Generator wird vom Industriepartner verwendet, um Security Gateways in einer möglichst realitätsnahen Umgebung zu testen.

Unsere Aufgabe bestand darin, gewisse Funktionen zu verbessern und den Generator mit zusätzlichen Komponenten zu erweitern. Des weiteren galt es, eine Fernbedienungssoftware fertigzustellen, welche im Rahmen einer Projektarbeit [PA] im Studiengang K&I angefangen wurde. Das Ziel war es, diese beiden Einzelarbeiten so zusammenwachsen zu lassen, dass dem Industriepartner ein gut funktionierendes, getestetes und dokumentiertes Softwarepaket zur Verfügung gestellt werden kann.

Anfänglich beschäftigten wir uns mit der vorhandenen Software und den Dokumentationen zu den beiden bestehenden Arbeiten. Nachdem wir die Software auf unseren Systemen installiert und zum Funktionieren gebracht hatten, begannen wir mit der Planung und Implementation der Änderungen. Dabei hatten folgende Punkte Priorität:

- Überarbeitung des gesamten Zufallsgenerator, denn es hatte sich herausgestellt, dass dieser zu oft die gleichen Werte lieferte.
- Implementation der Lastverteilung, damit die Pakete verteilt verschickt werden können, um so das Netz gleichmässig mit einem bestimmten Datenaufkommen zu belasten.
- Sicherstellung der Fernbedienbarkeit, damit der Generator von irgendeinem Client mit beliebigem Betriebssystem ferngesteuert und überwacht werden kann.
- Erhöhung der Flexibilität der Skriptsprache.
- Überwachung der Security Gateways.

Durch die komplette Überarbeitung des Zufallsgenerators war es uns möglich, den Quellcode sowie die Skriptsprache merklich zu vereinfachen. Mit der unter Java programmierten Software für die Fernbedienbarkeit, können nun mehrere Generatoren von zentraler Stelle gesteuert werden. Ist ein Simulator am Ende oder ein Security Gateway nicht mehr erreichbar, wird dies dem Benutzer mitgeteilt. Die Lastverteilung kann neu mit einem Befehl im Skript beeinflusst werden. Bei der Flexibilisierung der Skriptsprache konnten wir auf Tools zurückgreifen, welche bereits von Herrn Fernandez erstellt wurden. Bei den Änderungen wurde soweit als möglich auf die Abwärtskompatibilität geachtet. So gelang es uns, trotz relativ kurz bemessener Zeit, die oben genannten Punkte zu realisieren.

Winterthur, den 6.07.01

Lukas Valle Dejan Miletic

3 Aufgabenstellung

3.1 Beschreibung

Basis des Generatores ist eine Funktion, die den Versand beliebig aufgebauter IP−Pakete über den sogenannten Raw Socket von Linux ermöglicht. Durch ein Skript wird die Generatorfunktion so gesteuert, dass ein Paketstrom mit den gewünschten Eigenschaften entsteht und den Verkehr zwischen n Quellen und m Senken darstellt. Dieser Generator soll nun um weitere Funktionen ergänzt und weiterentwickelt werden wie etwa Lastprofile, Logging und Fernbedienbarkeit. Die Arbeit bietet nebst dem interessanten Thema auch Kontakte und Einblick in das Entwicklungs− und Anwendungsumfeld von Secure Data Networks.
Stichworte: Konzipieren, programmieren, integrieren, Softwareentwicklung unter Linux, C++, IP Protokoll Suite

3.2 Aufgaben

Nachdem wir uns mit der Funktionsweise des Paketgeneratores vertraut gemacht hatten, bot sich am Ende der zweiten Projektwoche die Möglichkeit eines persönlichen Gesprächs mit Herrn Fernandez und einer Besichtigung der Firma Omnisec. Anlässlich dieses Gesprächs wurden uns diverse Probleme erläutert, welche sich im alltäglichen Gebrauch des Paketgeneratores ergaben. Schliesslich wurde zusammen mit Herrn Fernandez und Herrn Weibel eine Prioritätenliste für die anstehenden Verbesserungen erstellt.

1. Überarbeitung des Random Generatores
2. Steuerung der Lastverteilung
3. Remote−Control und GUI
4. Alte Skriptsprache um die Skripte von Herr Fernandez erweitern
5. Überwachung der verschiedenen Gateways
6. Fragmentierung von grossen IP Paketen

3.3 Infrastruktur

Gebäude:	ZHW Winterthur, Gebäude E
Labor:	E523
Entwicklungs−PCs:	2 Stk.
CPU:	PIII−450MHz
RAM:	128MB
Ethernetkarte:	Digital DE 500 (PCI)
Betriebssystem:	SuSE Linux 7.1 Kernel 2.4
Generator / Senke:	
CPU:	Intel Celeron−700 Mhz
RAM:	64MB
Ethernetkarte:	3Com 3c905B 10/100 (PCI)
Betriebssystem:	SuSE Linux 7.1 Kernel 2.4

3.4 Software

Java:	JDK1.3	Java Entwicklungsumgebung
JBuilder Professional:	4.0	Entwicklung Java Das komplette JBuilder−Projekt liegt auf der CD im Verzeichnis /`Pojects/JBuilder` bei
KDevelop:	1.4	Enwicklung c++ Das komplette KDevelop−Projekt liegt auf der CD im Vezeichnis `/Projects/KDevelop` bei
KDE2:	2.1	Graphische Linux Oberfläche
StarOffice	5.2	Textverarbeitung / Tabellenkalkulation
tcpdump	3.4a6	Paketsniffer
ethereal	0.8.13	Paketanalyser
ksniffer	0.1.6	Bandbreitendarstellung

4 Einleitung

Ein Problem des Internets, das sowohl die staatliche als auch die kommerzielle Nutzung behindert, ist die mangelnde Sicherheit dieses Mediums. So kann sich ohne besondere Vorkehrungen niemand sicher sein, dass seine übertragenen Daten beim Empfänger ankommen, ohne abgefangen, mitgelesen und verändert zu werden. Im gleichen Masse wie die kommerzielle Nutzung des Internets wächst, nimmt auch der Bedarf an geschützter Kommunikation zu.

Ein Problem aller Systeme, welche auf dem IP−Protokoll aufbauen, ist, dass sie nach dem Store−and−Forward−Prinzip arbeiten. Die Pakete werden dabei über mehrere Router teilweise um die halbe Welt weitergeleitet. Wer Zugriff auf einen dieser Router hat, kann die ein− und ausgehenden IP−Pakete problemlos mitlesen, verändern oder fälschen. Durch Filterprogramme ist es möglich, automatisch nach bestimmten Mustern oder Absendern zu suchen. Letzteres ist sicher ein Feature, das nicht nur für Geheimdienste und Wirtschaftsspione, sondern auch für die Werbeindustrie von Interesse sein dürfte.

Ein weiteres Problem ist, dass ein Angreifer Pakete mit gefälschter Absendererkennung generieren kann. Der Empfänger kann nicht sicher sein, ob die Daten von dem System versandt wurden, von welchem sie zu kommen scheinen. Um nun die Übertragung zwischen zwei geschützten Netzwerken sicherer zu gestalten, wird der gesamte Verkehr zwischen diesen beiden Netzwerken verschlüsselt.

4.1 Virtual Private Network (VPN)

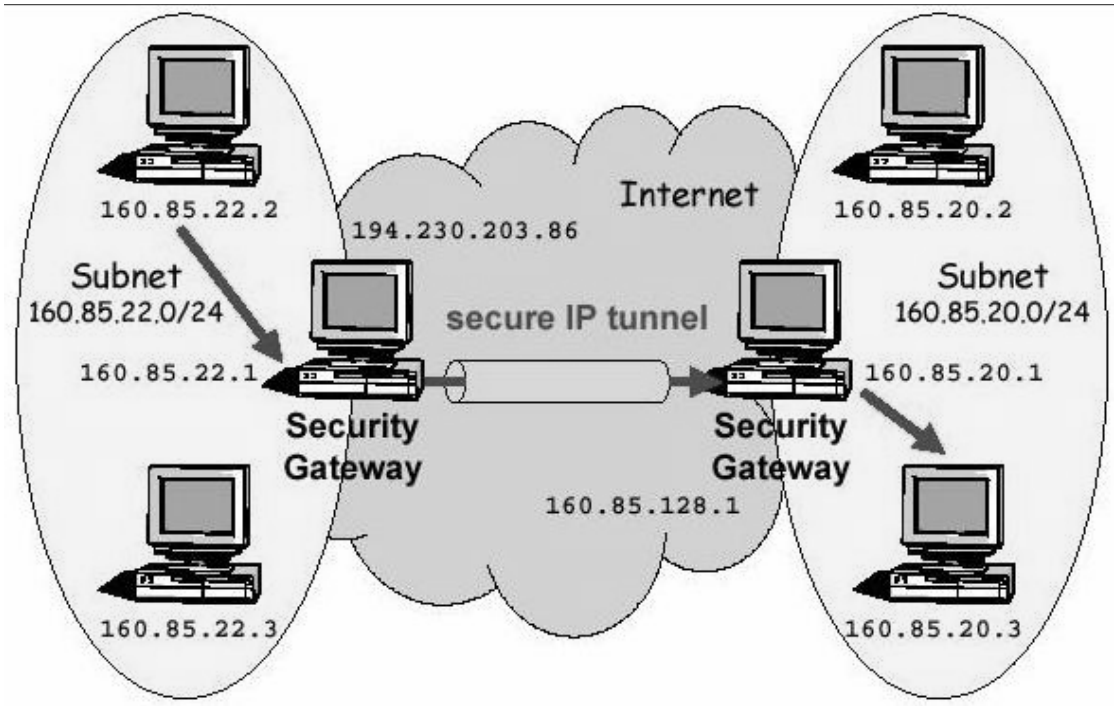


Abbildung 4.1 − Virtual Private Network

Das VPN stellt zwischen den beiden Firmensubnetzen einen virtuellen Tunnel dar, in dem nur Daten transportiert werden, die zwischen den beiden Endpunkten des VPN verschickt werden. Alle anderen Stationen im Internet können die VPN Pakete zwar sehen, aber nichts damit anfangen, weil ihre, normalerweise im Klartext, übertragenen Informationen zwischen den beteiligten Parteien verschlüsselt sind. Jedes Datenpaket im Netzwerk besteht aus einem Header der Informationen über Herkunft und Ziel des Pakets enthält und dem Teil mit den eigentlichen Informationen. Die VPN Software nimmt das Paket, verschlüsselt sowohl Nutzlast wie auch den ursprünglichen Header und packt einen neuen Header dazu. Am Ziel angekommen, entfernt der VPN−Gateway den temporären Header und entschlüsselt das Paket. Nun sieht die Information wieder so aus, wie vor dem Weg durch den Tunnel. Der Code zum Ver− und Entschlüsseln wird natürlich nicht im Paket mitge−schickt. Er beruht entweder auf einem Public Key Verfahren oder wurde vorher vereinbart.

4.2 IPSec & Security Gateways

IPSec bietet Security auf dem IP Network Layer indem es ein System anbietet, welches das benö−tigte Security Protokoll wählt, den Algorithmus für den Service und die kryptographischen Schlüssel installiert. Da der Sicherheitsdienst auf dem IP Layer angeboten wird, kann er von allen höheren Layern, wie TCP, UDP, ICMP, etc. benützt werden. IPSec kann dazu verwendet werden um einen oder mehrere Wege zwischen zwei Hosts, zwei Security Gateways oder zwischen einem Security Gateway und einem Host zu sichern. Normalerweise werden Systeme auf welchen IPSec implementiert ist als Security−Gateway bezeichnet. Beispiele sind Router oder Firewalls mit IPSec. Ein Security Gateway gilt als einfache und gute Lösung, um ein VPN mittels IPSec zu realisieren. Natürlich könnte der Verschlüsselungsvorgang auch jeweils auf jedem einzelnen PC realisiert werden. Dies würde jedoch einen erheblichen Mehraufwand bei der Administration mit sich bringen.

5 Änderungsübersicht

Dieses Kapitel soll kurz aufzeigen, welche Änderungen in das Programm einflossen, und welche zusätzlichen Features neu implementiert wurden. Ist bereits eine Testumgebung vorhanden, kann diese auch in Zukunft weiter verwendet werden. Die einzig notwendige Änderung gegenüber der älteren Version ist das Initialisieren des Zufallsgeneratores in den Skripten für den Generator.

5.1 Das Programm

5.1.1 Randomgenerator

Der Randomgenerator welcher verwendet wird, um bei der Paketgenerierung die Parameter zufällig zu gestalten um einen zufälligen Paketstrom zu simulieren, ist komplett überarbeitet worden. Dabei gelang es die Implementation des Zufallsalgorithmusses von der Programmlogik abzukoppeln, sodass künftige Änderungen ohne grösseren Aufwand getätigt werden könnten.

5.1.2 Lastverteilung

Neu kann das Datenaufkommen, welches die Generatoren auf das Netzwerk abgeben, vorgegeben werden. Dabei wurde eine möglichst gute Annäherung gewählt, weil eine exakte Simulation einer bestimmten Last unter den gegebenen Umständen nicht möglich ist.

5.2 Die Skriptsprache

5.2.1 Neue Befehle

Die Skriptsprache für den Generator hat sich in zwei Punkten verändert. Neu muss der Init−Befehl nur noch mit einem Wert `z.B. init (80)` aufgerufen werden. Es reicht, wenn dieser pro Skript einmal initialisiert wird. Die zweite Änderung betrifft die Steuerung der Lastverteilung, welche neu mit dem Befehl `speed (Mbit/s)` gesteuert werden kann.

