Zürcher
Hochschule
Winterthur

Hochschule
für Architektur,
Technik, Wirtschaft
und Sprache

Mitglied
der Fachhochschule
Zürich

H.323 Firewall-Proxy

Diplomarbeit 2000 – Sna00/01

Fach Kommunikationssysteme (KSy)

Dozent: **Prof. Dr. Andreas Steffen**

Studierende: **Roman Buser (IT3a), Michael Stolz (IT3a)**

Termin: **7.9. – 30.10.2000**

Labor: **E523**

Arbeitsplätze: **KSY007, KSY008, KSY112**

Inhalt

1 Zusammenfassung

Das Internet entwickelte sich von einem textbasierten Datennetzwerk zu einem Multimedianetz über das Bilder, Ton und Video übertragen werden. Da liegt es nahe auch Telefongespräche darüber zu übertragen. Firmennetze sind allerdings immer über einen Firewall mit dem Internet verbunden. Dieser Firewall muss überwunden werden, um Telefonverbindungen zwischen zwei beliebigen Teilnehmern, die irgendwo auf der Welt ans Internet angeschlossen sind, zu ermöglichen.

Das Überwinden eines Linux-Firewalls war das Ziel dieser Arbeit. Zu diesem Zweck sollte ein Firewall-Modul entwickelt werden, das eine H.323-Multimedia-Verbindung durch den Firewall ermöglicht. Unsere Lösung besteht darin, bei Bedarf dynamisch Ports zu öffnen und nach Beenden der Verbindungen wieder zu schliessen.

Um herauszufinden welche Ports das betrifft, werden die H.323-Meldungen analysiert, und dem Firewall wird mitgeteilt, welche Ports er öffnen muss. Um überhaupt an die Meldungen zu kommen, muss dem Firewall auch mitgeteilt werden, welche IP-Pakete er an das Modul übergeben soll. Nur solche Pakete werden analysiert.

Unser Modul unterstützt bislang Verbindungen zwischen zwei IP-Telefonen von Siemens. Der Anrufer muss dabei immer innerhalb des Firewalls sein. Es sind auch mehrere Verbindungen gleichzeitig möglich. NetMeeting wird noch nicht unterstützt. Es benutzt ein Feature von H.323, das unser Modul nicht versteht. Sämtliche Verbindungen werden in einem Log-File aufgezeichnet.

Neben dem Design und der Implementation des Moduls benötigte auch das Erarbeiten der Grundlagen viel Zeit. Das war nötig um die Arbeit überhaupt durchführen zu können. Das grösste Problem war jedoch, dass sich der Firewall von Linux noch in der Beta-Phase befindet. So standen wir vor einem Problem das gegen Ende der Diplomarbeit von den Linux-Entwicklern gelöst wurde. Zu diesem Zeitpunkt hatten wir bereits einen Workaround geschrieben.

2 Abstract

The Internet has evolved from a text-based data network to a multimedia net that is being used to transmit images, audio and video. It is obvious to make phone calls over the Internet as well. Corporate networks are always connected to the Internet through a firewall. The firewall has to be overcome in order to establish phone connections between two arbitrary participants connected to the Internet at some place on earth.

The goal of our project was to overcome the Linux firewall. This implicated the development of a firewall module that enables an H.323 connection through the firewall. Our solution to solve this problem is the dynamic opening of ports on demand, closing them when the connection is terminated.

In order to find out which ports are concerned, the H.323 messages are analysed. The firewall is then informed which ports it has to open. The firewall has to be told which IP packets it has to pass to the module so that they can be analysed. Other packets are not examined.

Our module currently supports connections between Siemens IP phones. The calling party has to be inside the firewall. Multiple concurrent connections are possible. NetMeeting is not yet supported. It uses an H.323 feature that our module is not capable of understanding at the time. All connections are logged to a file.

Apart from the design and the implementation of the module, much time had to be invested into getting familiar with the fundamentals. This was necessary in order to undertake the project at all. However the biggest problem was the Linux firewall still being in beta state. For example we faced a problem that was solved by the Linux developers just shortly before the end of our project. At this point we had already written a workaround.

3 Aufgabenstellung

Kommunikationssysteme (KSy)

Praktische Diplomarbeiten 2000 - Sna00/1

H.323 Firewall-Modul

Studierende:

- Roman Buser, IT3a
- Michael Stolz, IT3a

Termine:

- Ausgabe: Donnerstag, 7.09.2000 10:00 - 12:00 im E509
- Abgabe: Montag, 30.11.2000 12:00

Beschreibung:

H.323 basierte Multimedia-Anwendungen, wie zum Beispiel Microsoft NetMeeting, vertragen sich schlecht mit sicheren Firewalls, da im Verlauf der Kommunikation dynamisch eine ganze Anzahl von im voraus unbekannten UDP und TCP Ports für die Übertragung von Audio/Video-Kanälen, respektive Anwenderdaten geöffnet werden müssen.

In dieser Arbeit soll auf der Basis des Open H.323 Stacks unter Linux 2.4 ein Firewall-Modul erstellt werden, der das Q.931 Call Setup Protokoll auf dem TCP-Port 1720, sowie das anschliessende H.245 Control Protokoll analysiert und die darüber ausgehandelten Multimedia Ports für die Dauer der Kommunikation im Firewall öffnet und nachher wieder schliesst. Jede Verbindung soll in einem Logfile übersichtlich protokolliert werden.

Für den Fall, dass im Firewall IP Masquerading eingesetzt wird, soll ein Konzept erarbeitet und anschliessend realisiert werden, welches mehreren internen H.323 Clients erlauben soll, gleichzeitig durch den Firewall eine bidirektionale Verbindung nach aussen aufzubauen.

Ziele:

- Erstellen einer Software-Spezifikation für das H.323 Firewall Modul
- Codieren des H.323 Firewall-Moduls unter Linux 2.4 (GNU C/C++ Compiler / Debugger)
- Austesten des H.323 Firewall-Moduls mit folgenden Konstellationen:
    - H.323 Clients: Microsoft NetMeeting, Siemens HiNet IP Phone
    - Direkter Call Setup zwischen zwei H.323 Clients auf der Basis von IP Adressen

- Erstellen einer Installations- und Betriebsanleitung
- Dokumentation der Diplomarbeit

Infrastruktur / Tools:

- Raum: E523
- Rechner: 4 PCs mit Dual-Boot: SuSE Linux 7.0 / Windows NT 4.0
- VoIP: Microsoft NetMeeting 3.01, 2 Siemens HiNet IP-Phones
- SW-Tools: GNU C++ Compiler / Debugger, Open H.323 Source Code

Literatur / Links:

- Open H.323 Website:
  http://www.de.openh323.org/
- Open H.323 Mailing List:
  http://www.de.openh323.org/list.html
- ITU-T Recommendation H.323
  Packet-Based Multimedia Communications Systems
- ITU-T Recommendation H.225.0
  Call Signalling Protocols and Media Stream Packetization
- ITU-T Recommendation H.245
  Control Protocol for Multimedia Communication
- ITU-T Recommendation X.680
  Abstract Syntax Notation One (ASN.1): Specification of Basic Notation
- ITU-T Recommendation X.691
  ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)
- Linux Netfilter Home
  http://netfilter.filewatcher.org/

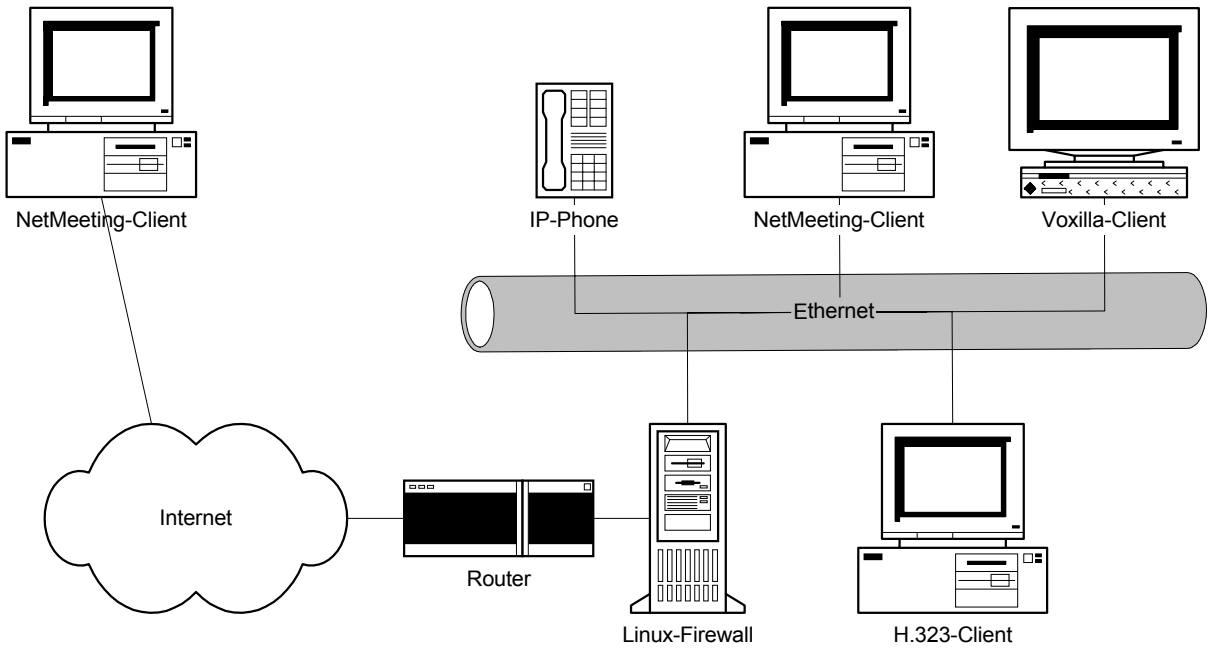
Winterthur, 7. September 2000

Dr. Andreas Steffen

4 Einleitung

4.1 Motivation

Die meisten Firmen besitzen ein Computernetzwerk. Parallel dazu existiert das Telefonnetz. Dies bedeutet den doppelten Verdrahtungsaufwand. Da ist es nur logisch, wenn Firmen über diese Computernetz auch den Telefonverkehr abwickeln möchten. Dabei darf aber der Schutz dieses Netzes gegen Aussen nicht vernachlässigt werden.
Stellen wir uns folgende Situation vor: Ein Betrieb hat ein geschütztes Intranet, das durch einen Firewall vom Internet getrennt wird. Um eine maximale Sicherheit vor Attacken aus dem Internet zu gewährleisten, schliesst der Firewall-Administrator alle nicht für ein spezifisches Protokoll notwendigen Ports und öffnet nur jene, für die ein Benutzer berechtigten Nutzen anmeldet.
Probleme ergeben sich bei dieser Konfiguration sobald eine Anwendung verwendet werden soll, die im Betrieb dynamisch neue Ports verwendet, welche zwischen den Beteiligten Kommunikationspartnern ausgehandelt werden. Zum Beispiel H.323-Clients wie NetMeeting und IP-Phones fallen in diese Kategorie.
An dieser Stelle setzt unsere Projektarbeit an: Der Firewall soll sich automatisch so rekonfigurieren, dass H.323-Verbindungen möglich werden. Falls im Intranet nur private IP-Adressen verwendet werden, sollte auch der Einsatz von NAT oder Masquerading möglich sein.



4.2 Hinweise zur Benützung dieses Dokumentes

Diese Dokumentation ist in folgende Kapitel unterteilt:

- Infrastruktur
  Eine Beschreibung der verwendeten Hard- und Software.
- Grundlagen
  Eine Einführung in Themen wie Networking und dem H.323-Protokoll, deren Kenntnis notwendig für das Verständnis unserer Diplomarbeit ist. In den nachfolgenden Kapiteln wird dieses Wissen als bekannt vorausgesetzt.
- Analyse
  Eine Zusammenstellung von Ideen, wie die diversen Probleme der Aufgabe gelöst werden können. Aus der Analyse erfolgte das Design.
- Design
  Jetzt wird es konkret. Die Klassenstruktur und die Zusammenarbeit der einzelnen Klassen wird hier festgelegt.
- Implementation
  Eine Beschreibung wie die Klassen in C++-Code umgesetzt werden. Dabei werden die komplizierten Code-Abschnitte erklärt.
- Test
  Eine Aufzählung der Testprozeduren und des Testergebnisses
- Installations- und Bedienungsanleitung
  Alles was man braucht um das H.323-Modul in Betrieb zunehmen.
- Schlusswort
  Ein Rückblick über die Arbeit und ein Ausblick auf zukünftige Erweiterungen.

5 Infrastruktur

Zu Beginn standen nur zwei PC-Arbeitsplätze (Nr. 7 und Nr. 8) im KSy-Labor E523 zur Verfügung und nicht wie in der Aufgabenstellung erwähnt vier.
Glücklicherweise konnte uns Herr Steffen einen dritten PC bereitstellen (KSY112), ohne den ein effizientes Arbeiten unmöglich gewesen wäre. Mit einem vierten PC wären uns einige Schwierigkeiten erspart geblieben, die dadurch entstanden, dass Fileserver und Firewall auf ein und demselben Rechner laufen mussten.

5.1 Eigener Fileserver

Für einen eigenen Fileserver lassen sich einige Gründe anführen:

1. Grösstmögliche Unabhängigkeit von der ZHW-IT-Infrastruktur.
2. Gesteigerte Datensicherheit durch selbstverwaltetes Backup und hohe Datenredundanz.
3. Ein NFS[1]-Server ermöglicht unter Linux ein effizientes Arbeiten mit gemeinsamen Datenbeständen. SMB[2]-Server, wie sie an der ZHW eingesetzt werden, verursachen Probleme mit Zugriffsberechtigungen und Dateiattributen. Speziell das von uns eingesetzte CVS [s. Kapitel: Concurrent Versions System (CVS)] setzt diese Funktionen zwingend voraus.
4. Die Performance eines eigenen, geswitchen 100 Mbps-Ethernet ist dem ZHW-LAN vorzuziehen.

5.2 Startschwierigkeiten

Auf den beiden Arbeitsplatz-PCs KSY007 und KSY008 war zu Beginn der Diplomarbeit keine saubere Windows-NT-Installation vorhanden, so dass die zuständigen Assistenten dies zuerst erledigen mussten. Auch die Linux-Installation auf KSY112 verzögerte sich, da für diesen PC anfänglich kein Monitor zur Verfügung stand.
Nach dem diese ersten Hürden überwunden waren, konnten wir die notwendige Infrastruktur aufbauen und auf den PCs die bzusätzliche benötigte Software installieren. Es folgt die detaillierte Beschreibung der Hardware- und Softwareausstattung.

5.3 Client-PCs

KSY007:
Arbeitsplatzrechner von Roman Buser.

Hardware:

- Pentium III (450MHz), 128MB RAM
- Siemens Multimedia-Keyboard
- 2 Ethernet-Karten (Digital DE500)
- interne SCSI-Harddisk (6 GB)
- EIDE-Harddisk (6 GB) in Wechselrahmen

Software:

- Windows NT 4.0 (Schulinstallation) auf interner SCSI-Disk
- SuSE Linux 7.0 (eigene Installation) auf interner SCSI-Disk
- gcc version 2.95.2 19991024 (release)
- Concurrent Versions System (CVS) 1.10.7 (client/server)

[1] Network File System
[2] Server Message Block

- GNU Emacs 20.7.1
- Netmeeting 3
- Open H.323 Project
- MS Project 97
- MS Office 2000 Professional
- Visio Technical 4.1

KSY008:

Arbeitsplatzrechner von Michael Stolz.

Hardware:

- Pentium III (450MHz), 128MB RAM
- 2 Ethernet-Karten (Digital DE500)
- interne SCSI-Harddisk (6 GB)

Software:

- Windows NT 4.0 (Schulinstallation) auf interner SCSI-Disk
- SuSE Linux 6.2 (Schulinstallation) auf interner SCSI-Disk
- Netmeeting 3
- MS Project 97
- MS Office 2000 Professional
- Visio Technical 4.1
- F-Secure SSH 1.1
- eXceed for Windows NT 5.0.1

5.4 Server-PC

KSY112:

Firewall der Versuchsanordung und Fileserver für die beiden Arbeitsplatzrechner.
Linux-Entwicklungs-Rechner von Michael Stolz.
Eine dritte Netzwerkkarte wurde nötig, um zwei separate Subnetze, getrennt durch den Firewall, einzurichten. Der Einbau der dritten Karte gestaltete sich unerwartet schwierig, da das Rechner-BIOS keine freien Ressourcen finden konnte. Erst nach Entfernen von ISDN- und Soundkarte wurde die Netzwerkkarte erkannt.

Hardware:

- Pentium III (550MHz), 128MB RAM
- 3 Ethernet-Karten (3Com 3c905C)
- EIDE-Harddisk (13 GB) in Wechselrahmen
- interne SCSI-Harddisk (6 GB)

Software:

- SuSE Linux 7.0 (eigene Installation) auf interner SCSI-Disk
- Linux-Kernel 2.4.0-test9
- Netfilter/IP-tables 1.1.2
- gcc version 2.95.2 19991024 (release)
- Concurrent Versions System (CVS) 1.10.7 (client/server)
- GNU Emacs 20.7.1
- Open H.323 Project [6]

5.5 Netzwerkkomponenten

1 Planet 10/100BaseT Ethernet-Switch FHSW-8080A
1 Planet 10Base-T Ethernet-Hub EH-1602R
1 Wandel&Goltermann DominoLAN DA-320 Netzwerk-Analyser (an Laptop)
4 Siemens HiNet IP-Phones LP5100

6 Grundlagen

Dieses Kapitel soll dem Leser, der sich mit Firewalls, Network Address Translation, dem H.323-Protokoll, TCP/IP und CVS noch nicht auskennt, helfen sich ins Thema unserer Diplomarbeit einzuarbeiten. Wer sich selbst als Experte auf dem einen oder anderen Gebiet betrachtet, kann getrost den entsprechenden Abschnitt überspringen. Für das Verständnis der Arbeit ist dieses Wissen aber Voraussetzung.

6.1 OSI-Referenzmodell

Das OSI-Modell basiert auf einem Vorschlag, der von der International Standards Organisation (ISO) entwickelt wurde und den ersten Schritt auf dem Weg zur internationalen Standardisierung der verschiedenen Protokolle darstellt. Dieses Modell trägt den Namen ISO-OSI-Referenzmodell (OSI – Open Systems Interconnection), weil es sich damit beschäftigt, offene Systeme miteinander zu verbinden, d.h. Systeme, die für die Kommunikation mit anderen Systemen offen sind.

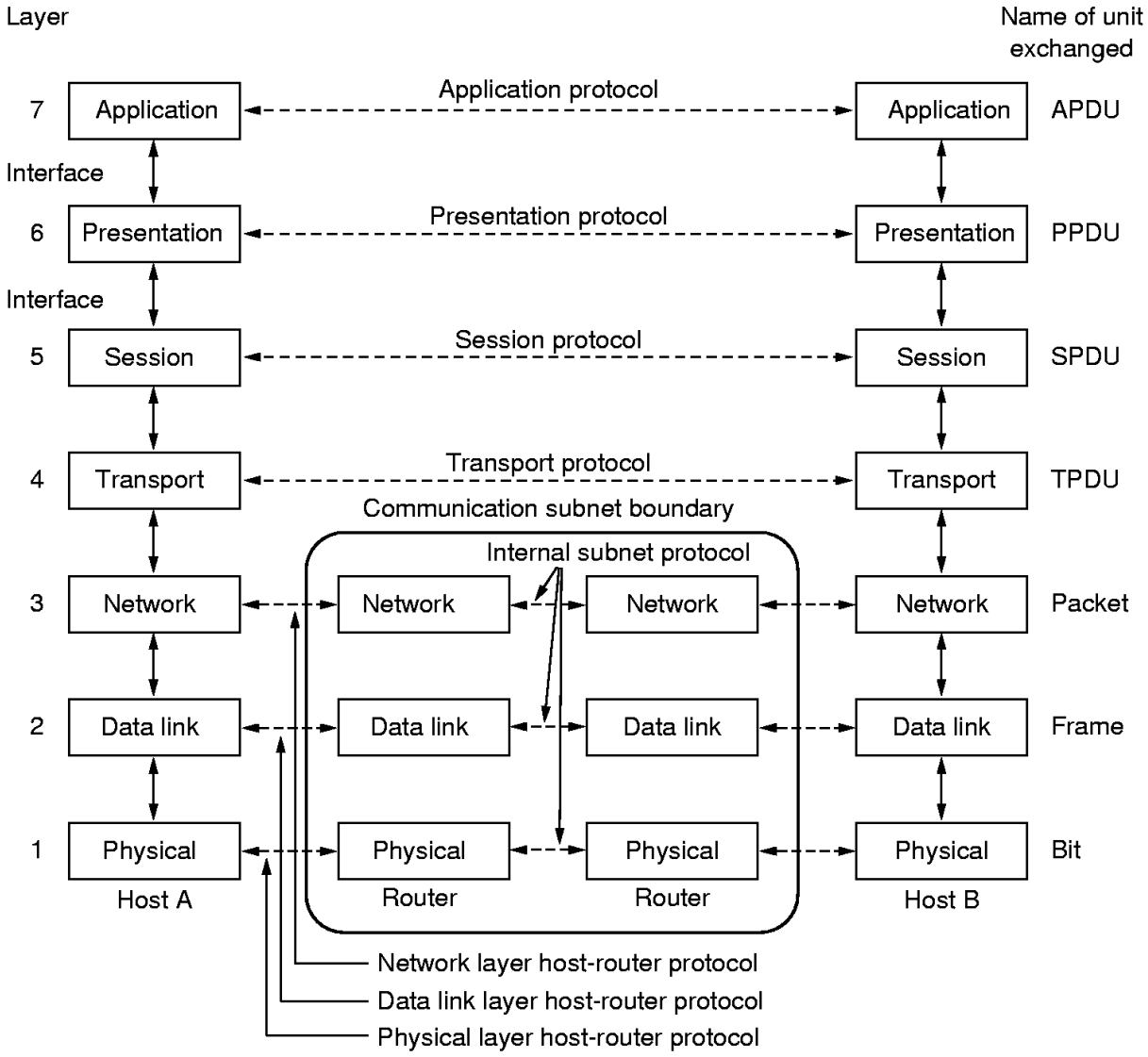
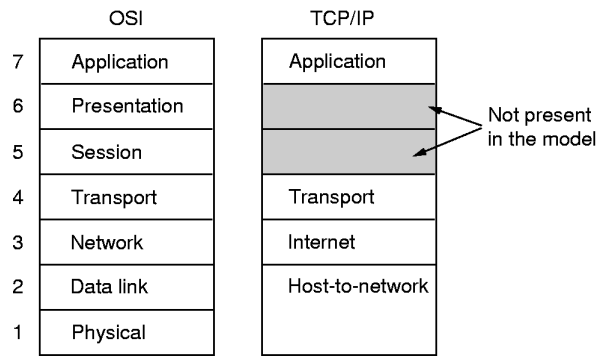


Abbildung 1: OSI-Referenzmodell

6.2 TCP/IP-Referenzmodell



TCP/IP wurde vom amerikanischem Verteidigungsministerium für das ARPANET[3] entwickelt. Weil das Internet auf diesem Protokoll aufbaut, hat es heute grosse Bedeutung für Computernetzwerke. Um die Funktionsweise dieses Protokolls theoretisch zu verstehen, hat man TCP/IP in vier Schichten, welche Protokolle repräsentieren, aufgeteilt. Jede Schicht stellt der jeweils darüberliegenden Schicht Dienste bereit.

