Institut für Informatik
der Ludwig-Maximilians-Universität München

Diplomarbeit

Das Internet-Protokoll Version 6 – Migrationsverfahren und Managementanforderungen

Robert Hoffmann

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Stephen Heilbronner, Dr. Bernhard Neumair
Abgabedatum: 20. August 1997

INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Diplomarbeit

Das Internet-Protokoll Version 6 – Migrationsverfahren und Managementanforderungen

Robert Hoffmann

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Stephen Heilbronner, Dr. Bernhard Neumair
Abgabedatum:	20. August 1997

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. August 1997

....

(Unterschrift des Kandidaten)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Einführung

Das exponentielle Wachstum des Internets, beschleunigt durch den Boom des World Wide Web, und die Erschließung neuer Anwendungsbereiche für Rechnernetze auf der Grundlage der Internet-Protokollfamilie haben dazu geführt, daß diese Basis den neuen Anforderungen nicht mehr gerecht wird.

In dem Ende der siebziger Jahre entwickelten Internet-Protokoll Version 4 (IPv4) wurde die Länge der Adressen auf 32 Bits festgelegt. Dies war ausreichend, um ein paar Tausend Rechner des amerikanischen Verteidigungsministeriums und später ein paar Millionen Rechner von auf der ganzen Welt verteilten wissenschaftlichen Einrichtungen miteinander zu verbinden.

Seit aber zunehmend Privatleute hauptsächlich mit den Diensten E-Mail und World Wide Web das Internet in Anspruch nehmen und kommerzielle Unternehmen große Gewinne wittern, wird der Adreßraum eng. Dazu kommt die Umstellung vieler Unternehmensnetze auf die TCP/IP-Technik („Intranet") und deren teilweiser Verbindung mit dem globalen Internet.

Routing

Würden nur die mit 32 Bits darstellbaren Zahlen – immerhin mehr als 4 Milliarden – bei der eindeutigen Zuordnung von Adressen zu Rechnern eine Rolle spielen, gäbe es damit auch in einigen Jahren noch keine Schwierigkeiten. Das Problem liegt in der verschwenderischen Vergabe ganzer Adreßblöcke an einzelne Netze zur Vereinfachung der Wegewahl in den Vermittlungsknoten, den Routern. Denn wäre für jeden einzelnen im Internet erreichbaren Rechner ein eigener Eintrag in den Tabellen nötig, anhand derer die Router entscheiden, welchen Weg ein Datenpaket nehmen soll, um sein Ziel zu erreichen, würde das Netz nicht funktionieren. Die Routing-Tabellen könnten nie in allen Routern auf einem aktuellen Stand gehalten werden und die Leistung der Router wäre nicht akzeptabel.

Statt Pakete an einzelne Rechner zu vermitteln, arbeiten Router vorzugsweise mit Adressen ganzer Netze. Die Adresse eines Rechners besteht, je nach Netzwerkklasse A, B oder C, aus einer ein, zwei oder drei Bytes langen Netzadresse und einer drei, zwei oder ein Byte langen Hostadresse. Es genügt für Router, eine Tabelle aus Netzadressen zu führen, um ein Paket zum gewünschten Zielnetz weiterzuleiten. Der Standort des Zielrechners kann dann mit Methoden der lokalen Netztechnik im Zielnetz ermittelt werden.

Besitzt ein größeres Unternehmen 10000 Rechner, die es mit dem Internet verbinden will, benötigt es 10000 Hostadressen. Diese werden z.B. von einem Klasse-B-Netz mit 65534 ($2^{16} - 2$) Hostadressen oder von 40 Klasse-C-Netzen mit jeweils 254 ($2^8 - 2$) Hostadressen je Netzadresse geboten. Hier zeigt sich die Verschwendung: Erhält das Unternehmen ein Klasse-B-Netz, bleiben 55534 mögliche Adressen ungenutzt. Wählt es aber 40 Klasse-C-Netze, werden zwar nicht viele Adressen verschwendet, aber die Routing-Tabellen wachsen um 40 Einträge und der Administrationsaufwand ist beträchtlich höher als bei einem Netz.

