

xDSL - Projekt

Softwarerouter  
unter Linux

Von: Karsten Dörges  
Leif Evers  
Fatih Gürsoy  
Klemens Mentel  
Edgar Thiel

Semester: I7I1

Betreut durch: Prof. Dr. Lübcke  
Prof. Myrzik

Abgabetermin: 13. Juni 2001

## Inhaltsverzeichnis:

1.	T-DSL Softwarerouter unter Linux .....	4
1.1.	Aufgabendefinition.....	4
1.2.	Teilprojekt eines Gesamtprojektes .....	4
1.3.	Projektdurchführung:.....	5
2.	Inhalt der mitgelieferten CD-Rom .....	6
3.	Firewall & Sicherheitskonzept .....	7
3.1.1.	Einleitung .....	7
3.1.2.	Was ist eine Firewall? .....	7
3.1.3.	Zielsetzungen einer Firewall .....	8
3.1.4.	IP Masquerading durch Firewall.....	8
3.1.5.	Funktionsweise eines Paketfilters.....	8
3.1.6.	Detaillierte Betrachtung von ipchains .....	9
3.1.7.	Gefahren aus dem Internet / Angriffsmethoden .....	12
3.1.8.	Firewall-Systeme ausspähen, um Sicherheitslücken zu entdecken .....	12
3.1.9.	Route-Tracing.....	13
3.1.10.	Paket Sniffer.....	13
3.1.11.	IP Spoofing .....	14
3.1.12.	SYN Flooding- Angriffe / DoS Angriffe durch halboffene Verbindungen.....	14
3.1.13.	Sicherheitsproblem UDP Pakete .....	15
3.1.14.	Einfügen gefälschter Nachrichten.....	16
3.1.15.	Tunneling .....	17
3.1.16.	Fragmentation Attack .....	18
3.1.17.	Flooding - DoS durch E-Mails oder UDP Pakete .....	18
3.1.18.	Gefahren durch e-mails und Download von Software / Anwender....	19
3.2.	Schlussfolgerungen.....	21
3.3.	Sicherheitstest.....	22
3.4.	Literatur .....	22
4.	Router:.....	23
4.1.	Installation des Routers .....	23
4.2.	Optionale Pakete des Software Routers.....	24
4.3.	Webadministrierung .....	24
4.4.	FTP .....	24
4.5.	Telnet.....	24
4.6.	Zeit Service.....	24
4.7.	Kommunikationsschnittstellen.....	25
4.8.	Integration im Intranet.....	25
4.8.1.	Grundinformation.....	25
4.8.2.	Installation der Tomcat Version 3.2 .....	26
4.8.3.	Installation des Admin-Tools .....	26
5.	Testklassen.....	27
5.1.	Testfälle und Testklassen, wofür sind Sie gut? .....	27
5.2.	Erstellung der Testfälle .....	28
5.2.1.	Router .....	28
5.2.2.	Firewall .....	29
5.3.	Erstellen der Testklassen.....	30
5.4.	Vor- und Nacharbeiten bei Tests .....	31
5.5.	Testberichte und Testergebnisse.....	33
5.5.1.	Wie und wann wurden die Testergebnisse aufgenommen.....	33

5.5.2.	Testergebnisse vom 02.05.2001 .....	33
5.5.3.	Testergebnisse vom 09.05.2001 .....	33
5.5.4.	Testergebnisse vom 16.06.2001 .....	34
5.5.5.	Testergebnisse vom 23.05.2001 .....	34
5.5.6.	Testergebnisse vom 30.05.2001 .....	34
5.6.	Test durch die Gruppe „Hard- und Softwarerouter“ .....	34
6.	Admin-Tool Frontend .....	36
6.1.	Beschreibung des Admin-Tool Frontend .....	36
6.2.	Detailliertere Beschreibung der Menüpunkte:.....	36
6.2.1.	Login .....	36
6.2.2.	IP-Settings .....	36
6.2.3.	IP-Range .....	37
6.2.4.	Disable All .....	37
6.2.5.	Enable All .....	37
6.2.6.	Start Router .....	37
6.2.7.	Stop Router.....	38
6.2.8.	Start Firewall.....	38
6.2.9.	Stop Firewall.....	38
6.2.10.	User .....	38
6.2.11.	Logout.....	38
6.3.	Vorgehensweise bei der Implementierung des Admin-Tools Frontend:.....	39
6.4.	Zusätzliche Implementierungen: .....	39
6.4.1.	Template-Generator:.....	39
6.4.2.	Validator-Klasse:.....	40
7.	Admin-Tool Middletier .....	41
7.1.	Die Implementierung des Admin-Tool Middletiers .....	41
7.1.1.	Klasse User .....	41
7.1.2.	Klasse IpSettings .....	42
7.1.3.	Klasse PortSettings .....	42
7.1.4.	Klasse UserException.....	42
7.2.	Die Eigenschaften vom Admin-Tool Middletier:.....	43
7.2.1.	Router .....	43
7.2.2.	Firewall .....	46
8.	Fazit: .....	49
9.	Anhang .....	50
9.1.	Anhang A .....	50
	Projektvorschlag der Gruppe .....	50
	Mentel, Dörges, Evers, Gürsoy und Thiel:.....	50
	Erstellen eines Softwarerouter für eine T-DSL Internetanbindung: .....	50
	Detaillierte Aufgabenbeschreibung des Projektes: .....	51
	Aufgabenverteilung im Projekt : .....	53
9.2.	Anhang A.1.1 .....	54
9.3.	Anhang A.1.2.....	56

# 1. T-DSL Softwarerouter unter Linux

## 1.1. *Aufgabendefinition*

Da unser Projekt die erworbenen Kenntnisse des vorletzten Semesters umsetzen sollte, haben wir uns für einen T-DSL-Router entschieden. Dieser sollte auf einem Linux-Rechner installiert werden und mittels einem webbasierenden GUI administrierbar sein.

Mittels diesem Router soll es einem kleinen Intranets möglich sein, über einen T-DSL Anschluss Internetseiten bzw. Anwendungen, wie z.B. Mail, Proxy usw. zu nutzen.

Die detailliertere Aufgabenstellungen entnehmen Sie bitte dem *Anhang A*.

## 1.2. *Teilprojekt eines Gesamtprojektes*

Aufgabenstellung dieses Projektes, welches sich über die Fachsemester 5 und 7 erstreckte, war es, die Technologie „xDSL“ kennenzulernen und die im letzten Semester erworbenen Kenntnisse anzuwenden.

Hierzu sollte ein Gemeinschaftsprojekt des Semesterverbandes I.7.1 erstellt werden. Um eine solche Aufgabe zu bewerkstelligen, mussten sich Gruppen bilden, die Teilprojekte übernehmen und realisieren.

Da eine Technologie wie xDSL bzw. hier in unserem konkreten Fall T-DSL kaum greifbar ist, wurden die meisten Teilprojekte in der Anwendung mit T-DSL gesucht und realisiert.

Da T-DSL eine sich immer mehr verbreitende Technik ist, wollten wir dem Heimanwender die Möglichkeit bieten, diese Technik für mehr als nur einen Rechner verfügbar zu machen. Ob dies vom Gesetzgeber in der Form erlaubt oder geduldet ist, ist fraglich. Dennoch liegt der Gedanke nah, dass man eine Technik die es einem ermöglicht, wie wir in den anderen Teilprojekten sehen, einen Videostream aus dem Internet bzw. eines Gesprächspartners anzusehen, nicht nur einem PC aus nutzen zu können. Auch andere Teilprojekte zeigen uns die Notwendigkeit, ein solches System auch für den Endanwender nutzbar zumachen, da es schon jetzt mehrere Hard- und Softwarerouter für T-DSL auf dem Markt sind. Aber nicht jeder Heimanwender kann sich einen Router für mehrere Hundertmark leisten.

Wenn man alle Teilprojekte in der Summe sieht, kann sich jeder T-DSL Nutzer ein kleines Intranet mit schnellen Datenaustauschmöglichkeiten je nach Platz und geldlichen Möglichkeiten zusammen stellen. Selbst je nach Kenntnisstand kann er aus den errungenen Kenntnissen und Realisationen profitieren.

So kann z.B. ein ungeübter Nutzer sich einen Hardwarerouter mit einem DataExpress Zugang installieren und sofort lossurfen. Für den geübteren Nutzer mit dem kleineren Geldbeutel ist eine Softwarerouterlösung mit einem normalen T-DSL Anschluss besser geeignet.

### **1.3. Projektdurchführung:**

Nachdem der Projektvorschlag unserer Gruppe von Herrn Lübcke und Herrn Myrzik angenommen wurde, haben wir das Gesamtprojekt in fünf Teilprojekte gegliedert, so dass jeder in der Gruppe für ein Teilgebiet verantwortlich war. Dieses war recht risikoreich, da jeder für den Erfolg des Projektes mitverantwortlich war und an die vorgegebenen Termine und Dead-Lines halten musste.

Daher haben wir versucht, gleich von Anfang an die Schnittstellen jeder Schicht zu beschreiben.

Jede Schicht hat eine grobe Schnittstelle angefertigt von Daten, die sie benötigt und die sie zur Verfügung stellen kann.

Mit diesen Beschreibungen aller Schichten mussten wir uns zusammensetzen und die Schnittstellenbeschreibungen aufeinander anpassen.

Dies war nötig, da die Schnittstellen der einzelnen Schichten nicht identisch waren. So benötigte das Middletier zum Beispiel eine Liste aller verfügbaren IPs. Dies war aber in der Schnittstelle der Router noch nicht vorgesehen.

Nachdem alle Schnittstellen auf einander abgestimmt worden waren, konnten wir anfangen, die Prototypen jeder Schicht zu erzeugen.

Jede Schicht sollte einen Prototypen erstellen, welcher die Schnittstellen mit Dummydaten abdeckt.

Dies war nötig, um damit beginnen konnte die „oberste Schicht“ (Frontend) auch ohne die reale Implementierung der unteren Schichten zu implementieren.

Mindestens alle 14 Tage haben wir uns getroffen, um offene Stellen zu besprechen bzw. sofort zu lösen. Während den 14 Tagen wurden alle Probleme und Anregungen sofort per Email in unserer Gruppe diskutiert und entschlossen.

## 2. Inhalt der mitgelieferten CD-Rom

Auf dieser CD befinden sich alle Sourcen, die benötigt werden, um den von uns in diesem Projekt erstellten Router mit der dazugehörigen Firewall und dem Administrationstool zu installieren und gegebenenfalls neu zu kompilieren.

Die CD besteht grundsätzlich aus zwei verschiedenen Teilen. Der erste Teil ist der kommerzielle Teil, welche den Tomcat (Webserver von Apache), JDK1.2 (Java Developer Kit von Sun) und das Fli4I Tool enthält. Wobei es hierbei nicht ganz richtig ist von kommerziell zu sprechen, da beide Programme frei verfügbar im Internet stehen und dort auch kostenlos heruntergeladen werden können.

Den Webserver kann man sich, falls man eine neuere Version des Tomcats haben will, unter <http://www.apache.org> herunterladen.

Die aktuelle Java Version JDK1.2, die auch als Java 2 bezeichnet wird, ist unter <http://java.sun.com> verfügbar. Sollte es in Zukunft Stabilitätsprobleme mit der Java-Version geben, kann eine neuere Version heruntergeladen werden. Dabei ist aber darauf zu achten, dass unser gesamter Quellcode auf Basis von Java 2 geschrieben wurde und somit mit neueren Versionen nicht lauffähig sein muss. Dies ist leider bei neueren Programmiersprachen nicht unüblich. Und Java macht da keine Ausnahme. Diese Problematik kommt aus dem immer noch hohen Entwicklungsgrad von Java. Jede instabile Methode wird von den neueren Java Versionen nicht verbessert, sondern es wird eine neue Methode geschrieben, die stabil(er) läuft. Die alte Methode wird mit einem „Dirty-Flag“ behaftet, um jedem Programmieren sofort zu zeigen, dass diese Methode nicht stabil läuft. Diese instabilen Methoden werden nach einigen Versionsänderungen nicht mehr unterstützt. Daher ist ein Versionswechsel bei Java-Programmen nicht immer einfach.

Das Fli4I Tool ist unter <http://fli4i.com> frei verfügbar. Auch hier kann es bei Neuinstallation sein, eine neuere Version herunter zu laden, da ältere Versionen nicht alle Netzwerkkarten unterstützen. Man kann diese zwar auch manuell einbinden, aber warum sollte man den Komfort nicht nutzen, den einem das Internet bietet.

Der zweite Teil ist unser Administrationstool, dass aus mehreren Bereichen besteht.

Das Admin-Tool als solches besteht aus den Java-Quelltexten und den Java-Classes. Die kompilierten Javaklassen sind der Übersicht halber in einem gepackten Archiv.

Da das Admin-Tool einige Java-Bibliotheken benötigt, die nicht im JDK1.2 enthalten sind, sind in einem extra Verzeichnis alle benötigten Bibliotheken in weiteren gepackten Archiven.

Darüber hinaus gibt es noch einige Text-Dateien, die das Admin-Tool erzeugt bzw. benötigt. Detailliertere Informationen über die Text-Dateien siehe „Installation des Administrationstools“.

### 3. Firewall & Sicherheitskonzept

#### 3.1.1. Einleitung

Auf den folgenden Seiten wird die Funktionsweise der paketfilternden Firewall ipchains untersucht, zudem werden verschiedene Angriffsszenarien auf die Firewall betrachtet, diese Angriffsmöglichkeiten werden analysiert und die Ergebnisse dieser Analysen sollen in die Konfiguration der Firewall einfließen und somit die Systemsicherheit erhöhen.

#### 3.1.2. Was ist eine Firewall?

Eine Firewall besteht aus Hard- und / oder Software und schützt den eigenen Rechner oder das Netzwerk vor unerlaubtem Zugriff von außen, beispielsweise dem Internet. Sie kontrolliert den Datenfluss, indem zum Beispiel die IP- und Port Adresse des Rechners, von dem ein Datenpaket empfangen wird, mit einer Liste erlaubter Sender verglichen wird.

Firewalls arbeiten auf Paketebene und fungieren als Paketfilter oder nutzen Funktionalitäten oberhalb der OSI Schicht 3, beispielsweise auf Application Level.

Einige Firewalls nutzen auch sogenannte „stateful inspections“, um Pakete anhand ihres Inhalts zu analysieren.

Auf den folgenden Seiten werden wir nur auf Firewalls eingehen, die Paketfilter verwenden, da unser Router die Firewall ipchains nutzt, diese arbeitet auf Paketfilterbasis.

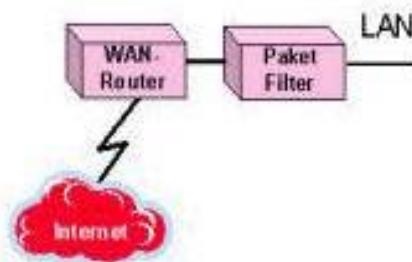


Abb.: Firewall zwischen Router und LAN

### **3.1.3. Zielsetzungen einer Firewall**

Eine Firewall dient dem Schutz eines internen Netzes z.B. LAN vor unberechtigten Zugriff durch externe Netze (z.B. dem Internet).

Unberechtigter Zugriff auf das zu schützende Netz kann dazu führen, dass Angreifer vertrauliche Daten ausspionieren, verändern oder löschen. Zudem besteht die Gefahr, dass Angreifer die Verfügbarkeit von Datenbeständen des internen LANs für andere Nutzergruppen einschränken (z.B. Blockade von Informationsserver).

### **3.1.4. IP Masquerading durch Firewall**

Hierzu baut die Firewall eine Tabelle aller aktiven Verbindungen Internet Hosts zum internen Netz (LAN) auf. Zurückkommende Pakete können somit den Hosts im LAN korrekt zugeordnet werden. Masquerading ähnelt NAT (Network Address Translation) und erhöht die Sicherheit, da die IP Adressen des LAN im Internet nicht sichtbar sind.

### **3.1.5. Funktionsweise eines Paketfilters**

Ein Paketfilter analysiert und kontrolliert die ein- und ausgehenden Pakete auf der Netzzugangs-, der Netzwerk- und der Transportebene. Dazu werden die Pakete, nicht nur TCP/IP-Protokolle, aufgenommen und analysiert.

Diese Analyse wird aufwändiger, sobald auch der Inhalt der einzelnen Pakete untersucht wird (wie es beispielsweise Firewalls mit stateful inspection Funktionen geschieht) ; daher wird normalerweise nur der Header durchsucht. Die beiden Netze, also internes und externes Netz, werden bei einer solchen Implementierung physikalisch entkoppelt. Ein Paket-Filter verhält sich wie eine normale Bridge und wird transparent in eine Leitung eingefügt.

Nach der Analyse des Pakets wird verifiziert, ob sie unter eine bestimmte Regel fallen und dementsprechende Reaktionen ausgeführt. In den Regeln wird vorzugsweise definiert, dass nur die notwendigste Kommunikation erlaubt ist; alle Verstöße werden abgewiesen (engl. deny) oder verworfen (engl. drop). Dadurch können sicherheitskritische Aktionen, wie zum Beispiel IP - Fragmentierung, von vornherein ausgeschlossen werden.