Application	Anwendungsschicht • eigentliche Anwendung: HTTP, Telnet, FTP, SMTP, NNTP, usw.
Transport	Transportschicht • verbindungsorientierte, zuverlässige Übertragung (TCP) oder • verbindungslose, unzuverlässige Übermittlung (UDP)
Internet	Vermittlungsschicht • Festlegen des Transportweges: Routing (IP)
Host-to-network	Sicherungs- und Bitübertragungschicht • Aufteilung der Daten in Rahmen • Beseitigen von Übertragungsfehlern

6.3 Protokolle im Überblick

Hier folgt eine Übersicht über die im Rahmen dieser Diplomarbeit benötigten Protokolle, mit Ausnahme von H.323. Speziell erwähnt werden alle Felder in den Headern, die für diese Arbeit von Relevanz sind.

Byteorder

Für das Speichern von Mehrbytegrössen wird entweder das Little-Endian-Format (niederwertiges Byte an der Startadresse) oder das Big-Endian-Format (höherwertiges Byte an der Startadresse) verwendet.
Alle Werte die im Netzwerk übertragen werden, liegen im Big-Endian-Format (auch Network-Byteorder genannt) vor. Bevor also mit Längenangaben oder ähnlichem gerechnet werden kann, müssen die Werte in die auf dem Host gebräuchliche Reihenfolge gebracht werden. Bei Intel-Rechnern ist das Little-Endian.

IP

Internet Protocol [22]

IP stellt die Basisdienste für die Übermittlung von Daten in TCP/IP-Netzen bereit. Hauptaufgaben des Internet Protokolls sind die Adressierung von Hosts und das Fragmentieren von Paketen. Diese Pakete werden von IP nach bestem Bemühen ("best effort") von der Quelle zum Ziel befördert, unabhängig davon, ob sich die Hosts im gleichen Netz befinden oder andere Netze dazwischen liegen. Garantiert ist die Zustellung allerdings

[3] Advanced Research Project Agency Network

nicht. Das Internet Protokoll enthält keine Funktionen für die End-zu-End-Sicherung oder für die Flusskontrolle.

Die Funktionen von IP umfassen:

- Die Definition von Datengrammen, welche die Basiseinheiten für die Übermittlung von Daten im Internet bilden.
- Definition des Adressierungsschemas.
- Übermittlung der Daten von der Transportebene zur Netzwerkschicht.
- Routing von Datengrammen durch das Netz.
- Fragmentierung und Zusammensetzen von Datengrammen.

Das IP ist ein verbindungsloses Protokoll, d.h. zur Datenübertragung wird keine Ende-zu-Ende-Verbindung der Kommunikationspartner etabliert. Ferner ist IP ein unzuverlässiges Protokoll, da es über keine Mechanismen zur Fehlererkennung und -behebung verfügt. Unzuverlässig bedeutet aber keinesfalls, dass man sich auf das IP Protokoll nicht verlassen kann. Es bedeutet in diesem Zusammenhang lediglich, dass IP die Zustellung der Daten nicht garantieren kann. Sind die Daten aber beim Zielhost angelangt, sind diese Daten auch korrekt.

Der Header sieht folgendermassen aus:

<table>
<tr><td colspan="2">0 7</td><td colspan="2">8 15</td><td colspan="2">16 23</td><td>24 31</td></tr>
<tr><td>Version</td><td>IHL</td><td colspan="2">Type of Service</td><td colspan="3">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td colspan="2">Protocol</td><td colspan="3">Header Checksum</td></tr>
<tr><td colspan="7">Source Address</td></tr>
<tr><td colspan="7">Destination Address</td></tr>
<tr><td colspan="6">Options</td><td>Padding</td></tr>
</table>

Für das Firewallmodul sind folgende Felder interessant:

- ‚IHL' (Internet Header Length) und ‚Total Length' um die Grösse der Payload dieses Paketes zu bestimmen.
- ‚Protocol' bezeichnet den Typ der Payload:

ip	0	IP	internet protocol, pseudo protocol number
icmp	1	ICMP	internet control message protocol
tcp	6	TCP	transmission control protocol
udp	17	UDP	user datagram protocol

- Die ‚Header Checksum' ist eine Prüfsumme nur über den IP Header. Sie muss angepasst werden, wenn der Header geändert wird. Und das ist bei NAT der Fall.
- Die Adressen, da sie bei NAT geändert werden.

In Linux heisst der `struct` dieses Headers `iphdr` und ist in `<netinet/ip.h>` definiert.

Private Adressen

IANA (Internet Assigned Numbers Authority) hat einige Adressblöcke für den Einsatz als nicht registrierte IP-Adressen in privaten Netzen reserviert. 'Registriert' bedeutet, dass der Adressblock von einer Registrierungstelle für IP-Adressen stammt. Damit ist sichergestellt, dass es sich um Adressen handelt, die im Internet eindeutig sind. Da die gleichen privaten Adressen in mehreren lokalen Netzen vorkommen, ist die Eindeutigkeit nicht gegeben. Aus diesem Grund werden private IP-Adressen im Internet nicht geroutet. Damit diese Adressen trotzdem geroutet werden können, benützt man entweder einen Router mit NAT (Network

Address Translation) oder 'Tunneling'. NAT macht nichts anderes als eine Adressumsetzung, d.h. die Source-Adresse im IP-Header wird umgewandelt, und zwar wird die private IP-Adresse (Bsp. 10.0.2.1) durch eine offizielle IP-Adresse (Bsp. 160.85.131.61) ersetzt. Dadurch lässt sich jetzt das Datenpaket im Internet routen. 'Tunneling' löst dieses Problem, indem es das ganze Datenpaket mit privaten IP-Adressen in ein neues Datenpaket verpackt, dessen IP-Header offizielle Adressen besitzt. Je nach der Grösse des lokalen Netzes benützt man die folgenden privaten IP-Adressen:

10.0.0.0	-	10.255.255.255	(Class A-Netz)
172.16.0.0	-	172.31.255.255	(Class B-Netz)
192.168.0.0	-	192.168.255.255	(Class C-Netz)

ICMP

Internet Control Message Protocol [23]

ICMP ist ein Zusatzprotokoll zu IP. Es liegt auf der gleichen Schicht wie IP, jedoch erweitert es dieses um einige Funktionen, wie z.B. rudimentäre Datenflusskontrolle, ICMP-Überlaufmeldungen, Echo Requests (ping) etc. Hier eine kleine Übersicht der wichtigsten ICMP Meldungen:

Typnummer im ICMP-Header	Nachricht	Beschreibung
08	Echo Request	Fragt eine Maschine ob sie noch läuft
00	Echo Replay	Maschine antwortet, dass sie noch läuft
03	Destination unreachable	Paket kann nicht zugestellt werden

Auch bei ICMP Paketen müssen die IP Adressen durch NAT geändert werden.

UDP

User Datagram Protocol [21]

UDP ist ein verbindungsloser, unzuverlässiger Datenpaketdienst und hat keine Möglichkeiten zur Datenflusskontrolle. Die Adressierung erfolgt über den IP-Header. Ausserdem ist es ein sehr schnelles Protokoll und wird deshalb gerne benutzt.

Der Header sieht folgendermassen aus:

0 7	8 15	16 23	24 31
Source Port		Destination Port	
Length		Checksum	

In diesem Header muss nur die Prüfsumme neu berechnet werden (s. Abschnitt Prüfsumme). Der ‚Source Port' ist optional, also muss er bei der Zuordnung von Paketen zu Verbindungen ignoriert werden.
In Linux heisst die `struct` dieses Headers `udphdr` und ist in `<netinet/udp.h>` definiert.

TCP

Transmission Control Protocol [24]

TCP hat die Aufgabe, einen verbindungsorientierten Übertragungsweg mittels der Nutzung vom IP zu realisieren. TCP ist zwar verbindungsorientiert, setzt aber auf IP auf, welches ein unzuverlässiges und verbindungsloses Protokoll ist. TCP behandelt die Pakete streng nach dem Motto "Keine Antwort ist ein Fehler". Unter TCP werden die Pakete auch Segmente genannt. Ein Datensegment enthält im TCP-Header eine Prüfsumme, welche vom Zielrechner

ausgewertet wird. Bei einem entdeckten Fehler wird das Paket verworfen und der Zielrechner bestätigt das Paket nicht. Wenn der Sender nach einer bestimmten Zeit keine Bestätigung bekommt, sendet er das Paket noch einmal. Bevor es überhaupt zu einer Kommunikation kommt, wird eine Kontrollinformation an den Zielrechner gesendet. Das erste Datensegment enthält eine Sequenznummer, welche dafür sorgt, dass die Segmente in der richtigen Reihenfolge bleiben. Der Zielrechner antwortet und schickt auch noch gleich eine ISN (initial sequence number) mit. Der Rechner, der die Daten schicken will, bestätigt nun, dass das Paket angekommen ist und fängt mit dem Senden von Daten an.

Der Header sieht folgendermassen aus:

<table>
<tr><td colspan="2">0 7</td><td colspan="2">8 15</td><td colspan="2">16 23</td><td colspan="2">24 31</td></tr>
<tr><td colspan="4">Source Port</td><td colspan="4">Destination Port</td></tr>
<tr><td colspan="8">Sequence Number</td></tr>
<tr><td colspan="8">Acknowledgment Number</td></tr>
<tr><td>Data Offset</td><td colspan="2">Reserved</td><td>Control Bits</td><td colspan="4">Window</td></tr>
<tr><td colspan="4">Checksum</td><td colspan="4">Urgent Pointer</td></tr>
<tr><td colspan="6">Options</td><td colspan="2">Padding</td></tr>
</table>

Wichtige Felder:

- Anhand der Portnummern können die Daten dem Signalling oder Control-Channel zugeordnet werden.
- ‚Data Offset' gibt die Grösse des TCP Headers an.
- ‚Checksum': Auch hier muss die Prüfsumme neu berechnet werden (s. Abschnitt Prüfsumme).
- ‚Control Bits': 6 Bits (von links nach rechts):
  - URG: Urgent Pointer field significant
    Das URG-Bit wird benutzt, um dem datenerhaltenden Benutzer anzuzeigen, dringenden Vorgängen nun die Priorität zu geben und zwar so lange, wie Daten existieren, deren Sequenznummer kleiner als der gesetzte Wert im URG-Feld ist.
  - ACK: Acknowledgment field significant
    Sagt eigentlich nur, dass im Bestätigungsfeld die Sequenznummer steht, welche der Sender des Paketes mit dem nächsten Paket vom Empfänger erwarten wird.
  - PSH: Push Funktion
    Wenn das PSH-Bit gesetzt ist bedeutet dies, dass das Segment des Senders Daten enthält, die vom Empfänger weitergeleitet werden sollen.
  - RST: Reset the connection (sagt ja alles)
    Bedeutet, dass der Sender die Verbindung beenden will. Ob das vom Empfänger akzeptiert wird, hängt von der Sequenznummer ab.
  - SYN: Synchronize sequence numbers
    Wird benutzt, um eine Verbindung zu initialisieren und um zu markieren, mit welcher Sequenznummer begonnen wird.
  - FIN:
    Bei einem gesetzten FIN-Bit bringt der Sender zum Ausdruck, dass er keine Daten mehr schicken wird.

In Linux heisst die `struct` dieses Headers `tcphdr` und ist in `<netinet/tcp.h>` definiert.

ITOT

ISO Transport Service on top of TCP [25]

Dies ist ein weniger bekanntes Protokoll. Es dient dazu das ISO Transport Model auf TCP nachzubilden. Das ermöglicht eine einfache Portierung von Applikationen die auf dem ISO Model aufsetzen. Bei ITOT werden Daten auf einem TCP-Stream wieder in Pakete gegliedert. Diese Pakete werden Transport Packets (TPKT) genannt.

Der Header sieht folgendermassen aus:

0 7	8 15	16 23	24 31
Version	Reserved	Length	

Dieses Protokoll wird von H.323 benutzt. Für das Firewallmodul ist nur das Längenfeld interessant. Damit lässt sich ermitteln ob schon das gesamte Paket angekommen ist oder nicht.
Dieser Header ist in Linux nicht vordefiniert.

Prüfsumme

Die Berechnung aller Prüfsummen erfolgt immer nach demselben Schema. Es wird das Einerkomplement der Summe aller 16-Bit-Halbwörter der zu überprüfenden Daten verwendet. Der Überlauf der Summe wird ebenfalls hinzu addiert. Bei einer ungeraden Anzahl Bytes wird mit einem 0-Byte aufgefüllt. Zur Berechnung dieses Algorithmus wird angenommen, dass das Prüfsummenfeld zu Beginn der Berechnung auf Null gesetzt ist.
Bei TCP und UDP erfolgt die Berechnung der Prüfsumme nicht nur über das ganze Paket inklusive Header, sondern auch über einen vorangestellten Pseudoheader, der folgenden Aufbau hat:

0 7	8 15	16 23	24 31
Source Address			
Destination Address			
Zero	Protocol	Packet Length	

Alle Werte werden aus dem IP Header übernommen. Da die ‚Packet Length' nicht explizit vorhanden ist, muss sie berechnet werden. Die Länge entspricht dem kompletten TCP/UDP Paket inklusive Header. Da sich die IP Adressen bei NAT immer ändern, muss die Prüfsumme immer neu berechnet werden, auch wenn sich das ganze UDP/TCP Paket nicht ändert.

6.4 Der Linux Firewall

Evolution

Der Linux-Kernel bietet seit Version 1.1 einen Packet-Filtering-Firewall [siehe Was ist ein Packet-Filtering-Firewall?]. Die erste Generation, welche noch auf ‚ipfw' von BSD basierte, wurde von Alan Cox Ende 1994 portiert. Für den Kernel 2.0 wurde dieser Teil des Kernels von Jos Vos und weiteren erweitert; das Userspace-Tool ‚ipfwadm' war damals für die Konfiguration des Firewalls zuständig.
Der komplette Linux-Firewall besteht immer aus zwei Teilen: Zum einen der eigentliche Paketfilter, welcher sich im Kernel befindet und zum anderen ein Benutzerprogramm, welches den Kernelteil steuert und eine Benutzerschnittstelle zur Verfügung stellt. Normalerweise ist der Kernelteil gemeint, wenn vom Linux-Firewall gesprochen wird.
Mitte 1998 überarbeitete Paul ‚Rusty' Russel zusammen mit Michael Neuling den Kernel ziemlich gründlich und führte das Userspace-Tool ‚ipchains' für den Kernel 2.2 ein. Schliesslich entstand Mitte 1999 die 4. Generation des Userspace-Tools und die letzte Überarbeitung des Kernels fand statt. Diese neueste Generation gehört zum Kernel 2.4, welcher spätestens anfangs 2001 als offizieller Release freigegeben werden soll[4]. In Kernel 2.4 heisst der Linux-Firewall ‚netfilter' und das darauf aufsetzende Userspace-Tool wird jetzt ‚iptables' genannt. Eine Beschreibung von ‚netfilter' und ‚iptables' findet sich im iX-Artikel „Ante portas: Linux 2.4" von Michael Riepe [siehe Kasten „Neuorganisation im Netzwerkcode"].

Was ist ein Packet-Filtering-Firewall?

Ein Packet-Filtering-Firewall schaut die Headers der ihn durchlaufenden Datenpakete an und entscheidet über das Schicksal des gesamten Pakets. Er kann sich dazu entschliessen, das Paket zu verwerfen (‚DROP'), zu akzeptieren (‚ACCEPT') oder etwas Komplizierteres damit anzustellen.
In Linux ist der Paketfilter Teil des Kernels und kann entweder als Modul geladen (seit Kernel 2.0) oder fest in den Kernel einkompiliert werden. Der Linux-Paketfilter beherrscht weit trickreichere Dinge, aber das Grundprinzip, die Paketheaders zu analysieren und über das weitere Schicksal der Pakete zu entscheiden, ist noch vorhanden.

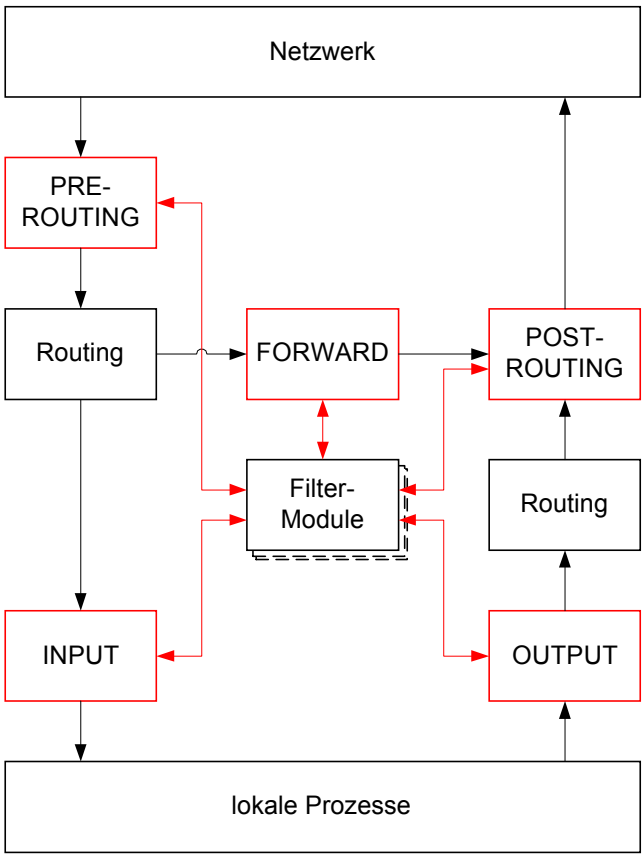
[4] Der ursprüngliche Zeitplan sah einen ersten stabilen Release Mitte 2000 vor! (Windows lässt grüssen…)

IP-tables

Mit iptables gibt es für Netfilter nur mehr ein einziges Werkzeug für alle Aufgaben, die mit NAT und Paketverarbeitung zusammenhängen. Diese werden in 3 Tabellen konfiguriert, woher auch der Name von iptables herrührt.
Jede dieser Tabellen setzt sich aus vordefinierten und benutzerdefinierten Chains zusammen, die wiederum Rules enthalten. Aufgrund dieser Rules wird entschieden, was mit Paketen geschieht, für die die Kriterien der jeweiligen Rule erfüllt sind.
Das Schicksal eines Paketes wird durch sogenannte Targets bestimmt. Ein Target kann eine benutzerdefinierte Chain sein oder eines der speziellen Targets ACCEPT, DROP, QUEUE oder RETURN. Bei ACCEPT wird das Paket durchgelassen, bei DROP verworfen. Pakete mit dem Target QUEUE können im Userspace weiter bearbeitet werden. RETURN beendet das Durchlaufen der aktuellen Chain, das Paket wird von der nächsten Rule in der vorhergehenden Chain betrachtet. Ist das Ende einer vordefinierten Chain erreicht oder in einer vordefinierten Chain das RETURN-Target erreicht, entscheidet die Policy der Chain über den Schicksal eines Paketes.
Beim Eintreten eines Paketes in den Rechner wird zunächst die nat-Tabelle berücksichtigt, dann fällt die Routing-Entscheidung. Anschließend werden die Rules der filter-Tabelle abgearbeitet.
Beim Austreten eines Paketes aus dem Rechner ist die Reihenfolge umgekehrt.

Neuorganisation im Netzwerkcode (iX 8/2000, S.98f [2])

Der nicht mehr zu bändigende Wildwuchs im Netzwerkcode des 2.2er-Kernels war der Anlass für eine gründliche Rodung und die anschliessende Neukonstruktion des gesamten Filtermechanismus. Das Ergebnis war der Netfilter, eine universelle Schnittstelle für die Handhabung von Netzwerk-Paketen. Wie in der Grafik zu sehen, erlaubt er anderen Teilen des Kernels, sich an fünf strategisch wichtigen Punk-ten in den Paketstrom einzuklinken, die durch-laufenden Pakete zu be-gutachten, zu zählen, um-zuleiten, wegzuwerfen oder zu verändern.