5.2.2 Automatische Generierung von Skripten

Neu können Umgebungsparameter für die Skripte festgelegt werden, sodass die Generator−Skripte automatisch erstellt werden. Dabei wird zwischen config und template Files unterschieden. In den config Files werden die Netzwerkparameter sowie der grobe Ablauf definiert, und in den template−Files der Aufbau der einzelnen Pakete beschrieben.

5.3 Java Remote Controll

5.3.1 Server

Um eine Loslösung von der Linux−Kommandozeile zu erreichen, ist neu ein Java GUI verfügbar, welches die wichtigsten Tätigkeiten rund um den Generator abdeckt. Auf dem Generator muss dazu ein Java Server gestartet werden. Dieser übernimmt die komplette Kommunikation mit dem Client auf dem entfernten System und steuert den Generator mittels spezieller Shell Skripte. Um die Übersicht nicht zu verlieren, was auf den einzelnenen Generator läuft, wird der Output des Generatores in ein Logfile umgeleitet. Um dieses Logfile jedoch nicht zu gross werden zu lassen, wird der Gernerator über die neu implementierte Option `−nooutput` aufgerufen. So werden nur die wichtigsten Meldungen in die Logfiles gespeichert.

5.3.2 Client

Über den Client welcher das Graphische User Interface implementiert, kann der Generator über eine beliebige TCP/IP Verbindung gesteuert werden. Weiter ist es auch möglich, bis zu 50 unterschiedliche Generatoren von einem zentralen Punkt aus zu steuern. Das GUI kann sowohl unter Windows als auch unter Linux in Betrieb genommen werden. Da die Logfiles auf dem Server im Unix Stile abgespeichert werden, gibt es Darstellungsprobleme unter den Windows Texteditoren.

6 Grundeinstellungen

6.1 Kernel Einstellungen

Die Vorgehensweise für die Inbetriebnahme der Software wird bereits ausführlich in der Diplom−arbeit [DA04] abgehandelt. Da sich diese Installationsanleitung jedoch noch auf den Linux Kernel 2.2.x bezieht, heute aber der neue Kernel 2.4.x aktuell ist, entschlossen wir uns eine neue Installa−tionsanleitung für den Kernel 2.4.x anzufertigen:

Als root einloggen
Verzeichnis wechseln: **`cd /usr/src/linux`**
Kernel−Menu öffnen: **`make menuconfig`**

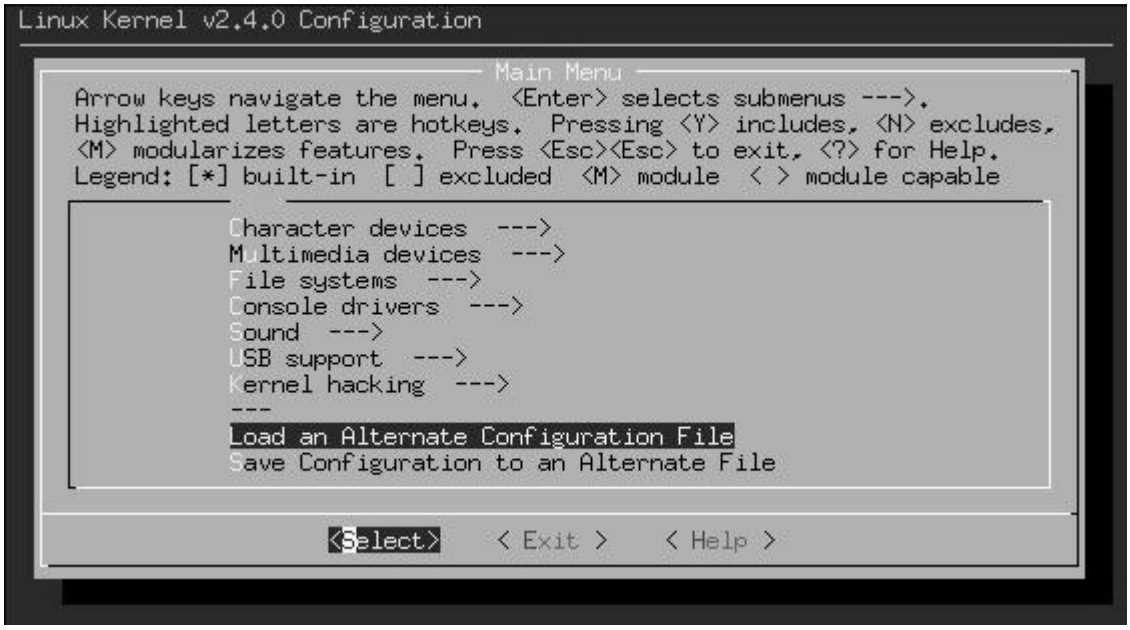


Abbildung 6.1 − Linux Kernel Konfiguration 2.4.0

Das Konfigurations−File mit den korrekten Einstellungen für den Linux Kernel 2.4.x liegt auf der CD im Verzeichnis `/etc/kerneleinstellungen.txt`. Diese Einstellungen werden unter anderem benötigt, um auch bei diesem Kernel mit IPChains als Firewallsoftware zu arbeiten.

Wichtig dabei ist, dass folgende Komponenten in den Kernel eingebunden werden:

- Networking Options → Packet Socket
- Networking Options → Network Packet Filtering
- Networking Options → TCP/IP Networking
- Networking Options → IP:Netfilter Configuration → IPChains (2.2−style) support

Weiter muss die korrekte Ethernetkartenunterstützung sowie der „dummy device“ im Kernel aktiviert werden. Nach dem Verlassen dieser Maske müssen folgende Befehle eingegeben werden, um den Kernel zu compilieren.

`make dep clean bzImage`
`make modules`
`make modules_install`
`make bzlilo`

Falls Fehlermeldung erscheint, dass kein lilo.conf gefunden wurde, muss der Lilo (z.B. im Yast) konfiguriert werden und auf der Konsole mit dem Befehl **`lilo`** installiert werden.

`make clean`

6.2 Java Installation

Sollen die Paketgeneratoren über das Remote Control Tool ferngesteuert werden, so müssen folgende Punkte zur Installation der benötigten Java Komponenten durchgeführt werden.

1. **Installation von JDK1.3 (Anleitung für SuSE Linux)**
   Mit dem Yast / Yast2 muss das Paket java2 (Java Development Kit 1.3) aus der Kategorie „pay“ installiert und das Paket JDK1.18 aus der Kategorie „d“ deinstalliert werden.

2. **Einen neuen Link auf das installierte jdk 1.3 setzen**
   Als root einloggen
   Wechseln des Verzeichnisses: **`cd /usr/lib`**
   Löschen des alten Links: **`rm java`**
   Setzen des neuen Links: **`ln -s jdk1.3 java`**
   Wieder als User einloggen
   Kontrolle des neuen Links: **`java -version`**

```
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.0)
Classic VM (build 1.3.0, J2RE 1.3.0 IBM build cx130-20001025
(JIT enabled:jitc))
```

3. **Pfad zu den Java-Binaries setzen.**
   Ausführen des export Befehles **`export PATH=$PATH:/usr/lib/jdk1.3/bin`**

6.3 Generator Einstellungen

Folgende Punkte dienen dazu, das Netzwerk in Betrieb zu nehmen und zu Überprüfen.

- Als root einloggen
- Auf der CD ins Verzeichnis `server` wechseln
- von der CD entpacken: **`tar -xvzf simulator.tar.gz`**

- Localhost pingen: **`ping localhost`**

  Falls nachfolgende Ausgabe erscheint:

```
PING localhost (127.0.0.1): 56 data bytes
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
```

  konfigurieren von Localhost: **`ifconfig lo 127.0.0.1`**
  oder standardmässig in `/etc/rc.config` einfügen: **`Start_Loopback="yes"`**

- Telnet auf selben PC: **`Telnet localhost`**

- Telnet von Remote auf Generator: **`Telnet [IP_von_Generator]`**

  Falls nachfolgende Ausgabe erscheint ... :

```
telnet localhost
Trying localhost...
telnet: connect to address localhost: Keine Route zum Ziel-
rechner
```

  ...ist wahrscheinlich im Netzwerk kein DNS-Server verfügbar. Dann müssen die IP-Adressen und ihre Hostnamen manuell in die Datei `/etc/hosts` eingetragen werden.

```
127.0.0.1       localhost
10.0.0.1        generator
10.0.0.2        senke
10.0.0.3        client
```

- Wichtig ist, dass die Routen zu den Zielhosts auf dem System korrekt gesetzt werden.

6.4 JBuilder Einstellungen

Die Installation des JBuilders ist für den Betrieb der Software nicht notwendig. Für weitere Entwicklungen am GUI oder am Java-Server, kann das bereits vorhandene JBuilder Projekt verwendet werden. Es ist im Verzeichnis `/Projects/JBuilder`.

6.5 Compilieren des Java-Sourcecodes

Nach dem Compilieren des Java-Codes, muss darauf geachtet werden, dass die beiden Klassen `ServerControlImpl_Skel.class` und `ServerControlImpl_Stub.class` frisch erstellt werden. Diese werden für die Kommunikation über RMI verwendet. Um neu zu compilieren ins Verzeichnis Java wechseln und dort folgenden Befehl eingeben:

```
rmic -classpath . Ipnetsim.ServerControlImpl
```

6.6 Starten des Programms

Server	**Client**
cd java killall rmiregistry rmiregistry& java ipnetsim.Server	cd java java ipnetsim.GUIFrame

Achtung: **RMIRegistry in der gleichen Konsole starten wie der Server.**
Script für den Generator muss am Ende der Datei mindestens ein `\n` haben.
Kommentare im Generator−Skript müssen nach // immer einen Leerschlag haben

7 Zufallsgenerator

Als eine erste und sehr umfangreiche Aufgabe stellte für uns die komplette Überarbeitung des Zufallsgeneratores dar. In der bisherigen Implementation musste für jedes Feld, welches zufällige Werte annehmen konnte, mit dem Skriptbefehl `init (x y z)` ein neuer Zufallsgenerator initialisiert werden.

Beispiel:

```
init (125 12 3) (125 12 3)
icmp 192.168.111.* 111.111.111.* 3
init (3 12 3)
icmp 192.168.111.1 111.111.111.222 *
init (3 5 3)
icmp 192.168.1.2 10.10.10.11 3 -code *
init (6 3 1)
```

Einerseits war dies bei umfangreichen Skripten sehr mühsam und andererseits waren die Pseudozufallsfolgen stark abhängig von den verwendeten Startwerten. Demzufolge war die Verteilung der Zufallswerte von den Angaben des Benutzers abhängig, was eigentlich nicht sein sollte. Unserer Meinung nach wurde der verwendete Zufallsalgorithmus nicht korrekt implementiert, was oft zu kurzen Zahlenfolgen führte (Kapitel 7.2).

7.1 Echte und Pseudozufallsfolgen

Bei den Methoden zur Erzeugung von Zufallszahlen kann man zwei grosse Klassen unterscheiden:

- Generatoren (generieren echte Zufallszahlen)
- Pseudozufallsgeneratoren

Generatoren für echte Zufallszahlen sind z.B. Würfel oder der Zerfall von radioaktiven Substanzen. Diese Werte sind im Normalfall nicht vorhersagbar, erfüllen aber die notwendigen statistischen Eigenschaften. Jedoch reichen solche Generatoren meist nicht aus, wenn innerhalb von kurzer Zeit viele Werte benötigt werden.

Aus diesem Grund wird versucht, Zufallszahlen durch die Verwendung von mathematischen Algorithmen zu bestimmen. Zur Erzeugung von Zufallszahlen mit Hilfe von Algorithmen bedarf es bei der Verwendung eines Computers nur kurzer Generierungszeiten und einer geringen Speicherkapazität. Auch sind die so erzeugten Zufallszahlen leicht reproduzierbar. Dadurch kann im Fehlerfall der Vorgang exakt simuliert werden. Die mathematische Erzeugungsmethode ist allerdings kein echter Zufallsprozess, man spricht daher auch von Pseudozufallszahlen, da die von ihr gelieferte Zufallszahlenfolge nach einem bestimmten Zyklus wieder ihren Ausgangswert erreicht.

Dass sich die Zufallsfolge wiederholt, ist nicht von Bedeutung, wenn die Folge lang genug ist. Dies ist der Fall, wenn das Experiment abgebrochen wird, bevor die gesamte Folge abgearbeitet wurde, oder wenn z.B. in einem Gerät weniger Werte zwischengespeichert werden können, als die Folge Werte liefern kann.