Auf Transportebene (OSI Schicht 4) findet bei UDP und TCP eine Überprüfung der Portnummern (Quell- und Ziel-Port) statt, bei TCP zusätzlich eine Überprüfung der Richtung des Verbindungsaufbaus statt.

Zusätzlich kann überprüft werden, ob der Zugriff in einem erlaubten Zeitrahmen geschieht.

Die entsprechenden Prüfinformationen werden aus dem Regelwerk entnommen und mit den Ergebnissen der jeweiligen Analysen verglichen.

### 3.1.6. Detaillierte Betrachtung von ipchains

Eine Chain ist eine Summe von Paketfilterregeln. Jede Regel prüft nun den Paket Headers des Paketes und entscheidet dann die weitere Vorgehensweise. Im Gegensatz zu anderen Firewalls wird das Paket mit allen Regeln in einer Chain verglichen. Am Ende der Chain angelangt, entscheidet der Kernel danach, was in der Chain default policy steht und führt die darin enthaltenen Befehle aus.

Der ipchains Paketfilter wird mit den Befehlen input, output und forward gesteuert. Diese Kommandos sorgen dafür, dass die zu filternden Pakete, nach Ziel- oder Quelladressen unterschieden werden .

Der Input Befehl ist für die Daten, die von einem externen Host eintreffen und an einen lokalen Host gesendet werden. Output betrifft alle Pakete, die von einem Rechner in das externe Netz gesendet werden (vom Router/ Firewall oder von einem Rechner hinter der Firewall). Forward setzt die Regeln, die für das Forwarding (weiterleiten) gelten.

Jede dieser 3 Varianten muss mit einer der vier folgenden Regeln besetzt sein (DENY, REJECT, ACCEPT und MASQ). DENY verwirft das Datenpaket - die Firewall ignoriert das Paket, als ob es nie existiert hätte. REJECT sendet im Gegensatz zu DENY ein icmp\_unreachable Paket an den Host zurück. ACCEPT erlaubt dem Datenpaket das Passieren durch die Firewall. Mit MASQ wird das jeweilige Paket maskiert.

Für jedes Datenpaket kann eine Quelle (source) und ein Ziel (destination) angegeben werden. So kann z.B. festgelegt werden, dass die Firewall Pakete nur von oder zu einer bestimmten IP Adresse durchlässt.

In einer Firewall existiert eine "default policy". Diese beschreibt das Verhalten, wenn keine Regel zutrifft. Es gibt zwei Möglichkeiten: Entweder die Firewall lässt alle Pakete passieren, oder sie blockt alle ab. In dem ersten Fall muss der Firewall explizit mitgeteilt werden, was verboten ist, in dem anderen Fall muss ihr mitgeteilt werden, was erlaubt ist, alles andere ist dann nämlich verboten. Wir empfehlen aus Sicherheitsgründen die zweite Möglichkeit.

Die einzelnen ipchains können entweder manuell geladen werden (über Eingabe von Kommandozeilenparametern) oder z.B. durch ein Skript , das alle Regeln enthält und die Firewall entsprechend konfiguriert.

#### Übersicht ipchains Parameter

Parameter:	Erklärung:
J	Der Parameter -j legt eine der Rules fest: accept, deny, masq oder reject
D	d steht für Destination=Ziel. Mit diesem Parameter koennen sie das Ziel des Datenpaketes festlegen.
S	s: Source. Hiermit wird die Quelle des Paketes bestimmt.

L	l: Log. Mit diesem Parameter werden die Zugriffe durch die einzelnen Chains im syslog (/var/log/messages) protokolliert.
P	p: Protokoll. 'tcp' oder 'udp'
I	i: Interface. Hiermit wird das Gerät bestimmt. z.B. eth0 (Netzwerkkarte) oder ippp0 (ISDN Karte)
A	Mit dem Parameter -A wird die Variante bestimmt. input, output oder forward.
P	P: Policy.. hiermit wird eine default policy gesetzt.

Beispiele ipchains Konfiguration :

**default policy – Pakete verwerfen, falls keine zutreffenden Regel existiert**

```
#Default Policy
ipfwadm -I -p deny
ipfwadm -O -p deny
ipfwadm -F -p deny
```

**Alles was von der ersten Netzwerkkarte Karte aus dem Internet kommt und auf 'telnet' zugreift soll verworfen werden:**

```
ipchains -A input -j DENY -I -s 0.0.0.0/0 -p tcp -i ippp0 -d 0.0.0.0/0 23
```

**Alle Adressen aus dem lokalen Subnetz sollen maskiert werden:**

```
ipchains -A forward -s 192.168.1.0/24 -j MASQ
```

**Alle Pakete in das externe Netz senden :**

```
ipchains -P output ACCEPT
```

**Ports von externen Hosts sperren:**

```
ipchains -A input -j DENY -p tcp -i ippp0 -I -d 0.0.0.0/0 {port}
```

Die folgende Grafik verdeutlicht den Transfer der Pakete durch ipchains.

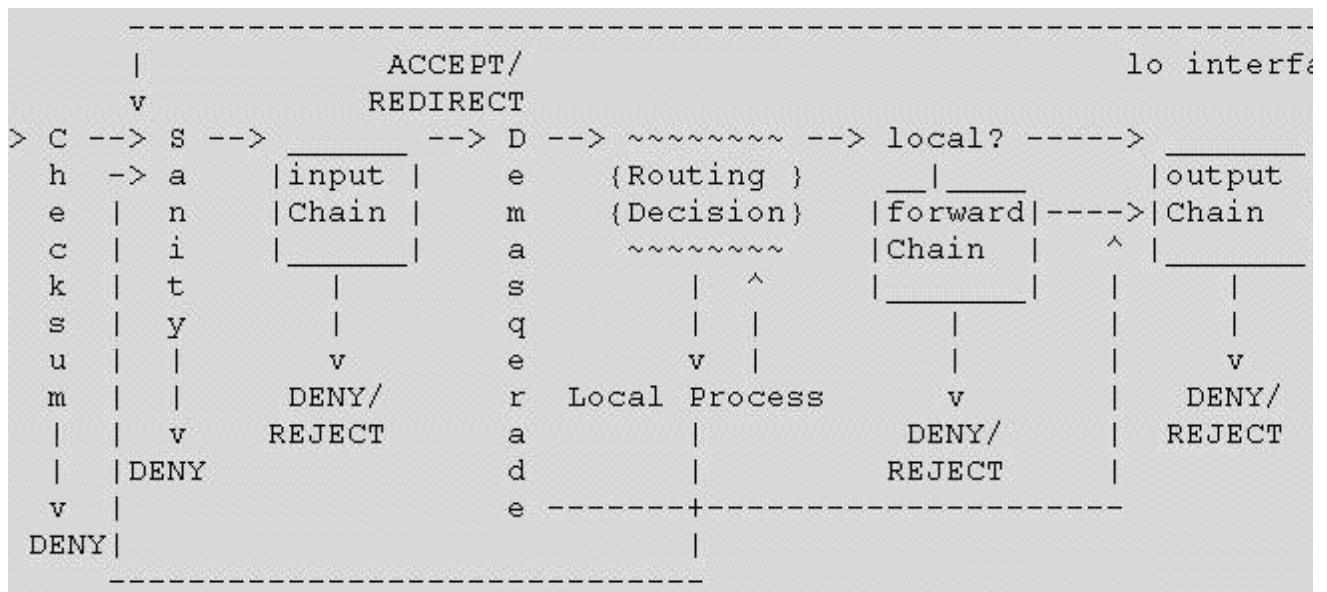


Abb.: Funktionsweise als Blockschaltbild von ipchains

Erläuterung wichtiger Begriffe in oben dargestellter Grafik :

Die **Checksum** ist eine Prüfsumme in dem IP- Header, die feststellt, ob ein Paket korrekt übertragen wurde. Stimmt die Prüfsumme nicht, so wird das Paket abgelehnt.

Der Sanity Mechanismus überprüft Pakete mit besonderen Flags im Header, die eventuell die korrekte Abarbeitung der Firewall-Regeln verhindern könnten (z.B. IP- Fragmente).

In der input chain werden die Pakete anhand der Filterregeln getestet. Pakete werden hier zurückgewiesen oder verworfen.

Die Funktion Demasquerade maskiert Pakete, die von einem Host im Intranet stammen, zu Absenderadressen mit der Firewall. Die eintreffende Antwort aus dem Internet werden der Adr. Im internen **ko** zugeordnet und weitergeleitet. Dies dient dem Verdecken der internen Netzwerk – Adressen und erhöht somit die Sicherheit. Wenn kein Masquerading aktiviert worden ist, ist diese Routine deaktiviert.

Die Zieladresse wird von der Routing Unterroutine routing decision untersucht. Hierbei entscheidet sich, an welchen **output chain** das Paket übergeben wird.

Die Fkt. **Local process** ermöglicht es, dass Paket von einem Programm entgegen genommen und weiterverarbeitet wird. An dieser Stelle kann z.B. ein Proxy oder ähnliches zur weiteren Filterung der Pakete installiert werden. Danach wird das Paket an die **output chain** übergeben.

Falls Pakete eines lokalen Programms an ein anderes Programm übergeben werden müssen, dann sind diese an das Interface **lo** (loopback Interface). Für dieses Interface existiert also auch eine **input chain**, die Ausgaben eines Programms wieder an den Kernel zwecks Weiterleitung übergeben kann.

Wenn ein Paket nicht von einem lokalen Programm (**local**) erzeugt wurde, dann wird dieses an die **forward chain** übergeben, ansonsten wird das Paket an die **output chain** direkt übergeben.

Die forward chain wird von jedem Paket durchlaufen werden, die von einem Interface kommend, an ein anderes weitergeleitet werden möchte.

In der **output chain** werden nochmals alle Pakete gefiltert, bevor sie das Interface verlassen.

### **3.1.7. Gefahren aus dem Internet / Angriffsmethoden**

Auf dem folgenden Seiten werden die gängigsten Angriffsmethoden diskutiert, diese Zusammenstellung erhebt keinen Anspruch auf Vollständigkeit.

Die hohe Anzahl der im Internet publizierten Angriffsmethoden auf Firewalls können im Rahmen dieser Ausarbeitung nicht behandelt werden, aber viele Angriffe sind nur Variationen der hier vorgestellten Vorgehensweisen.

### **3.1.8. Firewall-Systeme ausspähen, um Sicherheitslücken zu entdecken**

Firewall-Systeme und die darauf befindliche Software ist, wie auch ein individuelles Betriebssystem, durch diverse Techniken zu erkennen. Mit der Hilfe von Portscans und durch Abfangen von http Anfragen kann ein Angreifer von der Version bis hin zu den gesetzten Filter-Regeln alles für ihn notwendige herausfinden. Firewall-Systeme gelten als die größten Gegner von Eindringlingen und sind somit das erste Ziel eines Angreifers: Hat ein Angreifer erst einmal die Firewall ausgekundschaftet, kann er sich an die Kompromittierung derer machen.

Durch systematisches Scannen aller Ports kann ein Angreifer feststellen, welche Dienste und Prozesse auf der Zielmaschine derzeit ablaufen. Das Scannen kann bei einigen Betriebssystemen auf Grund von Implementierungsschwächen zu einem Denial-of-Service oder sogar zu Systemabstürzen führen. Die erhaltenen Informationen können für weitere Angriffe verwendet werden (z.B. Sniffing).

Gegenmaßnahmen:

Möglichst alle offenen Ports sperren, dies minimiert aber lediglich die Risiken.

### **3.1.9. Route-Tracing**

Die etwas leisere und subtilere Methode des Auskundschaftens von Firewall-Elementen ist das sogenannte Route-Tracing. Unter UNIX und seinen Derivaten kann man sich des Befehls traceroute und unter allen Windows-Versionen tracert.exe bedienen, um die Hops zu einem Ziel ausfindig zu machen. Unter UNIX/Linux verfügt Traceroute über den Parameter "-I", um den Trace anstatt mit der Übermittlung von UDP-Paketen mittels der ICMP-Technik durchzuführen.

Gegenmaßnahmen:

ICMP Pakete verwerfen.

### **3.1.10. Paket Sniffer**

Sniffer sind Programme, die es einem Administrator ermöglichen, Netzwerkleitungen anzuzapfen und den Netzwerkverkehr zu überwachen. Sniffer arbeiten in verschiedenen Protokollschichten und ermöglichen so die Fehlersuche in einem Netzwerk. Der Einsatz kann mit dem Abhören von Telefonleitungen verglichen werden. In der Hand eines Hackers werden sie hingegen oft dazu genutzt, vertrauliche Informationen wie z.B. User-Namen, Kennworte und E-Mails auszuspähen.

Die am häufigsten eingesetzten Netzwerke (wie z.B. Ethernet) nutzen zur Kommunikation einen gemeinsamen Kabelstrang. Das bedeutet, dass alle im Netzwerk übertragenen Daten an jedem Rechner dieses Netzes zur Verfügung stehen. Spezielle Filter in den Netzwerkkarten sorgen dafür, dass die Daten nur für den als Empfänger bestimmten Rechner lesbar sind. Sniffer umgehen diese Filterfunktion und ermöglichen es so, auf alle Datenpakete des Netzes zuzugreifen. Da der Einsatz von Sniffern selbst keinen Netzwerkverkehr erzeugt, da nur lesende Zugriffe erfolgen, ist es in der Praxis schwer, solche Attacken festzustellen.

Da die Datenpakete in einem Netzwerk im Binärformat vorliegen und somit nicht direkt lesbar sind, verfügen Sniffer oft über eine weitere Funktion, den sogenannten "Packet-Analyzer". Der Paket-Analyzer wandelt das binäre Datenformat in eine für den Anwender lesbare Form um und ermöglicht so das Mitlesen der Datenpakete. Wenn nun im Netz unverschlüsselte Passworte und Benutzerkennungen übertragen werden, sind diese ebenfalls in Klarschrift lesbar.

Darüber hinaus ist zu beachten, dass die oben beschriebene Vorgehensweise auch im Internet funktioniert, da auch im Web auf der gesamten Strecke zwischen Sender und Empfänger sämtliche Daten abgehört werden können.

Gegenmaßnahmen:

Die Bedrohung durch Sniffer im Ethernet oder zwischen x-DSL Router und Vermittlungsstelle kann vernachlässigt werden, aber Sniffer können durchaus im Internet Daten ausspionieren, daher sollten keine vertraulichen Daten unverschlüsselt übertragen werden.

### **3.1.11. IP Spoofing**

Ein kombinierter Angriff auf DNS und die Routingmechanismen beruhen darauf, dass ein Angreifer alle Requests an den DNS, Namen in IP-Adressen zu übersetzen, mithört und stattdessen die IP-Adresse eines zuvor bereits gekaperten Hosts liefert. Dadurch kann der Angreifer die gesamte Kommunikation ausspähen und Passworte sammeln. Somit sind DNS-Server ein sehr attraktives Angriffsziel. Dies impliziert, dass DNS-Server nur auf besonders gesicherten Rechnern installiert sein sollten.

Gegenmaßnahmen:

Keine – die Sicherheitsmassnahmen müssen auf dem entsprechen DNS Servern erfolgen.

### **3.1.12. SYN Flooding- Angriffe / DoS Angriffe durch halboffene Verbindungen**

TCP-SYN-Flooding kann auf zwei Arten für denial-of-service-Angriffe auf Server verwendet werden. Zum einen kann ein Angreifer auf Basis von IP-Spoofing halboffene Verbindungen etablieren. Der Angreifer-Client sendet SYN, der Opfer-Server antwortet mit SYN-ACK, aber der Client bestätigt nicht mit ACK. Solange diese Bestätigung fehlt, ist die Verbindung also halboffen. Bei einer gewissen Anzahl von halboffenen Verbindungen ist der Server nicht mehr in der Lage, neue Verbindungen anzunehmen. Dieser Angriff kann verhindert werden, indem z.B. eine Firewall IP-Spoofing nicht zulässt. Zum anderen ist TCP-SYN-Flooding auch ohne IP-Spoofing möglich, indem der Angreifer als Quelle eine Adresse angibt, zu der das SYN-ACK nicht geroutet werden kann, weil sie nicht existiert oder für den Server nicht erreichbar ist.

Gegenmaßnahmen:

Um diese Gefahr abzuwenden können zustandsabhängige Filter nur eine begrenzte Anzahl von SYN-Paketen zulassen. Dies kann mit aber ipchains nicht realisiert werden!

### 3.1.13. Sicherheitsproblem UDP Pakete

UDP ist ein Protokoll, welches kein ACK Bit benötigt. Dieses ACK Bit ist für die Firewall aber ein Zeichen, dass die Pakete passieren dürfen, sofern vorher ein Paket mit SYN Bit aus dem Intranet in das Internet versandt worden ist. Die Firewall speichert intern dann in einer HASH Tabelle, deren HASH - Werte sich aus Port - und IP - Nummer errechnen..