Benutzer haben mit der Netfilter-Schnittstelle keinen direkten Kontakt, sondern kommunizieren lediglich mit den darüberliegenden Filter-modulen. Sie können entweder eines der Emulationsmodule *ipfwadm* oder *ipchains* laden, um bei ihrem gewohnten Administrations-werkzeug zu bleiben, oder sich des *ip_tables* bedienen, das mit Programm iptables zusam-menarbeitet. Der Kernel lädt Moduls Module nicht automatisch dem zeptiert auch immer nur eins men-ihnen, ein Mischbetrieb ist diese drei möglich. Auch sollte man und er ak-hüten, das Firewall- von also nicht sich

Basismodul einfach wieder zu entfernen, um ein anderes zu laden – ebenso gut könnte man gleich den Firewall öffnen. *ip_tables* ist wiederum nur eine Zwischenstation; es stellt einige Basisfunktionen für alle Arten von IP-Paketfiltern bereit. Für das kommende Internet-Protokoll (IPv6) existiert übrigens schon ein entsprechendes Modul namens *ip6_tables*.
Wie der Arbeitstitel IP-Tables vermuten lässt, speichert der neue Kernel die Filter-Regeln in Form von Tabellen. Jede Tabelle kann mehrere Regelketten enthalten, die entweder einem der strategischen Punkte im Paketfluss zugeordnet sind oder aus anderen Ketten heraus aufgerufen werden (benutzerdefinierte Ketten). Einzelne Regeln bestehen aus einer Vergleichsfunktion (*match function*) und den von ihr benötigten Daten – IP-Adressen, Portnummern und so weiter – sowie einer Aktion (*target*). Kernelmodule können nach Belieben neue Tabellen anlegen oder zusätzliche Vergleichsfunktionen und Aktionen definieren. Tatsächlich ist nur ein kleiner Teil der vorhandenen Funktionen fest in *ip_tables* eingebaut, die übrigen sind als separate Module realisiert.
Drei verschiedene Tabellen sind zurzeit implementiert: *filter* enthält den eigentlichen Paketfilter, *nat* ist für Network Address Translation und Spezialformen wie Masquerading oder Port Forwarding zuständig, und *mangle* ermöglicht verschiedene andere Modifikationen von IP-Paketen, zum Beispiel das Ändern des Type of Service (ToS) oder das Markieren bestimmter Pakete. Für die gesamte Konfiguration zeichnet das Programm *iptables* verantwortlich. Es ist ebenfalls modular aufgebaut, damit es mit Veränderungen im Kernel besser Schritt halten kann; vielleicht erreicht es dadurch ja eine höhere Lebensdauer als seine drei Vorgänger.
Umsteiger, die vorher mit *ipchains* gearbeitet haben, werden sich mit *iptables* schnell anfreunden. Die Optionen der beiden Programme sind identisch, soweit das möglich war. Nur an einer Stelle heisst es aufpassen: bei *iptables* sind die Filter so angeordnet, dass jedes Paket nur noch einen der Filter *INPUT*, *FORWARD*, *OUTPUT* passieren muss; bei *ipchains* mussten weitergeroutete Pakete noch alle drei Filter nacheinander durchlaufen. Dies ist auch der Grund, warum die Namen der Ketten bei *ipchains* klein und bei *iptables* gross geschrieben werden – ein kleine Gedächtnisstütze für den Benutzer.

6.5 Network Address Translation (NAT)

Normalerweise passieren Pakete auf einem Netzwerk bei ihrer Reise von der Quelle (z.B. einem PC an der ZHW) zu ihrem Ziel (z.B. www.kernelnotes.org) viele verschiedene Router. Keiner dieser Router verändert das Paket grundsätzlich: Sie senden es nur weiter.
Falls einer der Router NAT durchführte, dann würde er Quelle oder Ziel des Paketes beim Passieren ändern. Man kann sich vorstellen, dass dies nicht der ursprünglich geplanten Funktionsweise des Systems entspricht, deshalb ist NAT eigentlich immer ein Murks. Üblicherweise erinnert sich der NAT-Router, auf welche Art er ein Paket verstümmelt hat, und wenn ein Antwortpaket in umgekehrter Richtung den Router passiert, macht er die Verstümmelung rückgängig, so dass alles funktioniert.

Warum wird NAT benutzt?

In einer perfekten Welt würde niemand auf diese Idee kommen…
Folgendes sind die Hauptgründe, NAT trotzdem einzusetzen:

- **Modemverbindungen ins Internet**
  Die meisten Internet-Service-Provider (ISP) geben ihren Kunden nur eine einzige IP-Adresse beim Einwählen. Nun kann man zwar Pakete mit einer beliebigen IP-Adresse abschicken, aber nur Antworten mit dieser einen Absenderadresse kommen zu einem zurück.
  Wenn man mehrere verschiedene Rechner (z.B. ein privates LAN) mittels dieser Verbindung aufs Internet bringen will, dann braucht man NAT.
  Dies ist heutzutage bei weitem die gebräuchliste Anwendung von NAT, in der Linux-Welt landläufig unter dem Namen ‚Masquerading' bekannt. Eine präzisere Bezeichnung ist Source NAT (SNAT), weil die Quellenadresse (source address) des Pakets geändert wird.
- **Mehrere Server**
  Manchmal möchte man beeinflussen, wohin Pakete, deren Ziel das eigene Netzwerk ist, hinfliessen. Häufig kommt man in diese Situation, wenn man nur eine einzige IP-Adresse hat (s. oben), aber es ermöglichen will, dass die Rechner hinter der „offiziellen" IP-Adresse von aussen zugänglich sind. Durch das Ändern der Zieladresse von eingehenden Paketen kann man dies erreichen.
- **Transparent Proxying**
  Es kann sein, dass man vortäuschen will, dass jedes Paket, dass den Router resp. Firewall passiert, für ein Programm auf dem Router selbst bestimmt ist. Diese Vorgehensweise wird benutzt, um sogenannte transparente Proxy-Server einzusetzen: Ein Proxy-Server ist ein Programm, welches zwischen dem eigenen Netzwerk und der Aussenwelt steht und die Kommunikation zwischen diesen beiden Gebieten vermittelt. Die Eigenschaft transparent bezieht sich darauf, dass das eigene Netzwerk überhaupt nicht bemerkt, dass es mit einem Proxy-Server kommuniziert. Jedenfalls nicht, solange der Proxy-Server funktioniert.
  Beispielsweise kann Squid, ein Web Proxy Cache, so konfiguriert werden, dass er als transparenter Proxy-Server arbeitet.

Zwei Arten NAT

Paul ‚Rusty' Russel, der Autor von Netfilter, unterscheidet zwei NAT-Typen: Source NAT (SNAT) und Destination NAT (DNAT).
Bei SNAT wird die Quellenadresse des ersten Pakets verändert, d.h. es wird geändert, woher die Verbindung kommt. SNAT wird immer nach dem Routing (post-routing) ausgeführt, gerade bevor das Paket den Router verlässt. Masquerading ist eine Spezialform von SNAT.

Bei DNAT wird die Zieladresse des ersten Pakets verändert, d.h. es wird geändert, wohin die Verbindung führt. DNAT wird immer vor dem Routing (pre-routing) durchgeführt, gerade nachdem das Paket neu im Router angekommen ist. Port-Forwarding, Load-Sharing und Transparent-Proxying sind Spezialformen von DNAT.

Masquerading

IP-Masquerading ist eine Netzwerkfunktion von Linux, ähnlich der NAT, die häufig in kommerziellen Firewalls und Netzwerkroutern anzutreffen ist. Wenn zum Beispiel ein Linux-Host mittels PPP, Ethernet, etc. mit dem Internet verbunden ist, erlaubt die Masquerading-Funktion anderen intern mit dem Linux-Rechner verbundenen Computern die Nutzung des Internet. Masquerading bietet diese Funktionalität sogar, wenn die internen Maschinen keine *offiziell zugeteilten Internet-IP-Adressen* haben.
IP-Masquerading erlaubt einem ganzen Satz von Maschinen *unsichtbares* Betreten des Internet, versteckt hinter einem Gateway-System, welches nach aussen hin als ein einzelnes System erscheint. Hinzu kommt, dass IP-Masquerading die Grundlage für eine sehr sichere Netzwerkumgebung bereitstellt. Zusammen mit einem gut aufgebauten Firewall sollte ein Einbruch in das System und interne LAN äusserst schwerfallen.

6.6 Was ist H.323?

Der PC entwickelt sich schnell zum primären Kommunikationsgerät für unzählige User. Während bisher hauptsächlich E-Mails verschickt wurden, beginnt der PC nun auch das Telefon zu verdrängen und bietet neue Möglichkeiten wie Videokonferenzen. Unterstützt wird das ganze durch die explosionsartige Verbreitung des Internets. Damit sich diese Technologien auch durchsetzen können, müssen die Endgeräte und Applikationen verschiedener Hersteller miteinander kommunizieren können. Dies ist nur möglich, wenn die Geräte ein gemeinsames Protokoll unterstützen, welches sämtliche benötigten Dienste, wie das Auffinden der Gegenstelle oder Rufsignal auslösen, zur Verfügung stellt. Und genau das wird von H.323 definiert, einer Basis für Audio-, Video- und Datenkommunikation über IP-basierte Netzwerke wie das Internet und alle LANs.
H.323 [15] stammt von der ITU-T (International Telecommunications Union) und definiert, wie Multimedia-Verbindungen über ein paketbasiertes Netzwerk übertragen werden können. Unterstützt werden alle gängigen Netzwerktechnologien wie TCP/IP und IPX über Ethernet, Fast Ethernet, Token Ring, ATM, etc.
Im einfachsten Fall besteht eine H.323 Verbindung aus zwei Endgeräten und dem Übertragungsmedium. Wie man Abbildung 2 entnehmen kann, gibt es noch weitere Komponenten in einem H.323 System. Unsere Arbeit beschäftigt sich nur mit der Verbindung zwischen zwei Terminals. Weitere Komponenten, wie z.B. der Gatekeeper, werden noch nicht unterstützt und deshalb im Rahmen dieser Einführung nicht erklärt. Für die weiteren Betrachtungen gehen wir von einem einfachen Aufbau aus, der nur aus zwei H.323 Terminals und dem Übertragungsmedium besteht.

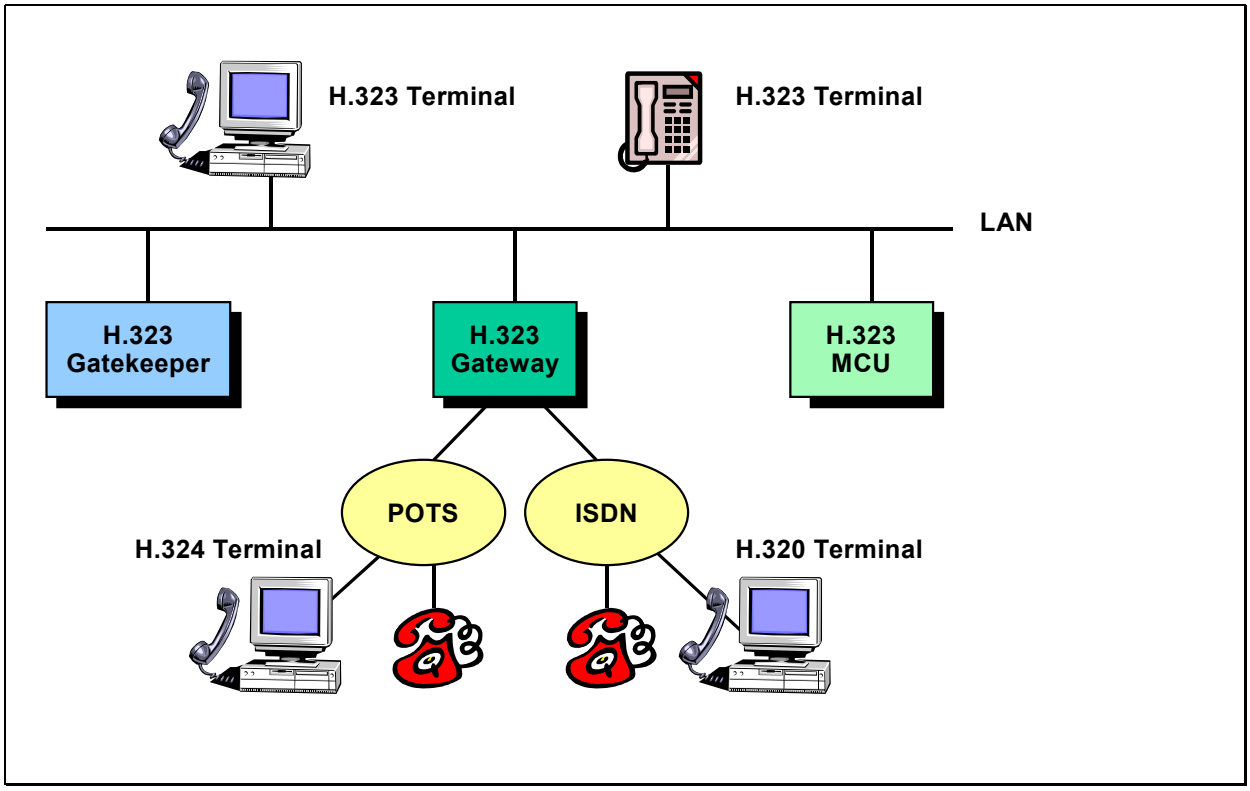


Abbildung 2: H.323 Übersicht

Später kommt dann noch unser Firewallmodul hinzu, welches aber nichts am Prinzip ändert, sondern nur einige Adressen im Protokoll austauscht. Es entspricht einem Gateway von H.323 nach H.323. Die Daten gehen unverändert hindurch.

Aufbau des Protokolls

Das Protokoll gliedert sich in drei Schichten, die aufeinander aufbauen:

- Den Signalling-Channel (Q.931) [16]
- Den Control-Channel (H.245) [17]
- Die Logical-Channels (G.7xx / H.26x / T.120)

Anhand eines Beispiels lässt sich der Ablauf einer einfachen Verbindung zwischen zwei Teilnehmern am besten verfolgen:

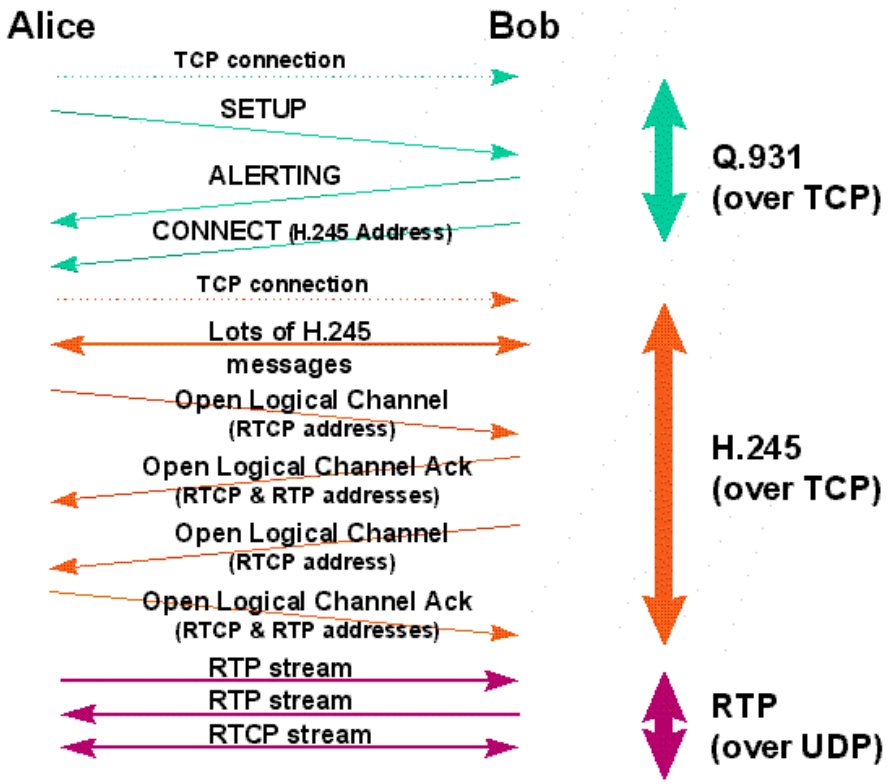


Abbildung 3: Verbindungsaufbau

Alice ruft Bob an. Zu Beginn wird der Signalling-Channel (codiert nach Q.931) geöffnet. Auf diesem wird der Verbindungsauf- und -abbau verwaltet. Als erstes wird darauf eine ‚Setup'-Meldung übermittelt. Bereits in dieser kann die H.245 Adresse enthalten sein. Üblicherweise wird sie mit der ‚Connect'-Meldung übertragen. Es folgen nun noch eine ‚Alerting'- und eine ‚Connect'-Meldung. Jetzt wird der H.245 Control-Channel geöffnet. Auf diesem findet die Capability-Verhandlung statt. Dabei wird zuerst ausgetauscht über welche Möglichkeiten das jeweilige Endgerät verfügt (z.B. wie wird Audio, wie Video kodiert). Anschliessend einigt man sich auf den kleinsten gemeinsamen Nenner den beide Geräte verstehen.
Für uns ist aber hauptsächlich die Verwaltung der Logical-Channels interessant, die ebenfalls hier stattfindet. Zwei solche, in jede Richtung einer, werden sogleich geöffnet. Auf dem Logical-Channel läuft das Real-time Transport Protocol (RTP). Dazu gehört auch eine Real-time Transport Control Protocol (RTCP) Verbindung. Nun muss Bob nur noch den Hörer abheben und das Gespräch kann stattfinden. Wenn einer der Beiden auflegt, wird die Verbindung getrennt. Das erfolgt in umgekehrter Reihenfolge. Zuerst wird auf dem Control-Channel ein ‚End Session Command' übermittelt, darauf folgt auf dem Signalling-Channel ein ‚Release Complete'. Damit wurde die Verbindung getrennt.
Zum Überblick nochmals die wichtigsten Protokolle von H.323 und deren Transport Service:

	Standard	Transport Service
RAS Channel	H.225.0	UDP
Call Signalling Channel	Q.931	TCP
Control Channel	H.245	TCP
openLogicalChannel	H.245	
Audio Channels	G.7xx	UDP / RTP / RTCP
Video Channels	H.26x	UDP / RTP / RTCP
User Data Channels	T.120	TCP
closeLogicalChannel	H.245	

Über den RAS (Registration, Admission and Status) Channel registriert sich ein Teilnehmer bei einem Gatekeeper. Dieses Feature von H.323 wird noch nicht unterstützt. Der Signalling und Control-Channel wurden bereits behandelt. Neu kommen die drei Arten von Logical-Channels hinzu. Alle werden über den Control-Channel verwaltet.

6.7 Concurrent Versions System (CVS)

Folgender Text ist die gekürzte Fassung des Artikels „Quelltext-Management nicht nur für Software-Entwickler" von Matthias Kranz, erschienen in Ausgabe 10/1998 (Teil 1) und 01/1999 (Teil 2) des Linux-Magazin [27].

GNU CVS

Das CVS Versions-Kontroll-System ist ein Programm zum Bearbeiten eines Softwareprojekts oder überhaupt eines aus Textdateien bestehenden Verzeichnisbaums.

Was ist CVS?

Vielleicht zunächst etwas zum Konzept von CVS (Concurrent Version System). CVS verwaltet die History eines Verzeichnisbaums, der Quelldateien beinhaltet. Konkret werden zu jeder bearbeiteten Datei die sukzessiven Veränderungen gespeichert. Dabei wird jede Änderung eines Files mit der aktuellen Zeit und einem Hinweis auf den Urheber der Änderung versehen. Zusätzlich kann die Datei mit beschreibenden Hinweisen versehen werden. Denn grundsätzlich stellt sich jedem Entwickler, zumindest in einer grösseren Gruppe die Frage, wer wie zuletzt ein File bearbeitet hat und warum.
Zentraler Bestandteil von CVS ist das Repository, in dem die Quelldateien liegen. Diese werden nicht direkt editiert. Statt dessen legt sich jeder beteiligte Entwickler ein eigenes Arbeitsverzeichnis an. In diesem kann er nun die Files beliebig editieren, Files hinzufügen oder löschen, usw. Entscheidend ist, dass diese Veränderungen immer nur die lokalen Kopien betreffen. Möchte man seine Version der Dateien in das Repository einspielen und damit jedem anderen Entwickler seine Veränderungen bekannt machen, müssen diese eingecheckt werden. Vor diesem Vorgang muss ein Update der eigenen Dateien erfolgen. Bei diesem Vorgang werden nur die Dateien aus dem Repository übernommen, die jüngeren Datums als die eigenen sind. Damit werden Inkonsistenzen vermieden. Schliesslich könnten

in der Zwischenzeit auch mehrere Leute an der gleichen Datei gearbeitet haben. Nach dem erfolgreichen Update werden die Veränderungen in das Repository übertragen. Jedes künftige Auschecken oder Updaten erfolgt also dann mit der aktualisierten Version.
Natürlich ist das Arbeiten nicht nur für grosse Software-Entwicklergruppen interessant. Man findet sehr schnell weitere Anwendungsgebiete in denen eine Versionskontrolle und ein Versionsmanagement sinnvoll sein können, z.B. beim Erstellen und Verwalten von vielen Web-Seiten. Nach der schnöden einführenden Theorie, nun die praktische Anwendung. Mögliche Probleme oder Schwierigkeiten werden nun an den entsprechenden Stellen erläutert.

Einrichten des Repository

Wie schon oben erwähnt geht beim Concurrent Version System nichts ohne das Repository. Bevor wir allerdings zur Einrichtung eines solchen kommen, setzen wir noch eine Environment-Variable - das erleichtert die Arbeit ungemein. Dazu geben Benutzer der *csh* (oder einer ihrer Abkömmlinge)

```
setenv CVSROOT /pfad/zum/repository
```

oder User der *bash*

```
export CVSROOT=/pfad/zum/repository
```

ein. Wird das Arbeiten mit CVS zum täglichen Bestandteil sollte man die Variable entsprechend in den Init-Skripten der Shells setzen. CVSROOT muss korrekt gesetzt werden, ansonsten kommt es z.B. zu solchen Fehlermeldungen:

```
$ cvs checkout quellcode
cvs checkout: No CVSROOT specified! Please use the '-d' option
cvs [checkout aborted]: or set the CVSROOT environment variable.
$
```

Das ist relativ aussagekräftig. Alternativ kann man also mit der '-d'-Option den CVS-Pfad explizit setzen. Dieser überschreibt auch eine möglicherweise gesetzte Environment-Variable. Zu Testzwecken könnte man jetzt also ein Verzeichnis anlegen, in dem unser Repository beheimatet sein soll. Im folgenden wird es sich um */home/developer/repository* handeln.