7.2 Lineare Kongruenzmethode

Diese Methode zur Berechnung von Pseudozufallsfolgen beruht auf dem Vorschlag von D.H. Lehmer aus dem Jahre 1951[Logi] und wird heute noch verwendet. Der rekursive Algorithmus berechnet Zufallszahlen im Bereich [0,m[mit Hilfe folgender Formel:

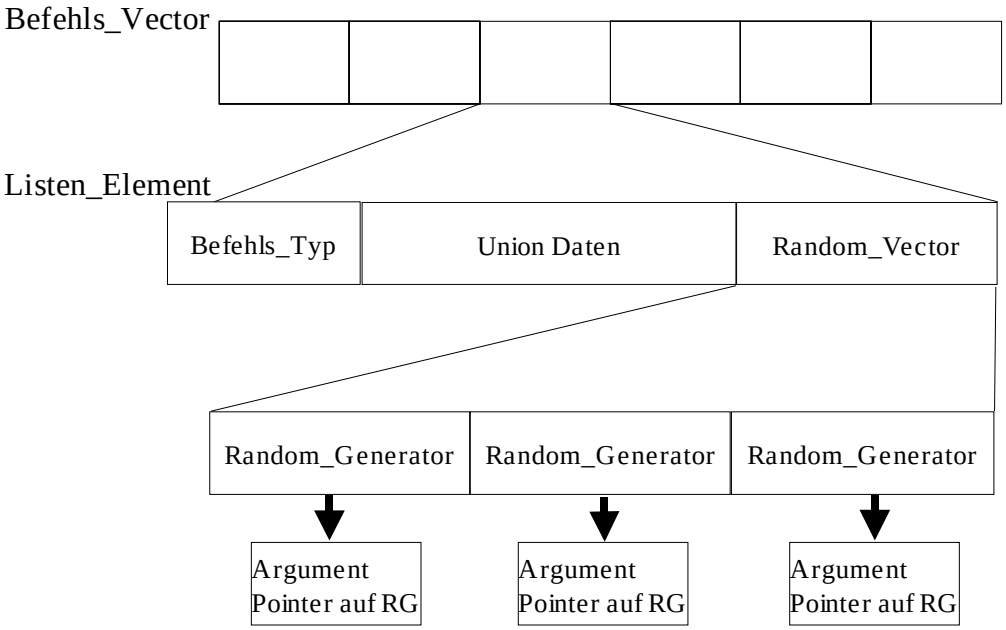
$$z_{i+1} = (a * z_i + b) \bmod m$$

Formel 7.1

In der bisherigen Implementation musste für jeden Zufallsbereich eine neue Pseudozufallsfolge mit den Werten a,b und m initialisiert werden. Es ist bei diesem Algorithmus aber sehr wichtig, dass für diese Werte geeignete Zahlenkombinationen verwendet werden. Dies ist ohne Tabellen aus einschlägiger mathematischer Literatur jedoch nicht garantiert. Weiter reicht im Normalfall ein einmaliges Bestimmen dieser drei Werte, da verschiedene Zufallszahlenfolgen durch unterschied−liche Vorgaben für den Startwert generiert werden können. Diese Methode ist nicht für alle Anwendungen geeignet, da es relativ einfach möglich ist, nach Abfangen von 3 folgenden Werten den weiteren Ablauf zu bestimmen. Eine Eigenschaft, die z.B. in der Kryptographie nicht gebraucht werden kann.

7.3 Bisherige Implementation

Bisher funktionierte das Auswählen von Zufallszahlen folgendermassen: Zuerst wurde im soge−nannten Befehls_Vektor eine Anzahl Listen_Elemente verwaltet, welche den Befehls−Typ, Daten und einen Random_Vector enthielten. In diesem Random_Vector wurde für jeden Parameter wel−cher eine Zufallszahl benötigt, ein eigener Random_Generator erstellt. Auf diese Weise wurden sehr schnell etliche neue Random_Generatoren erzeugt, welche alle die gleiche Aufgabe erledigten. Aus diesem Grund war es notwendig, so viele Random_Generatoren zu initialisieren wie Parameter mit Zufallszahlen im Skript vorhanden waren. Durch den Befehl `init (x y z)` wurden diese Parameter folgendermassen zugewiesen: a = x, b = y, m = z.



Zeichnung 7.1 − Bisherige Implementation des Random Generatores

7.4 Neue Implementation

Neu werden die Werte für a,b und m direkt im Quellcode im File `random.cpp` bestimmt. Bei den von uns verwendeten Zahlen: a = 24'298, b = 99'991 und m = 199'017 handelt es sich um die gleichen Zahlen, welche auch im Taschenrechner TI−59 von Texas Instruments verwendet werden. Die Periodenlänge dieses Zufallsalgorithmusses ist gleich m = 199'017. Da dieser Algorithmus immer Zahlen zwischen 0 und m liefert, müssen wir diese jeweils noch in einen gültigen Wert zwischen *min* und *max* abbilden. Dies geschieht mit der Formel:

$$temp = min + (max + 1 - min) * temp$$

Formel 7.2 − Zufallswerte zwischen min und max

Wichtig dabei ist, dass der `max`−Wert um eins höher angegeben wird als er wirklich ist, da später der float−Wert `temp` mit einer Typenumwandlung zu einem int−Wert konvertiert wird. Bei dieser Konvertierung wird immer nur der Ganzzahlbereich des float Wertes beachtet. Das führt dazu, dass die Zahl immer abgerundet wird.

```
// Funktion next_value berechnet nächsten Random-Wert. Aufruf min und max
unsigned long int random_geni::next_value(unsigned long int min, unsigned long
int max)
{
     unsigned long int wert;
     unsigned long int z;
     float temp;
     z = (a * z0 +b) % m;             //nächste Zufallszahl zwischen 0 und m
     z0=z;                            //1 mal initialiseren, danach wird Zahl
                                      //für die nächste Zufallszahl übernommen
     temp=1.0*z/m;                    //Zufallswert zwischen 0 und 1
     temp = min + (max+1-min) *temp;  // Zufallszahl zwischen max
                                      //und min max + 1, da typecast float to
                                      //int immer abrundet
     wert = (unsigned long int)temp   // float to int
     return wert;
};
```

Somit reduziert sich der Aufruf des Init−Befehls im Skript auf einen Parameter. Falls erwünscht, kann dieser beliebig oft neu initialisiert werden. Nach dem ersten Initialisieren liefert uns nun die Funktion
`random_geni::next_value(min, max)`
immer einen zufälligen Wert im Bereich von *min − max*.

Um die korrekte Funktionsweise des Randomgeneratores zu testen, haben wir den Quellcode so abgeändert, dass uns alle Zufallswerte am Bildschirm ausgegeben werden. Danach haben wir den Simulator mit folgendem Skriptbefehl gestartet und den gesamten Output während einiger Sekunden in das File `/etc/randomwerte.txt`, welches auch auf der CD beiliegt, abgespei−chert.

```
speed (0)
init (394)
loop endless
   loop 2000
         ip 10.0.0.* 10.0.0.* -ihl *1 -tosp * -tosd * -tost * -tosr * -id *
```

```
        -df * -mf * -fo * -ttl * -prot * -data *
   endloop
endloop
```

Diese Daten haben wir im Programm StarCalc von StarOffice importiert und ausgewertet. Leider konnte StarCalc nur 32’000 Werte verarbeiten. Um jedoch einen Überblick über die Verteilung der Werte zu bekommen, reichen diese 32’000 Werte vollends aus. Wir überprüften die Verteilung bei den Feldern: Destination & Source (0−255), ihl (5−15), tosd (0−1) und id (1−65’535).

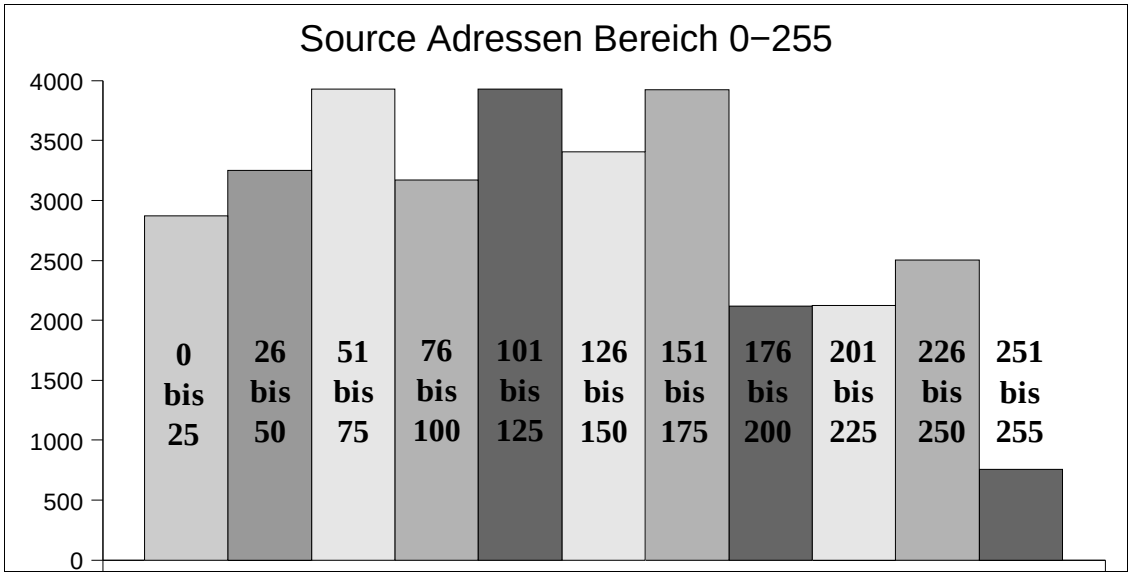


Diagramm 7.1 − Verteilung der Source Adressen aufgeteilt in 11 Gruppen à 25 Werte

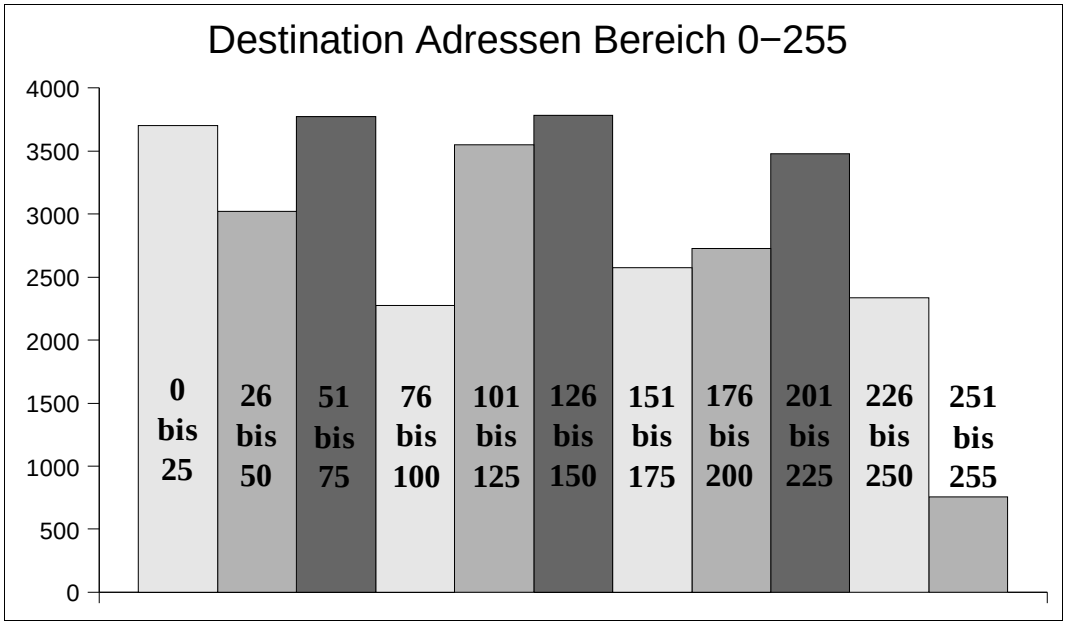


Diagramm 7.2 − Verteilung der Destination Adressen aufgeteilt in 11 Gruppen à 25 Werte