Genaugenommen ist das SYN Bit ein Zeichen für die Firewall, sich genau zu merken, von welchem Host aus eine Verbindung aus dem Intranet zu einem Host ins Internet aufgebaut werden soll, also für welchen Host aus dem Internet Antwortpakete zurück erwartet werden. Die Pakete, die aus dem Intranet zurück in das Intranet gesendet werden, sind durch das SYN Bit dann autorisiert. Die Firewall muss nur noch prüfen, ob diese ein ACK Bit besitzen, ob die Port - Adresse korrekt ist, und die IP - Nummer stimmt. TCP Verbindungen benutzen nach dem 3-Wege Handshake, also dem Verbindungsaufbau, höhere Ports über 1024, auch unprivilegierte Ports genannt . UDP hingegen hat einen wesentlich kleineren Overhead und ist somit viel schneller, insbesondere bei kleineren Datenmengen. Dafür besitzt es keinerlei Möglichkeit, verlorene Pakete zu erkennen, und diese wie bei TCP evtl. nochmals neu anzufordern. UDP Pakete benutzen die Ports, die für den Verbindungsaufbau verwendet wurden, auch weiterhin, außer die Programme selber haben diesen Mechanismus implementiert. Für die Erstellung von Prüfsummen ist ebenfalls das Programm selber verantwortlich. Da UDP Pakete ein SYN/ACK Mechanismus besitzen, kann eine Firewall bei diesen Paketen nicht feststellen, ob diese von einem Host im Intranet auch angefordert worden sind. Daraus folgt, dass die entsprechen Applikationen prüfen müssen, ob die empfangen UDP Pakete korrekt sind, oder aber alle UDP Ports auf der Firewall werden gesperrt, und die UDP Pakete aus dem Internet prallen an der Firewall ab.

Konkretes Angriffsszenario :

Ein solcher Angriff ist komplexer als man denkt, aber einfach zu zeigen. Wer mit einem älteren Netscape oder IE Browser animierte GIF Bilder betrachtet, der wird bemerken, daß hier GIF Bilder schnell hintereinander angezeigt werden. Problematisch wird dies erst dann, wenn das Format der Bilder plötzlich größer wird. Der intern vom Browser reservierte Speicherplatz für die Animation reicht dann plötzlich nicht mehr aus. Ist das Bild genügend groß, werden eventuell wichtige Bereiche im RAM der Arbeitsstation überschrieben. Ein sogenannter schwerer Ausnahmefehler (wo da die Ausnahme ist, weiß wahrscheinlich Microsoft selber nicht) tritt auf. In den meisten Fällen ist dieses Problem mit einem Absturz

verbunden. Wird jedoch die Bytefolge im GIF Bild so gewählt, daß sich ein sinnvolles Programm dahinter verstecken kann, dann kann der Angreifer hierüber auf der Arbeitsstation beliebige Programme starten. Diesen Effekt nennt man Buffer overflow und ist im Prinzip bei allen Protokollen, TCP sowie UDP möglich. Hier wurde der Effekt anhand einer GIF Animation aufgezeigt, dieser Effekt kann aber auch bei (REAL)VIDEO Sequenzen, DNS Paketen, NFS oder RPC Paketen o.ä. auftreten. Gelingt es einem Angreifer also, in einen UDP Datenstrom mit gespoofen Paketen eigene Daten einzuschleusen, dann kann er viel Unheil anstiften. Aus diesem Grunde ist bei RPC, NFS, DNS - Datenpaketen immer große Vorsicht angebracht. Ohne spezialisierte Proxys sollte man diese Pakete von einer Firewall tunlichst nicht transportieren lassen.

Gegenmaßnahmen:

UDP Pakete an der Firewall verwerfen oder die milde (aber unsichere) Methode wählen und die jeweils neuste Software installieren und hoffen, dass bestehende Sicherheitslücken behoben wurden.

### **3.1.14. Einfügen gefälschter Nachrichten**

TCP erkennt anhand der für jedes gesendete Paket neuen zufällig bestimmten Initial Sequence Number alte Pakete aus vorangegangenen Verbindungen (identifizierbar durch Quellsocket und Zielsocket). Da eine Verbindung erst dann zustande kommt, wenn beide Seiten jeweils den Empfang der Initial Sequence Number der Gegenseite quittiert haben, ist somit ein Sicherheitsmechanismus gegen replay attacks vorhanden. Kann allerdings ein Angreifer die Auswahl der Initial Sequence Number seines Opfers vorhersagen, so kann der Angreifer seinem Opfer eine Verbindung mit einer vertrauenswürdigen Maschine vortäuschen. Über Protokolle, die zur Authentisierung lediglich IP-Quelladressen verwenden (z.B. die "r"-Dienste), gelangt der Angreifer somit in das Zielsystem. Dieses Eindringen nach dem Vorherbestimmen der Initial Sequence Number kann durch eine Firewall verhindert werden. Um das Vorhersagen der Initial Sequence Number zu verhindern, empfiehlt es sich, den Zeitpunkt der Sequenznummernerhöhung zufällig zu bestimmen, wobei in kleinen Zeitabständen real zu wechseln ist.

Gegenmaßnahmen:

Sequenznummern werden vom TCP / IP Stack erzeugt, daher sollte der Administrator bei Sicherheitslücken des System die neuesten Patche installieren.

### 3.1.15. Tunneling

Ein lokaler User verbindet sich über `anonymous ftp` zu einem remote FTP-Server (Port 21). Um eine Datei zu übertragen, wählt der Client des Users zunächst einen beliebigen TCP-Port, an dem er auf die Datei wartet. Der Client sendet ein `PORT`-Kommando, um diese Wahl dem Server mitzuteilen. Der Server antwortet, indem er eine aktive TCP-Verbindung zu dem angegebenen Host öffnet und die Datei versendet. Ist das Netz des Clients durch einen Paketfilter geschützt, so wird die Datenübertragung fehlschlagen, da der Server ja einen internen Port öffnen möchte. Ein Applikationsfilter untersucht die Daten nach diesem speziellen `Port`-Befehl. Hat es diesen entdeckt, so lässt es eine Verbindung zwischen Remote Host und internem Client etablieren, da die Anfrage für eine solche Verbindung ja vom internen Client kam. Ist diese Verbindung erst etabliert, so kann sie beliebig lange bestehen bleiben. Öffnet ein bössartiger interner Nutzer eine FTP-Verbindung nach außen und gibt an, die Datei auf Port 23 (dem `Telnet`-Port) zu erwarten, so kann der Remote Host eine `Telnet` Verbindung zum Client erstellen. Dieser Angriff lässt sich mittels Java auch von externen Angreifern ausführen.

Durch die Fragmentierung von IP-Paketen existiert wenig Information, auf die man eine Filterentscheidung basieren kann, denn bis auf das erste Fragment besitzen die Fragmente keine Portnummern. Besitzt ein Angreifer einen Komplizen im zu schützenden Bereich, so können Fragmente ohne Portnummern zum Tunneln einer Firewall verwendet werden. Das erste Fragment beinhaltet zwar die Portnummer und kann entsprechend gefiltert werden. Wird dies nicht weitergeleitet, so ist das Paket im Inneren unvollständig, und die übrigen Fragmente werden schließlich vom Zielknoten verworfen. Ist der Zielknoten allerdings im Besitz eines Komplizen des Angreifers, so kann dieser die Fragmente zusammensetzen. Analog kann der Komplize im zu schützenden Bereich gefälschte Fragmente ohne Portnummern erstellen, die vom externen Komplizen auf einer äußeren Maschine zusammengesetzt werden können.

Gegenmaßnahmen:

Alle Fragmente ohne Portnummern bereits an der Firewall verwerfen oder dort zusammensetzen und überprüfen (dies kann mit `ipchains` nicht realisiert werden) werden. Dies ist mit zustandsabhängigen Filtern bzw. Paketfiltern, die Kontexte speichern können, möglich.

### **3.1.16. Fragmentation Attack**

Der Fragmentierungsangriff zielt darauf ab, die Implementierung der IP-Fragmentierung derart auszunutzen, daß Filterregeln von Paketfiltern nicht auf die fragmentierten IP-Pakete passen und somit diese Fragmente durchgelassen werden. Im [RFC 1858] wird der Tiny Fragment Attack beschrieben. Dabei zerlegt der Angreifer seine Nutzdaten (z.B. TCP-Pakete) in sehr kleine Fragmente. Dadurch wird z.B. ein TCP-Header auf mehrere Fragmente aufgeteilt und kann daher von statischen Paketfiltern nicht analysiert werden. Eine Filterung aufgrund von Regeln, die z.B. den TCP-Header betreffen, ist nicht mehr möglich. Als Gegenmaßnahme sollten Implementierungen von Paketfiltern verwendet werden, bei denen die Länge des Tiny Fragments nicht verändert werden kann.

Ein zweiter Angriff, der die IP-Fragmentierung ausnutzt, ist der sogenannte Overlapping Fragmenten. Die derzeitige Internet-Protokoll-Spezifikation beschreibt einen Reassemblierungsalgorithmus, der neue Fragmente produziert und dabei jeden überlappenden Teil der zuvor erhaltenen Fragmente überschreibt. Wird ein solcher Algorithmus angewendet, so könnte ein Angreifer eine Folge von Paketen konstruieren, in denen das erste Fragment (mit einem Offset der Länge Null) harmlose Daten beinhaltet (und dadurch von einem Paketfilter weitergeleitet werden kann). Ein beliebiges nachfolgendes Paket mit einem Offset, der größer als Null ist, könnte TCP-Header-Informationen (z.B. destination port) überlappen. Diese würden durch den Algorithmus modifiziert werden. Dieses zweite Paket wird von vielen Paketfiltern nicht gefiltert.

Gegenmaßnahme:

Gegenmaßnahme hierzu ist, Paketfilter zu verwenden, die ein Minimum an Fragment-Offset für Fragmente verlangen.

### **3.1.17. Flooding - DoS durch E-Mails oder UDP Pakete**

SMTP kann als Basis für einen Denial-of-Service Angriff dienen. Dieser Angriff zielt darauf ab, die legitime Benutzung einer Maschine zu verhindern. Angenommen, ein Angreifer sendet von 50 Maschinen aus jeweils 1000 1-MB Mails an sein Opfer, dann ist es wahrscheinlich, daß das System des Opfers dies nicht verarbeiten kann.

Es ist empfehlenswert, dass eine Organisation eine zentrale Einrichtung benennt, die für Mail zuständig ist. Dieser zentrale Mailserver sollte sich in einem Screened Subnet hinter dem äußeren Paketfilter befinden..

Da UDP eine Flusskontrolle fehlt, kann es das Datennetz lahm legen, indem es Router und Hosts mit Paketen überflutet. UDP hält sich an dieselben Konventionen für Port-Nummern und Server wie TCP, jedoch in einem eigenen Adressraum. Meist verwenden die Server niedrige Port-Nummern. Da es keine virtuellen Verbindungen gibt, werden alle Pakete, die für einen bestimmten Zielport bestimmt sind, unabhängig von Quelladresse oder Quellport an denselben Prozess weitergereicht.

Ein Denial-of-Service Angriff kann auch durch sogenannten UDP packet storm auf einem System oder zwischen zwei Systemen gefahren werden. Auf einem System bewirkt er eine Leistungsminderung. Zwischen zwei Systemen kann er zum Netzzusammenbruch führen. Daher sollten unnötige UAP-Dienste auf jedem Host abgeschaltet werden, insbesondere die Dienste Chargen (erzeugt eine lange Folge von Zeichen) und echo. Der Angriff erfolgt nach folgendem Muster: Sobald eine Beziehung zwischen zwei UDP-Diensten besteht, die jeweils Output produzieren, können diese Dienste eine hohe Anzahl an Paketen produzieren, die zu einem Denial-of-Service bei den Maschinen führen, auf denen die Dienste ablaufen. Für die Durchführung dieses Angriffs ist lediglich eine Netzverbindung notwendig.

Gegenmaßnahme:

Zentralen Mailserver verwenden (damit nur ein Rechner im schlimmsten Fall ausfällt) und UDP verbieten.

### **3.1.18. Gefahren durch e-mails und Download von Software / Anwender**

Viren, Würmer oder Trojanische Pferde aus dem Internet bedeuten eine immanente Gefährdung der IT Sicherheit. Aus Sicherheitsgründen sollten nur Betriebssysteme eingesetzt werden, die eine Nutzerverwaltung bezüglich abgestufter Rechtevergabe erlauben (z.B WinNt oder Linux) . Nur Administratoren sollen neue Software installieren dürfen, um die oben genannten Gefahren zu minimieren.

Gegenmaßnahmen:

Die Anwender müssen entsprechend geschult werden, dabei sollten die folgenden Sicherheitsrichtlinien beachtet werden:

Alle vorhandenen Sicherheitsfunktionen des Rechners sollten aktiviert werden (Passwort-Schutz , Passwörter regelmäßig ändern) , damit während der Abwesenheit des berechtigten Benutzers Unbefugte keine Möglichkeit haben, durch unbedachte oder gewollte Handlungen den Rechner zu gefährden.

Ein aktuelles Viren-Schutzprogramm mit aktuellen Signatur-Dateien einsetzen, das im Hintergrund läuft und bei bekannten oder erkannten Computer-Viren Alarm schlägt.

Im Browser sollte die Anzeige aller Dateitypen aktiviert sein.

Makro-Virenschutz von Anwendungsprogrammen (WinWord, Excel, Powerpoint, etc.) aktivieren und Warnmeldungen beachten.

Sicherheitseinstellungen von Internet-Browsern auf höchste Stufe einstellen (Deaktivieren von aktiven Inhalten (ActiveX, Java, JavaScript) und Skript-Sprachen (z.B. Visual Basic Script, VBS), etc.).

Keine Applikationsverknüpfung für Anwendungen mit potentiell aktivem Code (MS-Office) im Browser nutzen oder Anwendungen über Internet aktivieren.

Eingehende E-Mail können Viren, Trojaner oder Würmer einschmuggeln. Bei sicherheitsbewusstem Verhalten lassen sich hierbei schon die meisten Gefahren vermeiden.

Offensichtlich nicht sinnvolle E-Mails von unbekanntem Absendern sofort ungeöffnet löschen.

Bei E-Mail auch von vermeintlich bekannten bzw. vertrauenswürdigen Absendern prüfen, ob der Text der Nachricht auch zum Absender passt.

Verdächtig wäre beispielsweise ein englischer Text von deutschem Partner, oder auch fehlender Bezüge zu konkreten Vorgängen.

Besondere Vorsicht ist bei E-Mail Anlagen (Attachments) geboten (ausführbare Dateien, Skripte oder Office Dokumente sollten nicht geöffnet werden.)

Also kein "Doppelklick" bei ausführbaren Programmen (\*.COM, \*.EXE) oder Script-Sprachen (\*.VBS, \*.BAT), Vorsicht auch bei Office-Dateien (\*.DOC, \*.XLS, \*.PPT) sowie Bildschirmschonern (\*.SCR).

Auch eine E-Mail im HTML-Format kann aktive Inhalte mit Schadensfunktion enthalten.

Nur vertrauenswürdige E-Mail-Attachments öffnen (z. B. nach tel.

Absprache). Es ist zu beachten, dass die Art des Datei-Anhangs

(Attachment) bei Sabotageangriffen oft getarnt ist und über ein Icon nicht sicher erkannt werden kann.

Durch Beachtung der folgenden Maßnahmen kann die Gefahr reduziert werden, dass ein Endanwender unabsichtlich Computer-Viren verteilt.

E-Mails nicht im HTML-Format versenden, auch wenn es vom eingesetzten Mail-Programm her möglich wäre; ebenso sind aktive Inhalte in E-Mails zu vermeiden.

WinWord-Dokumente im RTF-Format versenden (Damit wird auch die Weiterleitung von ggf. vertraulichen Informationen im nicht direkt sichtbaren Verwaltungsteil der DOC-Datei verhindert.)

Keine unnötigen E-Mails mit Scherz-Programmen und ähnlichem versenden, da diese evtl. einen Computer-Virus enthalten können.

Daten und Programme, die aus dem Internet abgerufen werden, stellen einen Hauptverbreitungsweg für Computer-Viren und Trojanische Pferde dar, um Benutzerdaten auszuspähen, weiterzuleiten, zu verändern oder zu löschen. Es muss darauf hingewiesen werden, dass auch Office-Dokumente (Text-, Tabellen- und Präsentations-Dateien) Makro-Viren enthalten können.

Programme sollten nur von vertrauenswürdigen Seiten geladen werden, also insbesondere von den Originalseiten der Softwarefirma. Private Homepages, die bei anonymen Webspace-Providern eingerichtet werden, stellen hierbei eine besondere Gefahr dar.

Die Angabe der Größe von Dateien, sowie einer evtl. auch angegebenen Prüfsumme, sollte nach einem Download immer überprüft werden. Bei Abweichungen von der vorgegebenen Größe oder Prüfsumme ist zu vermuten, dass unzulässige Veränderungen, meist durch Viren, vorgenommen worden sind. Daher sollten solche Dateien sofort gelöscht werden.

Mit einem aktuellen Viren-Schutzprogramm sollten vor der Installation die Dateien immer überprüft werden.

Gepackte (komprimierte) Dateien sollten erst entpackt und auf Viren überprüft werden. Installierte Entpackungsprogramme sollten so konfiguriert sein, dass zu entpackende Dateien nicht automatisch gestartet werden.