Initialisierung des Repository

Vor der ersten Benutzung von CVS müssen wir natürlich zunächst festlegen, was überhaupt verwaltet werden soll. Dazu initialisieren wir in einem ersten Schritt das Repository.

```
$ cd /home/developer/repository
$ cvs init
```

Im Verzeichnis ist nun ein Unterverzeichnis *CVSROOT* erzeugt worden. Es ist unter anderem Heimat für diverse generierte Hilfsdateien. Im nächsten Schritt werden die Dateien des Verzeichnisses */home/developer/to_import/* unter CVS-Kontrolle gebracht. CVS findet über die Environment-Variable das Ziel.

```
$ cd to_import
$ cvs import -m 'Kommentar' src_import tag1 tag2
```

Durch diese Operation wurde im Repository ein Verzeichnis *src_import* erzeugt, in dem nun sämtliche Dateien abgelegt sind. Sie haben alle die Endung *,v* bekommen und bei näherer Betrachtung mit einem Editor fällt auf, dass sie mit Verwaltungs-Informationen erweitert worden sind. Verzichtet man auf obige *-m*-Option und den dazugehörigen *Kommentar*, wird automatisch ein Editor geöffnet. Dieser kann über die Environment-Variable CVSEDITOR gesetzt werden, ansonsten wird EDITOR ausgewertet. *tag1* und *tag2* sind sogenannte Vendor- und Release-Tags und zwingend vorgeschrieben. Nähere Informationen dazu folgen im zweiten Teil dieses Kurses. Vorerst kann man sie nach Belieben setzen, z.B. ein *login*- oder Namenskürzel angeben.

Auschecken eines Verzeichnisses

Zum Anlegen eines lokalen Arbeitsverzeichnisses benutzt man das *checkout*-Kommando. Auf diese Art und Weise kann man beliebig oft den Inhalt des Repositories reproduzieren. Man legt in seinem Heimat-Verzeichnis beispielsweise das Verzeichnis *project* an und führt das Folgende aus:

```
$ cd ~/project
$ cvs checkout src_import
cvs checkout: Updating src_import
U src_import/datei1.c
U src_import/datei2.c
U src_import/datei3.c
U src_import/datei4.h
```

Dann wird im Verzeichnis *project* das Unterverzeichnis *src_import* mit den entsprechenden Dateien angelegt. Diesen Vorgang kann man auch genauer spezifizieren. Bei Bedarf ist es z.B. über die Angabe eines *Tags* möglich, die beim Einchecken entsprechend markierte Datei auszulesen.

```
$ cvs checkout -r tag1 src_import
```

Oder es sollen beispielsweise alle Dateien von gestern ausgecheckt werden:

```
$ cvs checkout -D yesterday src_import
```

Dateien verändern

Jetzt arbeitet man also mit seinen und an seinen Dateien, entwickelt vielleicht sogar innerhalb einer grösseren Gruppe und nun stellt sich natürlich die Frage: Wie werden die Dateien konsistent gehalten bzw. wie befördert man seine veränderten Files wieder in das Repository? Im schwierigsten Fall haben ja zwei oder mehr Programmierer am gleichen Source-File gearbeitet. Wessen Veränderungen haben die höchste Priorität? Nun, CVS begegnet dieser Problematik zunächst einmal damit, dass man gezwungen wird vor dem Einchecken, dem sogenannten *committen*, ein Update seines Arbeitsverzeichnisses durchzuführen. Versucht man eine Datei einzuchecken, die zwischenzeitlich von jemandem erneuert worden ist, erhält man eine Fehlermeldung:

```
$ cvs commit datei4.h
cvs commit datei4.h
cvs commit: Up-to-date check failed for `datei4.h'
cvs [commit aborted]: correct above errors first!
```

Man wird also aufgefordert, zunächst ein Update vorzunehmen, was auf einfache Weise ausgeführt werden kann.

```
$ cvs update
cvs update
cvs update: Updating .
RCS file: /home/developer/repository/src_import/datei4.h,v
retrieving revision 1.1.1.1
retrieving revision 1.2
Merging differences between 1.1.1.1 and 1.2 into datei4.h
M datei4.h
```

Dieses Updaten verlief ohne Probleme, dass heisst die veränderte Datei *datei4.h* aus dem Repository konnte ohne Konflikte mit der Arbeitskopie verschmolzen werden. Dieses Verfahren hat sich in der Praxis als relativ zuverlässig erwiesen. Man sollte natürlich an dieser Stelle Vorsicht walten lassen und nach Merging-Meldungen anschliessend überprüfen, ob sich die neu entstandenen Files immer noch kompilieren lassen (im Falle der Verwaltung von Source-Codes). Verläuft das Verschmelzen allerdings nicht reibungslos, wird man von CVS durch Konflikt-Meldungen darauf aufmerksam gemacht.

```
$ cvs update
cvs update: Updating .
RCS file: /home/developer/repository/src_import/datei4.h,v
retrieving revision 1.1.1.1
```

```
retrieving revision 1.2
Merging differences between 1.1.1.1 and 1.2 into datei4.h
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in datei4.h
C datei4.h
```

CVS meldet in diesem Fall also, dass es nicht in der Lage war, die beiden verschiedenen Dateien zusammenzufügen und legt beide Varianten, entsprechend markiert, in der betreffenden Datei ab. Nun bleibt es dem Programmierer überlassen, den Inhalt zu überprüfen und anschliessend die Datei einzuchecken. Was in diesem Fall zu tun ist erfahren sie weiter unten im Abschnitt *Konflikte lösen*. Wurde eine Datei die man nur importiert, selbst aber gar nicht editiert, inzwischen verändert, wird sie einfach aktualisiert. CVS macht uns durch verschiedene Meldungen darauf aufmerksam. Wie oben zu sehen, steht am Anfang der Zeile vor jeder Datei ein einzelner Buchstabe. Dabei steht *U* für *Update*, *M* für *Modified* und *C* für *Conflict*. Durch das *M* wird angezeigt, dass man selbst als Letzter die Datei bearbeitet hat und etwaige Änderungen allen anderen noch nicht bekannt sind. Denn, dass sollte man sich immer wieder klar machen, diese Aktionen betreffen und manipulieren zunächst nur die Files im eigenen Arbeitsverzeichnis.

Veränderungen bestätigen

Nachdem das Updaten keine Fehler beim Verschmelzen vermeldet hat, sollte man seine bearbeiteten Sourcen in das Repository einchecken. Dazu dient der Befehl *commit*.

```
$ cvs commit datei4.h
Checking in datei4.h;
/home/developer/repository/src_import/datei4.h,v  <-  datei4.h
new revision: 1.2; previous revision: 1.1
done
```

Dabei wird automatisch der Editor geöffnet und man wird aufgefordert eine *Log-Message* einzugeben, eine kurze Beschreibung des Vorgangs. Mehr dazu später. Nun sind die Veränderunge für jedermann im Repository sichtbar und jeder andere wird beim Auschecken bzw. Updaten seiner Sourcen die neue Version erhalten. Wie man sehen kann wird bei obigem Vorgang automatisch die Revisionsnummer erhöht.

Dateien hinzufügen oder löschen

Für CVS bedeutet das Löschen oder Hinzufügen von Dateien nichts anderes als eine Änderung am bestehenden Datenbestand. Das hat auch einen einleuchtenden Grund. Wie schon erwähnt ist es ja unter CVS möglich, auf beliebige Versionen einzelner Dateien zurückzugreifen. Das heisst, man kann z.B. nach Belieben eine lauffähige Variante seines Programms auschecken ohne auf aktuellere, aber möglicherweise fehlerbehaftete Teile Rücksicht nehmen zu müssen. Nun kann es ja durchaus vorkommen, dass in dieser Version Dateien vorhanden sind, die in der weiteren Entwicklung wieder verschwunden sind. Sie werden aber weiterhin mitverwaltet und existieren immer noch im Repository, auch wenn sie beim jetzigen Checkout nicht mehr auftauchen würden. Ebenso weiss CVS nichts von neuen Dateien, die lediglich im Arbeitsverzeichnis, aber noch nicht im Repository auftauchen. Diese müssen zunächst mit einem *add* hinzugefügt und dann mit dem schon bekannten *commit*-Befehl eingecheckt werden.

```
$ cvs add datei5.c
cvs add: scheduling file `datei5.c' for addition
cvs add: use 'cvs commit' to add this file permanently
$ cvs commit
/home/developer/repository/src_import/datei5.c,v  <-  datei5.c
initial revision: 1.1
done
```

Wie man erkennen kann, beginnt jede Datei mit der initialen Revisionsnummer 1.1. Das Löschen funktioniert ganz ähnlich und wenn man das zugrundeliegende Konzept von CVS

verstanden hat, kann man sich schon vorstellen, wie es funktioniert. Zunächst löscht man mit Hilfe des gewohnten Unix-Befehls *rm* die Datei aus seinem Arbeitsverzeichnis. Anschliessend markiert man sie zum Löschen. Durch Ausführen des *commit*-Befehls wird die Datei *endgültig* entfernt. Mit endgültig ist in diesem Fall nur gemeint, dass sie beim nächsten *checkout* bzw. *update* nicht mehr erscheint.
In diesem Zusammenhang kann man sich diese Eigenschaft von CVS praktisch zunutze machen. Möchte oder muss man die Veränderungen, die man an einer Datei vorgenommen hat, rückgängig machen, sozusagen ein *Undo* ausführen, löscht man die entsprechende Datei einfach aus seinem Verzeichnis und führt dann den *update*-Befehl aus. CVS erkennt, dass eine Datei fehlt und man bekommt das File in seinem aktuellsten Zustand.
Komplizierter wird da schon ein Umbenennen von Files. Eine mögliche Lösung in diesem Fall ist mit den schon bekannten Kommandos zu erreichen. Zuerst gibt man der entsprechenden Datei einen neuen Namen. Im Anschluss muss man nun die alte Version aus dem Repository löschen. Dazu dient ja der *rm*-Befehl. Natürlich darf man nicht vergessen, das neu benannte File mit Hilfe des *add*-Kommandos dem Repository hinzuzufügen und die gesamte Aktion mit einem *commit* abzuschliessen. Nicht sehr elegant und auch mit einem Nachteil behaftet, denn die Log-Daten der Datei werden natürlich nicht übernommen.
Ein anderer Fall tritt auf, wenn man eine neue Datei in seinem Arbeitsverzeichnis angelegt hat und dann bewusst oder unbewusst *cvs update* ausführt. In diesem Fall steht CVS vor einem Problem und es gibt ein *?* gefolgt vom entsprechenden Dateinamen aus.

Zugriff auf ein Remote Verzeichnis

Das, was CVS zu einem vielgenutzten Utility in der Software-Entwicklung, und speziell der Freien Software macht, ist die Tatsache, dass das Arbeitsverzeichnis eines Programmierers nicht zwingend auf derselben Maschine liegen muss, welche das sogenannte Repository beheimatet, also das Verzeichnis, in dem die Original-Quellen liegen. Das macht das Concurrent Versions System geradezu geschaffen für Tausende von Entwicklern im Rahmen verschiedenster Projekte weltweit. Als Beispiele seien hier nur kurz genannt: KDE, GNOME, FreeBSD, Mozilla und viele mehr.

7 Analyse

7.1 Zielsetzung

Ziel dieser Diplomarbeit ist es, den im Linux-Kernel eingebauten Packet-Filtering-Firewall zu erweitern, so dass er H.323-basierten Multimedia-Netzwerkverkehr passieren lässt. Des weiteren soll im Bedarfsfall Masquerading eines privaten, hinter dem Firewall liegenden, Netzwerks unterstützt werden.

7.2 Herkömmliche Lösungsansätze

Bis heute gibt es zwei Möglichkeiten, wie ein Packet-Filtering-Firewall für ein Protokoll der Anwendungsschicht (OSI Layer 7) durchlässig gemacht werden kann:

1. Konventionelle Protokolle benützen bestimmte TCP- oder UDP-Ports (well-known ports). Der Firewall wird von der Systemadministratorin so konfiguriert, dass er jene IP-Pakete durchlässt, welche für diese Ports bestimmte Daten enthalten.
2. Es wird auf dem Firewall ein sogenannter Application-Proxy installiert. Ein Application Proxy ist ein Programm, welches zwischen dem eigenen Netzwerk und der Aussenwelt steht und die Kommunikation für ein bestimmtes Protokoll der Anwendungsschicht zwischen diesen beiden Seiten vermittelt. Dieser Proxy-Server muss den Clients bekannt sein, und sie müssen den Proxy-Betrieb unterstützen, damit diese Konfiguration funktionieren kann. Bekannte Vertreter der Gattung Application Proxy sind Web-Proxy-Server, wie z.B. der „Squid Web Caching Proxy".

7.3 Welcher Ansatz eignet sich für H.323-Multimedia-Anwendungen?

Ansatz 1 ist nicht brauchbar, da nicht zum Voraus bekannt ist, welche Ports im Laufe einer H.323-Verbindung von den beteiligten Clients verwendet werden. Die einzige Ausnahme stellt der Endpoint-Call-Signalling-Port dar, welcher fix definiert ist.
Ansatz 2 sollte grundsätzlich möglich sein, allerdings bedingt er einen grossen Aufwand. Denn ein richtiger H.323-Proxy-Server[5] tritt gegenüber den beiden kommunizierenden Clients als ihr Kommunikationspartner auf und stellt intern die Verbindung zwischen den beiden her. Damit das funktioniert, muss der Proxy-Server folgende Aufgaben übernehmen:

- Analyse des gesamten Verkehrs auf allen beteiligten Channels.
- Entgegennahme und Aufbau von TCP-Streams respektive
- Empfangen und Versenden von UDP-Paketen auf beiden Seiten des Firewalls.
- Weiterreichen der Daten.

Somit stellt ein Proxy-Server im Grunde genommen zwei komplette Clients dar.
Selbstverständlich müssen die Clients diesen Proxy-Betrieb unterstützen und dementsprechend konfiguriert werden. Dies lässt sich nur vermeiden, wenn der Proxy-Server transparent arbeiten kann. Notwendige Voraussetzung für einen Transparent Proxy ist Redirection, welche aber herkömmlicherweise nur für im Voraus bekannte Ports durchgeführt werden kann. Dies ist ein Teufelskreis.

7.4 Mehr Intelligenz im Firewall

Die Antwort auf diese Probleme nennen wir dynamisches Firewalling. Unter dynamischem Firewalling mit H.323-Unterstüzung verstehen wir die Fähigkeit des Firewalls, über ihn laufende H.323-Verbindungen zu analysieren und sich so zu rekonfigurieren, dass von den

[5] In H.323-Terminologie würde man wahrscheinlich eher von einem Gateway sprechen.

H.323-Clients neu vereinbarte Channels auf den entsprechenden TCP- resp. UDP-Ports freigegeben werden. Das heisst, der Firewall verarbeitet den ihn passieren Netzwerkverkehr nicht mehr nur bis zur Transportschicht (OSI Layer 4), sondern beherrscht sogar die Anwendungsschicht (OSI Layer 7). Für den Fall, dass Masquerading zum Einsatz kommt, muss der Firewall nicht nur den H.323-Verkehr analysieren, sondern zusätzlich die in den H.323-Meldungen transportierten IP-Adressen und Port-Nummern anpassen.
Recherchen im WWW ergaben, dass auch andernorts auf diesem Gebiet entwickelt wird [30,31].

7.5 Was bietet der Linux-Firewall (Netfilter)?

Netfilter bietet mit seiner modularen Architektur gute Voraussetzungen für dynamisches Firewalling. Es existieren bereits Kernelmodule, die Netfilter mit den Anwendungsprotokollen FTP[6] und IRC[7] vertraut machen. Es handelt sich dabei um sogenannte Connection-Tracking- und NAT-Hilfsmodule. Diese Module verfolgen die über den Firewall laufenden FTP- und IRC-Verbindungen mit und verarbeiten dazugehörende Pakete. Beispielsweise muss das NAT-FTP-Modul im FTP-Datenstrom die TCP-Ports und IP-Adressen austauschen.
Die logische Schlussfolgerung ist, für H.323-basierten Netzverkehr ein neues spezialisiertes Modul zu implementieren.

7.6 C++ im Kernel

H.323 ist eine sehr komplexe Protokollfamilie. Um den H.323-Verkehr zu analysieren bedienen wir uns einer C++-Klassenstruktur aus dem Open H.323 Project. Dies beschert uns ein neues Problem: C++ eignet sich nicht zum Programmieren von Teilen des Linux-Kernels. Und zwar wegen folgender Einschränkungen, die für die Kernelprogrammierung gelten (vgl. Posting: Linux kernel modules development in C++):

- Es ist kein Runtime Support für new(), delete(), etc. vorhanden.
- Es ist kein Exception Handling (throw(), try(), etc.) zulässig.
- Die Standard-Klassenbibliothek libstdc++ kann nicht verwendet werden.
- Da der Kernel in gcc (und NICHT ANSI-C) geschrieben wurde, kommen im Quelltext Bezeichner vor, die in C++ reservierte Schlüsselwörter sind.

Posting: Linux kernel modules development in C++

```
> -----Original Message-----
> From: linux-kernel-owner@vger.kernel.org
> [mailto:linux-kernel-owner@vger.kernel.org]On Behalf Of Horst
> von Brand
> Sent: Wednesday, September 27, 2000 9:32 PM
> To: Abel Muñoz Alcaraz
> Cc: Linux Kernel
> Subject: Re: Linux kernel modules development in C++
>
>
> =?iso-8859-1?Q?Abel_Mu=F1oz_Alcaraz?= <abel@trymedia.com> said:
>
> >     I want to develop a linux kernel module in C++ but I don't find
> > makefiles and/or sorce files examples to do this.
>
> Your problem is that you can't use anything of C++ that needs runtime
> support inside the kernel (there goes new() and friends), exception
> handling is out of the question (bye, bye, throw() et al),
> and (in case you
> hadn't thought of it) libstdc++ is way out of line (no fancy data
> structures or algorithms). What is left is not very much...
> and it won't
```

[6] File Transfer Protocol
[7] Internet Relay Chat

```
> make much of a difference in a driver, which by its nature is
> typically
> smallish. You have no place to go to find stuff you could
> inherit from, so
> you'd have to recreate all the supporting object infrastructure in the
> kernel in C++, or port it somehow from C (the object model inside the
> kernel doesn't really map into C++'s concepts cleanly!). Not worth the
> effort, AFAIKS.
>
> The kernel is written in plain gcc C (this is _not_ your ANSI
> C!), it uses
> identifiers that are reserved words in C++. And that won't
> change anytime
> very soon. But that would get you only to the point of using C++'s
> procedural aspects. To make it really worthwhile to write parts of the
> kernel in C++, the kernel would have to be redesigned from
> the ground up
> for C++. Nobody has ever volunteered (or even just hinted
> that they could
> be persuaded to volunteer) to do this job, and this
> discussion comes up
> each four months or so.
> --
> Dr. Horst H. von Brand
mailto:vonbrand@inf.utfsm.cl
Departamento de Informatica                     Fono: +56 32 654431
Universidad Tecnica Federico Santa Maria              +56 32 654239
Casilla 110-V, Valparaiso, Chile                Fax:  +56 32 797513
-
```

Fragen zum Thema C++ auf der Netfilter-Developer-Mailingliste wurden dementsprechend vehement mit negativem Unterton beantwortet. Erst die Erklärung, dass es nicht darum ging, C++ im Kernel zu verwenden, konnte die Kernelprogrammierer wieder einigermassen beruhigen. Einen kurzen Auszug aus unserer Email-Korrespondenz wollen wir an dieser Stelle zeigen:

```
An:           Mischa Stolz mstolz@gmx.ch
Von:          Daniel Stone daniel@dustpuppy.ods.org
Betreff:      Re: Troubles using libiptc inside C++ class
Datum:        Thu, 12 Oct 2000 20:55:16 -1100

> Hi!

Hi!

> I'm trying to use libiptc from inside a C++ class. Therefore I include
> libiptc.h in my C++ header file as follows:

C++. Mixed with C. BAD. BAD!
libiptc, if I'm not very much mistaken, is "library iptables C" - not C++, C.

> /usr/include/linux/netfilter_ipv4/ip_tables.h: In function `struct
> ipt_entry_target * ipt_get_target(ipt_entry *)':
> /usr/include/linux/netfilter_ipv4/ip_tables.h:295: ANSI C++ forbids
> using pointer of type `void *' in arithmetic
> /usr/include/linux/netfilter_ipv4/ip_tables.h:295: ANSI C++ forbids
> implicit conversion from `void *' in
> return

Err, too bad. The ANSI C standard clearly blesses void * with the role of generic pointer, and
explicitly allows arithmatic on it; this is just a C/C++ incompatibliity.

To quote Rusty in his kernel hacking HOWTO:
"Using C++ in the kernel is usually a bad idea ... and the include files are not tested for
it."

The kernel was built for C.

d
```

```
--
Daniel Stone
Kernel Hacker (or at least has aspirations to be)
daniel@dustpuppy.ods.org
http://dustpuppy.ods.org
```

Fazit

Ein Kernel-Modul, welcher den Open H.323 Stack benutzt, ist nicht realisierbar.