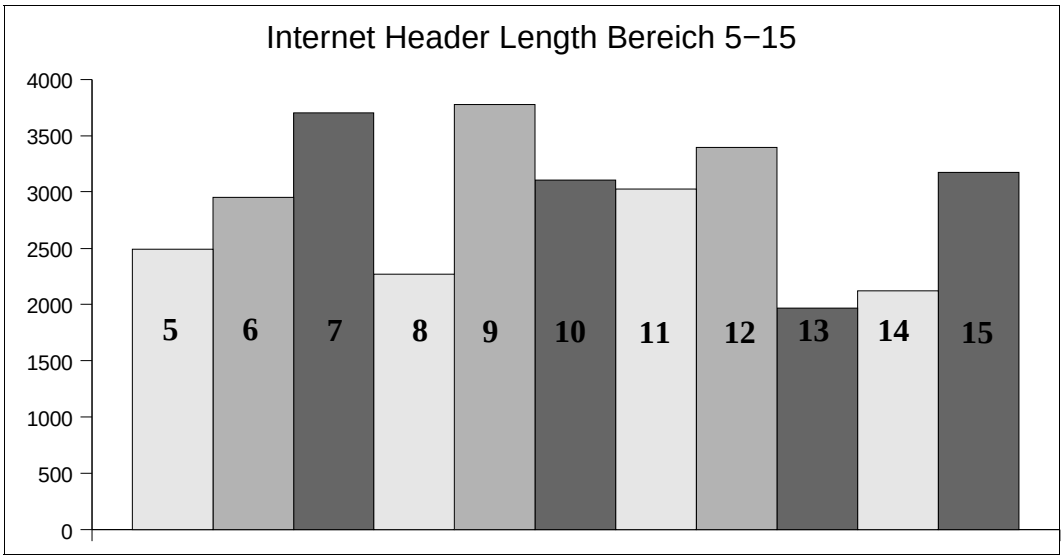


Diagramm 7.3 − Verteilung der Internet Header Length aufgeteilt in 11 Gruppen

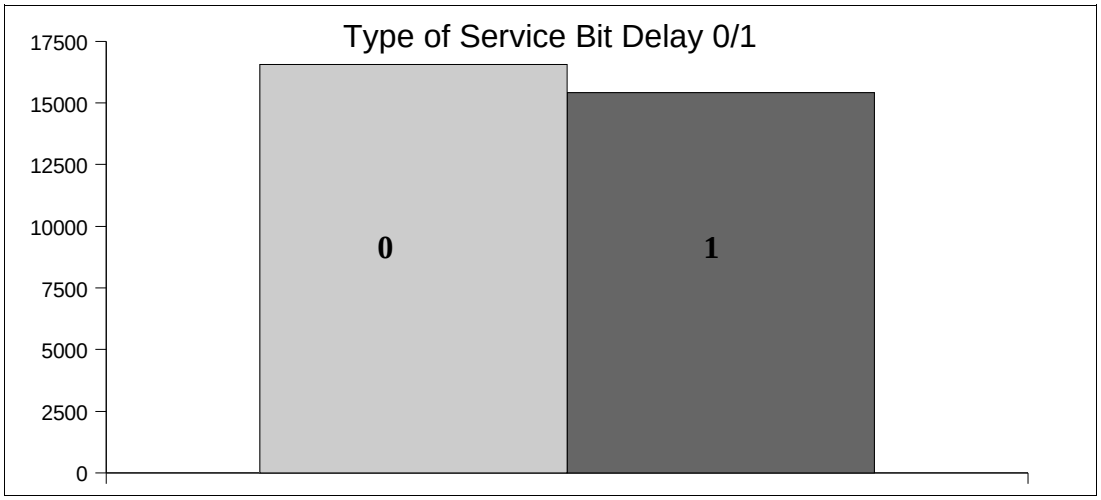


Diagramm 7.4 − Verteilung der Type of Service Bit Delay in 2 Gruppen

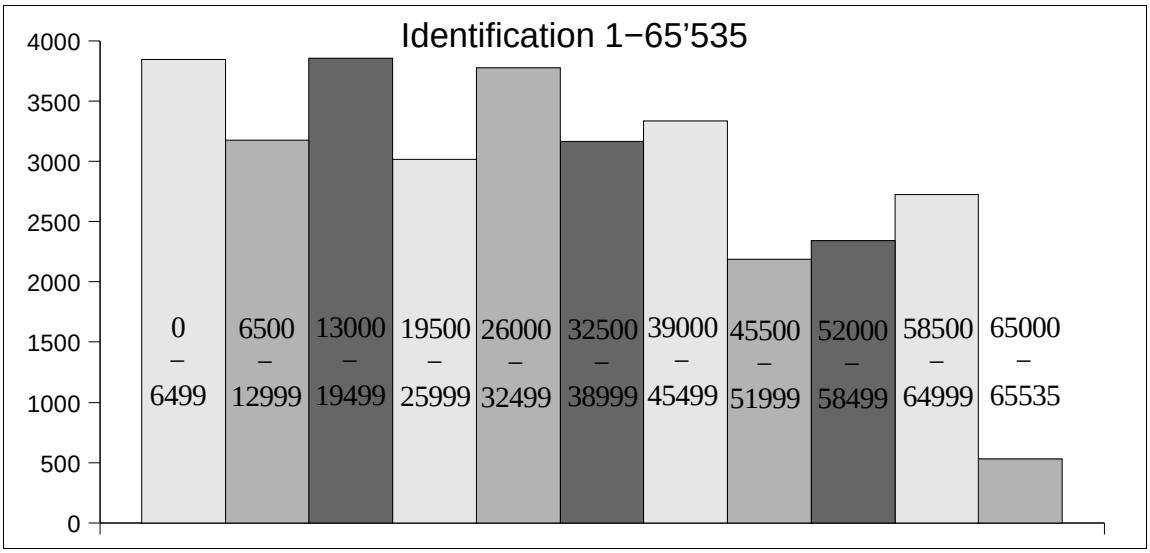


Diagramm 7.5 − Verteilung der Identification aufgeteilt in 11 Gruppen a 6500 Werte

Wie die oben aufgeführten Diagramme zeigen, sind keine Ausreisser in einem bestimmten Bereich festzustellen, und dies unabhängig davon, wie gross der Range zwischen `min` und `max` gewählt wurde. So zeigt sich besonders beim Diagramm 7.4, dass die Verteilung zwischen 0 und 1 beinahe gleich ist. Hierbei ist zu beachten, dass für die Erstellung der oben aufgeführten Diagramme nur die ersten 32’000 Werte von den insgesamt fast 200’000 möglichen Werten ausgewertet wurden.

Durch die vollständige Abkopplung vom Random−Generator und den Parametern für die Netz−werkpakete, ist in Zukunft die einfache Ersetzung der Random−Logik garantiert. Soll z.B. anstatt der Gleichverteilung der Werte ein Random−Generator verwendet werden, welcher Werte gleich der Standard−Normalverteilung liefert, so kann dies durch die Neuimplementierung der Random−logik in sehr kurzer Zeit realisiert werden. Ein weiterer Pluspunkt der neuen Implementation ist, dass bei einer allfälligen Erweiterung des Befehlssatzes für den Generator keine speziellen Aufrufe des Random Generatores gemacht werden müssen. Es reicht in jedem Fall ein Aufruf der Random Funktion mit dem Werten für `max` und `min`.

7.5 Änderungen im c++ Source Code

Das Feld Random−Vector (siehe Zeichnung 7.1) wurde ersatzlos gestrichen. Der Aufruf des Random−Generators geschieht neu im `simulator.cpp`, an der Stelle, wo es eine Zufallszahl braucht. Das Programm `parser.cpp` beschränkt sich neu nur auf das Auslesen des Skripts und nicht mehr um die Erstellung der Random−Generatoren.

8 Lastverteilung

Eine weitere unangenehme Eigenschaft des Paketgeneratores war es, dass er die Pakete immer mit der Geschwindigkeit an das Netzwerk weiterleitete, wie der zu verarbeitende Rechner in der Lage war diese zu generieren. So war es nicht möglich, z.B. für das eine Netzwerk einen hohen Durchsatz zu simulieren, während von einem anderen Netzwerk ein relativ geringer Durchsatz simuliert werden sollte. In diesem Fall füllten einfach beide Computer das Netzwerk und den Gateway mit so vielen Paketen wie es ihnen möglich war zu schicken. Aus diesem Grund erweiterten wir die Skriptsprache um einen weiteren Befehl. Mit dem Befehl `speed (Anzahl Mbit/s)`ist es nun möglich, das gewünschte Verkehrsaufkommen auf dem Netzwerk vorzugeben. Wird mit `speed (0)` initialisiert, werden die Pakete mit der maximalen Verarbeitungsgeschwindigkeit des jeweiligen Computers an das Netzwerk abgegeben.

8.1 Realisierung

Ursprünglich wollten wir das gewünschte Datenaufkommen auf dem Netzwerk dadurch erreichen, dass wir die Grösse des soeben versandten Paketes erfassten, und anhand dieser Grösse die Verzögerung bis zum Versenden des nächsten Paketes berechneten. Diese Berechung vernachlässigt allerdings die Zeit für das Generieren des jeweiligen Paketes, problematisch insbesondere für höhere Durchsatzraten und Pakete mit mehreren Zufallszahlen. Ein weiteres Problem stellen die kurzen Zeiten dar, welche zwischen den Paketen abgewartet werden müssen. Diese Zeiten können schnell Werte im unteren µs−Bereich annehmen, welche ohne ein Realtime−fähiges Linux nicht sauber aufgelöst werden können. Der normale Timer unter Linux bringt z.B. eine Auflösung von 10ms.

Berechnung der Verzögerung für eine Datenrate von 10MBit/s und einer Paketgrösse von durchschnittlich 200Bytes:

$$\frac{10MBit/s * 1'000'000}{(200*8)} = 6250\,Pakete/s$$

$$\frac{1}{6250}\,Pakete/s = 0{,}00016s$$

Formel 8.1 − Verzögerung zwischen den Paketen

Weiter muss die Zeit die für das Berechnen und Versenden des Paketes benötigt wird von der oben berechneten Zeit abgezogen werden. Es ergeben sich schnell Werte unter 100µs. Deshalb kamen wir von der Idee ab, für jedes Paket eine individuelle Verzögerung zu berechnen und entschlossen uns, die Verzögerung zwischen den Paketen für ein bestimmtes Intervall konstant zu halten. Um Probleme mit dem Timer auszuschliessen, entschieden wir uns die Verzögerung der Pakete durch eine dynamische for−Schleife zu realisieren. Beim Versenden eines Paketes wird immer die Länge von diesem aufaddiert. Die Länge der Pakete ermitteln wir in der Funktion `send_to` im File `send_packet.cpp` während diese auf den RAW Socket übergeben werden.

```
// Paket verschicken
if (sendto(sd, data, data_len, 0, (struct sockaddr *) &sa_desti-
nation, sizeof(struct sockaddr_in)) != -1 )
{
  Datasend += data_len+14;      //+14 da wir noch den Header des
                                //Packetes zu berücksichtigen haben
  sent = true;
}
```

Da im Integer `data_len` nur die Länge der Nutzdaten der Pakete angegeben wird, müssen noch zusätzliche 14 Bytes für den Header aufaddiert werden. Diesen Wert konnten wir mit dem Netz-werkanalysetool Ethereal ermitteln. Alle 0.1s wird vom Timer das Siganl SIG_ALRM ausgelöst. Dieses Signal wird im von uns erstellten Signalhandler im File `simulator.cpp` abgefangen. In diesem Signalhandler wird die Summe der versendeten Daten ausgewertet und die Summe wieder auf 0 zurückgesetzt. Je nachdem ob die gewünschte Datenrate erreicht wurde oder nicht, wird der Wert für die for-Schleifen korrigiert und das Signal SIG_ALRM wieder aktiviert. Am Anfang wird der Wert für die for Schleifen auf 0 initialisiert. Dadurch wird das Netzwerk am Anfang für eine kurze Zeit voll belastet. Dies hat den Vorteil, dass wenn keine Datenrate vorgegeben ist, der Computer mit voller Geschwindigkeit zu senden beginnt. Folgende for-Schleife ist im File `simulator.cpp` zweimal eingefügt worden. Einmal in der normalen Simulator Schleife und einmal in der loop Schleife. Immer umittelbar vor dem Versenden der Pakete. Ist die Datenrate auf 0 Mbit/s gesetzt, oder handelt es sich um einen Steuerbefehl wie z.B. endloop, wird sofort das nächste Paket generiert.

```
if (itr->befehl != loop && itr->befehl != endloop && DatenrateMbit
    != 0 && itr->befehl != init && itr->befehl != speed)
{
  for (int Schleifeaussen=0; Schleifeaussen<Wartekonstante;
        Schleifeaussen++)
  {
        for (int Schleifeinnen=0; Schleifeinnen<Wartekonstante;
             Schleifeinnen++)
        {}
  }
}
```

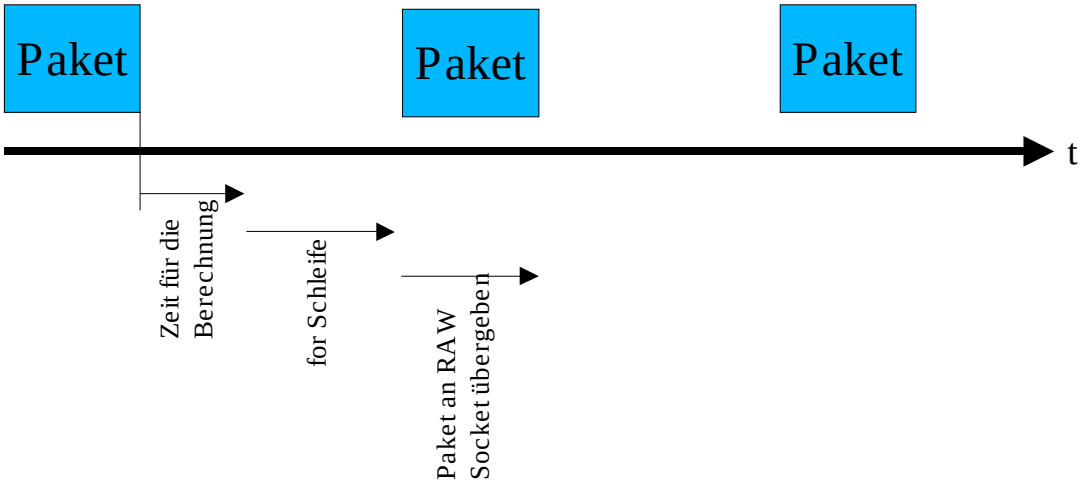
Da wir zwei in sich verschachtelte for Schleifen verwenden, welche beide durch die gleiche War-tekonstante beeinflusst werden, ist die Änderung der Wartezeit nicht linear zur Änderung der Wartekonstante. Diese beiden Werte haben eine quadratische Abhängigkeit. Wird die Wartekon-stante verdoppelt, erhöht sich die Wartezeit um das Vierfache. Dadurch wird eine schnelle Angleichung der Datenrate an den Soll-Wert auch dann erreicht, wenn diese weit auseinander-liegen. Bei der Berechnung für die Wartekonstante bis zum nächsten Signal, wird zwischen 8 möglichen Einflussfaktoren für die Wartekonstante unterschieden. Dabei wird je nach Abweichung zwischen Soll- und Istwert die Wartekonstante mehr oder weniger beeinflusst. Dabei muss beachtet werden, dass die Wartekonstante nicht beliebig reduziert werden kann, da sonst plötzlich ein nega-tiver Wert oder ein Überlauf auftreten kann. Wird eine Datenrate vorgegeben, welche vom Gene-rator nicht erreicht werden kann, so wird die Wartekonstante bald 0 sein und dort bleiben.

Folgende Änderungen der Wartekonstante werden im Signalhandler alle 0.1s durchgeführt.