### **3.2. Schlussfolgerungen**

Eine bekannte Weisheit im Computer-Sicherheits-Bereich ist es, alles zu sperren, dann einzelne Bereiche zu öffnen. Auf Grundlage dieser Maxime haben wir unsere Firewall konfiguriert,

Außerdem sollte auf der Firewall nur die nötigste Software laufen, dies war nicht konsequent umsetzbar, da wir ein System aus Firewall, Router und http Dämon installierten.

Die Gefahr durch Hackerangriffe kann aber als gering angesehen werden, da die Telekom bei jeder Einwahl dynamisch eine IP Adresse vergibt. Da wir ICMP (und somit ‚Ping‘) verbieten und fast alle Ports sperren, wird es für Hacker sehr schwer in das System einzubrechen.

Außerdem lässt sich das System nur mit lokalen IP Adressen der Form 192.168.X.X konfigurieren, diese Adressen werden im Internet nicht geroutet.

Die weitaus größere Gefahr geht von empfangenen E-Mails aus. Um diese Gefahr zu minimieren, müssen die Mitarbeiter über die möglichen Risiken informiert werden.

Hundertprozentige Sicherheit gibt es ohnehin nicht, der Systemadministrator kann nur die Sicherheit relativ erhöhen und damit den Aufwand für den Hacker so stark erhöhen, dass er in der Firewall kein lohnendes Angriffsobjekt sieht.

### **3.3. Sicherheitstest**

Um die Funktion der Firewall zu untersuchen, werden wir Portscanner einsetzen. Zudem werden wir die Firewall ‚anpingen‘. Sollte die Firewall nicht auf ‚Ping‘ antworten und zeigt der Portscan nur die explizit geöffneten Ports, kann das System als hinreichend sicher betrachtet werden.

### **3.4. Literatur**

Hackers Guide, anonymmer Author

Hackers Black Book, anonymmer Author

Hackerland, Dennis Moschito, Evrin Sen, unbekannter Verlag

<http://www.computer-security.de/>

<http://www.computec.ch/>

<http://www.firmen-info.de/firewall/>

<http://www.little-idiot.de/>

<http://www.securityfocus.com/>

<http://www.sicherheit-im-internet.de>

<http://www.tux-server.de/howto/ipchains.html>

## 4. Router:

Bei dem verwendeten Router handelt es sich um den sog. "fli4l". Dieses ist ein Open Source Projekt, welches ständig weiterentwickelt wird und speziell für die Einwahl mittels DSL Technologien geschrieben wurde.

Die Intention des Projektes ist, mit möglichst wenig Hardware Aufwand ein Maximum an Komfort und Sicherheit auf Seiten des Software Routers vorzuhalten. So nimmt dieser Software Router – beschränkt man sich auf die wesentlichen Funktionalitäten – lediglich eine Diskette in Anspruch. Ähnlich niedrig sind die Anforderungen an die benötigte Hardware. Ein in die Jahre gekommener und für anderweitige Aufgaben unbrauchbar gewordener Rechner der 486er Generation mit 8MB Speicher sowie einem Diskettenlaufwerk reichen vollkommen zum Betrieb der Software aus. Aus diesem Grund wird bei diesem Projekt auch von einem „1 Disketten Router“ gesprochen.

Komplettiert man die oben beschriebene Hardwarekonfiguration noch mit zwei Linux kompatiblen Netzwerkkarten, ist der Rechner fertig für den Betrieb als vollwertigen Software Router – selbst den Einbau einer Festplatte kann man sich in diesem Fall sparen, da die Software, die im Prinzip auf dem Linux Kernel 2.2.16 beruht, das komplette System in einer dynamisch eingerichteten RAM-Disk installiert und auch dort verbleibt. Die Startdiskette – auf der sich auch die zur Einwahl benötigten Zugangsdaten inklusive dem Passwort befindet, kann nach dem erfolgreichen Bootvorgang aus dem Laufwerk genommen und sicher im Tresor verwahrt werden – um bei Bedarf, z.B. nach einem Stromausfall, einfach wieder zur Herstellung des Normalzustandes des Systems eingesetzt zu werden. Die Eingabe weiterer Passworte, das Starten von zusätzlichen Programmen oder eine komplizierte Konfiguration entfallen, so dass sich dieser Software Router vor allem in den Zusammenhängen anbietet, in denen nicht permanent ein Netzwerkadministrator seinen Dienst verrichtet und einfache Tätigkeiten am Netzwerk auch von ungeübten Benutzern durchgeführt werden müssen.

### 4.1. *Installation des Routers*

Das Gesamtpaket des Routers ist auf der deutschsprachigen Internetseite des Projektes unter der Adresse <http://www.fli4l.de> zum Download freigegeben. Neben dem eigentlich Quellcode, der im Grunde zur Installation ausreicht, gibt es zusätzlich ein Administrationspaket, welches unter jedem beliebigen Windowsrechner ab der Version 95 lauffähig ist. Hiermit lässt sich die Startdiskette des Routers auch ohne größere Kenntnisse von Linux und dem Eingriff in komplizierte Konfigurationsdateien über eine grafische Benutzeroberfläche erstellen. Hierzu müssen lediglich die Zugangsdaten zum Provider bekannt sein, unterstützt werden bisher die Deutsche Telekom sowie der Österreichische Ableger, sowie Grundinformationen zum eigenen Netzwerk, wie statische IP Adressen der Client PCs müssen bekannt sein, um erfolgreich die Routing Tabelle anlegen zu können.

Wenn die Eingabe der Netzwerk- und Providerinformationen abgeschlossen ist, muss lediglich eine leere und formatierte Diskette eingelegt werden und der Prozess des Erstellens gestartet werden.

#### **4.2. Optionale Pakete des Software Routers**

Sind die Grundpakete des fli4l auf der Diskette untergebracht, verfügt der geneigte Benutzer zwar über einen lauffähigen Router, aber es verbleiben noch genügend Kilobytes auf der Diskette, um durch optionale Pakete das Leben im Netzwerk sowie die Administration des Routers angenehmer zu gestalten.

#### **4.3. Webadministration**

Bereits wenige Kilobyte reichen aus, um durch das Hinzufügen eines kleinen Webservers die Administration des Routers auch aus der Ferne zu gestatten. Hierbei ist es in der Grundausstattung bereits möglich, sich über den Status der Einwahl zu informieren, sowie den Router entweder zur Einwahl oder zum Beenden der Verbindung zu bewegen. Außerdem wird hierbei auch ein einfaches CGI (Common Gateway Interface) installiert, welches eine Kommunikationsschnittstelle u.a. zur Firewall und zu den Routing Funktionalitäten liefert, weshalb es im Rahmen unseres Projektes auch notwendig ist, dieses Paket mit zu installieren.

#### **4.4. FTP**

Mit weiteren wenigen Kilobyte Ballast lässt sich ein kleiner FTP Service installieren, der es ermöglicht sich aus dem internen Netz via FTP Client, wie er z.B. als Kommandozeilen Version von Windows mitgeliefert wird, auf dem Server einzuwählen und dort Skripte o.ä. zu hinterlassen. Auch von diesem Paket haben wir in der Test- bzw. Entwicklungsphase regen Gebrauch gemacht, um nicht immer wieder von Neuem eine Startdiskette neu erstellen zu müssen, wenn sich eines der für die Schnittstellen benötigten CGI-Skripte geändert hatten.

#### **4.5. Telnet**

Eine weitere sinnvolle Option ist der Telnet Dämon. Hierdurch ist es möglich, sich mit Telnet auf dem Server einzuloggen. Neben der Möglichkeit, die via FTP hochgeladenen Skripte mit den entsprechenden Benutzerrechten zu versehen, gibt es zusätzlich ein Programm namens „lmonc“, das einen kleinen Administrationsmonitor liefert. Neben Einwahl oder dem Auflegen ist hier zusätzlich auch die Möglichkeit zum Neustart des Systems gegeben.

#### **4.6. Zeit Service**

Eigentlich nicht relevant im Rahmen des Projektes, aber dennoch nützlich, ist das „Time“ Paket, mit welchem der Server in die Lage versetzt wird, sich die jeweils aktuelle Uhrzeit von einem NTPS Server aus dem Internet zu holen.

## **4.7. Kommunikationsschnittstellen**

Um den auf Projektseite sich und damit auf einem anderen Server befindlichen Servlets eine Kommunikation mit dem Router zu ermöglichen, sind wohldefinierte Schnittstellen notwendig. Bereits in der Beschreibung der zusätzlichen Pakete haben wir angesprochen, dass zum Betrieb unseres Projektes der Webserver auf dem Router installiert sein muss. Dieser Webserver fungiert als Schnittstelle zwischen der Kommandozeile des Routers, der sog. *bash*, und unseren Administrationservlets.

## **4.8. Integration im Intranet**

### **4.8.1. Grundinformation**

Anwendungsszenario wird in den allermeisten Fällen ein Netzwerk mit eventuell bereits vorhandenen Intranet sein. Basiert diese Intranetlösung bereits auf der Servlet Technologie, so ist die Integration in das bestehende System denkbar, da die Servlet Klassen nur in den dafür vorgesehenen Ordner des Webserver bzw. der Servlet-Engine kopiert werden müssen. Komplettiert wird die Installation durch kopieren der HTML Dateien in das Webverzeichnis des Servers. Dabei muss aber darauf geachtet werden, dass die Pfade zum Intranetserver bzw. der Port der Servlet-Engine angepasst werden.

Ist noch keine Intranetlösung vorhanden, so muss spätestens zu diesem Zeitpunkt ein Rechner dafür ausgewählt werden. Aus dem Blickwinkel der Sicherheitsaspekte - siehe dazu auch das Kapitel „Firewall“ - bietet es sich an, dafür einen eigenen Rechner dafür bereitzustellen.

Die im xDSL Projekt verwendete Servlet-Engine ist der Tomcat in der Version 3.2. Neuere Versionen wären nur als Beta Version erhältlich gewesen. In der Geschichte der Web- bzw. Applicationserver ist der Tomcat ein noch recht junges Projekt, welches eine Fortführung des Apache Webserver unter Integration der Jserv Engine darstellt, welche nicht mehr weiterentwickelt wird.

Tomcat lässt sich auch zusammen mit Apache als Webserver installieren, was aber nur einen Sinn ergibt, wenn die vom Webserver in der Mehrheit bearbeiteten Daten HTML Dateien sind, da der Apache seine Geschwindigkeit nur bei statischen Seiten voll ausspielen kann. Im Gegensatz dazu kann Tomcat neben der Ausführung von Servlets auch statische HTML Seiten verwalten, so dass wir uns für eine Stand-Alone Lösung mit Tomcat entschieden haben. Die Menge der statischen HTML Seiten in unserem Projektteil ist äußerst übersichtlich und fällt in der Gesamtheit nicht ins Gewicht.

## 4.8.2. Installation der Tomcat Version 3.2

Tomcat ist eine reine Java-Lösung, was bedeutet, dass auf dem ausführenden Rechner sowohl JDK, als auch JSDK installiert sein müssen. Danach kann das Tomcat Packet, das über die erstaunliche Größe von unter vier Mbyte verfügt, entpackt und so installiert werden. Unter Windows müssen dann noch die Umgebungsvariablen bzw. die Pfade richtig gesetzt werden und der Tomcat durch die Eingabe von „startup“ gestartet werden. Nun sollte man einen zumindest lauffähigen Server besitzen. Es ist selbstverständlich auch möglich, zur Kapselung des Administrationstools gegenüber den restlichen Intranetdateien eine eigene Webapplikation zu definieren. Eine Beschreibung dessen, über den Aufbau der Konfigurationsdateien des Tomcats Server.xml und insbesondere der Applikationskonfigurationsdatei web.xml, würde den Rahmen dieses Kapitels sprengen und den Fokus zu sehr vom eigentlich Thema der xDSL Technologien nehmen. Aus diesem Grunde verweisen wir bei der Frage nach der weiteren Konfiguration auf die einschlägige Fachliteratur zu diesem Thema bzw. auf die im Grunde ausreichende Dokumentation die sich im Internet auf der Seite des Tomcat Projektes unter <http://jakarta.apache.org/> befindet.

## 4.8.3. Installation des Admin-Tools

Um das Administrationstool zu installieren, muss man folgende Schritte beachten:

Als ersten Schritt sollte man das mitgelieferte JDK1.2, das man auf der CD unter D:\JDK1.2 finden, auf Ihrem System installieren.

Bei der Installation muss man nur ein Verzeichnis angeben, in welches das JDK installiert werden soll. Um eine einfache Handhabung zu haben, sollte man dieses in das Verzeichnis C:\JDK1.2 installieren.

Als nächsten Schritt kopiert man das Verzeichnis „xdsl“ von der CD auf die Festplatte nach C:\xdsl. In diesem Verzeichnis sind alle Java-Sourcen unter \src, die Java-Classes findet man unter \classes, alle benötigten Bibliotheken liegen unter \lib, unsere Userinstellungen liegen im \properties Verzeichnis und die Protokolldatei liegt im \log Verzeichnis.

Als nächstes muss man den Webserver Tomcat installieren. Diesen findet man auf der CD in dem Verzeichnis \Tomcat.

Eine detailliertere Installationsanleitung entnimmt man dem obigen Kapitel „Installation der Tomcat Version 3.2“

Das Root-Verzeichnis sollte auf C:\xdsl\html\_doc gesetzt werden. Ansonsten muss man auch hier dieses Verzeichnis in das Standardverzeichnis vom Tomcat kopieren.

Nun müssen die fehlenden Bibliotheken im Tomcat Classpath eingebunden werden.

Nun öffnet man eine Shell und gehen in das bin-Verzeichnis des Tomcats und starten den Tomcat, indem man startup eintippt und mit Return bestätigt.

Nun öffnet der Tomcat eine weitere Shell, in dem der Tomcat gestartet bleibt und sein Ereignisprotokoll schreibt.

Wenn man den Tomcat schließen will, geht man wieder in die erste Shell bzw. öffnet eine neue und gibt im bin-Verzeichnis shutdown ein.

Solange der Webserver läuft, kann man über jeden Webbrowser, der Frames unterstützt, den Router administrieren, in dem man die index.html Seite des Tomcats aufruft.

Zum Einloggen kann man den User „xdsI\_Benutzer“ mit dem Passwort „xD1!“ benutzen. Falls man eine Fehlermeldung bekommt, überprüft man, ob sich die „User.txt“ Datei im Verzeichnis „C:\xdsI\properties“ befindet. Falls dem so ist, kann man die Protokolldatei „Logging.log“ im \xdsIlog Verzeichnis einsehen. Dort werden alle Aktivitäten des Administrationstools mitgeschrieben.

## 5. Testklassen

### 5.1. Testfälle und Testklassen, wofür sind Sie gut?

Zu jedem Projekt gehört ein bestimmter Sicherheitsstandard.

Bei Softwareprojekten ist dieser Sicherheitsstandard in der Fehlerfreiheit und Absturzsicherheit zu finden.

Daher ist es wichtig, gleich mit Erstellung des Lasten- und Pflichtenheftes auch die Testfälle zu bestimmen. Somit kann zu jedem Zeitpunkt des Projektes die Einhaltung des Lastenheftes überprüft werden.

Bei kleineren Projekten könnte man zwar auch diese Testfälle selber „Händisch“ testen, aber mit diesen Klassen ist gewährleistet, dass immer die selben Fälle getestet werden und jeder Fehler, und sei er auch nur im Design, sofort aufgedeckt und protokolliert wird.

Dies ist wichtig, damit man bei Abgabe des Projektes dem Kunden die Testprotokolle mitliefert und dieser sich selber von der Vereinbarten Fehlerfreiheit überzeugen kann. Treten dennoch Fehler im Programm auf, kann der Kunde bei Fehlern, die auch mit den Testfällen abgedeckt worden sind, eine Verbesserung fordern. Ist der Fehler in einem Bereich, der nicht mit den Testfällen abgedeckt wurde, muss er ein neues Projekt in Auftrag geben, um diese Fehler beheben zu lassen. Somit ist es wichtig, alle wichtigen Testfälle (Szenarien) schon im Pflichtenheft festzuhalten.

Daher haben wir uns auch dazu entschlossen bei unserem Projekt Testfälle mit Testklassen abzudecken, die zu jeder Zeit gestartet werden können und alle Fehlermeldungen in eine Protokoll-Datei schreiben.

Diese Testfälle werden nach jeder Änderung am System, sei es auch nur ein geänderter Zeilencode, gestartet. Somit ist zu jeder Zeit die Sicherheit da, dass keine neuen Fehler auftreten, wenn ein anderer Fehler behoben wurde.

Auch bei Veränderungen im Hardwaresystem werden die Testklassen gestartet, da jede Hardware anders reagiert und somit auch Fehler in einem Rechnernetz bedeuten kann.

Um eine bessere Fehlerfreiheit bzw. Fehlermeldung zu erreichen, kann man diese Testklassen auch zeitgesteuert aufrufen lassen. Somit kann die Testklasse jede Viertelstunde gestartet werden und bei einem Fehler sofort

dem Systemadministrator eine Fehlermeldung zukommen lassen. Sei es via Email oder durch Aktivieren einer Fehlermeldung auf dem Bildschirm. In diesen Fällen kann der Administrator sofort den Fehler beheben und seine Mitbenutzer verständigen.