7.7 Vom Kernel in den Userspace

Wenn also der H.323-Stack nicht in den Kernel passt, dann müssen eben die H.323-Daten irgendwie in den Userspace kommen! Dort können sie unter Benützung der C++-Klassenstruktur von einem normalen Benutzerprozess verarbeitet werden.
Glücklicherweise wurde auch diese Eventualität von den Netfilter-Entwicklern berücksichtigt. Es existiert ein spezielles Target (QUEUE), welches als Ziel einer Regel angegeben werden kann. Dieses Target verschiebt die von der Regel herausgefilterten Pakete in eine Warteschlange, auf welche mittels Netlink-Device vom Userspace her zugegriffen werden kann. Das Netlink-Device ist eine spezielle Schnittstelle zwischen Kernel- und Userspace, die sich wie ein beliebiger anderer Socket benützen lässt. Eine C-Bibliothek (libipq) mit Funktionen, welche den Zugriff auf die Warteschlange ermöglichen, liegt dem Netfilter-Quellcode bei.
Über das weitere Schicksal der Pakete kann relativ frei entschieden werden. Trotzdem existieren zwei Einschränkungen:

1. Die Pakete können nur an dem Punkt (Hook) der Netfilter-Architektur zurückgespiesen werden, an dem sie herausgefiltert wurden.
2. Es existiert nur eine Warteschlange, welche von allen Userspace-Prozessen gemeinsam genutzt wird. Dieses Problem wurde von Harald Welte, Netfilter-Mitentwickler und Kernelhacker, erkannt und mittels einer Erweiterung der C-Bibliothek umgangen. Seine Lösung nennt sich libipqmpd und verwendet das MARK-Target zum Markieren der Pakete, damit sie später unterschieden und den beim ipqmpd-Daemon registrierten Benutzerprozessen zugeordnet werden können.

Die erste Einschränkung des QUEUE-Targets blieb leider nicht ohne Auswirkung auf die Implementation unserer Firewall-Erweiterung. Wie bereits im Kapitel Network Address Translation erwähnt, muss NAT vor dem Routing stattfinden, weshalb wir die für uns relevanten IP-Pakete am PREROUTING-Hook abgreifen müssen. Speisen wir sie nun an dieser Stelle zurück in die Netfilter-Architektur, dann werden die von aussen eingehenden Pakete der H.323-Verbindungen vom Connection-Tracking-Modul verworfen. Diese Netfilter-Erweiterung führt Buch über aus- und eingehende Verbindungen und verwirft Pakete, die sie nicht zuordnen kann. Die bei Netfilter eingebaute NAT macht ebenfalls Gebrauch vom Connection-Tracking.
Da die von unserem Benutzerprozess verarbeiteten Pakete leider für das Connection-Tracking unbekannte Verbindungen darstellen, ist unsere Lösung im aktuellen Zustand nicht mit Netfilter-NAT und Connection-Tracking kombinierbar.
Die Lösung zu diesem Problem wäre ein Connection-Tracking- und NAT-Helper-Modul. Da dieses nicht in C++ realisiert werden kann, gibt es das bereits erwähnte Problem mit dem Open H.323 Stack. Um diesen trotzdem benutzen zu können, müsste das Modul mit einem Benutzerprozess, welcher die H.323-Analyse durchführt kommunizieren, z.B. mit Hilfe des Netlink-Device.

Von Seiten der Kernelhacker wurde die Idee der Verarbeitung im Userspace eher schlecht aufgenommen, da eine ausreichende Performance bezweifelt wurde. Auch die Verwendung von C++ sei der Performance keinesfalls zuträglich, wurde angemerkt.

7.8 NAT für H.323

Beim herkömmlichen NAT werden nur die IP-Adressen sämtlicher passierender Pakete verändert. Für H.323 ist das nicht ausreichend, da in Q.931 und H.245 ebenfalls IP-Nummern übertragen werden. Auch hier müssen sie im Rahmen von NAT angepasst werden. Dies ist aber nicht ganz einfach. Bei H.323 kann man nicht wie von einfachen Protokollen gewohnt, einfach einen fixen Offset angeben, an dem die Adresse immer steht. H.323 basiert auf der Abstract Syntax Notation One (ASN.1)[18], welche optionale Felder erlaubt. Und H.323 macht davon ausgiebig Gebrauch. Um dennoch an die gewünschten Felder zu kommen, werden die ASN.1 codierten Daten in eine C++-Klassenstruktur decodiert. Das erfolgt nach den Packed Encoding Rules (PER)[19]. Dabei führt die C++-Klassenstruktur buch über die verwendeten optionalen Felder. Nun können die IP Adressen problemlos geändert werden.
Weiterhin müssen auch die IP-Adressen im IP-Header auf die gleiche Weise angepasst werden.

7.9 Wie verfolgt man eine H.323-Verbindung?

Wir wollen die Portnummern, die von H.323 verwendet werden, erhalten. Dazu muss die H.323-Verbindung mitverfolgt werden.
Eine H.323 Applikation wartet auf dem Endpoint-Call-Signalling-Port von H.323 (TCP-Port 1720) auf eingehende Verbindungen. Um eine neue Verbindung zu erkennen, muss nur überprüft werden, wann jemand eine ausgehende Verbindung auf den TCP-Port 1720 öffnet. Diese Verbindung kann von einem beliebigen Port ausgehen. Nun müssen die Daten auf dieser Verbindung analysiert werden und wenn eine Portnummer für die H.245 Verbindung vereinbart wird, müssen auf dem Firewall die der H.245 Verbindung entsprechenden Filterrules eingetragen werden, um diese Pakete passieren zu lassen. Gleichzeitig müssen auch die Rules eingetragen werden, welche die Pakete an das NAT-Programm übergeben, damit sie analysiert werden können.
Die Portnummer des Control-Channels wird auf dem Signalling-Channel vereinbart. Und zwar in einer der folgenden Meldungen: ‚Setup', ‚Alerting', ‚CallProceeding', ‚Connect', ‚Facility' oder ‚Progress'. Die Siemens IP-Phones verwenden die ‚Setup'-Nachricht, Netmeeting verwendet ‚Connect'. Je nachdem in welcher Meldung die Portnummer gesendet wird, kann die H.245 Verbindung vom Anrufer oder vom Angerufenen initialisiert werden. Sobald die Portnummer bekannt ist, muss auf die Initialisierung des Control-Channels gewartet werden. Auf dem Control-Channel läuft nun eine analoge Analyse ab, um das Öffnen eines oder mehrerer Logical-Channels zu registrieren. Dazu können die ‚Open Logical Channel Ack'-Meldungen verwendet werden, welche die Portnummern des RTPs & RTCPs enthalten. Sobald die nötigen Daten vorhanden sind, können auf dem Firewall wieder die dazugehörigen Rules eingetragen werden, welche die Pakete des Logical-Channels passieren lassen. Diese Pakete enthalten die eigentlichen Nutzdaten (Ton, Bild, ...), welche nicht mehr analysiert werden müssen.

7.10 Zusammenfassung der Erkenntnisse

Verwendung von Netfilter

Alle geschilderten Probleme zeigten sich erst im Laufe der Implementations- und Testphase, wodurch die Zeit für eine Berücksichtigung in unserer Lösung nicht reichte. Trotzdem konnten wir eine lauffähige Implementation erreichen, die zwar einigen Einschränkungen unterliegt, welche sich aber mit zusätzlichem Aufwand sicher beseitigen liessen.
Um die Rules des H.323-Moduls von den übrigen Regeln abzutrennen, wird eine eigene Table (h323nat) und Chain (h323mod) benötigt. Alle erzeugten Rules werden dann nur in diese Table und Chain eingetragen.

Funktionsweise des H.323-Moduls

Auf dem Signalling-Channel (Q.931) wartet das Modul auf das Öffnen des Control-Channels (H.245). Sobald der gefunden wurde, wartet es auf dem Control-Channel auf das Öffnen eines Logical-Channels (G.7xx / H.26x / T.120). Die Daten darauf werden unverändert weitergegeben. Zusätzlich wird auf allen Channels NAT in den IP-Headern durchgeführt.

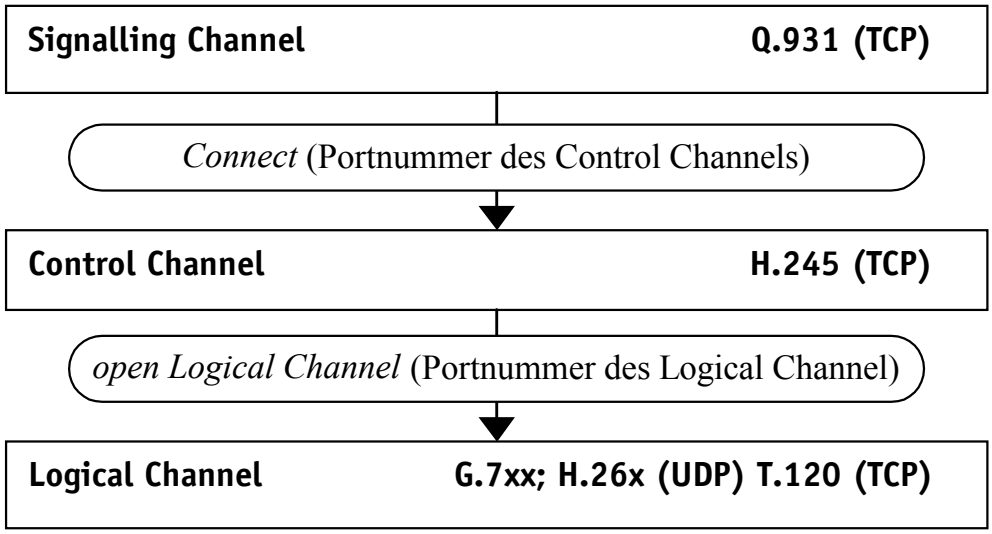


Abbildung 4: Vereinfachte Funktionsweise von H.323

8 Design

8.1 Idee

Das Design beruht auf der Idee, die Struktur einer H.323-Verbindung in eine Klassenstruktur abzubilden. UUConn repräsentiert diese Verbindung. Sie besteht aus dem Control-Channel (CtrCh) und dem Signalling-Channel (SigCh). Beide sind bidirektional und werden deshalb durch zwei Instanzen gebildet, die nur eine Richtung des Channels repräsentieren. Ausserdem können beliebig viele Logical-Channels (LogCh) zu einer Verbindung gehören. Der UUConnWrapper wurde aus programmiertechnischen Gründen eingefügt. Darauf wird im Kapitel Implementation näher eingegangen. Das H.323-Modul unterstützt mehrere Verbindungen. Diese werden in der Klasse H323mod verwaltet.

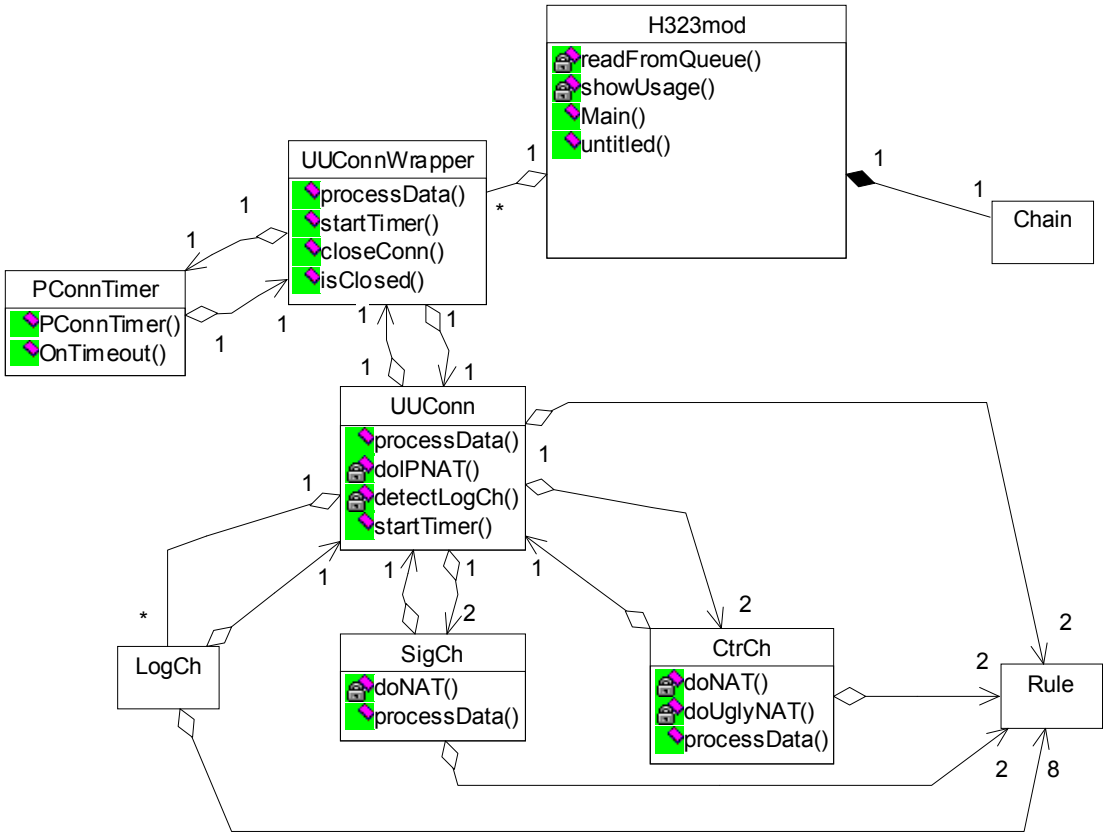


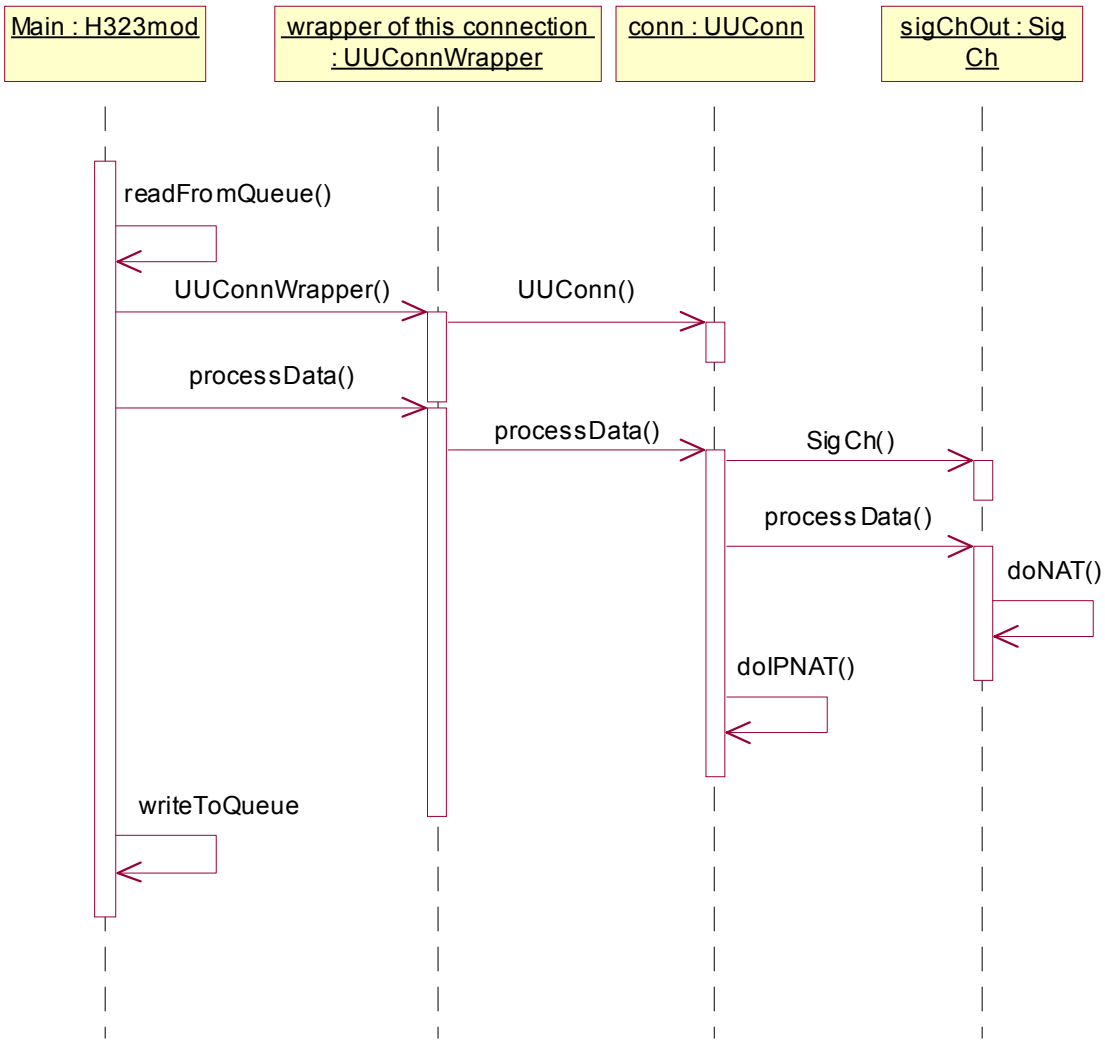
Abbildung 5: Klassendiagramm

8.2 Beschreibung der Klassen

- H323mod:
  - Initialisiert die Anwendung und liest in einer Endlosschlaufe Pakete aus der QUEUE.
  - Die Pakete werden anhand der IP-Adressen an die dazugehörende UUConn-Klasse weitergegeben. Sollte noch keine passende UUConn existieren, so wird sie erzeugt.
- UUConnWrapper:
  Eine einfache Lösung um UUConns löschen zu können, obwohl der Main-Loop in einem Blocking-read ist. Die Details werden im Kapitel Implementation besprochen.
- UUConn:
  - Umfasst eine vollständige User to User Verbindung. Eine Verbindung wird durch die IP-Adressen der beiden Kommunikationspartner identifiziert.
  - Die Daten werden anhand des Protokolls (TCP oder UDP) und den Portnummern den einzelnen Channels zugeordnet.
  - Es wird überprüft, ob ein Logical-Channel geöffnet wurde.
  - Es wird NAT auf IP-Ebene durchgeführt.
- SigCh:
  - Hier wird der Signalling-Channel mit den dazugehörigen Rules verwaltet und analysiert.
  - Es wird NAT in den Q.931-Daten durchgeführt.
- CtrCh:
  - Hier wird der Control-Channel mit den dazugehörigen Rules verwaltet und analysiert.
  - Es wird NAT in den H.245-Daten durchgeführt.
- LogCh:
  - Hier werden alle Rules eines Logical-Channels verwaltet. Es werden nur die UDP-Protokolle G.7xx und H.26x unterstützt.
- Rule:
  - Legt eine Rule in den IP-Tables an. Sie wird beim Erzeugen der Instanz definiert. Beim Löschen wird sie wieder aus den Tables entfernt.
- Chain:
  - Diese Klasse legt eine benutzerdefinierte Chain in der FORWARD-Chain der filter-Table an.
  - In dieser Chain und in der POSTROUTING-Chain der h323nat-Table werden auch gleich die Rules angelegt, welche die für TCP-Port 1720 bestimmten Daten an unser Programm übergeben.
- PConnTimer:
  - Sobald der Timer einer UUConn abgelaufen ist, meldet er seinem UUConnWrapper, dass die Verbindung nicht mehr benötigt wird und gelöscht werden kann.

Es gibt noch einige Hilfsklassen auf die erst im Kapitel Implementation eingegangen wird.

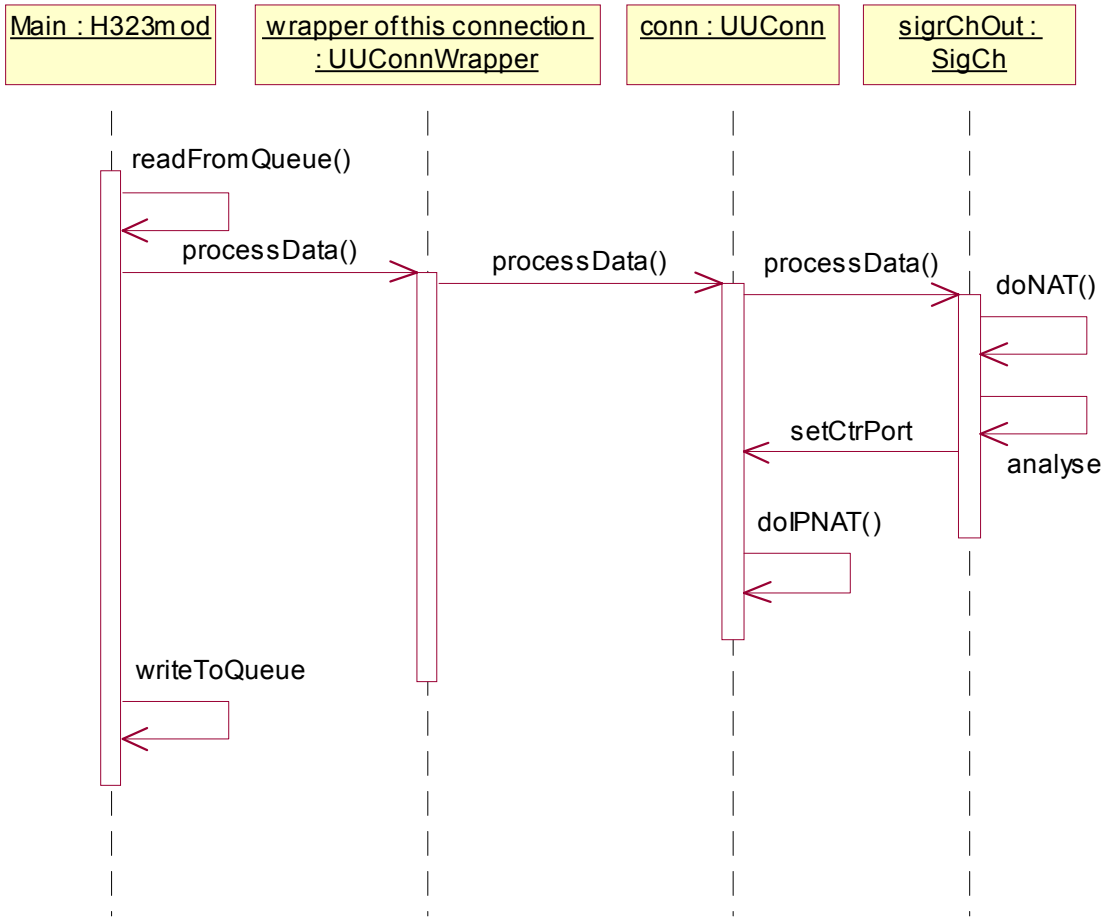
8.3 Ablauf beim Aufbau einer neuen Verbindung



1. Main liest ein Paket aus der QUEUE und stellt, fest ob es zu einer bereits bestehenden Verbindung gehört.
2. Da dies nicht der Fall ist, wird ein neuer Wrapper erzeugt, welcher wiederum eine neue UUConn erzeugt.
3. Nun werden die Daten via Wrapper an die UUConn übergeben.
4. Diese stellt fest, dass es sich um eine neue Verbindung handelt und erzeugt einen SigCh.
5. Diesem werden die Daten übergeben, und er führt falls nötig das H.323-NAT durch. Da es sich beim ersten Paket immer um ein TCP-SYN-Paket handelt, welches keine Daten enthält, wird das nicht nötig sein.
6. Jetzt kann die UUConn das IP-NAT ausführen.
7. Das fertige Paket wird nun zurück in den Kernel geschrieben.