Abweichung Istwert/Sollwert	Wartekonstante
< 40%	−100
40 − 80%	−30
80% − 95%	−3
95% − 100%	−1
100% − 105%	+1
105% − 120%	+3
120% − 160%	+30
160,00%	+100

Tabelle 8.1 − Änderung der Wartekonstante

So erreichen wir eine komplette Loslösung von einem exakten Timer und sind nicht von der Verarbeitungsgeschwindigkeit des jeweiligen Computers abhängig, da sich die Verzögerung jeweils dynamisch anpasst. Nachteilig hingegen ist, dass zwischen allen Paketen während 0.1s der gleiche Abstand besteht. So wird gewährleistet, dass über den gesamten Zeitraum Daten auf die Leitung gesendet werden. Besser wäre, die Länge jedes einzelnen Paketes in die Berechnung einzubeziehen. Dies dürfte aber mit heutigen Computern und einer Standard−Linuxinstallation nicht sauber zu realisieren sein. Allenfalls wäre hier der Einsatz von Realtime Linux eine Alternative. Jedoch stellt sich auch dort das Problem, dass die Berechnung der Verzögerung in extrem kleinen Zeitintervallen vonstatten gehen müsste. Um ein normales Verkehrsaufkommen einer bestimmten Bandbreite zu simulieren, dürfte die von uns implementierte Variante vollends genügen. In einer realen Umgebung dürften die Verzögerungen zwischen den einzelnen Paketen ebenfalls nicht absolut korrekt stimmen.



Zeichnung 8.1 − Einfluss der for Schleife auf den Paketversand

Da wir exakt x*1’000’000 Bits/s inkl. Header auf das Netzwerk senden, kann es sein, dass sich je nach Überwachungstool für das Netzwerk, einen tieferen Wert anzeigt. Dies weil in gewissen Tools die Datenrate in MBit/s angezeigt wird. Dabei wird das Datenaufkommen zweimal mit 1024 divi−diert. Ebenfalls Abweichungen können sich ergeben, wenn vom Überwachungstool nur die Daten und nicht das ganze Paket inkl. Header registriert wird. Jedoch sollte immer eine konstante Daten−rate angezeigt werden.

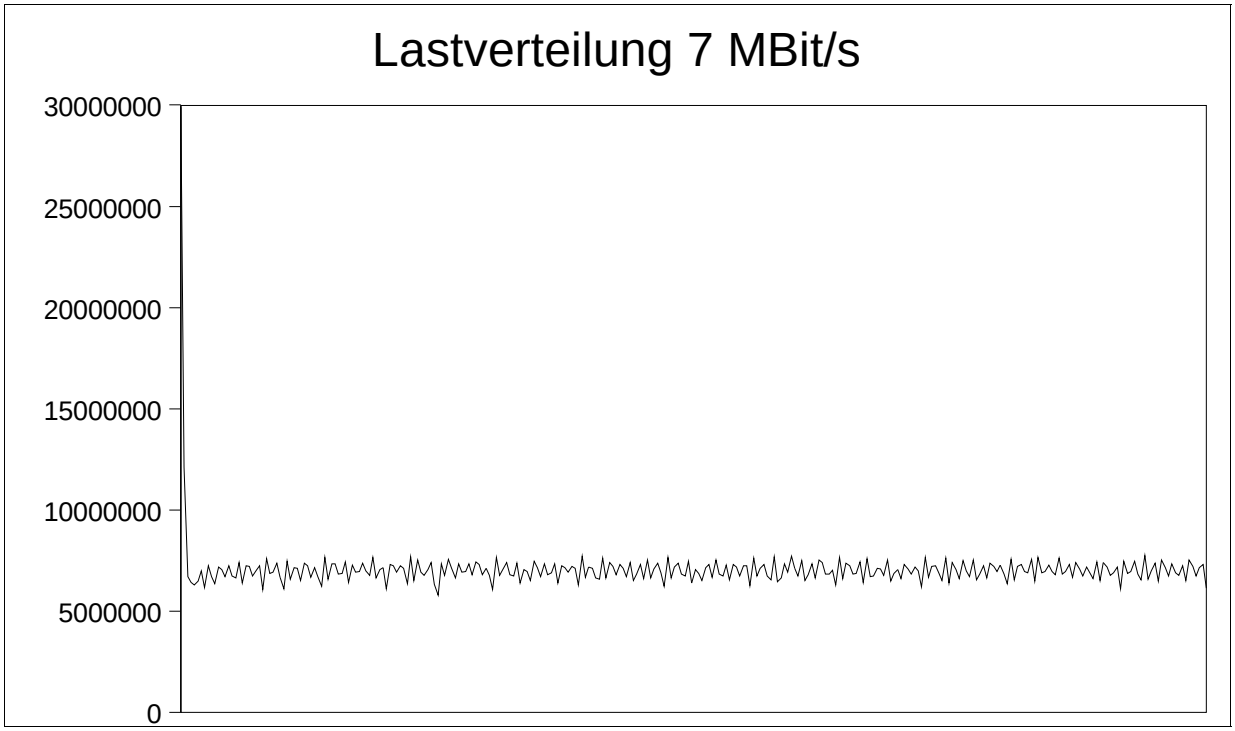


Diagramm 8.1 − Lastverteilung von 7 MBit/s während 300s

Wie im Diagramm 8.1 aufgezeigt, wird zuerst während einer kurzer Zeit mit maximaler Geschwindigkeit gesendet. Jedoch wird die Last sehr schnell auf den gewünschten Wert reduziert, wo sie sich dann auf die 7MBit/s ±5% einpendelt.

9 Remote−Control / GUI

Um eine möglichst realitätsnahe Netzwerkumgebung zu simulieren, werden oft etliche Computer als Generatoren eingesetzt. Um all diese Generatoren effizient und zentral zu verwalten, wird nun ein Graphisches User Interface zur Verfügung gestellt.

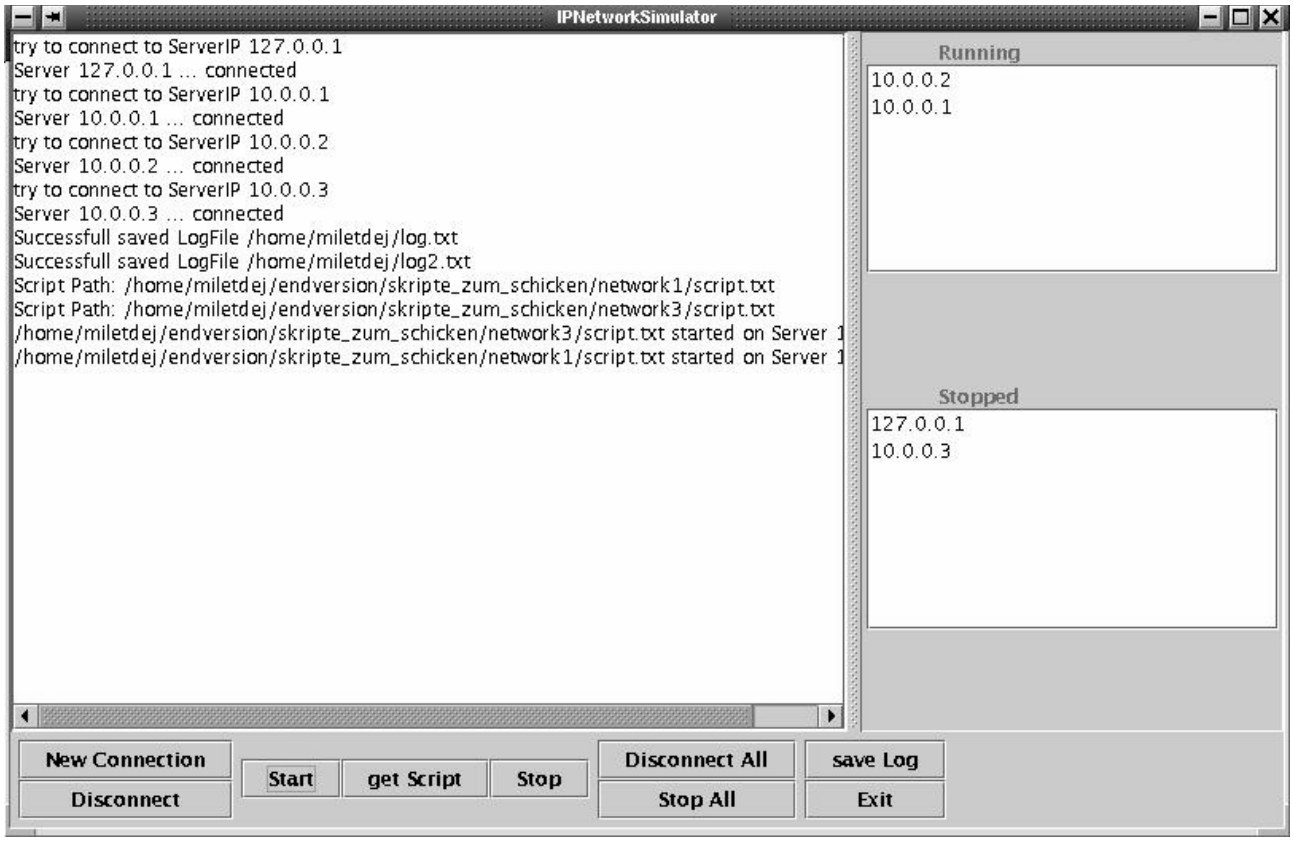


Abbildung 9.1 − GUI Output mit mehreren Verbindugen

Als Vorlage für die Fernbedienungssoftware diente uns die Projektarbeit [PA]. Ursprünglich sollte diese Software, so wie sie realisiert wurde, in die Diplomarbeit [DA04] eingebaut werden. Es stellte sich aber schon bald heraus, dass das gesamte GUI frisch entwickelt werden musste. Da unter Windows entwickelt und das Programmierte nie auf einer Linux−Maschine getestet wurde, mussten etliche Features überarbeitet, gelöscht oder frisch programmiert werden. Der Server−Teil konnte fast übernommen werden, und funktionierte bis auf ein wesentliches Detail gut, welches wir beim gründlichen Testen herausfanden. Das Übertragen des Files „`script.txt`“ vom Client auf den Server, funktionierte nur beim ersten Mal. Das heisst, wenn ein Server mehrere Male mit dem gleichen Skript gestartet werden sollte, so funktionierte dies nicht. Grund war ein BufferedReader der nicht sauber mit dem Inputstream vom Socket kommunizierte.
Auf Client Seite vereinfachten wir den Code wesentlich. Im Hauptprogramm „`GUIFrame.java`“ erstellten wir einen ActionListener, welcher alle Buttons des GUI's überwacht.

Anstatt des Balkendiagramms unserer Vorgänger, wurde ein TextPanel realisiert, auf das alle wesentlichen Vorgänge geschrieben werden. Der Benutzer hat so die Kontrolle und Übersicht über alle Generatoren. Wir benutzten keine Sonderklasse für die Ausgabe von Texten, sodass das GUI problemlos auf Windows sowie Linux lauffähig ist.
Des Weiteren wurde die Idee des „Speichern des Logfiles“ fertiggestellt, sodass der Benutzer die Möglichkeit hat, den Output des Servers in einem File abzuspeichern. Auf Wunsch des Industrie−partners, wird in dieses Logfile ein „ERROR“ geschrieben, falls ein Gateway nicht mehr erreichbar ist. Der Benutzer erhält sogleich eine Warnmeldung. So kann man auf einfache Weise die Erreichbarkeit der Gateways kontrollieren.
Die Anzahl möglicher Server−Verbindungen wurde von anfänglich 5 auf 50 erhöht. Mit jeder Serververbindung wird ein „Struct“ erstellt. Nachfolgend sind Unterschiede zwischen Java und C++ aufgelistet:

Java	C++

public class Book{
 	
public int nPages;
public double price
};
	struct Book{
	
	int nPages;
	
	double price;
	
	};

Java	C++

Book myBook;
myBook.nPages = 321;
myBook.price = 123.45;
	Book myBook = new Book();
	myBook.nPages = 321;
	myBook.price = 123.45;

Unsere Klasse mit dem Serverstruct sieht folgendermassen aus. Jede neue Server−Verbindung wird in ein solches struct abgebildet.

```
public class Serverstruct {

public String ip;            // IP des Servers z.B. 127.0.0.1
public boolean connect;      // ist Server connected ?
public String scriptpath;    // Pfad des Scripts
public String scriptname;    // Name des Scripts
public String logpath;       // LoggingPfad Output des Servers
public boolean scriptstarted;// Ist Script gestartet worden
public int free;             // Speicherplatz 0=frei,1=besetzt
public boolean error;       // hat es ErrorMeldung im logfile.log
}
```

9.1 Ablauf

1. New Connection Neue Verbindung zu einem Server erstellen (Kapitel 9.2.1).

2. get Script Skript auswählen, welches der Server abarbeiten soll (Kapitel 9.2.3).

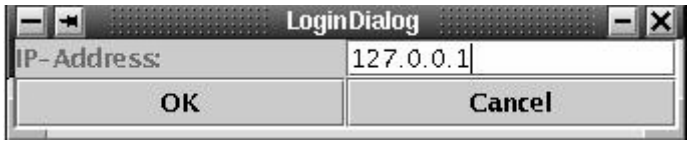
3. save Log File auswählen, in das Serverlogging Informationen geschrieben werden. Gateway Überwachung wird so automatisch gestartet (Kapitel 9.2.8).

4. Start Start dieses Skriptes (Kapitel 9.2.4).

9.2 Beschreibung des GUI

9.2.1 New Connection

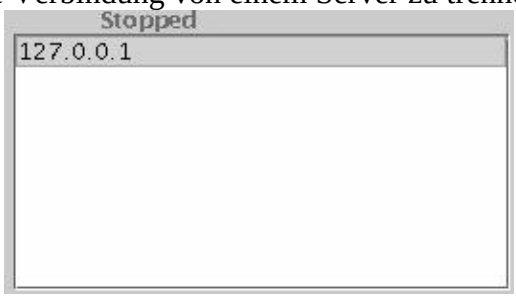
Um eine neue Verbindung zu einem Server zu erstellen, drückt man den Button New Connection und es erscheint das Login−Fenster



Hier gibt man die IP des Servers an, auf dem man ein Skript ausführen möchte.