## **5.2. Erstellung der Testfälle**

### **5.2.1. Router**

Als erstes stand der Funktionsumfang des Routers zur Diskussion. Hierbei sollte natürlich jede Funktion des Routers in mehr oder weniger gründlichen Testroutinen getestet werden.

Da kein Programm zu 100 Prozent getestet werden kann, haben wir die Hauptaufgaben des Routers versucht, so gründlich wie in diesem Projektumfang möglich, in die Testfälle aufzunehmen.

Daher haben wir uns dazu entschlossen, die zu testenden IPs freizugeben und über diese dann eine externe Webseite aufzurufen. Dies sollte laut des Funktionsumfangs des Routers möglich sein.

Danach wird die IP des Rechners gesperrt und wieder eine Seite im Internet aufgerufen. Jetzt sollte der Router keine Verbindung zulassen.

Um diesen Test auf alle IPs in einem Rechnernetz auszudehnen, bedarf es einem größeren Aufwand. Man müsste entweder ein RMI gestütztes System schreiben, welches auf jedem Rechner installiert ist und bei dem Test im richtigen Augenblick eine Internetseite aufruft. Oder man müsste eine Telnetverbindung zu diesem Rechner öffnen, um über diesen einen „Ping“ zu einem Rechner, der im Internet steht, abzusenden.

Beide Methoden sind sehr programmieraufwändig. Daher haben wir uns dazu entschlossen den Test der IP Freischaltung und Sperrung nur mit dem Rechner durchzuführen, auf dem die Testklassen laufen.

Der Test der IP Freischaltung und Sperrung wird wie folgt aussehen:

Lokale IP beim Router sperren und zwei Verbindungen öffnen, eine ins Internet und eine im Intranet. Wobei die erste Verbindung fehlschlagen muss und die zweite hergestellt werden muss. Danach wird die IP nochmals gesperrt und der Test wiederholt. Dies ist nötig, damit auch keine Fehler auftreten, wenn eine IP von zwei Personen gleichzeitig (nacheinander) gesperrt wird. Falls eine Fehlermeldung vom Router kommen sollte, wird diese sofort protokolliert. Nun wird die IP wieder freigegeben und die beiden Verbindungen wieder aufgebaut. Jetzt müssen beide Verbindungen hergestellt werden. Auch hierbei wird die IP noch mal freigeschaltet und der Test wiederholt.

Diese Systematik ist auch für diesen Umfang eines Projektes tragbar, da wir nur ein Intranets mit ca. 10 Rechnern haben und wir davon ausgehen können, dass der Router alle IPs gleich behandelt.

Falls Widersprüche auftreten, kann man diese Testklassen auch von verschiedenen Rechnern gleichzeitig bzw. nacheinander laufen lassen und bekommt somit die Tests für verschiedene Rechner IPs.

## 5.2.2. Firewall

Als nächstes stand die Funktionalität der Firewall zur Diskussion.

Hierbei war die Findung der Testfälle nicht gerade einfacher.

Da die Hauptfunktionalität einer Firewall das Sperren und Freigeben einzelner Ports einer bzw. mehrerer IPs ist. Somit mussten wir uns wieder darüber einig werden, ob wir die Tests auf eine IP beschränken wie bei den Routertests, oder ob wir hierbei den großen Aufwand in Kauf nehmen können, um alle IPs abzudecken.

Wir haben uns auch hierbei dazu entschlossen die Tests nur auf eine IP zu beschränken, da wir auch hier von einer einheitlichen Funktionalität der Firewall ausgehen können bzw. bei Widersprüchlichkeiten der Firewall auch die Testklassen auf verschiedenen Rechnern laufen lassen können.

Als die Frage der IPs geklärt war, kam eine noch größere Frage auf uns zu. Sollte man alle Ports testen oder genügt es einige wichtige Port unter die Lupe zu nehmen? Bzw. kann man auch einen möglichen Angriff auf einem bestimmten Port von außen simulieren?

Um die Lesbarkeit des Testprotokolls zu erhalten, haben wir uns dazu entschieden, nur die wichtigsten Ports zu testen. Diese Ports sind 80 für die allgemeine http-Verbindung, 8080 für die Proxy-Verbindung, 110 für eine Telnet-Verbindung, 21 für eine ftp-Verbindung, 25 für eine SMTP-Verbindung und 134 für eine Pop3-Verbindung. All diese Verbindungen sollten zwar offen sein, um richtig im Internet arbeiten zu können. Aber gerade der Telnet und FTP Port sollte von außen nicht erreichbar sein.

Dies stellt auch gleich wieder die Frage, ob wir einen Angriff von Außen simulieren können und in unseren Testprotokollen mitschreiben können.

Aber leider bräuchten wir hierfür zwei verschiedene Netzzugänge , um eine eigene Testinstanz auf einem Rechner des zweiten Netzes laufen zu lassen.

Aber leider hatten wir in unserem Testumfeld keinerlei Möglichkeiten dies zu simulieren. Bei Installation im RDÜ-Labor kann über einen solchen Testfall noch nachgedacht werden.

Bis dahin haben wir uns auf folgende Testfälle konzentriert :

Alle Ports sperren und die oben genannten Ports auf Ihre Durchlässigkeit zu prüfen. Danach werden die o.g. Ports wieder freigeschaltet und wiederum auf ihre Durchlässigkeit geprüft. Als letztes sperren wir jeden einzelnen Port und prüfen, ob er noch eine Verbindung ins Internet herstellen kann.

Danach wird ein Port noch genauer getestet. Dieser Port wird, obwohl er schon gesperrt ist, noch einmal gesperrt. Hierbei darf die Firewall keine Fehlermeldung geben. Zusätzlich wird geprüft, ob die Firewall den Port immer noch gesperrt hat.

Nun wird der Port zweimal nacheinander freigeschaltet und jeweils geprüft, ob die Firewall keine Fehler meldet bzw. den Port auch wirklich freigeschaltet hat.

### 5.3. Erstellen der Testklassen

Beim Erstellen der Testklassen sollte als erstes angemerkt werden, dass alle Testklassen von einem Testclient aufgerufen werden. Dadurch können wir bei einem Testdurchlauf alle Testklassen nacheinander durchlaufen lassen.

Dieser Testclient hat einige Übergabeparameter, um die erforderlichen Angaben über Testklassen anzugeben. In einem Statischen Testsystem könnte man diese Sicherlich auch direkt in den Quelltext schreiben. Aber da wir verschiedene Testfälle und somit auch Klassen haben, schien es für uns einfacher, dem Testclient bei jedem Aufruf die Testklassen als Parameter mitzugeben. Bei der statischen Methode müsste der Quell-Code bei jeder Änderung erneut kompiliert werden.

In unserem Fall gibt es verschiedene Arten, die Funktionen des Routers und der Firewall zu testen. Entweder man testet die Funktionen direkt mit den CGI-Scripten, welche auf dem Rechner des Routers/Firewall liegen oder man testet sie mit den Klassen, welche von unserem Admin-Tool verwendet werden, da diese auch direkt auf unsere CGIs zugreifen. Wir haben uns dazu entschieden, nur den ersten Wege zu beschreiten. Dies hat zur Folge, dass man bei einem fehlgeschlagenen Testfall den auftretenden Fehler schlechter einkreisen kann.

Bei Testklassen, die beide Varianten testen würden, könnte man schneller sehen, in welchem Teil (CGI oder Java-Klasse) der Fehler liegt. So muss die Java-Klasse erst debuggt bzw. mit Ausgaben versehen werden, um zu sehen, ob die Schnittstelle zwischen dem Router bzw. Firewall und den Java-Klassen richtig funktioniert. Die Tests für unser das Middle-Tier“ unseres Admin-Tool findet man unter *xdsl.tools.testtool.UserTest.java*. Die Dateiangabe ist in der Java spezifischen Notation geschrieben. Jeder Name vor der Java-Datei bezeichnet ein Package. Dies ist mit den Unterverzeichnissen gleichzusetzen. Die Dateien sind in dem Verzeichnis „D:\java\xds\src\xds\tools\testtool“ zu finden. Die kompilierten Klassen sollten Sie unter „D:\java\xds\classes\xds\tools\testtool“ finden. Aber auch bei voll funktionsfähigen Klassen kann es passieren, dass der Administrator den Router bzw. Firewall nicht einstellen kann. Dies kann an einen Fehler an dem Servlet-Programm liegen oder an der Servlet- Engine.

Daher sollte man diese Testfälle nicht nur direkt an den dafür zuständigen Klassen testen, sondern auch über das GUI (Graphical User Interface).

Diese Art des Testens ist etwas schwieriger, da man bei einem GUI den Benutzer anmelden und alle relevanten Daten in einer Session speichern muss. Daraus resultiert eine schwierigere Aufrufmethodik. Man muss die Aufrufparameter aus der erhaltenden HTML-Seite heraus parsen und diese wiederum aufrufen. Somit kann man auch das GUI auf seine Funktionalität überprüfen.

Auf der anderen Seite kann man durch einfaches Aufrufen der einzelnen Seiten schon heraus finden, ob die Servlet-Engine reagiert, oder ob diese einen Fehler hat, der sich durch einen einfachen Reboot beheben lässt.

Daher haben wir uns dazu entschlossen, das GUI nur in bestimmten Punkten zu testen.

Als erster und einfachster Test wird das GUI mit einem Usernamen und Passwort aufgerufen um zu sehen, ob die Servlet-Engine reagiert und den Benutzer erkennt.

Als zweiten Test wird die lokale IP freigeschaltet, und es wird eine Verbindung ins Internet hergestellt. Nachdem die Verbindung hergestellt werden konnte, wird die lokale IP gesperrt und erneut wird versucht, diese Verbindung herzustellen. Jetzt sollte der Router keine Verbindung zulassen.

Durch diesen einfachgehaltenen Test kann festgestellt werden, ob das GUI reagiert und dem User es erlaubt, IPs zu verändern.

Es ist somit nur ein sogenannter „still-alive-Test“, da nur getestet wird, ob die Anwendung noch „lebt“. Es ist zwar auch ein funktionaler Test, aber in einem sehr geringen Umfang.

Diesen Testfall finden Sie in der Java-Datei *xdsl.tools.testtool.ConnectionTest*. Für weitere Informationen ist der Quelltext mit Inline-Dokumentation im Anhang A.1.3 zu finden.

Diese Testklasse sollte in bestimmten Abständen immer wieder aufgerufen werden, um einen Absturz, der unbemerkt bleibt, zu vermeiden.

## 5.4. Vor- und Nacharbeiten bei Tests

Da wir uns für eine dynamische Variante des Testclients entschieden haben, müssen wir bei jedem Test dem Testclient die Testklassen als Parameter mit übergeben. Diese Aufrufe des Testclients mit verschiedenen Testklassen (Testfällen) sollte praktischerweise mittels einer Batch-Datei gemacht werden. Da man sich somit eine Reihe von Testszenarien als Batch-Dateien anlegen und bei Bedarf einfach ausführen kann.

Will man die Testklassen von einem anderem Rechner starten, muss man leider den Quellcode der Testklassen einmal erneut kompilieren, da es eine statische Variable gibt, die die lokale IP-Adresse des Rechners beinhaltet. Diese ist wichtig, da wir nur die lokale IP sperren und testen können. Man hätte auch diesen Parameter mit in die Übergabeparameter übernehmen können. Aber da es in einem Testsystem entweder immer von einem Rechner getestet wird bzw. alle Entwickler ein eigenes Testsystem haben und diese auch immer einen Compiler auf dem Rechner haben, ist es übertrieben, diesen Parameter bei jedem Start mit zu übergeben.

Ein Aufruf des Testclients könnte wie folgt aussehen :

*Javaw -classpath*

```
"D:\java\xdsl\Java\classes;D:\java\Vib\servlet.jar;D:\java\Vib\junit3.2\junit.jar;
D:\java\Vib\xerces.jar;D:\java\Vib\jaxp-1.1\crimson.jar;D:\java\Vib\jaxp-
1.1\jaxp.jar;D:\java\Vib\jaxp-
1.1\xalan.jar;D:\java\Vib\xml.jar;D:\java\Vib\vt.jar;D:\java\Vib\v18n.jar;D:\java\Vib
\dt.jar;D:\java\Vib\tools.jar;D:\java\Vib\jpda.jar;D:\java\Vib\viimp.jar"
xdsl.tools.testtool.TestClient xdsl.tools.testtool.UserTest
xdsl.tools.testtool.ConnectionTest
```

Das Programm Javaw.exe findet man in dem Verzeichnis vom Jdk1.2. Alle Angaben, die in den Anführungszeichen hinter *classpath* stehen, beschreiben die benötigten Bibliotheken. Hinter diesem Parameter steht die auszuführende Javaklasse (TestClient). Alle dahinterstehenden Angaben sind die Übergabeparameter für den Testclient. In diesem

Beispiel wurden zwei Testklassen (UserTest und ConnectionTest) übergeben.

Diesen Aufruf findet man auch unter „D:\java\User\_Connection\_Test.bat“ auf der CD. In diesem Verzeichnis finden Sie auch noch mehrere Batch-Dateien zum Aufrufen eines Testfalles. Hierbei ist zu beachten, dass man alle Angaben im classpath überprüft und gegebenenfalls ersetzen muss, da die Angaben alle auf das Laufwerk D zeigen. Aber in einem anderen Testsystem kann dieses Laufwerk auch ganz anders heißen. Falls man selbst mit der Abänderung dieser Dateien immer noch Fehler bekommt, sollten man die Jdk1.2 Installation überprüfen und sicherstellen, dass das Jdk1.2 Verzeichnis im System bekannt ist, da ansonsten die Javaw Datei nicht gefunden wird.

Beide Testfälle werden durchgetestet und schreiben alle Ihre Ergebnisse bzw. Meldungen in die Protokolldatei. Diese findet man unter *D:\java\Vog\Logging.log*“. Nach einem erfolgreichem Test sollte der Testclient als letztes in die Protokolldatei folgende Zeilen schreiben :

```
(Mo 28.05.2001)10:14:29(120) ; info ; ALL TESTS OK! (Total: 2 tests)
(Mo 28.05.2001)10:14:29(170) ; info ; All Tests for xdsl.tools.testtool.ValidatorTest
completed in 0,22 sec.
(Mo 28.05.2001)10:14:29(170) ; info ;+++++
(Mo 28.05.2001)10:14:29(230) ; info ; TEST-REPORT for Mon May 28
10:14:29 GMT+02:00 2001
(Mo 28.05.2001)10:14:29(230) ; info ; +-----+
(Mo 28.05.2001)10:14:29(230) ; info ; Total TestClasses: 2
(Mo 28.05.2001)10:14:29(230) ; info ; Total TestCases : 10
(Mo 28.05.2001)10:14:29(280) ; info ; +-----+
(Mo 28.05.2001)10:14:29(280) ; info ; Total Failures : 0

(Mo 28.05.2001)10:14:29(280) ; info ; Total Errors : 0
(Mo 28.05.2001)10:14:29(280) ; info ; +-----+
(Mo 28.05.2001)10:14:29(340) ; info ; Testrun completed in 0,44 seconds.

(Mo 28.05.2001)10:14:29(340) ; info ;+++++
```

Mit dieser Ausgabe wird angezeigt, wie viele Testklassen abgearbeitet werden und wie viele Fehler bzw. Errors es bei diesen Tests gab. Detailliertere Angaben über eine solche Protokolldatei nach einem Test findet sich im Anhang A.1.6 und A.1.7

Nach einem Test sollte man die Protokoll-Datei unter einem sinnvollem Namen abspeichern, da diese beim nächsten Testdurchlauf gelöscht und erneut geschrieben wird. Darüber hinaus sollte man den Urzustand wieder herstellen, da alle Tests die IP und Port Einstellungen verändern werden. Somit kann es sein, dass einige Rechner nach dem Test anders reagieren als vor dem Test.

In unseren Tests wird zwar nur die lokale IP des Testrechners gesperrt, aber wenn man nun die Tests auf verschiedenen Rechnern startet, müssen die Einstellungen dieser Rechner wieder in den Urzustand gestellt werden. Daher macht es Sinn, einen eigenen Rechner für die Testklassen abzustellen bzw. einen Rechner zu nehmen, der niemals ins Internet gehen muss, wie z.B. ein Print-Server, um Folgefehler zu vermeiden.

## **5.5. Testberichte und Testergebnisse**

### **5.5.1. Wie und wann wurden die Testergebnisse aufgenommen**

In den ersten Wochen des Projektes, als wir die Schnittstellen festgelegt haben, konnten noch keinerlei Tests durchgeführt werden, da es noch keinerlei testfähigen Code gab.

Ab dem 2.5.01 haben wir angefangen, die Prototypen der einzelnen Gruppen zusammenzuführen und diese auch zu testen.

Mit Anfang unserer Testdurchläufe kam uns schnell eine Frage wie können wir diese Testergebnisse gut dokumentieren? Nicht jedes Testergebnis ist wichtig, sondern es sollte nur den aktuellen Stand wieder spiegeln.