8.4 Ablauf beim Öffnen des Control-Channels

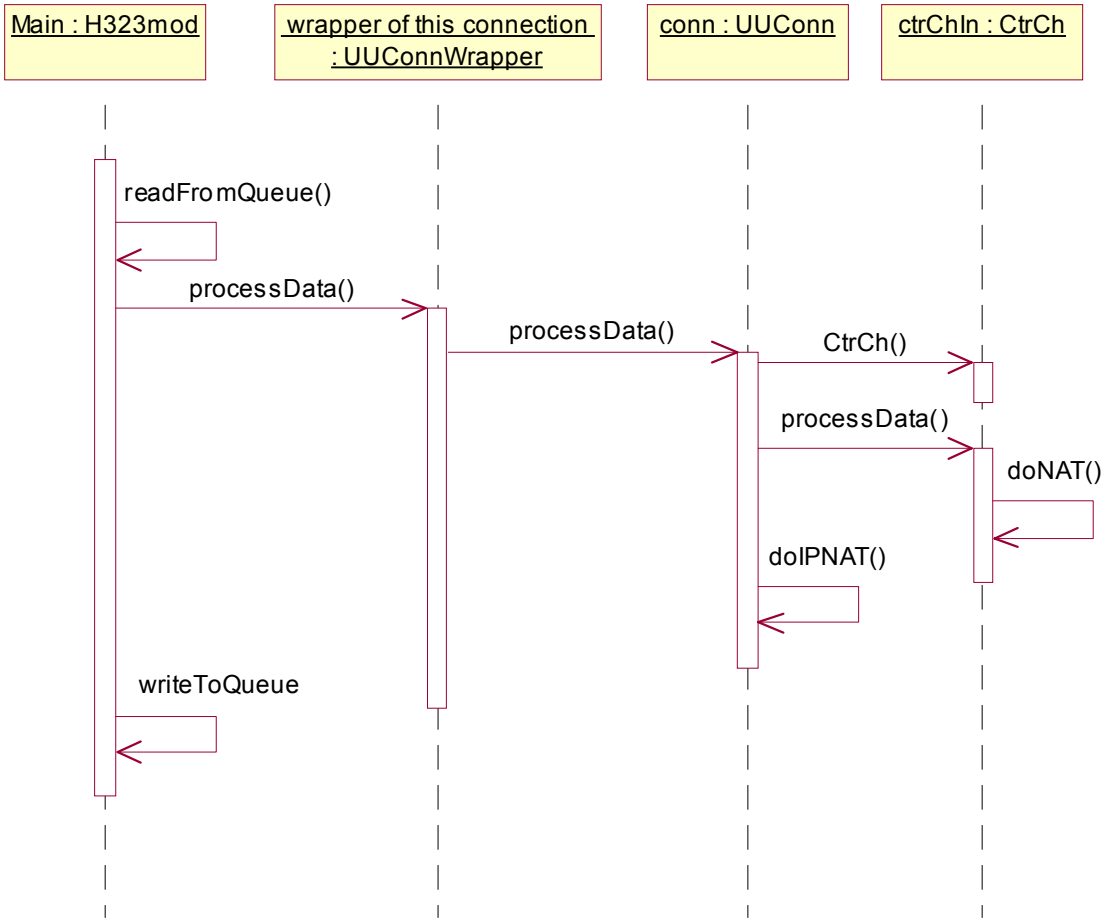
Ermitteln des Control-Channel Ports



Bei den IP-Phones wird der Port in der ‚Setup'-Meldung übermittelt. Deshalb wird hier ein Paket, das diese Meldung enthält, betrachtet.

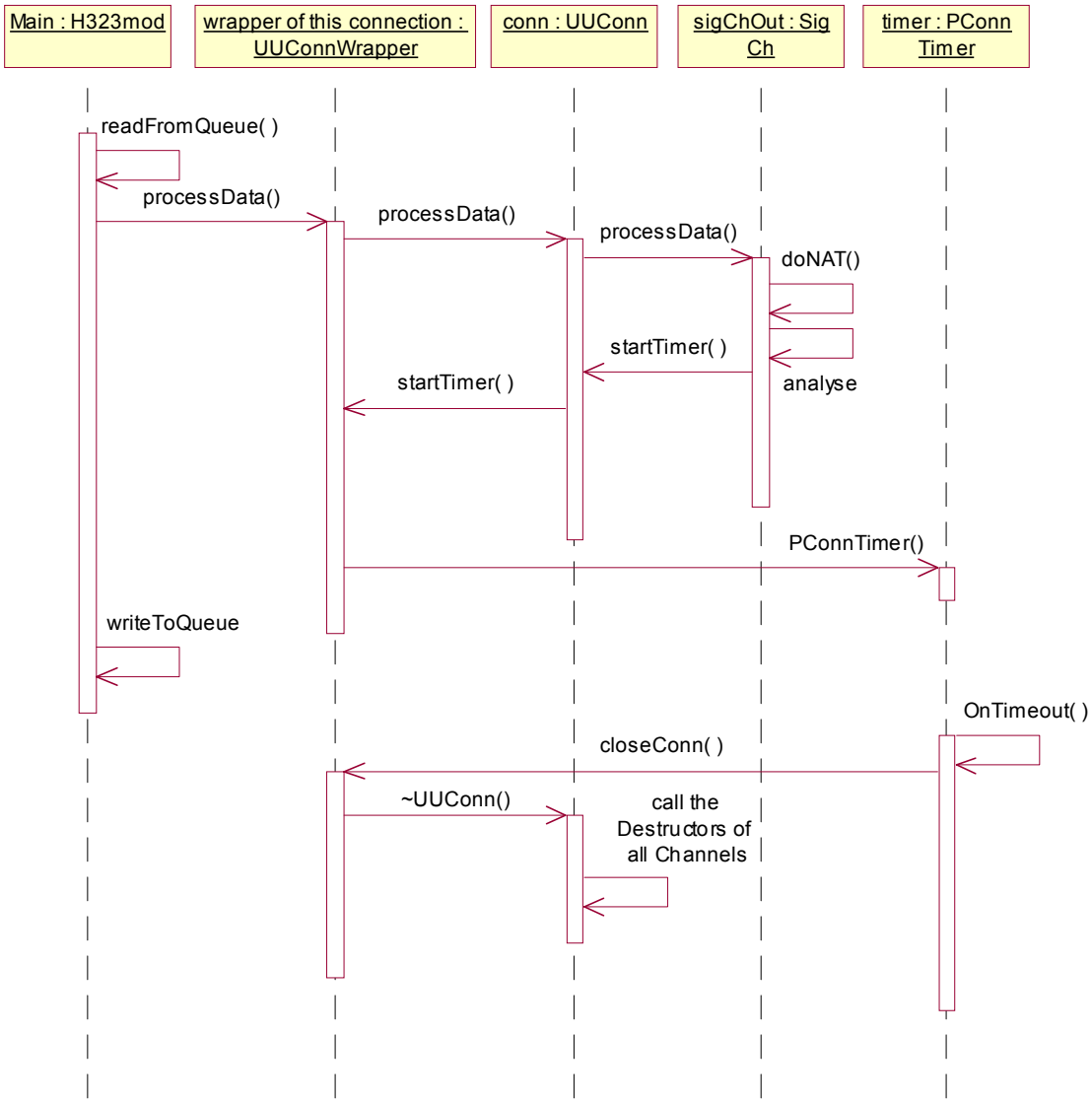
1. Wenn das Paket übermittelt wird, wird es auf dieselbe Weise wie bei einer neuen Verbindung an den SigCh übergeben. Nur existiert die Verbindung und der SigCh bereits und sie müssen deshalb nicht mehr erzeugt werden.
2. Nachdem der SigCh das NAT durchgeführt hat, stellt er bei der Analyse der Daten fest, das es sich um eine ‚Setup' Meldung handelt, die den Control-Channel Port enthält. Dieser wird an die UUConn übergeben.
3. Das IP-NAT und die Rückgabe des Paketes läuft analog zum Ablauf bei einer neuen Verbindung

Die erste Meldung für den Control-Channel



1. Nun wird das erste Paket des Control-Channel gelesen. Es wird auf die bekannte Weise an die UUConn übergeben.
2. Sobald die UUConn anhand des zuvor ermittelten Ports erkennt, dass es sich um den Control-Channel handelt, wird ein neuer CtrCh erzeugt.
3. Diesem werden die Daten übergeben, und er führt falls nötig das H.323-NAT durch. Auch hier handelt es sich beim ersten Paket immer um ein TCP-SYN-Paket, so dass kein NAT nötig ist.
4. Der Rest ist wieder identisch mit dem Ablauf beim Aufbau einer neuen Verbindung

8.5 Ablauf beim Schliessen einer Verbindung



1. Der SigCh erhält das Paket auf dem bekannten Weg und führt das H.323-NAT aus.
2. Bei der Analyse stellt er fest, das es sich um eine ‚Release Complete'-Meldung handelt.
3. Er meldet das via die UUConn dem Wrapper.
4. Dieser erzeugt einen neuen Timer, der in 5 Sekunden abläuft.
5. Das Paket wird wieder zurückgegeben.
6. Nach den 5 Sekunden wird vom Timer OnTimeout() aufgerufen.
7. Von dort wird dem Wrapper mitgeteilt, dass die Verbindung beendet ist.
8. Der Wrapper löscht die Verbindung und mit ihr alle Channel Objekte.

Der Wrapper selbst wird nicht sofort gelöscht. Er wird erst vom Hauptprogramm gelöscht, wenn dieses durch ein weiteres Paket aus dem blocking-wait der Funktion readFromQueue() befreit wird.

9 Implementation

9.1 Allgemeines

Der Quellcode wurde mit doc++-Kommentaren versehen. Der aus diesen Kommentaren generierte HTML-Output befindet sich auf der CD.

Bibliotheken

In dieser Arbeit wurde der H.323 Stack des Open H.323 Project [6] verwendet. Daraus wurden nur die Klassen in den Dateien q931.h, h225.h und h245.h verwendet. Diese repräsentieren die Struktur von H.323. Der H.323 Stack basiert auf der PWLib [7] von Equivalence Ltd. Deshalb wurde auch diese Library verwendet. Man erkennt alle Klassen der PWLib daran, dass sie mit ‚P' beginnen. Das Ziel der PWLib ist es, eine plattformunabhängige Standard-Library bereitzustellen. Um dies zu erreichen, wird ausgiebig mit Macros gearbeitet, was die Lesbarkeit des Codes nicht gerade erhöht. Als Lesehilfe wurde an diesen Stellen erklärender Kommentar eingefügt. Eine ausführliche Dokumentation ist auf der Homepage vorhanden.
Für die Kommunikation mit den IP-Tables wird noch die libipq benötigt. Die Verwendung wird in der Beschreibung der Klasse H323mod erläutert.

Byteorder

Um Werte von Network-Byteorder in Host-Byteorder zu konvertieren, stellt Linux folgende Funktionen zur Verfügung: `ntohs()`, `htons()`, `ntohl()`, `htonl()`. Diese Kürzel stehen für „Network to Host short / long" und umgekehrt.

IP-Adressen

Beim Umgang mit IP-Adressen, müssen sie in die für Menschen lesbare Dot-Notation (z.B. 192.168.131.59) gebracht werden können. Dazu stehen die Funktionen `inet_aton()`, `inet_ntoa()` zur Verfügung. Das bedeutet „Internet Ascii to Network" und umgekehrt. IP-Adressen werden immer in eine `struct in_addr` abgelegt (aus `<netinet/in.h>`). Diese Struct enthält nur das Feld `s_addr`, welches die Adresse enthält.

9.2 Inhalt der Dateien

Dateiname	Inhalt
main.h, main.cxx	Definition der UUConn-Hashtable mittels PDICTIONARY() class H323mod
uuConn.h, uuConn.cxx	const H323_SIGNALLING_PORT = 1720 struct tpkhdr struct pseudo_header class LogCh class CtrCh class SigCh class UUConn class PConnTimer class UUConnWrapper
tcpstream.h, tcpstream.cxx	#define OUTGOING 0 #define INCOMING 1 class TcpStreamKey class TcpStreamData class H225withH245

	class Q931withH225
rule.h, rule.cxx	class Rule
chain.h, chain.cxx	class Chain
iptable_h323nat.c	unser h323nat-Table

9.3 Erklärungen zu den Klassen

Die folgenden Erklärungen sollten zusammen mit dem Code gelesen werden.

H323mod

Als erstes werden die Parameter der Kommandozeile geparst. Dazu wird die Klasse PArgList verwendet. Dieser Klasse wird ein String übergeben, der sämtliche möglichen Optionen enthält. Beim verwendeten String `"a:hi:v"` bedeutet das, dass die Optionen -a, -h, -i und -v vorhanden sein können, wobei nach -a und -i ein Parameter stehen muss. Der Aufruf von h323mod hat folgende Syntax: `h323mod [-a IP-Address] [-h] [-i Interfacename] [-v]`
Jetzt wird mit Hilfe der libipq die Verbindung zum Kernel erstellt. Dazu wird die Funktion `ipq_create_handle()` aufgerufen. Von dieser erhält man einen Handle mit dem die QUEUE identifiziert werden kann. Nun muss ihr noch gesagt werden, dass Pakete bis zu einer Grösse von 64kByte übergeben werden sollen. Das geschieht mit `ipq_set_mode()`.
Nun beginnt die Endlosschleife in der Funktion readFromQueue(). Mittels `ipq_read()` werden die ankommenden Pakete einzeln ausgelesen. Diese Funktion kehrt nur zurück, wenn ein Paket empfangen wurde. Daraus entstanden verschiedene die Probleme mit dem Programmende und dem löschen von Verbindungen nach deren Beendung. Nun überprüft `ipq_message_type()`, ob es sich um ein gültiges Paket handelt. Falls das der Fall ist, muss noch `ipq_get_packet()` aufgerufen werden um das IP-Paket zu erhalten. Dieses wird an die dazugehörende Verbindung übergeben. Zu diesem Zweck werden die Verbindungen in einer Hashtable verwaltet. Als Hashkey dient die IP-Adresse des Teilnehmers, der sich ausserhalb des Firewalls befindet.
Bei Beenden des Moduls sollten alle Regeln wieder gelöscht werden. Zur Zeit funktioniert das noch nicht. Eine Lösung wäre es ein Signal Handler zu schreiben der Ctr-C abfängt und dann die Rules löscht. Das ist nicht ohne weiteres möglich, da das Hauptprogramm in einem eigenen Prozess läuft und wir dadurch keinen Zugriff mehr auf dessen Variablen haben. Für eine andere Lösung muss die Funkton `ipq_read()` verändert werden, damit sie auch auf beim Programmende zurückkehrt.

UUConnWrapper

Diese Klasse wurde eingeführt um folgendes Problem zu lösen: Die Funktion `ipq_read()` aus der Libipq ist blockierend. Das heisst das Programm kehrt nur vom Aufruf zurück, wenn ein Paket gelesen wurde. Gleichzeitig sollte aber eine Verbindung sofort nach der letzten Meldung gelöscht werden. Schliesslich müssen die Firewall-Rules wieder gelöscht werden. Da sich die letzte Meldung aber nur bestimmen liesse, indem man den TCP Verbindungsabbau mit sämtlichen Spezialfällen mitverfolgen würde, haben wir darauf verzichtet und benutzen stattdessen einen Timer, der ein paar Sekunden nach dem H.323 Verbindungsabbau auslöst. Da wir den H.323 Verbindungsabbau problemlos erkennen können, ist das viel einfacher.

Der Ablauf:

Es trifft das Timersignal ein. Doch das Hauptprogramm wartet schon wieder auf ein neues Paket. Damit wir die Verbindung trotzdem löschen können, haben wir als Zwischenschicht den Wrapper eingeführt. Die Verbindung wird nun aus dem Wrapper gelöscht und nicht aus

dem Hauptprogramm. Der Wrapper wird später vom Hauptprogramm folgendermassen gelöscht:

```
for ( PINDEX i = 0; i < uuConns.GetSize(); i++ ) {
    // is connection still in use?
    if ( uuConns.GetDataAt(i).isClosed() ) {
        // it's not. close it
        uuConns.RemoveAt( uuConns.GetKeyAt(i) );
        // we've removed an element. So we've to decrement the counter
        i--;
    }
}
```

Speziell an dieser Schleife ist, dass der Zähler innerhalb der Schleife dekrementiert wird. Das ist nötig, wenn wir ein Element aus der Liste löschen und dadurch alle anderen Elemente um eins nach vorne rutschen. Ausserdem ändert sich der Wert der Funktion im Abbruchkriterium. Wenn man den Zähler nicht dekrementierte, würde ein Element übersprungen werden.
Im Wrapper wird dafür gesorgt, dass nicht auf die Verbindung zugegriffen wird, nachdem sie gelöscht wurde. Da das Überprüfen und Schreiben aber nicht als atomare Operation ausgeführt wird, sind Fehler möglich. In einer weiteren Version sollte das behoben werden. Da eine Verbindung normalerweise aber nicht beendet und sofort wieder geöffnet wird, sollte diese Situation eigentlich nie eintreten.

UUConn

Die verschiedenen Portnummern werden von den Channel-Klassen direkt gesetzt. Ein Logical-Channel wird erzeugt, sobald alle Portnummern gesetzt wurden. Anschliessend werden die Nummern zurückgesetzt und sind für den nächsten Channel bereit.
Die verschiedenen Logical-Channels werden in einer einfachen Liste (PList) verwaltet.

SigCh und CtrCh

Hier werden die Daten analysiert. Dazu werden die Klassen des Open H.323 Projects verwendet. Für den Signalling-Channel benutzen wir noch die Hilfsklasse Q931withH225 (siehe unten).
In diesen Klassen gab es ein Problem mit der ASN.1-Codierung, die von der PWLib zur Verfügung gestellt wurde. Wenn wir die Daten die wir erhielten decodierten und ohne sie zu verändern sofort wieder codierten, entsprach das Ergebnis nicht mehr den ursprünglichen Daten. Da das ganze nach den komplizierten PER (Packed Encoding Rules) codiert wird, konnten wir nicht endgültig feststellen, ob überhaupt und falls ja, wo genau der Unterschied oder Fehler liegt.
Beim SigCh genügt es die zusätzlichen Bytes zu ignorieren, es funktioniert trotzdem. Das ging beim CtrCh nicht so einfach. Die Funktion doNAT() welche das NAT auf korrekte Weise durchführt, kann nicht benutzt werden. Stattdessen verwenden wir die Funktion doUglyNAT(). Diese decodiert die Daten nicht, sondern durchsucht sie nach der IP-Adresse und ersetzt diese direkt. Solange das Bytemuster der IP-Adresse nicht in einem anderen Zusammenhang vorkommt, klappt das wunderbar. Aber trotzdem lauert hier eine potentielle Fehlerquelle.

Der zentrale Loop von doUglyNAT() für Verbindungen nach aussen:

```
for ( __u16 i = 0; i < len - 3; i++ ) {
    if ( memcmp( &per[i], &conn->localAddr.s_addr, 4 )==0 ) {
        memcpy( &per[i], &conn->fwAddr.s_addr, 4 );
    }
}
```

Die Daten sind im Array `per` gespeichert, `len` ist die Länge des Arrays. Da die IP-Adresse 4 Bytes lang ist, können wir 3 Bytes vor dem Ende des Arrays mit Vergleichen aufhören. Gesucht wird nach der localAddr (Adresse des Teilnehmers innerhalb des Firewalls) welche durch fwAddr (Adresse des Firewalls) ersetzt wird.

Rule und Chain

Die beiden folgenden Hilfsklassen stellen die Schnittstelle zu iptables dar. Ursprünglich war geplant, diese beiden Klassen mit Hilfe der ‚libiptc' zu realisieren, einer C-Library, welche Funktionen zum Konfigurieren der iptables-Schicht zur Verfügung stellt. Diese Library ist nicht genügend dokumentiert, abgesehen von ein paar knappen Kommentarzeilen im Quellcode. Einige der für die Rule-Klasse benötigten Funktionen erwarten Aufrufparameter, welche wohl höchstens für Kernelhacker intuitiv verständlich sind.
Aufgrund dieser Widrigkeiten schien es vernünftiger, eine Stufe höher anzusetzen: Das iptables-Programm bedient sich ebenfalls der ‚libiptc', warum also nicht auf dessen Quellcode abstützen? Die einfache Lösung war gefunden: do_command() aus iptables.c aufrufen.