Eine Zwischenlösung für dieses Problem wurde 1994 und 1995 mit dem Classless Inter-Domain Routing (CIDR) eingeführt. Mit CIDR treffen Router ihre Entscheidungen nicht mehr anhand fester Netzwerkpräfixe, sondern anhand variabler Präfixlängen der einzelnen Routen. Dies erlaubt eine effizientere Vergabe von Adressen und das Zusammenfassen vieler Routing-Tabelleneinträge zu einem einzigen.

Weitere Probleme

Jedoch wird auch das auf Dauer nicht ausreichen, um der Adressenknappheit und den wachsenden Routing-Tabellen wirkungsvoll zu begegnen. Außerdem hat das alte Internet-Protokoll noch eine Reihe anderer Schwächen, die nicht oder nur unbefriedigend auf anderer Ebene umgangen werden können:

- Der Konfigurationsaufwand einzelner Hosts ist hoch. Es fehlen standardmäßig vorhandene Autokonfigurationsmechanismen.

- Es gibt es keine Möglichkeit, auszuwählen, welcher Dienst über welchen Dienstanbieter abgewickelt werden soll.

- Authentifizierung und Verschlüsselung der Daten müssen von Applikationen vorgenommen werden, die nicht überall vorhanden sind.

- Ebenfalls nicht überall vorhanden ist die Unterstützung von mobilem Computing und Multicastadressierung.

- Ströme zusammengehörender Pakete können auf unterschiedlichen Wegen in falscher Reihenfolge zu ihrem Ziel gelangen. Eine konstant verfügbare Mindestbandbreite kann nicht garantiert werden. Dadurch können Anwendungen, die einen konstanten Datenstrom voraussetzen, wie Echtzeit-Audio- und -Videoübertragungen nicht korrekt arbeiten.

- Die Leistungsfähigkeit von Netzwerken könnte gesteigert werden, indem zeitaufwendige, überflüssige Berechnungen von Prüfsummen oder die Fragmentierung von zu großen Paketen in den Routern vermieden werden.

All diese und weitere Probleme versucht die Entwicklung des Internet-Protokolls Version 6 (IPv6) zu lösen. Welche Features das neue Internet-Protokoll bietet, wie bestehende IPv4-Netze zu IPv6-Netzen umgebaut werden können und was das für das Netzmanagement bedeutet, darum geht es in dieser Diplomarbeit.

1.2 Vorgehensweise

Im zweiten Kapitel wird zunächst ein Überblick über das neue Internet-Protokoll gegeben. Die Header der Pakete, die Adreßarchitektur und die Autokonfigurationsmechanismen werden vorgestellt.

Das dritte Kapitel behandelt mögliche Migrationsverfahren zu IPv6, stellt die dazu verwendeten Mechanismen vor und entwickelt einige typische Migrationsszenarios. Ferner werden der Einfluß auf die Anwendungsentwicklung untersucht und dazu Beispiele zur Socket-Programmierung entwickelt und nach IPv6 portiert.

Im vierten Kapitel wird beschrieben, wie diese Migrationsverfahren teilweise in die Praxis umgesetzt wurden, indem ein am Institut installiertes IPv6-Testnetz an das globale IPv6-Testnetz *6bone* angeschlossen wurde.

Dabei gewonnene Erfahrungen fließen in die im fünften Kapitel untersuchten Managementanforderungen der Migration von IPv4 hin zu IPv6 und während des Betriebs von IPv6-Netzen ein.

Im sechsten Kapitel werden einige für bestimmte Teilbereiche dieser Managementanforderungen wichtige Standards und ihre Neuentwicklungen für IPv6 besprochen.

Nach einer Zusammenfassung der untersuchten Migrationsverfahren und Managementanforderungen und einem Ausblick auf mögliche weiterführende Projekte im siebten und achten Kapitel beschreibt Anhang A ausführlich die Installation des IPv6-Testnetzes am Institut.

In Anhang B sind die Quelltexte der in Kapitel 3 untersuchten Beispiele zur Anwendungsentwicklung abgedruckt.

Anhang C enthält ein Verzeichnis der verwendeten Abkürzungen.