Dies ist darüber hinaus notwendig, da wir keine 100 Seiten an Testergebnissen in den Anhang hängen wollten. Dies würde zwar jeden kleinsten Schritt unseres Projektes gut dokumentieren, hätte aber den Nachteil, dass keiner sich die Mühe machen würde, alle Testergebnisse durchzulesen.

Daher haben wir uns dazu entschieden, jede Woche jeweils eine Protokolldatei, der UserTest- und ConnectionTest Klassen, abzuspeichern.

Somit können wir den Verlauf unseres Projektes dokumentieren und halten die Protokolle in einem überschaubaren Rahmen.

Im Folgenden werden die Testergebnisse der einzelnen Wochen erläutert. Die dazugehörigen Protokolldateien befinden sich im Anhang bzw. auf der CD im „xDSL/log“ Verzeichnis. Jede Datei ist mit dem Datum des Mittwoches der jeweiligen Wochen und dem Namen der Testklasse gekennzeichnet.

### **5.5.2. Testergebnisse vom 02.05.2001**

Bei diesen Tests waren noch alle Dummydaten in unseren Prototypen. Daher wurden keine Exceptions ausgelöst, aber es gab immer wieder Fehler, da man eine Seite im Internet aufrufen konnte, da die IP Adressen noch nicht gesperrt wurden. Auch die Ports wurden noch nicht im realen Router verändert.

### **5.5.3. Testergebnisse vom 09.05.2001**

In dieser Woche hatten wir schon eine reale Anbindung an den Router vorgenommen.

Diese Anbindung war aber nur auf die IP-Settings beschränkt. Daher wurden keine Fehler mehr bei den IP-Änderungen angezeigt, aber leider immer noch den gesperrten Port. Dort waren immer noch unsere „Dummy-Daten“ im Einsatz.

#### **5.5.4. Testergebnisse vom 16.06.2001**

Nun war auch die Anbindung an die Firewall gelungen. Nur leider ergaben sich mehr Schwierigkeiten als wir dachten, da es nicht so einfach war, die Firewallregeln wieder aus der Firewall herauszunehmen bzw. zu überschreiben. Daher sind diesmal alle Testfälle, in denen versucht wird einen geschlossenen Port wieder freizugeben, fehlgeschlagen. Es wäre auch sicher möglich gewesen, die Firewall erst zu stoppen, da diese Methode alle Firewallregel heraus löscht. Aber leider war diese Methode auch noch nicht implementiert.

#### **5.5.5. Testergebnisse vom 23.05.2001**

In dieser Woche gab es die echte Anbindung an unsere Firewall auch mit der Sperreigenschaft. Nun konnten wir IP sperren und wieder freigeben und genauso auch bei den Port.  
Nun fehlten nur noch die Methoden zum Starten und Stoppen der Firewall und des Routers.

#### **5.5.6. Testergebnisse vom 30.05.2001**

Diese Woche sollte unsere Firewall endlich auch von anderen Usern (der Gruppe Soft- und Hardware-Router) getestet werden. Dafür war auch alles soweit fertig. Die Firewall und der Router konnten jetzt gestoppt und gestartet werden. Die IPs und Ports waren einzeln veränderbar, und es gab auch sonst keine weiteren Störungen. Nur leider, wie in unserem Bericht zu lesen, kam es nicht zu diesem Test.

### **5.6. Test durch die Gruppe „Hard- und Softwarerouter“**

In unserem Projekt „xDSL“ sollten alle Einzelprojekte mehr oder weniger eine große Projektarbeit ergeben. Daher lag die Idee sehr nahe, den von uns entwickelten Router mit seiner Firewallfunktionalität nicht nur durch Testklassen testen und bewerten zu lassen, sondern auch durch andere Gruppen unseres Projektes.

Hierzu schien uns die Gruppe „Hard- und Softwarerouter“ wie geschaffen, da sie sich die nötigen Vergleichswerte durch ihre Tests der kommerziellen Router angeeignet hat.

In Absprache mit der Gruppe sollte unsere Router-Version ab dem 30.5.2001 getestet werden. Für diesen Test haben wir versucht unseren Zeitplan um einige Wochen nach vorne zu verlegen und unseren Router schon vor dem 30.05.2001 im RDÜ-Labor zu installieren. Bei der

Installation gab es leider schwerwiegende Hardwareprobleme. Diese bestanden im wesentlichen darin, dass der TDSL-Zugangrechner, welcher im Labor steht, eine Dual-Netzwerkkarte hat, die nicht von Linux unterstützt wird. Auch auf der Webseite des Herstellers war kein Linuxsupport zu finden. Darüber hinaus waren die Bezeichnungen des Herstellers auf der Webseite gegensätzlich. Dazu kam, dass wir den „xDSL-Projekt-Rechner“ mit einer neuen Festplatte versehen und dort eine Win-NT Version installiert haben. Diese wollte bei der Installation die MAC-Adressen der vorhandenen Netzwerkkarten haben. Diese waren leider, nach unserem damaligem Wissen, nicht herauszufinden. Um die MAC-Adressen zu bekommen, haben wir den Rechner in seiner normalen Installation mit Windows 2000 gestartet und in den Netzwerkeinstellungen nachgesehen. Auch über einen „arp -a“ Aufruf konnten wir keine Adressen sehen. Nach einer Rücksprache mit der Gruppe haben wir erfahren, dass die Netzwerkkarten eine Besonderheit haben, ihre Mac-Adressen sind aufgedruckt. Da wir aber an dem anderen Problem (ohne neue Netzwerkkarten), die wir voraussichtlich am 16.06.2001 erhalten, nicht vorbei kamen, haben wir die Installation auf ihren ursprünglichen Termin zurückgesetzt.

Somit blieb als einzige Auswahlmöglichkeit nur noch ein Test in unserem heimischen Testaufbau. Welchen wir leider nicht mit in das RDÜ-Labor bringen konnten, da es in einem laufendem Betrieb eingesetzt und getestet wird.

Diese Möglichkeit zu testen war zwar gut, aber nach Meinung der Gruppe „Hard- und Software-Router“ leider nicht sehr aussagekräftig, da keine Vergleichsmöglichkeiten bestehen. Man kann keine zwei Router in zwei verschiedenen Netzen testen. Einen solchen Test kann man auch am Schreibtisch ohne Netz machen, da jedes Intranet verschieden schnell ist und mit unterschiedlichen Netzanbindungen versehen ist. Somit hätte man nur einen Vergleich des Funktionsumfangs erstellen können.

## 6. Admin-Tool Frontend

Das Admin-Tool soll in erster Linie dem Netzwerkadministrator eine benutzerfreundliche und einfache Umgebung schaffen, mit dem er den Router und die Firewall administrieren/konfigurieren kann.

Das Admin-Tool Frontend erhält und übergibt ausschließlich Werte (Variablen) auf die Schnittstellen des Admin-Tool Middletier.

### 6.1. *Beschreibung des Admin-Tool Frontend*

Das Admin-Tool bietet dem Administrator oder dem Anwender folgende Möglichkeiten:

- Login
- IP-Settings
- IP-Range
- Disable All
- Enable All
- Start Router
- Stop Router
- Start Firewall
- Stop Firewall
- User
- Logout

### 6.2. *Detailliertere Beschreibung der Menüpunkte:*

#### 6.2.1. Login

Beim Start des Admin-Tools wird der Benutzer aufgefordert, ein Login-Namen und ein Login-Passwort einzugeben. Dies soll bewirken, dass nicht jeder X-beliebige Benutzer die Möglichkeit hat, am Router und Firewall Änderungen vorzunehmen, sondern ausschließlich von autorisierten Benutzern. Nach dem Login wird der erfolgreich eingeloggte Benutzer an eine Session gebunden, um den Missbrauch z.B. nach einem Logout durch den „Zurück-Button“ zu verhindern.

Der Login-Name sowie das –Passwort werden dann an das Middletier weitergereicht und dort verifiziert. Bei einem fehlerhaften Login wird eine entsprechende Meldung ausgegeben.

#### 6.2.2. IP-Settings

Hier werden alle IP-Adressen, die Hostnamen zu der jeweiligen IP, der Status der jeweiligen IP (freigegeben/gesperrt) und die Möglichkeit zu jeder IP bestimmte Ports freizugeben bzw. zu sperren (Firewall-Regeln).

Bei einem „Klick“ auf den jeweiligen Status der IP-Adresse wird dieser sofort geändert. Man kann also direkt die PCs im Netz sperren bzw. freigeben.

Um Firewall-Regeln, also Ports, sperren oder freigeben, zu definieren ist der Button „modify“ vorgesehen. Dabei wird die aktuelle IP-Adresse angezeigt, damit keine Unklarheiten entstehen.

Zunächst wird eine Tabelle mit den sieben wichtigen Ports (21 FTP, 23 Telnet, 25 SMTP, 80 WWW, 110 POP3, 143 IMAP und 8080 alternativer WWW) angezeigt. Dort kann man die Auswahl freigeben bzw. sperren für den jeweiligen Port zu der IP-Adresse treffen.

Ferner ist es möglich, die anderen Ports im Bereich von 0 – 49151 zu sperren oder freizugeben. Dieses geschieht entweder durch die Eingabe des einzelnen Port oder durch Eingabe des Port-Bereichs. Z.B. 200 – 500 und die jeweilige Auswahl „freigeben“ oder „sperren“.

Die freigegebenen Ports werden dann an die Tabelle der sieben Ports angehängt.

Das Freigeben bzw. Sperren der Ports geschieht hierbei nur bei der ausgewählten IP-Adresse des Clients.

### **6.2.3. IP-Range**

Hier können Bereiche von IP-Adressen, also PCs, für den Netzwerkverkehr freigegeben oder gesperrt werden.

Bei der Eingabe des IP-Bereichs wird nur die Eingabe des letzten Segments der IP-Adressen, also die Hostadresse, benötigt. Das Middletier fügt die gesamte IP-Adresse zusammen.

Bei einem Netzwerk von bis zu 10 PCs, mit dem wir gearbeitet haben, ist dieses Feature zwar noch nicht so bedeutend, aber bei einem Netzwerk, wo z.B 100 PCs arbeiten, ist es dann doch sehr mühsam, einzelne zu sperren oder freizugeben.

### **6.2.4. Disable All**

Ermöglicht ein schnelles Sperren aller IPs im Netzwerk.

### **6.2.5. Enable All**

Ermöglicht ein schnelles Freigeben aller IPs im Netzwerk.

### **6.2.6. Start Router**

Es gibt zwei Wählverfahren des Routers: „Manuell“ und „Dial on demand“. Bei dem manuellen Wählverfahren muss der Administrator die Verbindung zum Internet selber herstellen. „Dial on demand“ reagiert auf jede Anfrage eines Clients zum Internet und wählt selbstständig.

Der Menüpunkt „Start Router“ setzt den Zustand von „Manuell“ auf „Dial on demand“.

### **6.2.7. Stop Router**

Beendet die Verbindung zum Internet durch ein Auflegen des Modems und das Wählverfahren wird von „Dial on demand“ auf „Manuell“ gesetzt.

### **6.2.8. Start Firewall**

Hier werden die Firewall-Regeln aktiviert.

Alle Ports bis auf die sieben Ports:

21 - FTP

23 - Telnet

25 - SMTP

80 - WWW

110 - POP3

143 - IMAP

8080 - alternativer WWW

werden geschlossen. Dieses gilt für jede IP im Netzwerk, d.h. alle Clients haben nur Rechte auf die Standard-Internetanwendungen.

### **6.2.9. Stop Firewall**

Stoppt alle Firewall-Regeln. Alle Ports werden geöffnet, somit haben alle Clients uneingeschränkten Zugriff ins Internet und sind nicht mehr sicher.

### **6.2.10. User**

Es können zusätzliche Konten für Administratoren durch Eingabe eines neuen Login-Namen und Login-Passwort angelegt werden. Der Name und das Passwort werden ebenfalls auf Korrektheit überprüft, d.h. es wurde genau definiert welche Zeichen für Login-Name und Login-Passwort gültig sind ([siehe 6.4.2](#)).

### **6.2.11. Logout**

Bei einem Logout wird die Session zerstört, und man muss sich erneut einloggen, um Veränderungen am Router und der Firewall vorzunehmen.

### **6.3. Vorgehensweise bei der Implementierung des Admin-Tools Frontend:**

Um die Implementierung des Frontends beginnen zu können, mussten erst die Schnittstellen des Middletiers festgelegt werden, da das Frontend auf diese zugreift.

Nachdem die Phase der Schnittstellenfestlegung abgeschlossen war, waren die Schnittstellen ersteinmal so stabil, dass mit der Implementierung des Frontends begonnen werden konnte.

Da das Middletier noch nicht die richtigen Daten des Routers lieferte, wurden zunächst Dummy-Daten an das Frontend geliefert bzw. vom Frontend an das Middletier. Somit war zunächst eine Implementierung am eigenen PC zu Hause möglich.

Da das Admin-Tool webbasierend ist, wurde es in Java (Servlets) und Html implementiert. Es gab bei der Java-Implementierung des Admin-Tools Frontend zwei Möglichkeiten. Die Implementation mit mehreren Servlets, oder die Implementation mit nur einem Servlet.

Mehrere Servlets haben den Vorteil, dass der Programmcode eben in diesen Verteilt wird und somit übersichtlicher wird. Der Nachteil ist der, dass auftretende Fehler schwerer zu finden sind und etwaige Änderungen, z.B. durch ein Übergabeparameter, sich durch alle Servlets ziehen.

Wir haben uns für ein Servlet entschieden. Obwohl hier der Programmcode grösser und evtl. unübersichtlicher ist. Daher ist der Programmcode gut ausdokumentiert (siehe CD → xds\src\xds\AdminServletHaupt.java).

Wichtig hierbei ist noch zu sagen, dass nach einem Login ein Benutzer an eine Session gebunden wird. Bei jedem Login wird geprüft, ob es eine gültige Session gibt. Dies hat z.B. den Vorteil, dass nach einem Logout nicht der „Button Back“ des Browsers missbraucht wird, um von jemanden, der nicht autorisiert ist, Änderungen vornehmen zu lassen.

Als dann das Middletier an die echten Daten vom Router angepasst wurde und somit diese auch an das Frontend lieferte, musste das Frontend ebenfalls angepasst werden. Dies war dann nicht mehr zu Hause möglich, so dass sich alle Projektteilnehmer bei einem Kommilitonen trafen, der den Platz und das nötige Equipment bereitstellte.

Bei der Anpassung an die „echten“ Daten stellte sich heraus, dass zuvor gut mit den Dummy-Daten gearbeitet wurde und es fast keine Änderungen gab.

### **6.4. Zusätzliche Implementierungen:**

#### **6.4.1. Template-Generator:**

Der Template-Generator hat die Aufgabe, Werte, die zuvor in einem Vector gespeichert wurden wie z.B. Die IP-Adressen, durch die Tags <eField> und <EObjectField> im Html-Quellcode zu ersetzen.

Die Reihenfolge spielt dabei keine Rolle. Wenn in einem EObjectField ein EField enthalten ist, ist es egal, ob es vor oder nach dem Einsetzen des EObjectFields gefüllt wird.

#### **6.4.2. Validator-Klasse:**

Die Validator-Klasse wird benötigt, um korrekte Eingaben durch den Benutzer zu gewährleisten. So wird z.B. beim Festlegen der Ports, die zu sperren sind, ob es sich nur um Zahlen handelt und keine Buchstaben eingegeben wurden. Der Port-Bereich, der gesperrt werden soll, wird ebenfalls überprüft, ob der von-Bereich nicht grösser als der bis-Bereich ist.

Ferner werden bei dem Hinzufügen von Benutzern darauf geachtet, dass der Login-Name nur die Zeichen:

„abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789\_“

und das Login-Passwort nur die Zeichen:

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789,,:-\_\*!?" enthalten darf.

## 7. Admin-Tool Middletier

Das Middletier stellt die Schnittstellen für das Admin-Tool Frontend zur Verfügung. Das Frontend benutzt diese Schnittstellen, um den Router bzw. die Firewall zu konfigurieren. Die User müssen in einer Datei, die in einem bestimmten Format abgespeichert wird, auf der Festplatte in einem festen Pfad bereitgestellt werden. Das Middletier benutzt diese Datei, um die User, die sich im Frontend einloggen, zu verifizieren.

### 7.1. Die Implementierung des Admin-Tool Middletiers

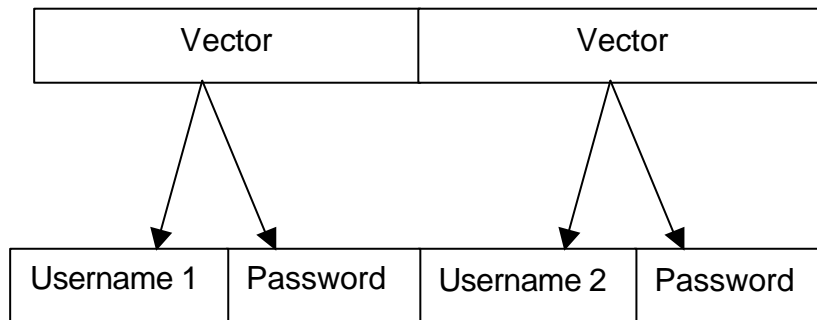
Die IP- bzw. Port-Adressen und die User, die in der User.txt abgespeichert sind, werden in Vektoren gespeichert. Der Vektor für die User heißt „*mUserList*“ und für die IP- und Port-Adressen „*mUser*“. Für die Darstellung der IP- und Port-Adressen beim Frontend werden die Methoden vom Middletier benutzt. Ändert sich der Zustand eines Ports bzw. einer IP-Adresse, wird über die Methode das jeweilige Skript aufgerufen und der Vektor aktualisiert.