9.2.2 Disconnect

Um eine Verbindung von einem Server zu trennen, markiert man die gewünschte IP im rechten Fenster

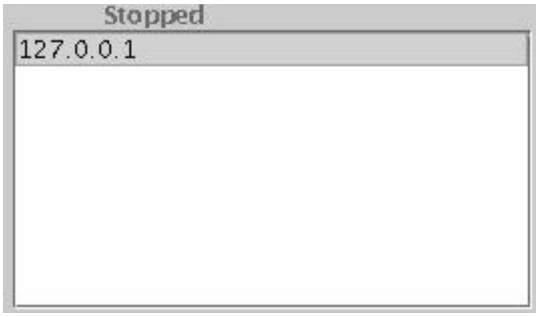


und drückt dann auf den Button Disconnect. Nach einer Bestätigung dass man die Verbindung wirklich trennen will, wird man von diesem Server getrennt. Zuerst wird, falls erforderlich, ein Skript zuerst gestoppt.

9.2.3 get Script

Falls noch kein Skript ausgewählt wurde, muss man den Button get Script drücken. Über den Button Start wird die ausgewählte Datei zum Server transferiert.

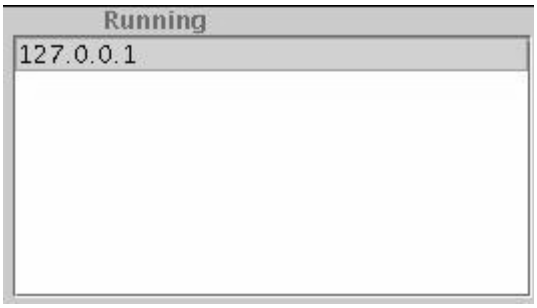
9.2.4 Start

Nachdem man eine Server−IP im rechten Feld 

ange−klickt und das Skript ausgewählt hat, kann man dieses Skript auf dem gewünschten Server durch Drücken des Buttons 

starten.

9.2.5 Stop

Nachdem man eine Server−IP im rechten Feld 

angeklickt hat, kann man dieses Skript auf dem gewünschten Server durch Drücken des Buttons 

stoppen.

9.2.6 Disconnect All

Nach dem Drücken des Buttons 

, werden alle Skripte zuerst gestoppt und dann die Verbindungen zu den Servern beendet.

9.2.7 Stop All

Mit dem Button 

werden alle laufenden Skripte auf den verschiedenen Servern gestoppt.

9.2.8 save Log

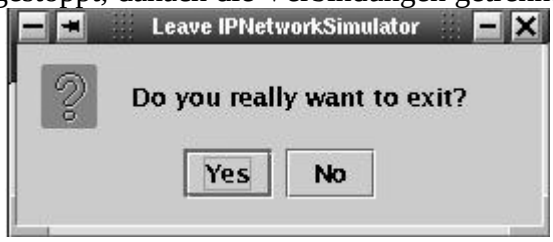
Falls der Server sein Skript fertig abgearbeitet hat, schreibt er in sein `logfile.log` das Wort ENDE. Dieses `logfile.log` wird zum Client transferiert und als Datei mit dem gewünschten Namen, welches vorher mit 

angegeben wurde, abgespeichert. Alle 10 Sekunden wird dieses File geöffnet und nach den Schlüsselwörtern ERROR und ENDE durchsucht. Findet er ENDE, wird dem Benutzer eine Meldung ausgegeben, dass der Server sein Skript abgearbeitet hat.

Findet er ERROR, wird dem Benutzer eine Meldung ausgegeben, dass der Server einen Gateway nicht mehr anpingen kann.

9.2.9 Exit

Beim Drücken des Buttons 

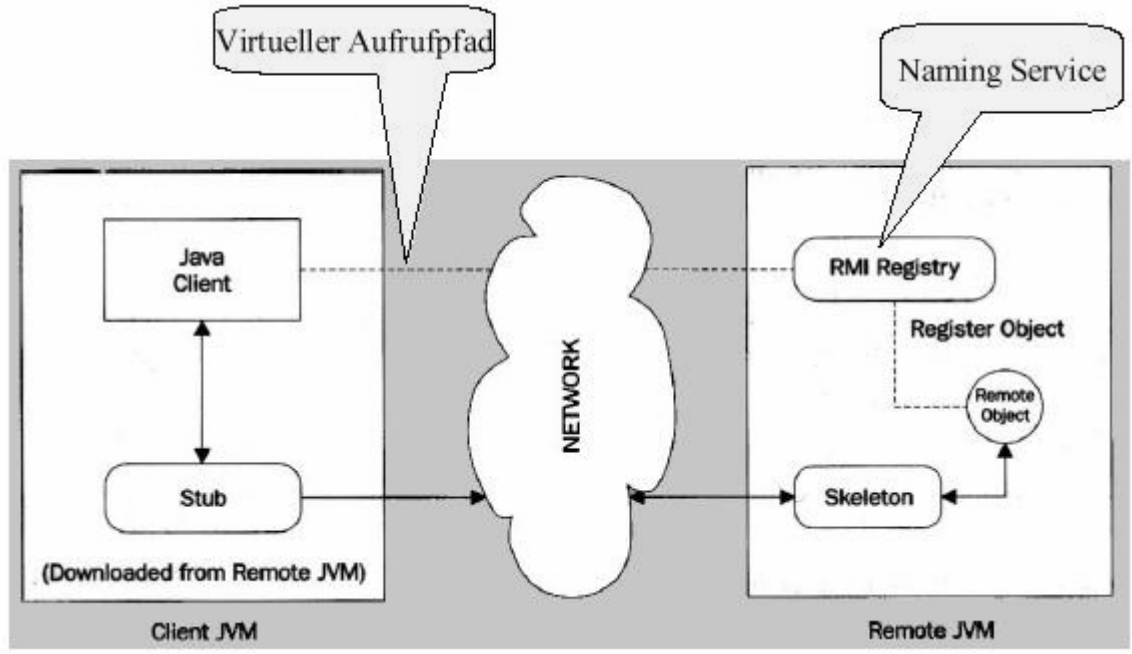
, oder dem Schliessen des Fensters, werden zuerst alle Skripte gestoppt, danach die Verbindungen getrennt, und dann das GUI nach der Bestätigung dieses Fensters 

geschlossen.

9.3 Aufbau und Funktionsweise des Remote Control

9.3.1 RMI

Die Kommunikation zwischen Client und Server basiert auf RMI (Remote Method Invocation).



Auf der Client Seite wird ein Interface mit Methoden definiert, das als Schnittstelle zwischen Client und Server dient. Es definiert das Verhalten des Servers. Die Implementation dieses Interfaces ist auf dem Server.

Der Server wird durch eine Namensauflösung gefunden. Ein einfacher Dienst ist bei Java die „rmiregistry“. Dieser Dienst läuft auf dem Server, der Services zur Verfügung stellt. Ein neuer Dienst instanziert zuerst ein lokales Objekt mit der Implementierung und registriert dieses bei der RMI Registry. Die Auflösung erfolgt mit lookup() und einem URL der Form *rmi://host_name[:name_service_port]/service_name*

Die Kommunikation erfolgt Stream−Basiert, in der Regel über TCP/IP. Darüber wird das Java Remote Method Protocoll (JRMP) verwendet. Dieses existiert in zwei Versionen:

- In JDK 1.1 definiert mit Skeleton Klassen auf dem Server
- In Java 2 SDK definiert, performance−optimiert, ohne Skeleton Klassen.

Mit dem Befehl „`rmic`“ werden die Klassen Stub und Skeleton erstellt. Auf der Client Seite der Stub, auf der Server Seite der Skeleton.

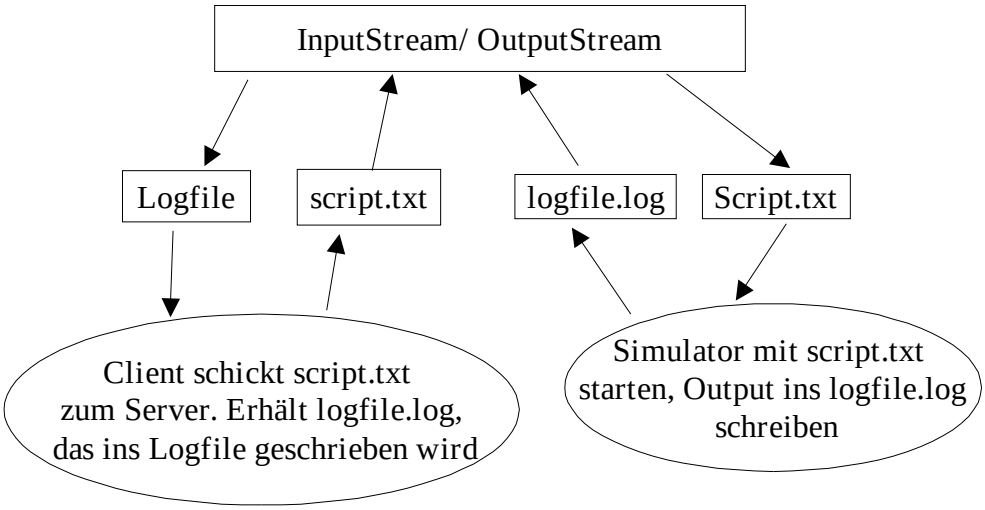
9.3.2 Interface des GUI

Unser Interface heisst `ServerControl.java` und die Implementierung `ServerControlImpl.java`. Folgende Funktionen sind in diesem File definiert.

- start(String scriptname): Das Script „`SimStart.sh`“ auf dem Server ausführen.
- stop(String scriptname): Das Script „`SimStop.sh`“ auf dem Server ausführen.
- fertig(String scriptname): Das Script „`SimFertig.sh`“ auf dem Server ausführen.
- ping(String scriptname): Das Script „`SimAlive.sh`“ auf dem Server ausführen.
- getstatus(): Fragt den Status des Simulators ab. Während der Simulator läuft existiert auf dem Server ein File namens `sim.loc`, welches beim Stoppen des Servers gelöscht wird.
  gestoppt (sim.loc gelöscht) = Status 3
  gestartet (sim.loc erstellt) = Status 2
- disconnect(): Den Client vom Server trennen

9.3.3 Transfer von Files script.txt und lofile.log

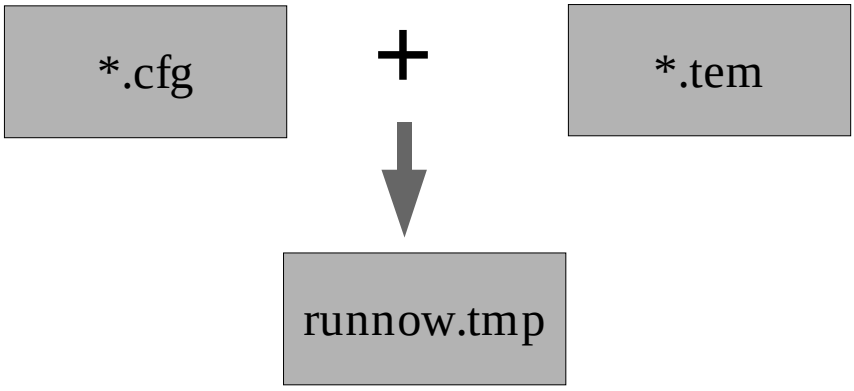
Auf dem Server wird im File `ServSocket.java` ein Thread gestartet der erkennt ob `script.txt` empfangen oder ein `logfile.log` gesendet werden soll. Client und Server kommunizieren über einen Input bzw. OutputStream.



Daher ist es zwingend, dass auf dem Server ein File namens „`logfile.log`“ existiert, sowie das Script das zum Server transferiert wird „`script.txt`“ heisst !

10 Skriptsprache

Wie oben schon erwähnt, wurde die vorhandene Skriptsprache um einen Befehl (speed) erweitert und ein Befehl modifiziert (init). Daneben hat Herr Fernandez von der Firma Omnisec einige weiter Shell−Skripte erstellt, welche das Erstellen von Skripten für den Simulator wesentlich vereinfachen. Weiter ist es mit diesen Shell−Skripten auch möglich, die Netzwerkkarte für das jeweilige virtuelle Netz automatisch zu konfigurieren. Beim Steuern des Simulators über das Graphische User Interface werden diese Shell−Skripte automatisch in der richtigen Reihenfolge abgearbeitet.



Das auszuführende Skript muss nicht mehr neu geschrieben werden, welches bei vielen Netzwerken schnell gross werden kann, sondern es ist ein config− und ein template−File zu erstellen. Diese beiden werden dann gemeinsam ausgewertet und daraus ein für den Generator lesbares File `runnow.tmp` generiert. Dieses File ist vom exakt gleichen Stil, wie die bisherigen Skript−Files für den Generator. Es können also auch bestehende Skripte weiterverwendet werden. Einzige Einschränkung ist das Arbeiten mit der Remote−Control Software. Hier können nur config−Files übertragen werden. Achtung, die benötigten Template Files müssen sich in diesem Fall bereits alle auf dem Server im Unterverzeichnis `../etc/` befinden. Dies liegt an der sehr unflexiblen Programmierung der Remote Software auf welcher wir aufbauten.