Kapitel 2

Das neue Internet-Protokoll

Das Hauptproblem des Internet-Protokolls Version 4 ist, wie in der Einleitung bereits erwähnt, der zu kleine Adreßraum, verursacht durch die mit 32 Bits zu knapp bemessene Adreßlänge, verbunden mit einer durch Netzklassen bedingten verschwenderischen Vergabe der Adressen. Diesem Problem begegnet das neue Internet-Protokoll Version 6 mit der Vervierfachung der Adreßlänge auf 128 Bits und einer klassenlosen, hierarchisch strukturierten Adreßarchitektur.

Weitere Unzulänglichkeiten des alten Protokolls wie fehlende Mechanismen zur Autokonfiguration, Echtzeitübertragung, Authentifizierung und Verschlüsselung sowie überflüssiger Rechenaufwand beim Routing sollen durch die Neugestaltung des Paket-Headers und die Entwicklung neuer Zusatzprotokolle beseitigt werden.

Dieses Kapitel gibt einen Überblick über das neue Internet-Protokoll. Zunächst wird der Header der IPv6-Pakete beschrieben und mit dem Header von IPv4-Paketen verglichen. Die folgenden Abschnitte behandeln Erweiterungsheader, Adreßarchitektur, Neighbor Discovery und Autokonfiguration.

2.1 Der IPv6-Header

Der IPv6-Header und seine Erweiterungsheader sind in [Deering 96] spezifiziert. Der Haupt-IPv6-Header besteht aus insgesamt acht Feldern, die zusammen 40 Bytes umfassen (vgl. Abbildung 2.1). Die ersten 64 Bits enthalten die Felder

Version: 4 Bits; Versionsnummer des Internet-Protokolls, hier: 6

Priority: 4 Bits; Ermöglicht es einer Quelle, die gewünschte Priorität des Pakets relativ zu anderen Paketen aus derselben Quelle zu bestimmen. Werte zischen 0 und 7 werden für Datenverkehr mit Flußkontrolle verwendet, Werte zwischen 8 und 15 für Echtzeit-Übertragungen. In dem Entwurf [Hinden 97a] wird dieses Feld in **Class** umbenannt. An seiner Bedeutung ändert sich nichts.

Flow Label: 24 Bits; Die zufällig gewählte Identifikationsnummer soll Routern erlauben, alle zu einer Ende-zu-Ende-Verbindung gehörenden Pakete anhand ihres Flow Labels weiterzuleiten, ohne den Rest des Headers auswerten zu müssen.

Payload Length: 16 Bits; Länge der Nutzlast, i.e. der auf den IPv6-Header folgende Rest des Pakets inklusive Erweiterungsheader, TCP-Header, Datenbereich, etc.

Next Header: 8 Bits; Typ des direkt auf den IPv6-Header folgenden Headers

Hop Limit: 8 Bits; wird von jedem Knoten, der das Paket weiterleitet, um 1 dekrementiert. Das Paket wird verworfen, wenn das Hop Limit Null wird.

Daran schließen sich die zwei jeweils 128 Bits langen Felder für Quell- und Zieladresse an.