Das Middletier besteht aus 4 Klassen:

#### 7.1.1. Klasse User

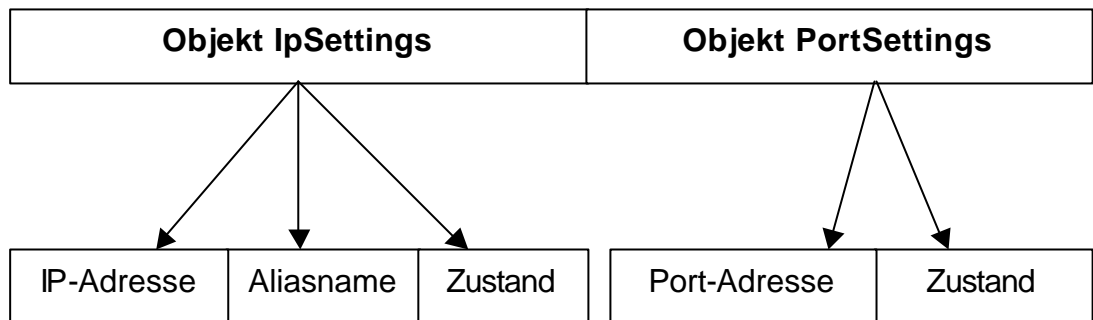
Die Klasse User alle Methoden zur Verfügung, die auf Klassen IpSettings, PortSettings und UserException zugreifen. In der Klasse User werden auch die beiden Vektoren „*mUserList*“ und „*mUser*“ verwaltet.

Der Vektor „*mUserList*“ für die User sieht folgendermaßen aus:



Der Username und das Passwort sind vom Typ *String*. **Bei der Eingabe des Passworts sollte auf Groß- und Kleinschreibung geachtet werden.**

Der Vektor „*mUser*“ für die IP- und Port-Adressen sieht folgendermaßen aus:



Der Vektor wird bei jedem Verändern einer IP- bzw. einer Port-Adresse aktualisiert. Da nur ein User zurzeit das Tool benutzen kann, sind die aktuellen IP- bzw. Port-Adressen im Vektor zwischengespeichert.

#### **Objekt IpSettings:**

Die IP-Adresse, der Aliasname und der Zustand sind vom Typ *String*. Die variable Zustand kann auch vom Typ *Boolean* sein. Da man in den Vektor keine Boolean-Werte abspeichern kann, müsste man bei jedem Abspeichern den *Boolean-Wert* in einen *String* umwandeln. Da man sich IP-Adressen schlechter merken kann, haben wir den Aliasnamen der IP-Adresse mit abgespeichert.

#### **Objekt PortSettings:**

Die Port-Adresse und der Zustand sind ebenfalls vom Typ *String*.

### **7.1.2. Klasse IpSettings**

In dieser Klasse wird die IP-Adresse gespeichert. Jede IP-Adresse hat zwei Zustände freigegeben oder gesperrt.

### **7.1.3. Klasse PortSettings**

In dieser Klasse wird die Port-Adresse gespeichert. Jede Port-Adresse hat zwei Zustände freigegeben oder gesperrt.

### **7.1.4. Klasse UserException**

Die Klasse *UserException* wird benutzt, um Fehlermeldungen abzufangen und weiterzuleiten.

## 7.2. Die Eigenschaften vom Admin-Tool Middletier:

Das Admin-Tool Middletier stellt Methoden zur Verfügung, die per Http-Verbindung verschiedene Skripte aufrufen, um den Router bzw. die Firewall zu konfigurieren. Beim Einloggen werden die aktuellen Einstellungen des Routers und der Firewall übernommen.

### 7.2.1. Router

Für die Konfiguration des Routers werden die jeweiligen Methoden vom Admin-Tool Middletier aufgerufen. Diese Methoden benutzen folgende CGI-Skripte:

<i>stoprouter.cgi</i>	– Stoppen des Routers
<i>startrouter.cgi</i>	– Starten des Routers
<i>allowedips.cgi</i>	– Zustand aller IP-Adressen zurückgeben
<i>deny.cgi</i>	– IP-Adresse sperren
<i>allow.cgi</i>	– IP-Adresse freigeben

Die Methoden werden aufgelistet und kurz erläutert. Für genauere Information wird auf den Quellcode auf der CD verwiesen, der unter folgendem Pfad zu finden ist: **xdsl/src/User/User.java**

- **Router stoppen  
stopRouter ()**

Der Router wird gestoppt und es gibt keine Internetverbindung mehr. Das Stoppen des Routers wird durch die Methode **stopRouter ()** realisiert.

Diese Methode ruft per http-Verbindung das Cgi-Skript „*stoprouter.cgi*“ auf und wartet auf die Antwort des Routers. Der Skript-Aufruf sieht wie folgt aus:

....

```
URL aUrl = new URL (mCgiPath + "stoppRouter.cgi");
URLConnection aConnection = aUrl.openConnection();
InputStreamReader aIStream = new
InputStreamReader(aConnection.getInputStream());
BufferedReader aReader = new BufferedReader(aIStream);
```

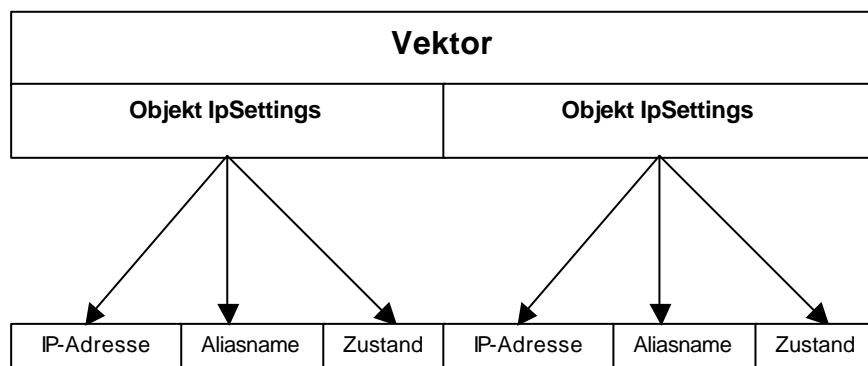
.....

Nach dem das Cgi-Skript ausgeführt ist, wird eine HTML-Seite zurückgegeben. Wurde das Skript erfolgreich ausgeführt, so steht in der Html-Seite ein „OK“.

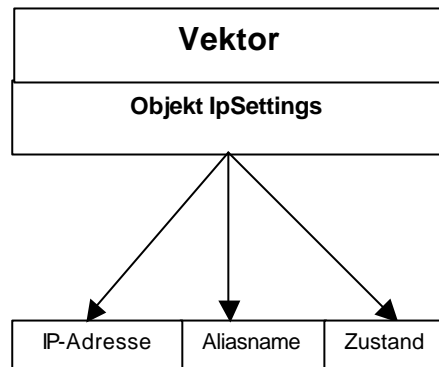
- **Router starten  
startRouter()**

Alle IP-Adressen im lokalen Netz werden freigegeben. Das Starten des Routers erfolgt ebenfalls über die Methode **startRouter()**, die ein Cgi-Skript aufruft.

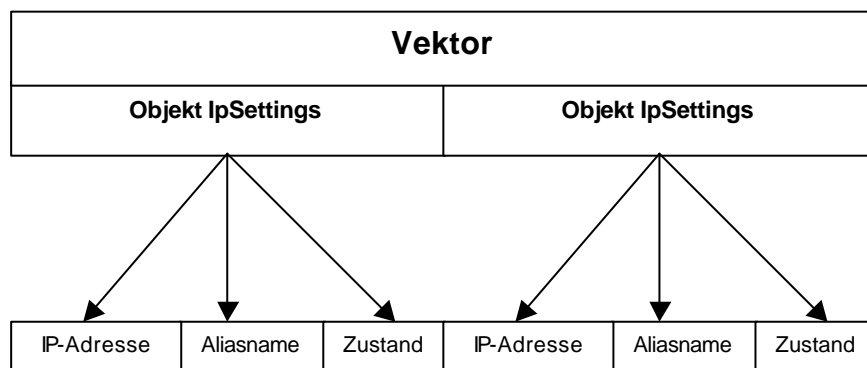
- Bereich von IP-Adressen sperren bzw. freigeben**  
**setIpSettings(int fromIp, int toIp, String inAktive)**  
 Die Methode wird mit 3 Parametern aufgerufen, wobei die ersten beiden Parameter vom Datentyp *Integer* sein müssen. Der dritte Parameter ist vom Datentyp *String* und gibt den Zustand der IP-Adresse an („gesperrt“ oder „freigegeben“).  
 Der erste Parameter gibt den Anfangsbereich und der zweite den Endbereich der IP-Adresse an.  
 Da wir ein lokales Netz haben und die ersten 3 Stellen (192.168.17.???) der IP-Adressen gleich sind, braucht man nur die letzte Nummer der IP-Adresse angeben. Die beiden Werte dürfen nicht gleich bzw. der erste Wert darf nicht größer sein als der zweite.
- Eine IP-Adressen sperren bzw. freigeben**  
**setIpSettings (String inIp, String inAktive)**  
 Diese Methode wird mit 2 Parametern aufgerufen. Beide Parameter sind vom Datentyp *String*. Der erste gibt die IP-Adresse an und der zweite gibt den Zustand („gesperrt“ oder „freigegeben“) an.
- Alle IP-Adressen sperren bzw. freigeben**  
**setIpSettingsAll(String inAktive)**  
 Diese Methode hat nur einen Parameter, und der ist vom Typ *String*. In diesem *String* steht entweder „gesperrt“ oder „freigegeben“. Somit werden alle IP-Adressen entweder „gesperrt“ oder „freigegeben“.
- Bereich von IP-Adressen ausgeben**  
**getIpSettings(int fromIp, int toIp)**  
 Die Methode wird mit 2 Parametern aufgerufen, wobei beide Parameter *Integerwerte* enthalten müssen. Der erste Wert gibt den Anfangsbereich und der zweite den Endbereich der IP-Adresse an.  
 Auch hier wird nur die 4-te Stelle der IP-Adresse angegeben. Für die Erklärung siehe **Bereich von IP-Adressen sperren bzw. freigeben**.  
 Als Rückgabewert erfolgt ein Vektor, der folgendermaßen aufgebaut ist:



- **Eine IP-Adresse ausgeben**  
**getIpSettings(String inIp)**  
 Als Parameter benötigt diese Methode eine IP-Adresse vom Datentyp *String*. Als Rückgabewert erfolgt ein Vektor, der den aktuellen Zustand der IP-Adresse zurückgibt. Dieser Vektor ist folgendermaßen aufgebaut:



- **Alle IP-Adressen ausgeben**  
**getIpSettingsAll()**  
 Diese Methode wird ohne Parameter aufgerufen und gibt einen Vektor zurück, der alle IP-Adressen, den Zustand jeder IP-Adresse und den Aliasnamen enthält. Der Vektor ist wie folgt aufgebaut.



## 7.2.2. Firewall

Für die Konfiguration der Firewall werden ebenfalls die jeweiligen Methoden vom Admin-Tool Middletier aufgerufen. Diese Methoden benutzen folgende CGI-Skripte:

<i>startFirewall.cgi</i>	– Starten der Firewall
<i>stopFirewall.cgi</i>	– Stoppen der Firewall
<i>denyPorts.cgi</i>	– Port Sperren
<i>allowPorts.cgi</i>	– Port freigeben
<i>allowedAllPorts.cgi</i>	– Alle Ports einer IP-Adresse sperren
<i>denyAllPorts.cgi</i>	– Alle Ports einer IP-Adresse freigeben

Auch in diesem Abschnitt werden die Methoden aufgelistet und kurz erläutert. Für genauere Information wird auf den Quellcode auf der CD verwiesen, der unter folgendem Pfad zu finden ist:

**xdsl/src/User/User.jav**

- Firewall starten  
**startFirewall()**

Beim Starten der Firewall werden folgende Ports freigeschaltet.

Port 21	-> File Transfer Protocol (FTP)
Port 23	-> Telnet
Port 25	-> Simple Mail Transfer Protocol (smtp)
Port 80	-> World Wide Web HTTP (www-http)
Port 110	-> Post Office Protocol (pop3)
Port 143	-> Internet Message Access Protocol (imap)
Port 8080	-> HTTP Alternate (see port 80)

Diese Ports sind die Grundeinstellungen beim Starten der Firewall. Alle anderen Ports sind gesperrt.

- Firewall stoppen  
**stopFirewall()**

Wenn die Firewall gestoppt wird, sind alle Ports freigeschaltet.

- **Portbereich** sperren bzw. freigeben

Es gibt zwei Möglichkeiten, Portbereiche im lokalen Netz zu sperren.

- a) den **Portbereich** einer IP-Adresse sperren bzw. freigeben  
**setIpPortSettings(String inIp, int fromPort, int toPort, String inAktive)**

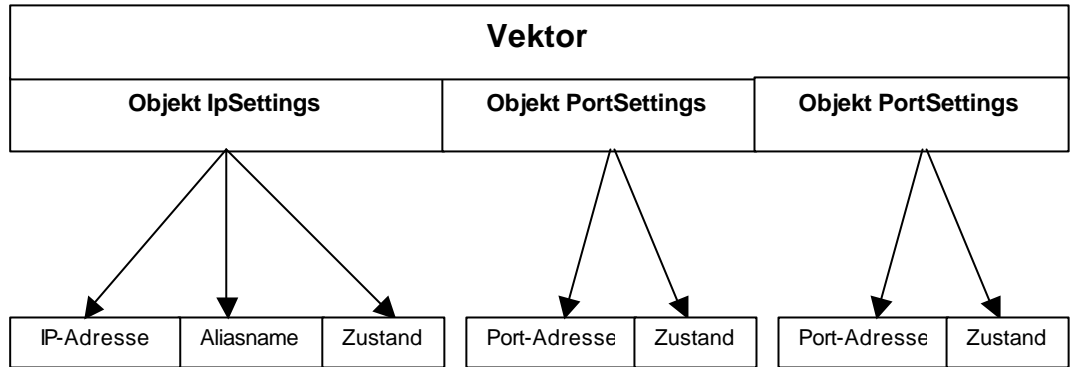
Die Methode hat 4 Parameter, die folgende Datentypen haben:

Der erste Parameter ist ein *String*, der die komplette IP-Adresse beinhaltet.

Der zweite und dritte Parameter sind *Integer*, die den Portbereich angeben.

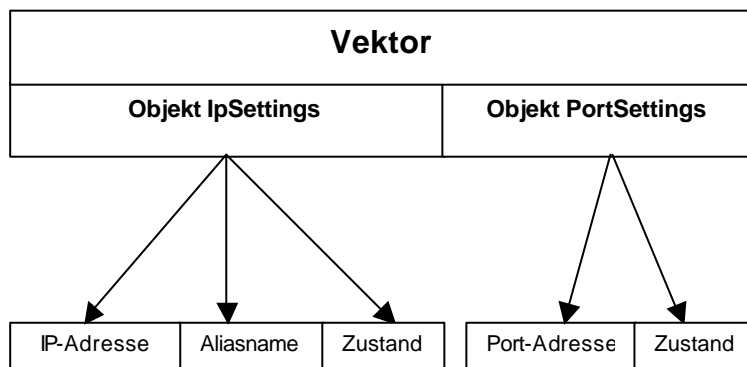
Der vierte Parameter ist auch ein *String*, der gibt den Zustand des Portbereiches an, der aus folgender Zeichenkette besteht: „gesperrt“ oder „freigegeben“.