C/C++-Inkompatibilitäten

Dies führte zur nächsten Hürde: Eine in der C-Library enthaltene Funktion war nicht C++-kompatibel! Genauer gesagt wurde void-pointer-Arithmetik betrieben. Also passten wir die entsprechende Headerdatei (ip_tables.h) an, und ein Test zeigte, dass es funktioniert. Diese Anpassung haben wir Rusty Russel und Harald Welte mitgeteilt. Letzterer befand die Änderung für brauchbar.
Bald zeigten sich jedoch neue Schwierigkeiten. Es funktionierte nur beim Anlegen der ersten Rule. Beim Debugging zeigt sich, dass der Fehler innerhalb von do_command() beim Aufruf der Systemlibrary-Funktion getopt() auftritt. An dieser Stelle schien weiteres Debugging nicht mehr sinnvoll. Kurz vor Abgabe der Diplomarbeit lieferte Harald Welte eine Erklärung für das Fehlverhalten [s. Kasten: „Harald Welte's posting to the netfilter-devel mailing list"].

Simple and Dirty

Nun musste eine unschöne, jedoch einfach zu realisierende Lösung gefunden werden: Der Aufruf des iptables-Programms direkt aus dem C++-Code. Damit liessen sich beide vorher geschilderten Klippen umschiffen.

Rule im Detail

Es können vier verschiedene Arten von Regeln angelegt werden. Diese unterscheiden sich in der Anzahl der spezifizierten TCP- resp. UDP-Ports. Im allgemeinsten Fall werden gar keine Ports festgelegt, sondern nur die IP-Adressen angegeben. Dann gibt es noch die Varianten, dass entweder Quell- oder Zielport und IP-Adressen spezifiziert werden. Und last but not least kann eine Regel erzeugt werden, die Quell-, Zielport und IP-Adressen berücksichtigt. Dazu stehen vier verschiedene Konstruktoren zur Verfügung.
Die Parameter und der Typ der Regel werden in Instanzvariablen gespeichert, da diese Informationen beim Ausführen des Destruktors wieder benötigt werden, um die Regeln zu löschen. Die meisten der Konstruktorparameter werden als char-Pointer übergeben, weshalb die Hilfsfunktion initVars() diese Zeichenketten in die Instanzvariablen kopiert.

Wie bereits erwähnt, werden die Regeln durch Ausführen des Programms iptables im Firewall angelegt. Das Ausführen von iptables geschieht mittels Aufruf des execl()-Systemcall, nachdem zuvor mit fork() ein Kindprozess erzeugt wurde.

Chain im Detail

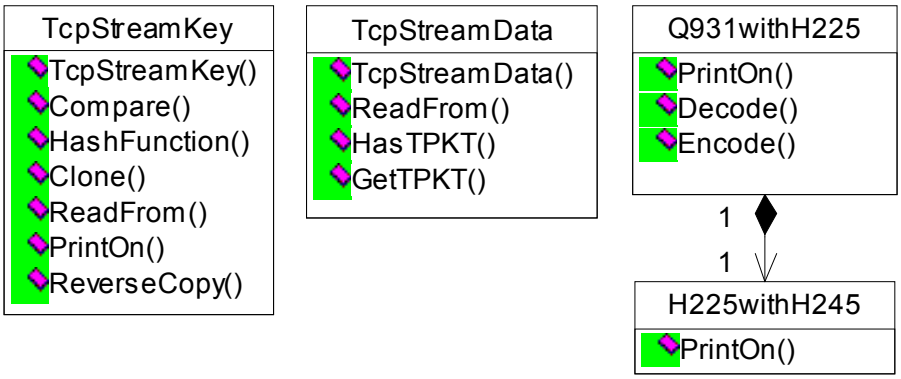
Diese Klasse erzeugt eine benutzerdefinierte Chain in der filter-Table, und legt Regeln in der FORWARD-Chain der filter-Table an, welche allen Verkehr in diese benutzerdefinierte Chain umleiten. In ihr werden die Regeln eingetragen, welche die Daten des H.323-Endpoint-Signalling-Channel passieren lassen. Gleichzeitig wird in unsere h323nat-Table eine Rule eingetragen welche diese Daten ins QUEUE-Target weiterleitet.
Analog zur Rule-Klasse wird das Programm iptables ausgeführt, um diese Änderungen an der Firewallkonfiguration durchzuführen.

Harald Welte's posting to the netfilter-devel mailing list

```
From netfilter-devel-admin@us5.samba.org Thu Oct 26 07:32:27 2000
From: Harald Welte <laforge@gnumonks.org>
To: Derrik Pates <dpates@andromeda.dsdk12.net>
Cc: Netfilter Development Mailinglist <netfilter-devel@lists.samba.org>
Subject: Re: Ok, this _is_ a libiptc problem...
Sender: netfilter-devel-admin@lists.samba.org
List-Archive: http://lists.samba.org/pipermail/netfilter-devel/
Date: Thu, 26 Oct 2000 09:26:51 +0200

On Sat, Oct 21, 2000 at 01:37:08AM -0600, Derrik Pates wrote:
> Ok, my last issue was not libiptc's fault. However, I have been able to
> reproduce a new issue - if I try to do more than one rule operation or
> chain creation during a libiptc session, the second instance will segfault
> when it free()s the old session handle (it actually SEGV's in the
> free()). I have been able to track this down to, apparently,
> alloc_handle() not allocating enough memory, and somewhere along the line,
> something is getting overwritten that causes the free() to segfault. If I
> just do a single operation (add a rule, create a chain), all's happy and
> fine. I know it's not a problem with my rule creation polluting anything,
> because if I just create two chains (no creating rules in my code), the
> result is the same.


mmh... I ran into the same difficulties while re-writing iptables-restore.
There are more than one issue involved:

1. The alloc_handle() problems as described
2. If anybody calls do_command() from iptables.c more than one time
   he has to
    - Reset the getopt() system by setting optind = 0 prior to calling
      any getopt function
    - Reset the match-specific flags prior to call any match->save
      function
3. Another problem I was unable to track down yet, which results in only
the last "append" operation being executed.

It seems like a lot of stuff in libiptc is untested in the 'more than
one operation per transaction' case. I'm going to fix this over the next
few days.

I already have preliminary patches for the issues with (2) but don't want
to put them into cvs until everything is tested. But if anybody needs them
now, mail me.

> Derrik Pates      | Sysadmin, Douglas School|    _   #linuxOS on EFnet

--
Live long and prosper
- Harald Welte / laforge@gnumonks.org                http://www.gnumonks.org
============================================================================
GCS/E/IT d- s-: a-- C+++ UL++++$ P+++ L++++$ E--- W- N++ o? K- w--- O- M-
V-- PS+ PE-- Y+ PGP++ t++ 5-- !X !R tv-- b+++ DI? !D G+ e* h+ r% y+(*)
```

Die Hilfsklassen

Diese Klassen werden verwendet, um die Handhabung von ITOT über TCP und Q.931 mit H.225 zu vereinfachen. Grösstenteils sind sie aus dem H323dump, das Bestandteil des Open H.323 Projects ist, entnommen. Nur das Encoding musste hinzugefügt werden.



TcpStreamKey:

Diese Klasse enthält die IP-Adressen und Portnummern einer TCP-Verbindung. Die Klasse kann auch als Hashkey verwendet werden, sodass damit eine UUConn identifiziert werden kann.

TcpStreamData:

Diese Klasse ist ein Array in dem TCP Daten abgelegt werden. Es lässt sich überprüfen ob ein vollständiges TPKT in den Daten ist, und in diesem Fall kann man die Payload des Paketes auslesen.

Q931withH225:

In Q.931 wird nur übertragen um welche Art von Meldung es sich handelt (z.B. Setup). Die eigentliche Meldung wird in H.225 übertragen. Diese Klasse fasst diese Strukturen zusammen.

H225withH245:

H.225 erlaubt das Tunneling von H.245. Diese Klasse überprüft, ob dies der Fall ist, falls ja werden die H.245-Daten decodiert.

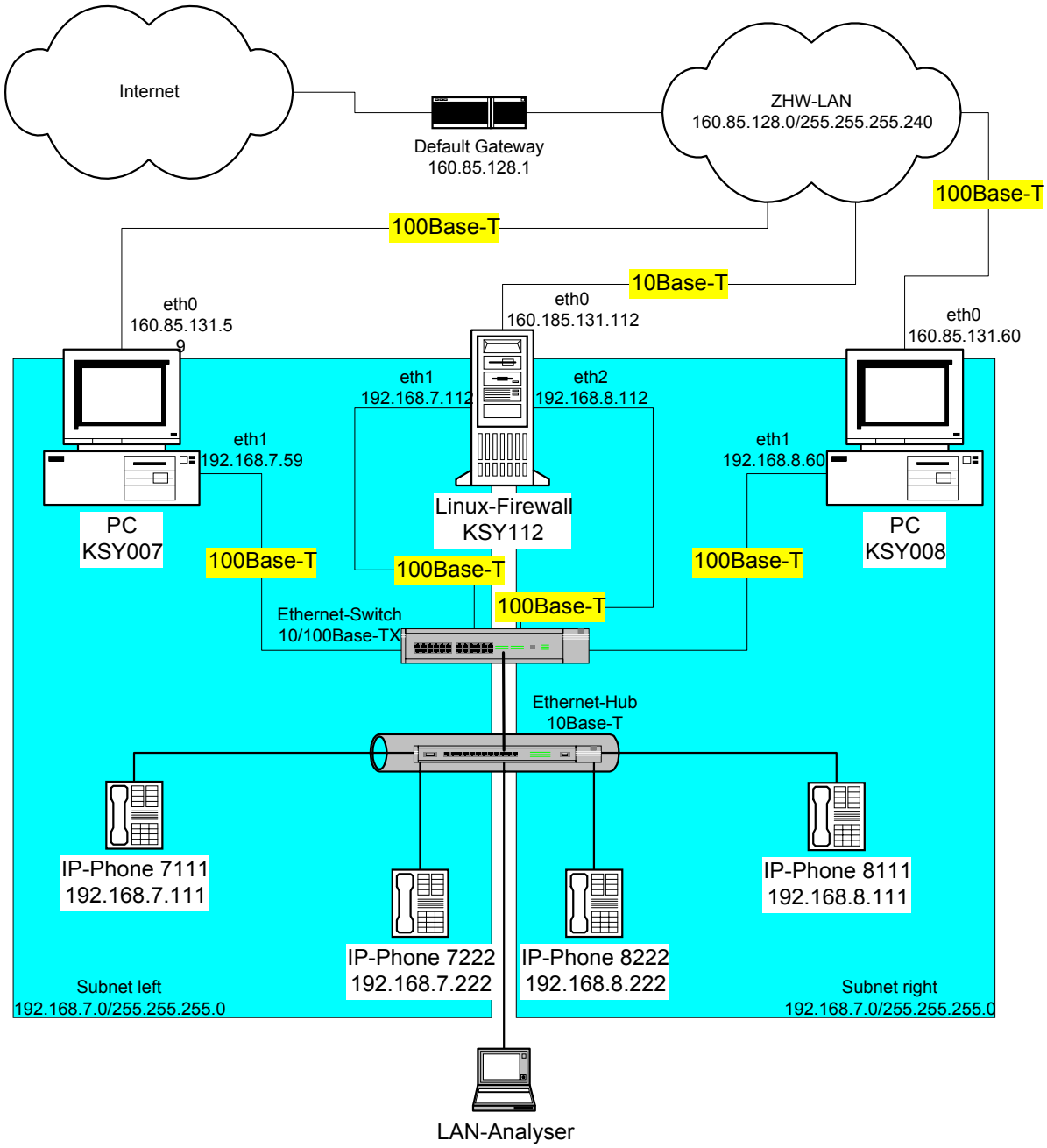
9.4 Die wichtigsten Header aus Linux im Überblick

`in.h`	`struct in_addr`
	Definition sämtlicher IP-Protokolle:
	`IPPROTO_TCP = 6`
	`IPPROTO_UDP = 17`

`ip.h`	`struct iphdr`
`tcp.h`	`struct tcphdr`
`udp.h`	`struct udphdr`

10 Test

10.1 Testanordnung



Logische Struktur

Die Testanordnung besteht aus zwei C-Klasse-Subnetzen (`left´ und `right´), welche durch den Firewall getrennt resp. verbunden sind. Der Vollständigkeit halber ist in der Abbildung die gesamte Netzstruktur dargestellt, inklusive der Anbindung der drei PCs ans ZHW-LAN. Für die Tests ist dies irrelevant, da hier nur die beiden Subnetze interessieren. Subnetz `left´ stellt das geschützte, hinter dem Firewall liegende, Intranet dar; Subnetz right repräsentiert das Internet.

Physikalische Struktur

Auffällig ist, dass die beiden logischen Subnetze über den gleichen Switch geführt werden. Das ist völlig praktikabel, da die Routing-Tabellen der PCs so konfiguriert wurden, dass die beiden Subnetze trotzdem getrennt sind [s. Routing-Konfiguration].
Da ein Ethernet-Switch in der Funktionweise einer Multiport-Bridge entspricht, so dass die einzelnen Ports eigene Ethernet-Segmente darstellen, bekommt der LAN-Analyser nichts vom Netzwerkverkehr auf den übrigen Ports mit. Deshalb muss zusätzlich ein Ethernet-Hub verwendet werden, an welchem der LAN-Analyser angeschlossen wird. Die IP-Phones oder PCs deren Kommunikation mit dem Firewall oder untereinander mitverfolgt werden soll werden auch an dem Hub angeschlossen.

10.2 Routing-Konfiguration

Die beteiligten Geräte (PCs und IP-Phones) wurden auf die beiden Subnetze verteilt. Als Gateway für das jeweils andere Subnetz wurde immer der Firewall eingetragen. Die Details der Konfiguration, wie z.B. IP-Adressen und Subnetzmasken, können der Abbildung entnommen werden.

Firewall (KSY112)

In SuSE-Linux kann die gewünschte Routing-Konfiguration bequem in der Datei `/etc/route.conf` eingetragen werden. Auf dem Firewall sieht das für unsere Testanordung folgendermassen aus:

```
Destination          Gateway              Netmask              Device
192.168.7.0          0.0.0.0              255.255.255.0        eth1
192.168.8.0          0.0.0.0              255.255.255.0        eth2
```

Netmeeting-Clients (KSY007, KSY008)

Unter Windows NT werden die Routing-Tabellen mit dem route-Kommando manipuliert. Die Syntax lautet folgendermassen:

```
route /p ADD 192.168.7.0 MASK 255.255.255.0  192.168.7.112 IF 3
                  Ziel^      ^Maske          ^Gateway         ^
                                                 Schnittstelle^
```

"/p" braucht nur angegeben zu werden, falls eine permanente Eintragung der Route gewünscht wird. Permanent bedeutet in diesem Fall, dass die Route über einen Neustart des Rechners hinaus eingetragen bleibt.

IP-Phones (7111, 7222, 8111, 8222)

Wie man das Routing auf den IP-Phones konfiguriert, steht im Kapitel Konfiguration der IP-Phones [siehe Kapitel 11.1].

10.3 Testablauf

IP-Phones

1. Starten unserer Firewall-Extension (./h323mod)
2. Starten des LAN-Analysers
3. Anruf von Phone 7111 zu Phone 8111
4. Anruf annehmen
5. Verbindung prüfen
6. Anruf von Phone 7222 zu Phone 8222
7. Anruf annehmen
8. Verbindung prüfen
9. Beide Verbindungen beenden
10. Anruf von Phone 7111 zu Phone 8111

11. Anruf annehmen
12. Verbindung prüfen
13. Verbindung beenden

Netmeeting

1. Verbindung von KSY007 auf KSY008 aufbauen.
2. Verbindung prüfen
3. Verbindung trennen.

10.4 Testresultate

Um die korrekte Funktion unseres Programms zu überprüfen, benutzen wir einen LAN-Analyser. Der LAN-Analyser sammelt alle IP-Pakete die vom oder zum externen IP-Phone gehen.
Wenn zwischen den Telefonen eine Verbindung zustande kommt, wissen wir bereits, dass nichts Falsches gemacht wird. Ob aber das Gewünschte gemacht wird, erkennt man erst in der Ausgabe des Analysers. Das Wichtigste sieht man auf den ersten Blick: Es gelangen keine privaten IP-Adressen nach aussen. Alle Pakete scheinen zwischen dem IP-Phone (192.168.8.111) und dem Firewall (192.168.8.112) hin und her zu gehen. Und das ist der Sinn von NAT.

LAN-Analyser Dump

Dieser Dump enthält nur Verbindungauf- und –abbau einer Verbindung, da er sonst viel zu umfangreich geworden wäre.

Destination	Source	Interpretation
Verbindungsaufbau:		
192.168.8.111	192.168.8.112	TCP D=1720 S=1026 SYN
192.168.8.112	192.168.8.111	TCP D=1026 S=1720 SYN
192.168.8.111	192.168.8.112	TCP D=1720 S=1026
Übermitteln des Control-Channels Ports (= 1027):		
192.168.8.111	192.168.8.112	H.225.0 SETUP User_Msg_Type=Setup Addr(IPv4)=192.168.8.112 Port=1027
192.168.8.112	192.168.8.111	TCP D=1026 S=1720
Öffnen des Control-Channels:		
192.168.8.112	192.168.8.111	TCP D=1027 S=1024 SYN
192.168.8.111	192.168.8.112	TCP D=1024 S=1027 SYN
192.168.8.112	192.168.8.111	TCP D=1027 S=1024
Datenübertragung auf dem Control-Channel beginnt (LAN-Analyser erkennt den Port in der Setup Meldung leider nicht):		
192.168.8.111	192.168.8.112	TCP D=1024 S=1027 PUSH
192.168.8.112	192.168.8.111	H.255.0 ALERTING User_Msg_Type=Alerting
192.168.8.111	192.168.8.112	TCP D=1720 S=1026
Daten auf dem Control-Channel (gekürzt):		
192.168.8.112	192.168.8.111	TCP D=1027 S=1024 PUSH
192.168.8.111	192.168.8.112	TCP D=1024 S=1027 PUSH
192.168.8.112	192.168.8.111	TCP D=1027 S=1024 PUSH
192.168.8.112	192.168.8.111	TCP D=1027 S=1024
192.168.8.111	192.168.8.112	TCP D=1024 S=1027
...	...	
192.168.8.111	192.168.8.112	TCP D=1024 S=1027 PUSH
192.168.8.112	192.168.8.111	TCP D=1027 S=1024
192.168.8.112	192.168.8.111	H.225.0 CONNECT User_Msg_Type=Connect
192.168.8.111	192.168.8.112	TCP D=1720 S=1026
Daten auf dem Logical-Channel (gekürzt):		
192.168.8.112	192.168.8.111	UDP D=1893 S=1890 Len=500
192.168.8.112	192.168.8.111	UDP D=1893 S=1890 Len=500

```
      ...              ...
  192.168.8.112   192.168.8.111  UDP D=1893 S=1890 Len=500
  192.168.8.112   192.168.8.111  UDP D=1893 S=1890 Len=500

  192.168.8.111   192.168.8.112  TCP D=1024 S=1027 PUSH
  192.168.8.112   192.168.8.111  TCP D=1027 S=1024
  192.168.8.112   192.168.8.111  TCP D=1027 S=1024 PUSH
  192.168.8.111   192.168.8.112  TCP D=1024 S=1027 PUSH
```
Control-Channel wird geschlossen:
```
  192.168.8.111   192.168.8.112  TCP D=1024 S=1027 FIN
  192.168.8.112   192.168.8.111  TCP D=1027 S=1024
```
Verbindung ist beendet:
```
  192.168.8.111   192.168.8.112  H.225.0 RELEASE COMPLETE
                                  User_Msg_Type=Release Complete
  192.168.8.112   192.168.8.111  TCP D=1027 S=1024 PUSH
  192.168.8.111   192.168.8.112  TCP D=1024 S=1027
  192.168.8.111   192.168.8.112  TCP D=1720 S=1026 FIN
  192.168.8.112   192.168.8.111  TCP D=1026 S=1720
  192.168.8.112   192.168.8.111  TCP D=1027 S=1024 PUSH
  192.168.8.111   192.168.8.112  TCP D=1024 S=1027
  192.168.8.112   192.168.8.111  TCP D=1027 S=1024 FIN
  192.168.8.111   192.168.8.112  TCP D=1024 S=1027
```
Signalling-Channel wurde geschlossen:
```
  192.168.8.112   192.168.8.111  TCP D=1026 S=1720 FIN
  192.168.8.111   192.168.8.112  TCP D=1720 S=1026
  192.168.8.111   192.168.8.112  TCP D=1720 S=1026 RESET
```

Logdatei h323.log

Die erste Verbindung:
```
Sat Oct 28 22:17:53 Connection from 192.168.7.111 to 192.168.8.111 opened
    Connection 192.168.7.111 to 192.168.8.111 opens Signalling-Channel from
    port 1047 to 1720
    Connection 192.168.7.111 to 192.168.8.111 opens Control-Channel from
    port 1048 to 1034
    Connection 192.168.7.111 to 192.168.8.111 opens Logical-Channel with:
        media channel 3826 --> 3820 and media Control-Channel 3827 --> 3821
```
Die zweite Verbindung:
```
Sat Oct 28 22:18:05 Connection from 192.168.7.222 to 192.168.8.222 opened
    Connection 192.168.7.222 to 192.168.8.222 opens Signalling-Channel from
    port 1054 to 1720
    Connection 192.168.7.222 to 192.168.8.222 opens Control-Channel from
    port 1055 to 1041
    Connection 192.168.7.222 to 192.168.8.222 opens Logical-Channel with:
        media channel 2151 --> 2159 and media Control-Channel 2152 --> 2160
```
Beide Verbindungen werden geschlossen:
```
Sat Oct 28 22:18:18 Connection from 192.168.7.111 to 192.168.8.111 closed
Sat Oct 28 22:18:26 Connection from 192.168.7.222 to 192.168.8.222 closed
```
Die dritte Verbindung:
```
Sat Oct 28 22:18:29 Connection from 192.168.7.111 to 192.168.8.111 opened
    Connection 192.168.7.111 to 192.168.8.111 opens Signalling-Channel from
    port 1049 to 1720
    Connection 192.168.7.111 to 192.168.8.111 opens Control-Channel from
    port 1050 to 1035
    Connection 192.168.7.111 to 192.168.8.111 opens Logical-Channel with:
        media channel 3830 --> 3824 and media Control-Channel 3831 --> 3825
```
Die dritte Verbindung wird wieder geschlossen:
```
Sat Oct 28 22:18:41 Connection from 192.168.7.111 to 192.168.8.111 closed
```

Fazit

Die Verbindung kommt durch den Firewall hindurch zustande. Alle andern Ports bleiben gesperrt, so dass z.B. ein Ping nicht mehr durch den Firewall geht. Ausserhalb des Firewalls sind keine internen Adressen zu empfangen. Also funktioniert auch die NAT.
Leider konnten wir mit Netmeeting keine Verbindung aufbauen. Vorerst werden also nur die Siemens IP-Phones unterstützt.