10.1 Template File

In den Template Files `*.tem` wird das Skriptfile für den Generator definiert. In diesen kommen die gleichen Befehle vor wie im Skriptfile für den Generator. Hier werden die einzelnen Pakete mit all ihren Optionen definiert. Jedoch müssen die IP Adressen für den Sender und Empfänger nicht mehr direkt in das Skript geschrieben, sondern können dynamisch bei der Generierung erzeugt werden. Dadurch kann das gleiche Skript ohne grossen Aufwand für beliebige Netzwerke erstellt werden. Wichtig ist dabei die Verwendung von folgenden Schlüsselwörtern im template−File, da diese bei der Generierung je nach config−File ersetzt werden.

`KWsrcAddrStart:`	Dieser Wert wird durch den src−startaddr Wert aus dem config File ersetzt (3 Spalte nach @simloop).
`KWsrcAddrStop:`	Dieser Wert wird durch den src−stopaddr Wert aus dem config File ersetzt (4 Spalte nach @simloop)
`KWdstAddrStart:`	Dieser Wert wird durch den dst−startaddr Wert aus dem config File ersetzt (5 Spalte nach @simloop)
`KWdstAddrStop:`	Dieser Wert wird durch den dst−stopaddr Wert aus dem config File ersetzt (6 Spalte nach @simloop)

Beispiel eines template−File Eintrages:

```
udp KWsrcAddrStart−KWsrcAddrStop KWdstAddrStart−KWdstAddrStop *
−data 100−1400
```

10.2 Config File

Als nächstes muss ein config−File erstellt werden. In diesem wird festgelegt, welche Netzwerkkarte welches Netz in welchem IP−Range simuliert. Weiter kann für jedes virtuelle Netzwerk ein sepe−rates template−File angegeben werden, welches Infos über den Simulationsablauf enthält. Als letztes kann man noch eine beliebige Anzahl von Gateways angeben, welche während der Ausfüh−rung der Simulation auf ihre Erreichbarkeit überprüft werden. Das config−File arbeitet mit fol−genden Schlüsselwörtern:

- @sink
- @simloop
- @simbody
- @gateway
- #

10.2.1 @sink

Zeilen welche mit dem Schlüsselwort @sink beginnen, dienen dazu, die Netzwerkkarten im lokalen Computer so einzurichten, dass während der Simulation alle Pakete über die dafür vorgesehenen Interfaces verschickt werden. Dabei können beliebig viele solcher Zeilen angegeben werden. Es ist jedoch darauf zu achten, dass jede dieser Zeilen folgendem Muster entspricht:

```
@sink <interface> <netname> <virtualOffset> <startaddr> <stopaddr>
```

Beispiel:

```
@sink EastNetwork01 eth0 1 172.20.16.1 172.20.16.16
@sink EastNetwork01 eth0 32 172.20.17.1 172.20.17.16
@sink EastNetwork01 eth0 64 172.20.18.1 172.20.18.16
...
```

`virtualOffset:` Der virtualOffset dient dazu, die virtuellen Interfaces einer einzigen Netzwerkarte zu unterscheiden. So muss darauf geachtet werden, dass der Virtual Offset bei jeder Zeile höher ist, als bisher Netzwerkinterfaces initialisiert wurden. Im obigen Beispiel werden für die IP's von 172.20.17.1 bis 172.20.17.16 die Netzwerkinterfaces von eth0:32 bis eth0:47 eingerichtet.

Die Auswertung aller Zeilen des config−Files mit dem Schlüsselwort @sink am Anfang der Zeile erfolgt durch das Shell Skript `sink`. Diesem Skript muss als erster Parameter mit `−f filename` der Filename des config−Files übergeben werden. Als zweiter Parameter muss dem Skript mit `up/down` mitgeteilt werden, ob die Netzwerkkarten für eine Simulation eingerichtet werden sollen, oder ob die Simulation beendet ist, und die Netzwerkkarten wieder in den ursprünglichen Zustand zurückgesetzt werden sollen. Zum Einrichten der Netzwerkarten wird das Skript `virthostup` aufgerufen. Die gesamten Änderungen an der Netzwerkkonfiguration, legt dieses Skript in das File `sink_down_log.tmp` ab. Um die Netzwerkkarten wieder in ihren ursprünglichen Zustand zurückzubringen, wird dieses File vom Skript `virthostdown` wieder eingelesen, abgearbeitet und wieder gelöscht.

10.2.2 @simloop / @simbody

Alle Zeilen welche mit dem Schlüsselwort @sim beginnen, dienen dazu, die verschiedenen template−Files mit den verschiedenen virtuellen Netzwerken zu kombinieren. So kann ein bestimmter Simulationsvorgang für verschiedene Netzwerke simuliert werden, ohne im endgültigen Skriptfile jede Zeile für die verschiedenen IP Adressen mehrmals zu schreiben. Mit dem Befehl @simloop können Wiederholungen angegeben werden, welche genau gleich definiert werden, wie in den Skript Files für den Generator. Die Zeilen für @simloop und @simbody im Config File müssen folgendem Muster entsprechen:

```
@simloop <loopvalue>
@simbody <name> <template file> <src−startaddr> <src−endaddr>
         <dst−startaddr> <dst−endaddr>
@simloop <endloop>
```

Beispiel:

```
@simloop loop endless
@simbody Eastnetwork01 tstsys_ftest01.tem 172.20.16.1 172.20.16.16
         172.30.16.1 172.30.16.16
@simloop endloop
```

Nun werden alle Zeilen mit @sim vom Shell−Skript `simu` ausgewertet und das endgültige Generator Skript `runnow.tmp` erstellt. Dem Skript `simu` muss mit `−f filename` der Filename für das config−File übergeben werden. Als weiterer Parameter muss noch `run` übergeben werden

damit das File für den Generator erstellt wird. Dabei werden die Schlüsselwörter `KwsrcAddr-STar, etc.` des template−Files durch die korrekten Werte für das Netzwerk ersetzt.

10.2.3 @gateway

Zeilen welche mit dem Schlüsselwort @gateway beginnen, dienen dazu, die IP−Adressen der Security Gateways zu definieren. Somit können die zu überwachenden Gateways dem Server vom Remote Client mitgeteilt werden. Exakte Arbeitsweise der „Gateway Überwachung“ siehe Kapitel 11. Die Zeilen für @gateway im config−File müssen folgendem Muster entsprechen:

```
@gateway <IP−Adresse>
```

Beispiel:

```
@gateway 10.0.0.1
```

10.2.4 # (Kommentarzeilen)

Als letztes sind noch die Kommentarzeilen zu erwähnen. Obwohl zur Zeit nur die Zeilen ausgewertet werden, welche mit einem definierten Schlüsselwort beginnen, ist es doch ratsam, alle Kommentarzeilen mit folgendem Zeichen zu beginnen „#“. Erstens wird die Lesbarkeit stark erhöht, und zweitens können künftige Änderungen leichter implementiert werden, wenn in allen config−Files das gleiche Schlüsselzeichen für die Kommentarzeilen verwendet wurde. Sollte aus irgend einem Grund dieses Schlüsselzeichen für die Kommentarzeilen nicht mehr genügen, so kann das bis zur heutigen Version noch problemlos ersetzt werden, da zur Zeit alle Zeilen zum Kommentar gezählt werden, welche nicht mit einem definierten Schlüsselwort beginnen.

11 Gateway Überwachung

Da es das Ziel des Generatores ist, die Security Gateways zu testen, werden diese automatisch überwacht und der Benutzer bei einem Ausfall automatisch informiert. Es muss beachtet werden, dass nur die Verfügbarkeit der diversen Gateways überprüft wird. Die korrekte Arbeitsweise der Gateways muss mit anderen Tools überwacht werden.

11.1 Ablauf

Falls der Button save Log gedrückt und das File angegeben wird, in welches die Logging Informationen abgespeichert werden sollen, wird ein Thread gestartet. Dieser Thread transferiert alle 10 Sekunden das "`logfile.log`" vom Server zum Client und startet das Shell−Skript `SimAlive.sh` auf dem Server, wodurch alle Gateways überprüft werden. Dort öffnet er es und liest jede Zeile aus. Falls er ein "ERROR" ausgelesen hat, gibt er eine Fehlermeldung an den Benutzer aus. Der Benutzer sieht im Logfile in der Zeile unterhalb des Errors die IP des Gateways, die nicht erreicht worden ist.

Möglicher Eintrag im „`logfile.log`" bei korrekter Ausführung des Simulators.

```
START

Parser Start
Parser Ende
Simulator start
Simulator Ende

ENDE
```

Möglicher Eintrag im „`logfile.log`" bei Nichterreichen des Gateways 160.120.48.54

```
START
Parser Start
Parser Ende
Simulator start
Simulator Ende
ERROR
160.120.48.54
ENDE
```

11.2 Funktionsweise

Die Überwachung der Gateways erfolgt zur Zeit noch über das Shell Skript `SimAlive.sh`. Dieses Skript wertet das übergebene config−File aus und überwacht alle Gateways, welche dort definiert worden sind. Dazu wird an alle Gateways der Reihe nach ein Ping verschickt. Der Output des Ping Programmes wird im gleichen Skript ausgewertet, und je nach Status der ERROR−Eintrag im `/log/logfile.log` erstellt.

```
set pingok=‘ping $gatew −c 1 −w 1 |grep transmitted|cut −f4 −d\  ‘
echo "$gatew"
 if ($pingok < 1) then
    echo "ERROR" >> ../log/logfile.log
    echo "$gatew" >> ../log/logfile.log
 endif
```

Mit den Optionen −c 1 und −w 1 wird definiert, dass nur einmal gepingt werden soll, und dass maximal 1 Sekunde auf die Bestätigung gewartet werden soll. Da nur einmal gepingt wird, kann es unter Umständen sein, dass der Ping nicht korrekt beantwortet wird. Solche Vorfälle werden zur Zeit noch nicht abgefangen. Werden die Anzahl Pings pro Gateway oder die maximale Antwortzeit erhöht, muss darauf geachtet werden, dass bei Nichterreichen von mehreren Gateways die Abar−beitungszeit des Skriptes weiterhin unter dem 10s Intervall liegt, in welchem das Skript ausgeführt wird. Bei zu hohen Werten ist die Einhaltung diser 10s nicht mehr garantiert, da die Pings nicht parallel abgearbeitet werden. Mit der aktuellen maximalen Verzögerung von einer Sekunde ist die korrekte Arbeitsweise für bis zu 8 Gateways garantiert. Unsere Implementation stellt erst eine ganz grobe Überprüfung der Gateways dar. Doch zeigt sie das Prinzip auf, wie eine Überwachung der Gateways implementiert werden kann. Auch hier wurde auf die Unabhängigkeit zwischen dem Generator und der Überwachung geachtet, sodass die heute bestehende Logik auf einfache Weise durch eine komplexere ausgetauscht werden könnte.

12 Datei−Struktur

Alle Dateien welche zur Ausführung sowohl als Generator, Senke oder als GUI−Client verwendet werden, sind im File `/simulator/server.tar.gz` zusammengefasst. Weiter ist ein Zip−File `/client/GUI.zip` vorhanden, welches nur das Directory ipnetsim enthält. Dies kann eingesetzt werden, falls die Steuerung der Generatoren von einem Windows Client erfolgen soll. Übersicht über den Inhalt des `server.tar.gz`.

```
-/
 |
 |-bin
 |
 |-etc
 |
 |-java
 |
 |-log
 |
 |-skripte
```

12.1 bin−Directory

Im Unterverzeichnis `bin` ist das bereits compilierte und ausführbare Programm `netsim`. Netsim ist der eigentliche Paketgenerator. Je nach Verwendung von unterschiedlichen Linux Distributionen, kann es sein, dass dieses Binary nicht sauber ausführbar ist. Dann empfiehlt es sich, die Sourcen des Programmes von der CD auf den lokalen Rechner zu kopieren, und dieses neu zu compilieren.

12.2 etc−Directory

Hier sind alle Template−Files abgelegt, welche config−Files benötigen, die mittels Fernbedienung auf den Server geladen werden. Das jeweils aktuelle config− File, welches mittels Java−GUI aktiviert wurde, wird immer in dieses Directory unter dem Namen `script.txt` abgelegt.

12.3 Java−Directory

Im Unterverzeichnis `ipnetsim` sind alle Klassen, die sowohl der Server wie auch der Client braucht, vorhanden.

12.4 Log−Directory

Hier werden alle Log−Dateien abgespeichert.

`logfile.log` In diesem File wird der reduzierte Output des Generatores sowie des

	Gateway−Überwachungsskriptes abgespeichert. Dieses File wird jeweils neu erstellt, wenn auf dem Client eine neue Simulation gestartet wird. Weiter Infos siehe Kapitel 9.2.8
`sim.loc`	Dieses File dient dem Client zur Erkennung, ob auf dem System bereits ein Server läuft oder nicht. Weiter Infos siehe Kapitel 9
`sink_down_log.tmp`	Hier werden die Infos abgespeichert, die benötigt werden, um die Netzwerkinterfaces sauber zu deaktivieren.

12.5 Skripte−Directory

Hier werden alle Shell−Skripte abgelegt, welche irgend einer Form mit dem Simulator zu tun haben. Dabei handelt es sich entweder um `csh` oder `sh` Skripte. Weiter wird in diesem Directory auch das File `runnow.tmp` abgelegt. Dieses File enthält immer das zuletzt generierte Skriptfile für den Generator. Bei jedem neuen Starten des Generatores über das Remote Control Interface, wird dieses File automatisch neu generiert.

12.5.1 SimAlive

Das Skript `SimAlive.sh` realisiert die Überwachung der Security−Gateways. Sollte einmal eine umfangreichere Überwachung für die Gateways benötigt, muss beachtet werden, dass das neu programmierte Programm die Error Meldungen in das richtige File schreibt. So könnte von diesem Skript aus auch ein komplexeres Programm gestartet werden. Weiter Infos siehe auch Kapitel 11.