- b) Oder den Portbereich aller IP-Adressen sperren bzw. freigeben.  
**setPortSettingsAll(int fromPort, int toPort, String inActive)**  
Die Methode hat 3 Parameter, die folgende Datentypen haben:  
Der ersten beiden Parameter sind vom Datentyp *Integer* und geben den Portbereich an. Der dritte Parameter ist ein *String*, der gibt den Zustand des Portbereiches an, der aus folgender Zeichenkette besteht: „gesperrt“ oder „freigegeben“.
- **Eine** Port-Adresse einer IP-Adresse sperren bzw. freigeben  
**setIpPortSettings(String inIp, int inPort, String inActive)**  
Um einen Port zu sperren bzw. freizugeben, muss die IP-Adresse, der Port und der Zustand als Parameter angegeben werden. Die IP-Adresse und der Zustand sind vom Datentyp *String* und der Port vom Typ *Integer*.
- **Alle** Port-Adressen sperren bzw. freigeben  
Es gibt zwei Möglichkeiten alle Ports im lokalen Netz zu sperren.
  - a) alle Ports einer IP-Adresse sperren bzw. freigeben oder  
**setIpPortSettingsAll(String inIp, String inActive)**  
Die Methode hat 2 Parameter vom Typ *String*. Im ersten *String* steht die IP-Adresse und der zweite Parameter gibt den Zustand der Ports an, die aus folgender Zeichenkette bestehen: „gesperrt“ oder „freigegeben“.
  - b) Oder alle Ports aller IP-Adressen sperren bzw. freigeben.  
**setPortSettingsAll(String inActive)**  
Die Methode hat nur einen Parameter, der vom Datentyp ein *String* ist. Dieser *String* gibt den Zustand der Ports an und besteht wie in der ersten Variante aus folgender Zeichenkette: „gesperrt“ oder „freigegeben“.
- **Portbereich** von einer IP-Adresse ausgeben  
**getIpPortSettings(String inIp, int fromPort, int toPort)**  
Die Methode hat 3 Parameter, die folgende Datentypen haben:  
Der erste Parameter ist ein *String*, der die IP-Adresse beinhaltet.  
Der zweite und dritte Parameter sind *Integer*, die den Portbereich angeben. Nachdem alle Ports in diesem Bereich ausgelesen sind, wird ein *Vector* zurückgegeben, der wie folgt aufgebaut ist:

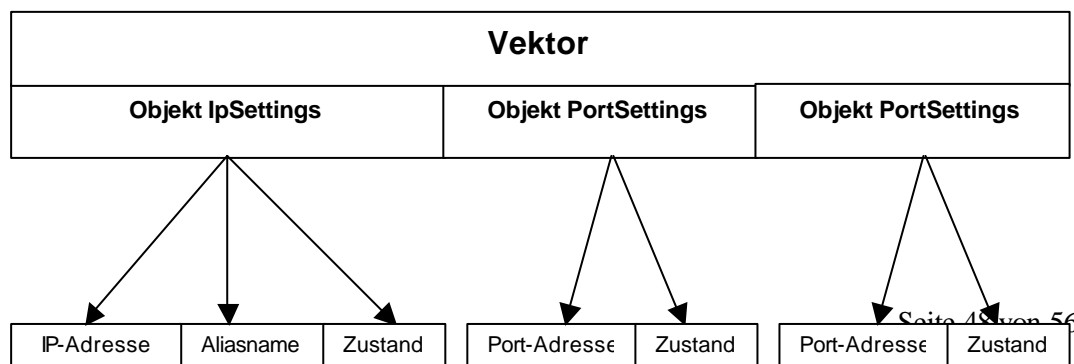


Das erste Element ist ein Objekt der Klasse IpSettings und jedes weitere Element ein Objekt der Klasse PortSettings, was die Portnummer und den Zustand beinhaltet.

- **Eine** Port-Adresse von einer bestimmten IP-Adresse ausgeben  
**getIpPortSettings(String inIp, int inPort)**  
 Als Parameter benötigt diese Methode eine IP-Adresse vom Datentyp *String* und die Portnummer vom Datentyp *Integer*. Die Methode gibt einen Vektor zurück, der aus 2 Elementen besteht. Das erste Element ist ein Objekt der Klasse IpSettings und beinhaltet die IP-Adresse, den Aliasnamen und den Zustand. Das zweite Element ist ein Objekt der Klasse PortSettings und beinhaltet die Portadresse und den Zustand.



- **Alle** Port-Adressen einer IP-Adresse ausgeben  
**getIpPortSettingsAll(String inIp)**  
 Diese Methode hat als Parameter eine IP-Adresse vom Datentyp *String*. Alle Ports dieser IP-Adresse, die gesperrt sind, werden in einem Vektor gespeichert zurückgegeben. Der Vektor ist wie folgt aufgebaut:



## **8. Fazit:**

Da es bei Projekten mit mehreren Teilnehmern immer auf jeden einzelnen ankommt, muss man sich auf jeden verlassen können. Das Zusammenspiel der Projektteilnehmer in diesem Projekt hat sehr gut funktioniert. Probleme und Fragen wurden immer in den Treffen gelöst und besprochen. Außerhalb der Treffen kommunizierten wir via Email und dem herkömmlichen Telefon, dieses lief auch reibungslos. Jeder war somit immer auf dem neuesten Stand der Dinge und Missverständnisse wurden sofort behoben. Das Arbeiten in diesem Team hat sehr viel Spaß gemacht, obwohl man in dieser Besetzung zuvor noch nie gearbeitet hatte.

## 9. Anhang

### 9.1. Anhang A

#### Projektvorschlag der Gruppe

**Mentel, Dörges, Evers, Gürsoy und Thiel:**

#### **Erstellen eines Softwarerouter für eine T-DSL Internetanbindung:**

##### Software Router

- Betriebssystem : Linux
- Internetanbindung: T-DSL
- Intranet : 1 bis 10 Rechner mit Win 9x / Win NT

##### Administration

- Administration via Browser mit Benutzername und Passwort.
- Einzelne Rechner ausmaskierbar.
- Starten und Stoppen des Routers via Tool.
- Zeitbezogene Kontrolle, ob Verbindung noch aktiv ist, ansonsten erneut herstellen.

##### Firewall

- Einzelne Ports via Internet/Intranet Tool ab- und anschaltbar.
- Nur Anfragen von bestimmte Rechner ins Internet durchlassen.

##### Mögliche Erweiterungen:

- Zeitliche Beschränkung einzelner Rechner bzw. des gesamten Intranets.
- Änderungen einfügen, ohne das Gateway neu zu starten.

## Detaillierte Aufgabenbeschreibung des Projektes:

### Allgemeine Systemanforderungen:

- Alle Clients werden über einen Hub oder Switch mit dem Router verbunden.
- Das Netzwerk kann 1 bis 255 Rechner beinhalten.
- Getestet werden nur max. 10 Rechner.
- Die Clients können als Betriebssystem WinNT, Win9x oder Linux haben.
- Die Internetverbindung ist eine TDSL-Verbindung.
- Routerbetriebssystem ist Linux.
- Auf dem Router muss mindestens die Linux-Kernel Version **xxx.xx** installiert sein.
- Auf dem Router muss ein Servletrunner installiert sein. Wobei es sich hierbei um den Jrun, Tomcat oder den von SUN mitgelieferten Servletrunner handeln kann.

### Router :

- Systemkomponenten und Schnittstellen herausfinden und überprüfen.
- Routerdienst via Script oder direkt mit Java aufrufen, je nach dem, ob es Javaschnittstellen gibt.
- Falls die Dienste die Änderungen auch ohne Neustart übernehmen, wird dieses implementiert.
- Falls die benutzte Internetverbindung eine Flatrate ist, kann der Benutzer alle 15 oder 30 Minuten die Verbindung zum Internet überprüfen lassen. Falls diese nicht mehr besteht, wird sie wieder hergestellt.

### Firewall :

- Systemkomponenten und Schnittstellen herausfinden und überprüfen.
- Firewalldienste via Script oder direkt mit Java aufrufen je nachdem, ob es Javaschnittstellen gibt.
- Alle Ports können einzeln ab- und angeschaltet werden.
- Alle IP-Adressen können einzeln ab- und angeschaltet werden.
- Aktive IP-Adressen können auch zeitlich begrenzt gestartet werden.
- Falls die Dienste die Änderungen auch ohne Neustart übernehmen, wird dieses implementiert.
- Sicherheitskonzept ist zu erstellen und durchzuführen.

## QS / Test :

- System-, Performanz- und Sicherheitstest.
- Als erstes sollen ausreichend Testszenarien erarbeitet werden.
- Während der Systementwicklung und nach Abschluss der Implementierung des Systems werden diese Szenarien durchgespielt und dokumentiert.
- Das System muss alle Tests in ausreichender Form bestehen.

## Admin Tool :

### Frontend :

- Das Tool wird über den Internet-Browser aufgerufen.
- Diese geschieht mittels einem Servlets, welche auf dem Router läuft.
- Über das Tool (Servlets) wird der Username und das Passwort von Benutzer eingelesen und bei Korrektheit wird dieser eingeloggt.
- Der Benutzer kann alle Einstellungen der Firewall und des Routers verändern und die neuen Einstellungen übernehmen lassen.
- Der Admin-User (Admin,xDsl01) kann neue User anlegen.
- Das verwendete Internetprotokoll ist aus Sicherheitsaspekten https.

### Middletier :

- Diese Javaklassen stellen die direkte Verbindung an die Systemdienste her.
- Das Passwort wird mittels einer abgespeicherten Datei verglichen.
- Das Passwort liegt verschlüsselt in der Datei vor.
- Um beide Passwörter zu vergleichen wird das eingegebene Passwort verschlüsselt, und dann werden beide verschlüsselten Passwörter verglichen.
- Das Middletier stellt folgende Methoden zur Verfügung:
  - Routereinstellungen ändern
  - Routereinstellungen übernehmen
  - Router starten
  - Router stoppen
  - Firewalleinstellungen ändern
  - Firewalleinstellungen übernehmen
  - Firewall starten
  - Firewall stoppen
  - IP-Adressen nach definiertem Zeitraum wieder schliessen

## **Aufgabenverteilung im Projekt :**

Klemens Mentel : Firewall

Karsten Döriges : Router

Leif Evers : Admin-Tool Frontend

Fatih G. : Admin-Tool Middletier

Edgar Thiel : QS / Test

## 9.2. Anhang A.1.1

Auszüge einer Protokolldatei eines Tests ohne Anbindung an ein Livesystems. Somit schlagen alle Tests fehl:

```
(Do 07.06.2001)16:58:18(510) ; info ; TestClient is preparing to run tests for: ;
    xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:18(620) ; info ; Testclass #1: xdsl.tools.testtool.UserTest ;
    xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:18(620) ; info ; ;
    xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:18(670) ; info ; See end of file for Report Statistics! ;
    xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:18(670) ; info ; ;
    xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:18(890) ; info ; Cannot find init()-Method! Using standard connection...
; xdsl.tools.testtool.TestClient.createTestClasses(TestClient.java:111) ; main
(Do 07.06.2001)16:58:18(950) ; info ; >> ;
    xdsl.tools.testtool.TestClient.runTests(TestClient.java:130) ; main
#
# Starting Tests for xdsl.tools.testtool.UserTest
(Do 07.06.2001)16:58:19(000) ; info ; Testcase #1... ;
    xdsl.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Do 07.06.2001)16:58:19(110) ; EXCEPTION ; java.lang.NullPointerException >> ;
    xdsl.tools.testtool.TestClient$1.addError(TestClient.java:141) ; main
# java.lang.NullPointerException
# at xdsl.tools.testtool.UserTest.test001addUser(UserTest.java:48)
# at java.lang.reflect.Method.invoke(Native Method)
# at junit.framework.TestCase.runTest(TestCase.java:155)
# at junit.framework.TestCase.runBare(TestCase.java:129)
# at junit.framework.TestResult$1.protect(TestResult.java:100)
# at junit.framework.TestResult.runProtected(TestResult.java:117)
# at junit.framework.TestResult.run(TestResult.java:103)
# at junit.framework.TestCase.run(TestCase.java:120)
# at junit.framework.TestSuite.run(TestSuite.java, Compiled Code)
# at xdsl.tools.testtool.TestClient.runTests(TestClient.java:150)
# at xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code)
# at xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code)
(Do 07.06.2001)16:58:19(170) ; info ; Testcase #2... ;
    xdsl.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Do 07.06.2001)16:58:19(170) ; ERROR ; Failure#junit.framework.AssertionFailedError:
User wasn't found ; xdsl.tools.testtool.TestClient$1.addFailure(TestClient.java:145) ; main
.
.
.
(Do 07.06.2001)16:58:20(210) ; info ; Testcase #32... ;
    xdsl.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Do 07.06.2001)16:58:20(210) ; EXCEPTION ; java.lang.NullPointerException >> ;
    xdsl.tools.testtool.TestClient$1.addError(TestClient.java:141) ; main
# java.lang.NullPointerException
# at xdsl.tools.testtool.UserTest.test021setIpSettingsWithSeconds(UserTest.java:466)
# at java.lang.reflect.Method.invoke(Native Method)
# at junit.framework.TestCase.runTest(TestCase.java, Compiled Code)
# at junit.framework.TestCase.runBare(TestCase.java, Compiled Code)
# at junit.framework.TestResult$1.protect(TestResult.java, Compiled Code)
# at junit.framework.TestResult.runProtected(TestResult.java, Compiled Code)
# at junit.framework.TestResult.run(TestResult.java, Compiled Code)
# at junit.framework.TestCase.run(TestCase.java:120)
# at junit.framework.TestSuite.run(TestSuite.java, Compiled Code)
# at xdsl.tools.testtool.TestClient.runTests(TestClient.java:150)
# at xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code)
# at xdsl.tools.testtool.TestClient.main(TestClient.java, Compiled Code)
(Do 07.06.2001)16:58:20(270) ; info ; SOME TESTS FAILED! (Total: 32 Failed: 1 Errors: 27)
; xdsl.tools.testtool.TestClient.runTests(TestClient.java:164) ; main
(Do 07.06.2001)16:58:20(270) ; info ; All Tests for xdsl.tools.testtool.UserTest completed in
1,27 sec. ; xdsl.tools.testtool.TestClient.runTests(TestClient.java:166) ; main
(Do 07.06.2001)16:58:20(270) ; info ;
+++++ ;
    xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:20(270) ; info ; TEST-REPORT for Thu Jun 07 16:58:20 GMT+02:00
2001 ; xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Do 07.06.2001)16:58:20(270) ; info ; +-----+ ;
    xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
```

```

(Do 07.06.2001)16:58:20(320) ; info ; Total TestClasses: 1 ;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(320) ; info ; Total TestCases : 32 ;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(320) ; info ; +-----+;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(320) ; info ; Total Failures : 1 ;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(320) ; info ; Total Errors : 27 ;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(320) ; info ; +-----+;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(320) ; info ; Testrun completed in 1,27 seconds. ;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main ;
(Do 07.06.2001)16:58:20(380) ; info ; ;
+++++ ;
xdsl.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ;

```

### 9.3. Anhang A.1.2

Protokolldatei eines Test, der ohne Fehler durchlief.

```
(Mo 28.05.2001)10:14:28(240) ; info ; TestClient is preparing to run tests for : ;
xdsI.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:28(290) ; info ; Testclass #1: xdsI.tools.testtool.UserTest ;
xdsI.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:28(290) ; info ; Testclass #2: xdsI.tools.testtool.ValidatorTest ;
xdsI.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:28(350) ; info ; ;
xdsI.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:28(350) ; info ; See end of file for Report Statistics! ;
xdsI.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:28(350) ; info ; ;
xdsI.tools.testtool.TestClient.main(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:28(460) ; info ; >> ;
xdsI.tools.testtool.TestClient.runTests(TestClient.java:130) ; main
#
# Starting Tests for xdsI.tools.testtool.UserTest
(Mo 28.05.2001)10:14:28(510) ; info ; Testcase #1... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(570) ; info ; Testcase #2... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(620) ; info ; Testcase #3... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(680) ; info ; Testcase #4... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(680) ; info ; Testcase #5... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(680) ; info ; Testcase #6... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(730) ; info ; Testcase #7... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(730) ; info ; Testcase #8... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(730) ; info ; ALL TESTS OK! (Total: 8 tests) ;
xdsI.tools.testtool.TestClient.runTests(TestClient.java:159) ; main
(Mo 28.05.2001)10:14:28(840) ; info ; All Tests for xdsI.tools.testtool.UserTest completed in
0,22 sec.; xdsI.tools.testtool.TestClient.runTests(TestClient.java:166) ; main
(Mo 28.05.2001)10:14:28(900) ; info ; >> ;
xdsI.tools.testtool.TestClient.runTests(TestClient.java:130) ; main
#
# Starting Tests for xdsI.tools.testtool.ValidatorTest
(Mo 28.05.2001)10:14:28(900) ; info ; Testcase #1... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:28(950) ; info ; Testcase #2... ;
xdsI.tools.testtool.TestClient$1.startTest(TestClient.java:135) ; main
(Mo 28.05.2001)10:14:29(120) ; info ; ALL TESTS OK! (Total: 2 tests) ;
xdsI.tools.testtool.TestClient.runTests(TestClient.java:159) ; main
(Mo 28.05.2001)10:14:29(170) ; info ; All Tests for xdsI.tools.testtool.ValidatorTest completed in
0,22 sec.; xdsI.tools.testtool.TestClient.runTests(TestClient.java:166) ; main
(Mo 28.05.2001)10:14:29(170) ; info ; ;
+++++ ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(230) ; info ; TEST-REPORT for Mon May 28 10:14:29 GMT+02:00
2001 ; xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(230) ; info ; +-----+;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(230) ; info ; Total TestClasses: 2 ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(230) ; info ; Total TestCases : 10 ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(280) ; info ; +-----+;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(280) ; info ; Total Failures : 0 ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(280) ; info ; Total Errors : 0 ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(280) ; info ; +-----+;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(340) ; info ; Testrun completed in 0,44 seconds. ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
(Mo 28.05.2001)10:14:29(340) ; info ; ;
+++++ ;
xdsI.tools.testtool.TestClient.initializeTests(TestClient.java, Compiled Code) ; main
```