11 Installations- und Bedienungsanleitung

11.1 Konfiguration der IP-Phones

Sämtliche Einstellungen werden im Administrations-Menu durchgeführt.
Öffnen des Administrations-Menu:

1. Mit den Pfeiltasten Service auswählen
2. OK drücken
3. Ziffer ‚6' drücken
4. Administrator Passwort eingeben (Default = 123456)
5. OK drücken

Folgende Einstellungen müssen in diesem Menu vorgenommen werden:

- 03: DHCP = Off
- 05: HiNet = Off
- 07: Terminal IP addr. = IP-Adresse
- 08: Terminal mask = Gemäss Netzwerkplan
- 13: IP routing = Gemäss Netzwerkplan

Alle übrigen Einstellungen können nach Bedarf gesetzt werden. Details dazu befinden sich auf der CDROM die dem IP-Phone beiliegt.
Die folgenden Einstellungen sind zwar nicht nötig, aber sie vereinfachen die Benutzung. Sie erfolgen im Menu Service -> Configuration:

- 10 : Dialing Mode = IP
- 11 : Incoming Call Display = IP

11.2 Installation

Die folgenden Schritte sind auf dem Firewall-Rechner durchzuführen.

Linux-Kernel

Der bei SuSE-Linux 7.0 mitgelieferte Standard-Kernel trägt erst Versionsnummer 2.2 und auch der optional installierbare „Hacker"-Kernel 2.4.0-test1 ist nicht auf dem letzten Stand. Deswegen muss als erstes ein neuerer Kernel-Source-Tree installiert werden. Damit kann ein aktueller Kernel generiert (kompiliert) werden.
Zum Erzeugen eines neuen Kernels stehen make-Befehle zur Verfügung. Somit wird der Quellcode mit dem C-Compiler übersetzt. Die Kernelquellen befinden sich üblicherweise im Verzeichnis /usr/src/linux. Vor der Kompilierung wird der Kernel konfiguriert. Dazu stehen drei verschiedene Methoden zur Verfügung:

- `make config` Konfiguration über Kommandozeile
- `make menuconfig` Durch Menü im Textmodus
- `make xconfig` Durch Menü unter dem X Window System

Die Konfigurationsscripts enthalten zahlreiche Fragen, die im Normalfall mit ‚y'(Ja) oder ‚n' (Nein) beantwortet werden können. Viele Treiber bieten ausserdem die Option ‚m' an, dies steht für Modul. Dadurch wird der Treiber zwar kompiliert, aber nicht direkt in den generierten Kernel eingebunden, sondern als ladbares Modul bereitgestellt.
Die folgenden Optionen sind für eine tadellose Funktion unserer Firewall-Erweiterung zu aktivieren:

Code maturity level options --->

- CONFIG_EXPERIMENTAL=y

Loadable module support --->

- CONFIG_MODULES=y

General setup --->
- CONFIG_NET=y

Networking options --->
- CONFIG_NETLINK=y
- CONFIG_NETLINK_DEV=m
- CONFIG_NETFILTER=y

IP: Netfilter Configuration --->
- CONFIG_IP_NF_QUEUE=m
- CONFIG_IP_NF_IPTABLES=m
- CONFIG_IP_NF_FILTER=m

- CONFIG_NET_FASTROUTE=n

Eine detaillierte Beschreibung, wie man den Linux-Kernel kompiliert, findet sich in [8].

IP-Forwarding einschalten

Als root muss folgender Befehl aufgeführt werden:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

Nun arbeitet der Linux-Kernel als Router, jedenfalls solange nicht neu gebootet oder das Forwarding wieder abgeschaltet wird.
Der Befehl zum Abschalten lautet wie folgt:

```
# echo "0" > /proc/sys/net/ipv4/ip_forward
```

In SuSE-Linux kann IP-Forwarding auch permanent mittels Konfigurationsdatei eingeschaltet werden. Dazu muss in der Datei `/etc/rc.config` folgende Eintragung gemacht werden:

```
# runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"
```

Kopieren der Binaries

Kopieren der Dateien `h323mod` und `iptable_h323nat.o` in ein beliebiges Verzeichnis.
cp /cdrom/bin/* /usr/local/sbin

Module laden

Folgende Module müssen von Hand geladen werden:

```
modprobe ip_tables
modprobe netlik_dev
modprobe ip_queue
```

Und aus dem Programmverzeichnis:

```
insmod iptable_h323nat.o
```

11.3 Bedienung

Wenn alle oben genannten Schritte ausgeführt wurden, kann das Modul gestartet werden.

Aufruf von der Kommandozeile

```
h323mod [-a IP-Address] [-h] [-i InterfaceName] [-v]
```

Options:

`-h`	Gibt eine Beschreibung der Optionen aus
`-a IP-Address`	IP-Address ist die Adresse der äusseren Firewall-Netzwerkkarte
`-i InterfaceName`	InterfaceName ist die Bezeichnung der äusseren Firewall-Netzwerkkarte (z.B. eth1)
`-v`	Verbose. Alle H.323 Meldungen werden in Klartext ausgegeben

Logfile

Im Logfile werden sämtliche Verbindungen mit allen verwendeten Ports protokolliert. Natürlich wird auch notiert, wann die Verbindung stattfindet. In einem Beispiel sind die verschiedenen Einträge am Besten zu sehen. Das Logfile heisst `h323.log` und ist immer im selben Directory wie `h323mod`.

Öffnen einer Verbindung:

```
Sat Oct 28 22:18:29 Connection from 192.168.7.111 to 192.168.8.111 opened
```

Ports des Signalling-Channels:

```
    Connection 192.168.7.111 to 192.168.8.111 opens Signalling-Channel from
    port 1049 to 1720
```

Ports des Control-Channels:

```
    Connection 192.168.7.111 to 192.168.8.111 opens Control-Channel from
    port 1050 to 1035
```

Ports des Logical-Channels:

```
    Connection 192.168.7.111 to 192.168.8.111 opens Logical-Channel with:
       media channel 3830 --> 3824 and media Control-Channel 3831 --> 3825
```

Schliessen der Verbindung:

```
Sat Oct 28 22:18:41 Connection from 192.168.7.111 to 192.168.8.111 closed
```

12 Schlusswort

12.1 Rückblick

Für diese Arbeit war ein sehr grosses Grundlagenwissen im Bereich Linux-Firewall und H.323 nötig. Bereits die Analyse des H.323 Stacks war sehr aufwändig, insbesondere da die Dokumentation bei Open Source Software meist recht knapp ist.
Als wir dann zur Analyse der Quelltexte übergingen, waren wir erst einmal von den mehreren tausend Zeilen Quelltext überwältigt. Und Kommentar haben wir meist vergeblich gesucht.
Trotzdem konnten wir die Arbeit erfolgreich beenden und zeigen, dass dynamisches Firewalling möglich ist.

12.2 Einschränkungen der vorliegenden Software

- Zur Zeit werden nur die Siemens IP-Phones unterstützt. NetMeeting funktioniert nicht und andere Clients wurden nicht getestet.
- Wenn auf dem Firewall normales NAT mit dem dazugehörigen Connection Tracking stattfindet, werden unsere Pakete von diesem verworfen.
- Das Beenden des Moduls funktioniert noch nicht sauber. Es bleiben Rules im Firewall zurück.
- T.120 Datenverbindugen werden nicht unterstützt.

12.3 Ausblick

Es gibt unzählige Weiterentwicklungsmöglichkeiten. Einige gehen aus diesem Dokument bereits hervor, andere sind uns während der Arbeit eingefallen. Ein kleine Liste der Möglichkeiten:

- Behebung der obengenannten Einschränkungen
- Die Unterstützung von weiteren Endgeräten und H.323 Applikationen
- NAT Support von ICMP-Nachrichten

12.4 Dank

Wir bedanken uns bei allen Personen, die uns bei unserer Arbeit unterstützt haben. Besonderen Dank verdient Herr Steffen, der uns immer mit Rat und Tat zur Seite stand. Dank geht auch an Roger Zuber, der uns fleissig mit dringend benötigter Hardware belieferte. Im weiteren danken wir den Linux Kernelhackern, allen voran Harald Welte, die auf der Mailingliste auf unsere Sorgen eingingen.
Und last but not least danken Tux für sein aufmunterndes Grinsen.

30. Okt. 2000 M. Stolz R. Buser

13 Anhänge

13.1 Glossar

ASN.1 (Abstract Syntax Notation 1)
Standardisierte Objektdefinitionssprache, die im SNMP und H.323 benutzt wird. Wie fast alles an OSI ist sie gross, komplex und nicht besonders effizient.

10Base-T
Bezeichnung einer neueren Technologie zur Verkabelung von Ethernet-LANs mit 10 Mbps. Das Signal wird über Leiterpaare geführt. Verdrillte Kabelpaare gibt es mit zwei, vier, sechs oder mehr Leitern pro Kabel.

Bridge
Ein Speichervermittlungsrelais, mit dem LAN-Segmente verbunden werden. Eine Bridge speichert LAN-Rahmen und leitet sie weiter, hat aber keine Kenntnis über den Inhalt dieser Rahmen.

Chain
Regelkette, in welcher Rules eingetragen werden. Diese Rules werden nacheinander abgerarbeitet, bis eine Übereinstimmung gefunden wird.

Datengramm (Datagram)
In Zusammenhang mit Rechnernetzen wird damit eine Dateneinheit bezeichnet, die ohne vorherigen Verbindungsaufbau übertragen werden kann. Datengramme beruhen auf dem verbindungslosen Prinzip, im Gegensatz zum verbindungsorientierten Ansatz. Jedes Datengramm muss die volle Adresse des Empfängersystems enthalten. IP und CLNP sind z.B. Protokoll, die Datengramme austauschen.

Ethernet
Ein LAN-Technologie auf der Grundlage der IEEE-Norm 802.2 bzw. 802.3. Diese Technik mit gemeinsamer Mediumnutzung wurde anfänglich in 10 Mbps mit Koaxialkabeln oder verdrillten Kabelpaaren angewandt. Die an Ethernet-Netze angeschlossenen Stationen verwenden die CSMA-CD-Methode für die gemeinsame Nutzung eines physischen Übertragunsmediums. Eine schnelle Version läuft mit 100 Mbps (Fast-Ethernet).

Firewall
Eine Firewall ist eine moderne Ausführung der mittelalterlichen Art, riesige Wälle und Mauern rund um Burgen und Ortschaften zu bauen. Jeder, der den Wall passieren wollte, wurde inspiziert. Bei Netzen ist die gleich Vorgehensweise möglich. Ein Unternehmen kann viele LANs zusammenschliessen, jedoch fliesst der gesamte Verkehr nach aussen und nach innen durch einen elektronischen Wall (Firewall).

Gateway
Ein Gateway (dt. Übergang) bezeichnet den Übergang in ein Datennetz. Das öffentliche Telefonnetz und das Internet oder ein Intranet können durch ein Gateway verbunden sein. Es stellt die Verbindung zwischen zwei Kommunikationsnetzen dar und ermöglicht es zum Beispiel, dass die Verbindung zwischen zwei normalen Telefonanschlüssen über Internet erfolgen kann.

H.323
Der H.323-Standard legt technische Details über Datenpakete und von Empfangs- und Sendeeinrichtungen fest, die für Multimedia-Verbindungen über LANs verwendet werden.

Internet
Bezeichnung für eine Ansammlung von Netzen (Netzverbund), die den gesamten Globus überspannen und unter dem Internet-Protokoll (IP) laufen.

Intranet
bezeichnet jedes private Computernetzwerk, das mit der gleichen Software funktioniert und die gleichen Standards zur Übertragung benutzt, wie das Internet. Zu diesen Standards gehört zum Beispiel TCP/IP.

IP-Adresse
Eine IP-Addresse bezeichnet eine aus 32 Bit zusammengesetzte Zahl, über die jeder Computer (Host-Rechner) im Internet, der über TCP/IP verbunden ist, eindeutig gefunden werden kann. IP-Adressen werden normalerweise als eine Folge von Zahlen angegeben, die durch Punkte unterbrochen ist. Beispielsweise: 128.127.50.224. Jeder Host hat seine eigene feststehende IP-Adresse.

IP-Table
Eine Verwaltungstabelle, welche Chains beinhaltet. Die IP-Tables werden nacheinander entsprechend ihrer Priorität durchlaufen.

IP-Phone
Ein Telefonapparat, welcher direkt an einem IP-Netz angeschlossen wird und das H.323-Protokoll beherrscht, um Telefongespräche übers Netz zu führen.

IP Telefonie
Technologie, die es ermöglicht Telefonanrufe übers Internet oder andere IP-Netze zu führen. Als Endgerate eignen sich der PC, IP-Phones oder via Gateways normale Telefonapparate.

ISP
ist das Kürzel für Internet Service Provider. Es bezeichnet Firmen, welche die Verbindung zwischen einem Internet-Zugang und einem Benutzer ermöglichen. ISPs bieten eine oder mehrere Möglichkeiten für Verbindungen ins Internet an. Dazu gehören: Einwählverbindungen über Modem, über ISDN oder E-1-Verindungen.

Kernelspace
Einer der grundlegenden Aspekte von Linux ist es, dem Kernel einen eigenen Speicherbereich zuzuordnen, den man auch *Kernelspace* nennt. Der Kernel wiederum verwaltet den Rest des Arbeitspeichers für die Anwendungsprozesse. Dieser Bereich wird als *Userspace* bezeichnet.

LAN (Local Area Network)
Lokales Netz, das innerhalb eines Gebäudes oder eines Firmengeländes installiert wird und in dem private Übertragungsmedien und Vermittlungsanlagen benutzt werden. Zu den LAN-Technologien zählen Ethernet, Token-Ring, FDDI und neuerdings auch LATM. Grössere Entfernungen werden von MANs und WANs abgedeckt.

Modem (Modulator-Demodulator)
Breitbandübertragungsgerät im Telekommunikationsbereich, das einen Träger mit einem Vorwärtssignal moduliert und einen empfangenen Träger demoduliert, um das Signal heraus-zuziehen.

Multimedia
Fachbereich, der sich mit der rechnergesteuerten Integration von Text, Grafik, Stand- und Laufbild sowie Ton und anderen Medien befasst. Alle Informationsarten können gemischt dargestellt, gespeichert, übertragen und digital verarbeitet werden.

Protokollstapel
Eine Aufstellung von Protokollen, die ein bestimmtes System nutzen kann.

Router
Bezeichnung für einen Knoten oder Vermittler in einem verbindungslosen paketvermittelten Netz, z.B. einem IP-Netz. Router dienen als Verbindungspunkte zwischen LAN-Segmenten oder als Gateway zwischen LANs und WANs.

RTP
Kürzel für "Real-time Transport Protocol". Es erfüllt Funktionen bei der Echtzeit-Datenübertragung, und kann bei der Übertragung von Audio-, Video-, oder Simulationsdaten eingesetzt werden.
Rule
Eine (Filter-)Regel in einer Chain von IP-Tables. Eine Regel gibt Kriterien an, mit welchen Pakete übereinstimmen müssen, damit eine Aktion ausgelöst wird
SMTP (Simple Mail Transfer Protocol)
SMTP ist ein einfaches ASCII-Protokoll. Nach dem Aufbau der TCP-Verbindung (im Internet) zu Port 25 wartet die sendende Maschine, die als Client fungiert, bis die empfangende Maschine, die als Server auftritt, zuerst mit der Kommunikation beginnt. Der Server beginnt durch Aussenden einer Textzeile, durch die er sich identifiziert und mitteilt, ob er E-Mail annehmen kann. Andernfalls löst der Client die Verbindung auf und versucht es später noch einmal.
SNMP (Simple Network Management Protocol)
Systematische Methode der Überwachung und Verwaltung eines Rechnernetzes. Dieses Rahmenwerk und das Protokoll wurden inzwischen häufig in kommerziellen Produkten implementiert und hat sich zum De-facto-Standard für das Netzmanagement entwickelt.
Tunneling
Mechanismus, durch den ein Paket wie folgt an einen mobilen Benutzer gesendet wird: Das Paket wird an das Heimat-LAN des Benutzers weitergeleitet, weil das aus der Adresse hervorgeht, und vom Heimatagenten des Heimat-LANs des betreffenden Benutzers aufgenommen. Der Heimatagent schlägt die neue (vorübergehende) Adresse des mobilen Benutzers nach, dann kapselt er das Paket in das Nutzdatenfeld eines äusseren Pakets und sendet es an den Fremdagenten. Bei Erhalt des gekapselten Pakets nimmt der Fremdagent das Originalpaket aus dem Nutzdatenfeld und sendet es als Rahmen der Sicherungsschicht an den mobilen Benutzer.
Userspace
s. Kernelspace
Verbindungslos
Ein Netzzugriffsmodus, bei dem keine vorherige Autorisation in Form einer expliziten Verbindung erforderlich ist, sonder Informationen unmittelbar übertragen werden können. Das bedeutet, dass Rahmen oder Pakete ohne Verbindungsaufbau übertragen werden können. Das ist der Fall bei den meisten heutigen LANs (Ethernet, Token-Ring, FDDI) und IP- oder SDMS-WANs. Verbindungslos charakterisiert auch die Regeln für Ende-zu-Ende-Kommunikationssysteme, d.h. die Transportschicht. UDP ist ein Beispiel eines verbindungslosen Transportprotokolls. Das Gegenteil ist verbindungsorientiert.
Verbindungsorientiert
Ein Netzzugriffsmodus, der bedeutet, dass Rahmen oder Pakete nur über Verbindungen ausgetauscht werden können, die dem Netz bekannt sind. Das der Fall bei leitungsvermittelten Netzen, X.25, ATM und Frame-Relay. Verbindungsorientiert charakterisiert auch die Regeln für zwei kommunizierende Endsysteme auf der Transportschicht. TCP ist ein Beispiel eines verbindungsorientierten Transportprotokolls, bei dem ein Endsystem nicht ohne vorherige Vereinbarung mit dem Empfänger in Form einer logischen Verbindung Pakete übertragen kann. Das Gegenteil ist verbindungslos.

13.2 Literaturverzeichnis

Grundlagen

[1] The Netfilter Project: Packet Mangling for Linux 2.3+
http://netfilter.filewatcher.org/

[2] Kernissage: Neuer Anwender-Kernel 2.4 / Neuorganisation im Netzwerkcode
iX 8/2000 Seiten 98-100

[3] A Primer on the H.323 Series Standard
http://www.databeam.com/standards/index.html

[4] H.323 and Firewalls by Intel
http://support.intel.com/support/videophone/trial21/h323_wpr.htm

[5] Web ProForum Tutorial: H.323
http://www.iec.org/tutorials/h323/

[6] The Open H.323 Project
http://www.openh323.org/

[7] Portable Windows Libaray
http://www.equival.net/pwlib/

HOWTOs

[8] The Linux Kernel HOWTO
http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html

[9] Linux 2.4 Packet Filtering HOWTO
http://netfilter.kernelnotes.org/unreliable-guides/packet-filtering-HOWTO.html

[10] Linux Networking-concepts HOWTO
http://netfilter.kernelnotes.org/unreliable-guides/networking-concepts-HOWTO.html

[11] Linux 2.4 NAT HOWTO
http://netfilter.kernelnotes.org/unreliable-guides/NAT-HOWTO.html

[12] Linux netfilter Hacking HOWTO
http://netfilter.kernelnotes.org/unreliable-guides/netfilter-hacking-HOWTO.html

[13] Unreliable Guide To Hacking The Linux Kernel
http://netfilter.kernelnotes.org/unreliable-guides/kernel-hacking/lk-hacking-guide.html

[14] Linux 2.4 Advanced Routing HOWTO
http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html

ITU-T Recommendations

[15] ITU-T Recommendation H.323:
Packet-Based Multimedia Communications Systems

[16] ITU-T Recommendation H.225.0:
Call Signalling Protocols and Media stream Packetization

[17] ITU-T Recommendation H.245:
Control Protocol for Multimedia Communication

[18] ITU-T Recommendation X.680:
Abstract Syntax Notation One (ASN.1). Specification of Basic Notation

[19] ITU-T Recommendaton X.691:
ASN.1 encoding rules – Specification of Packed Encoding Rules (PER)

RFCs (Request for Comment)

[20] A site where you can find all RFCs
http://www.rfc-editor.org/rfc.html
[21] RFC 768 (UDP)
User Datagram Protocol
[22] RFC 791 (IP)
Internet Protocol
[23] RFC 792 (ICMP)
Internet Control Message Protocol
[24] RFC 793 (TCP)
Transmission Control Protocol
[25] RFC 2126 (ITOT)
ISO Transport Service on top of the TCP

Tools

[26] CVS - Concurrent Versions System
http://www.gnu.org/software/cvs/
[27] Quelltext-Management nicht nur für Software-Entwickler (Teil 1)
http://www.linux-magazin.de//ausgabe/1998/10/CVS/cvs.html
[28] Squid Web Proxy Cache
http://www.squid-cache.org/

Manuals

[29] HiNet LP 5100 IP Telephone: Operation and Administration Manual
Product Documentation on CD, Order No. C39453-Z5-C68

Weitere Literatur

[30] SPF: Stateful Packet Filtering.
ftp://ftp.interlinx.bc.ca/pub/spf
[31] The Sinus Firewall Page.
http://www.ifi.unizh.ch/ikm/SINUS/firewall/

13.3 Sourcecode