Aufruf: `./SimAlive.sh Parameter`

Parameter: config−File mit den Gateway IP's. (File muss in `../etc/` liegen.)

Verändert: `../log/logfile.log` (Eintrag der nicht erreichbaren Gateways)

12.5.2 SimFertig

Hat ein ein Generator sein Befehlsskript fertig abgearbeitet, so schreibt er das Schlüsselwort ENDE in das Logfile und beendet sich. Nun existiert aber noch immer das Statusfile sim.loc auf dem Server, welches gelöscht werden muss. Daneben müssen die Netzwerkinterfaces wieder korrekt deaktiviert werden. Aus diesem Grund wird dieses Skript immer dann vom Client aufgerufen, wenn dieser das Schlüsselwort ENDE im Logfile findet.

Aufruf: `./SimFertig.sh` Parameter

Parameter: config−File mit den IP's für die Netzwerkinterfaces. (File muss in `../etc/` liegen.)

Verwendet: `../skripte/sink`

Löscht: `../log/sim.loc`

12.5.3 SimStart

Soll über den Client ein Server gestartet werden, so müssen einige Änderungen vorgenommen werden, damit das Arbeiten des Generatores funktioniert. Zuerst wird das Statusfile `sim.loc` erstellt. Damit wird sichergestellt, dass kein weiterer Generator über den Client auf diesem Server gestartet wird. Danach wird versucht, bereits bestehende Netzwerkinterfaces zu deaktivieren. Nun werden die Netzwerkinterfaces anhand des config−Files aktiviert. Aus dem config−File wird das Generatorskript `runnow.tmp` im aktuellen Verzeichnis erstellt. Der Generator kann nun gestartet werden. In diesem Skript wird der Generator mit der neuen Option `−nooutput` gestartet. Es werden nur wichtige Statusmeldungen abgespeichert und nicht ein riesiges File erstellt.

Aufruf: `./SimStart.sh Parameter`

Parameter: config−File mit den IP's für die Netzwerkinterfaces. (File muss in `../etc/` liegen.)

Verwendet:
```
../skripte/sink
../skripte/simu
../bin/netsim
../skripte/runnow.tmp
```

Erstellt: `../log/sim.loc`

12.5.4 SimStop

Soll eine Simulation gestoppt werden, bevor das Befehlsskript fertig abgearbeitet wurde, so wird vom Client das Skript `SimStop.sh` aufgerufen. Dabei wird zuerst das File `sim.loc` gelöscht. Danach wird mit dem `killall` Befehl der Generator terminiert. Am Schluss werden die Netz−werkinterfaces wieder deaktiviert.

Aufruf: `./SimStart.sh Parameter`

Parameter: config−File mit den IP's für die Netzwerkinterfaces. (File muss in `../etc/` liegen.)

Verwendet: `../skripte/sink`

Beendet: `../bin/netsim`

Löscht: `../log/sim.loc`

12.5.5 firewall

Das Firewall Skript dient dazu, um die benötigten Firewall−Einstellungen automatisch auszu−führen.

12.5.6 simu

Das simu−Skript dient dazu, aus dem Config−File das Generator−Skript `runnow.tmp` zu erstellen. Weitere Infos siehe Kapitel 10

Aufruf: `../simu Parameter`

Parameter:

`−f Filename`	Filename ist Config File in `../etc/`
`run`	es soll gestartet werden.

Erstellt: `../skripte/runnow.tmp`

12.5.7 sink

Das sink Skript wertet das config−File so aus, dass die Netzwerkinterfaces richtig aktiviert werden können. Weitere Infos sieht Kapitel 10.

Aufruf: `./sink −f Parameter`

Parameter:

`−f Filename`	Filename ist Config File in `../etc/`
`up/down`	Ob die Netzwerkinterfaces aktiviert oder deaktiviert werden sollen

Verwendet: `../skripte/virthostdown`
`../skrripte/virthostup`

12.5.8 virthostup

Das virthostup Skript aktiviert die für die Simulation benötigten Netzwerkinterfaces. Achtung: es werden nur die Netzwerkkarten aktiviert. Die Routen zu den Ziel Hosts müssen immer von Hand eingegeben werden.

Aufruf: `./virthostdown Parameter`

Parameter:

`log`	Logfile zum sauberen deaktivieren der Netzwerkinterfaces

Erstellt: `../log/sink_down_log.tmp`

12.5.9 virthostdown

Das virthostdown Skript deaktiviert die für die Simulation benötigten Netzwerkinterfaces. Dieses Skript kann teilweise Warnmeldungen bringen, da Linux alle Netzwerkinterfaces eines Sunetz−werkes löscht, wenn das erste Interface des Subnetzwerkes gelöscht wird. Diese Warnmeldungen können ignoriert werden.

Aufruf: `./virthostdown Parameter`

Parameter:	interface	Netzwerkinterface
	virtoffset	Offset um die Interfaces zu unterscheiden
	startaddr	Start des IP−Address Ranges
	stopaddr	Ende des IP−Address Ranges

Verwendet: `../log/sink_down_log.tmp`

Löscht: `../log/sink_down_log.tmp`

13 Schlusswort

13.1 Fazit

Diese Projektarbeit erwies sich als sehr lehr− und abwechslungsreich. Wir hatten die Möglichkeit uns in den Sprachen Java und C/C++, sowie uns im Schreiben von Shell−Skripts zu vertiefen. Weiter konnten wir die Client/Server Kommunikation über RMI praktisch testen, wobei zuerst das korrekte Aufsetzen der Betriebssysteme auf den Clients und Servern von Nöten war. Der Einstieg in den bereits vorhandenen Quellcode erwies sich besonders am Anfang als äusserst schwierig. Dies insbesondere, weil wir uns zuerst einen Überblick sowohl im C/C++ als auch im Java Code verschaffen mussten. Durch die Erstellung von Struktogrammen erarbeiteten wir uns das Verständnis, das es braucht um den Programmablauf zu verstehen. Die letztjährige Diplomarbeit war soweit ausgereift und dokumentiert, dass sich ein tieferes Einarbeiten in den umfangreichen Quellcode als möglich erwies. Im Gegensatz dazu steht die Projektarbeit des GUI's, die weder ausgereift noch dokumentiert war. Durch den fast komplett neu erstellten Quellcode konnten die Zielvorgaben trotzdem erreicht werden. Ein Umstand der uns einen enormen zusätzlichen Zeitaufwand bescherte, mit welchem am Anfang nicht gerechnet werden konnte. Allgemein ist das gesamte Projekt des Netzwerk−Simulators mit den von uns vorgenommenen Änderungen und Erweiterungen einen grossen Schritt weitergekommen. So konnten alle zum Ziel gesetzten Punkte wunschgemäss implementiert werden.

13.2 Weiterentwicklungsmöglichkeiten / Ausblick

Es bleiben offene Punkte, welche im alltäglichen Gebrauch sicherlich nützlich wären. Funktionen wie das Logging sind nicht vollständig implementiert. Ein erster Grundstein konnte aber gelegt werden. Weiter dürfte eine komplexere Realisierung der Lastverteilung z.B. unter RealTime Linux, eine grosse Herausforderung darstellen. Dazu müssten sehr gute Analysetools für den Netzwerkverkehr zur Verfügung stehen, die auch höhere Bandbreiten verarbeiten können. Bei der Fragmentierung müsste abgeklärt werden, ob externe Tools existieren, die solch grosse Pakete unterstützen, oder ob eine komplett andere Hardware von Nöten wäre. Gedanken über eine zukünftige Erweiterung auf IPv6 sind nötig und dürfte trotz der Abkopplung der Netzwerk−API's von der Programmlogik eine echte Herausforderung darstellen.

13.3 Dank

Wir bedanken uns bei unserem Industriepartner Omnisec, speziell bei Herrn Fernandez und unserem Dozenten Herrn Weibel für die gute Zusammenarbeit.

14 Zeitplan

14.1 Soll Zeitplan

Zeitplan der Pojektgruppe Valle / Miletic Projektleiter

Arbeiten	Woche 21					Woche 22					Woche 23					Woche 24					Woche 25					Woche 26					Woche 27				
	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F
Plan erstellen	X			Auffahrt							Pfingstmontag			Absolvententag														Absolventenkongress							
Besprechung 13.00		X					X					X					X					X					X					X			
Einarbeiten	X																																		
Arbeitsplatz einrichten & 4 PC's		X																																	
Programme austesten			X		X																														
Server integrieren auf Linux Host Bedienung via Windows−Client						X																													
Random−Generator abspecken, neue Algorithmen prg.								X	X	X																									
Log−File erstellen Txt−File												X	X																						
Lastprofil erstellen															X																				
Firmenbesichtigung										X																									
Lastprofil erstellen																X	X	X	X	X															
Lastprofil erstellen																					X														
Installationsmittel erstellen																						X	X	X	X										
Dokumentation																										X	X		X	X					
Dokumentation																															X	X	X	X	X

14.2 Ist−Zeitplan

Zeitplan der Pojektgruppe Valle / Miletic Projektleiter

Arbeiten	Woche 21					Woche 22					Woche 23					Woche 24					Woche 25					Woche 26					Woche 27				
	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F	M	D	M	D	F
Plan erstellen	X			Auffahrt							Pfingstmontag			Absolvententag														Absolventenkongress							
Besprechung 13.00		X					X					X					X					X					X					X			
Einarbeiten	X																																		
Arbeitsplatz einrichten & 4 PC's		X																																	
Programme austesten			X		X																														
Server integrieren auf Linux Host Bedienung via Windows−Client						X	X	X	X	X																									
Random−Generator abspecken, neue Algorithmen prg.												X	X		X	X	X	X	X	X															
Lastprofil erstellen												X	X		X	X	X	X	X	X	X	X	X												
Firmenbesichtigung										X																									
Skriptvereinfachung Kontrolle der Gate−ways mittels Ping																									X	X	X		X	X					
GUI																					X	X	X	X	X	X	X		X	X	X				
Tests																										X	X		X	X	X	X	X		
Dokumentation																										X	X		X	X	X	X	X	X	X

15 Literaturverzeichnis

15.1 Webseiten

[Logi]
Lineare Kongruenzmethode
http://www.logic.at/lvas/krypto/Hillebrand/html/node13.html

[Cosy]
Zufallszahlen
http://www.cosy.sbg.ac.at/~aichhorn/diskretesimulation/html/node5.html

15.2 Dokumentation

[Fei]
Patrick Feisthammel
ZHW−Unterlagen Fach Softwaresysteme
Verteilte Systeme RMI
http://www.zhwin.ch/~fei

[Sna]
Andreas Steffen
ZHW−Unterlagen Fach Kommunikationssysteme
VPN Netzwerke
http://www.zhwin.ch/~sna

15.3 Bücher

[BStr]
Bjarne Stroustrup
Die C++ Programmiersprache
Addison−Wesley, 1998, 3. Auflage

[Hors]
Cay S. Horstmann / Gary Cornell
Core Java 2 Band 1 − Grundlagen
Markt + Technik, 1999, 2. Auflage

[Wiel]
M. Wielsch / J. Prahm / H.−G. Esser
Linux intern
Data Becker, 1999, 1. Auflage

[Hero]
Helmut Herold
Linux/Unix Systemprogrammierung
Addison−Wesley, 1999, 2. Auflage

[Rich]
W. Richard Stevens
Unix Network Programmming
Prentice hall software series, 1990

15.4 ZHW Arbeiten

[DA04]
Andreas Klötzli / Roland Reichlin
IP Netzwerk−Simulator zum Test von VPN−Komponenten
ZHW, 2000, Wei 00/4

[PA]
Daniel Kern / Reto Strassmann
Graphical User Interface für IP−Netzwerk−Simulator
ZHW, ????

16 CD−Aufbau

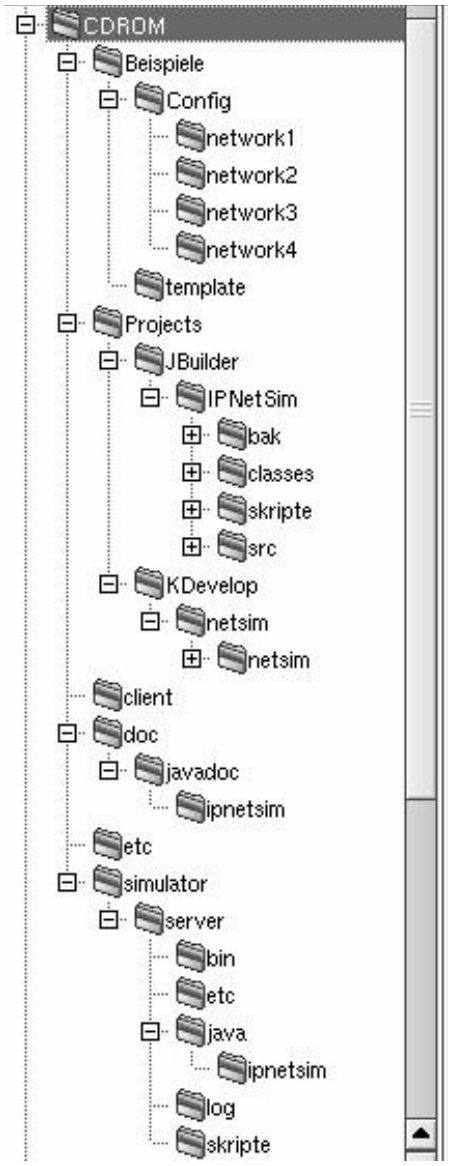


Abbildung 16.1 − Dateistruktur der CD