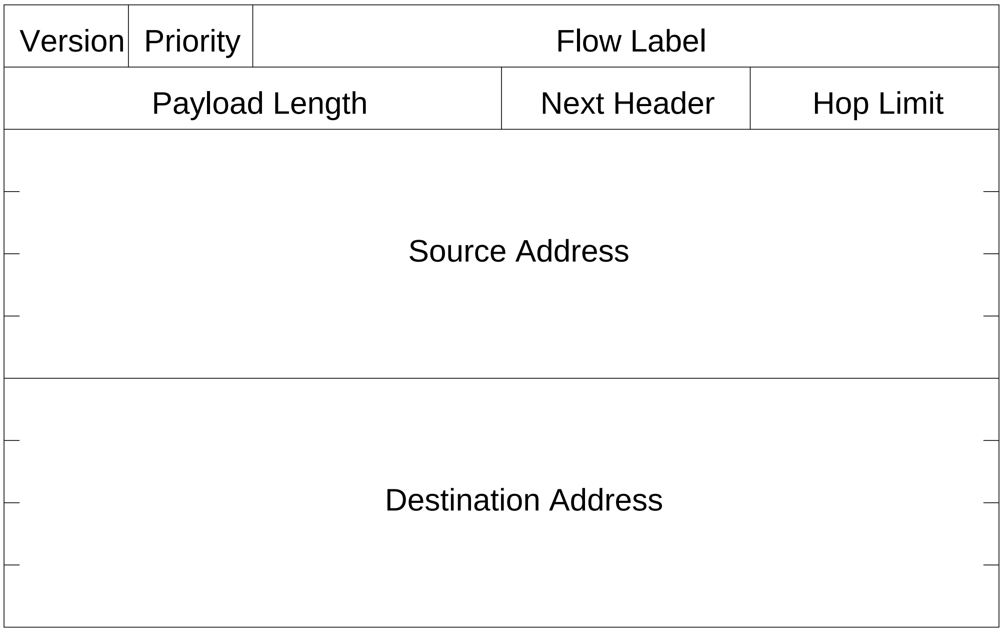

Abbildung 2.1: Der IPv6-Header

Zum Vergleich ist in Abbildung 2.2 der IPv4-Header dargestellt. Man erkennt leicht die Vereinfachung des neuen gegenüber dem alten Header. Obwohl eine IPv6-Adresse viermal so lang ist wie eine IPv4-Adresse, wird der neue Header nur knapp doppelt so lang wie der alte. Der IPv4-Header enthält insgesamt 13 Felder, davon 2 Adreßfelder und ein Optionsfeld von variabler Länge.

Die sechs Felder Header Length, Type of Service, Identification, Flags, Fragment Offset und Header Checksum wurden gestrichen. Die drei Felder Total Length, Protocol Type und Time to Live wurden umbenannt und zum Teil neu definiert.

Der IPv6-Header enthält im Gegensatz zum IPv4-Header keine optionalen Elemente. Optionen können stattdessen in separaten Erweiterungsheadern (Extension Headers), die an den IPv6-Header angehängt werden, übermittelt werden (siehe Abschnitt 2.2). Somit beträgt die Länge des IPv6-Headers immer 40 Bytes und eine Längenangabe wie das Header-Length-Feld bei IPv4 ist unnötig.

<table>
<tr><td>Version</td><td>IHL</td><td colspan="2">Type of Service</td><td colspan="4">Total Length</td></tr>
<tr><td colspan="4">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="4">Header Checksum</td></tr>
<tr><td colspan="8">Source Adress</td></tr>
<tr><td colspan="8">Destination Adress</td></tr>
<tr><td colspan="6">Options</td><td colspan="2">Padding</td></tr>
</table>

Abbildung 2.2: Der IPv4-Header

2.2 Die Erweiterungsheader

Um eine spezielle Behandlung von IPv6-Paketen veranlassen zu können, gibt es die Möglichkeit, an den Haupt-IPv6-Header sog. Erweiterungsheader anzuhängen. Gegenwärtig sind folgende sechs Erweiterungsheader definiert:

Routing Header: Ermöglicht die Angabe einer bestimmten Route, die das Paket auf dem Weg zu seinem Zielrechner nehmen soll.

Fragment Header: Im Gegensatz zu IPv4 fragmentieren IPv6-Router Pakete, die für die Strecke zum nächsten Router zu groß sind, nicht, sondern weisen sie zurück und melden dies dem Quellrechner mit einer ICMP-Fehlermeldung. Pakete können jedoch bereits vor dem Absenden fragmentiert werden, indem zwischen dem Haupt-IPv6-Header und der fragmentierten Nutzlast dieser Pakete ein Fragment Header eingefügt wird, der es dem Zielrechner ermöglicht, diese Pakete wieder richtig zusammenzusetzen.

Authentication Header: Gibt dem Empfänger eines Pakets die Möglichkeit, die Authentizität dieses Pakets festzustellen, d.h. festzustellen, ob die Quelladresse tatsächlich die angegebene ist und ob das Paket während der Übermittlung nicht verfälscht wurde.

Encrypted Security Payload: Garantiert, daß nur legitime Empfänger den Inhalt des Pakets lesen können.

Destination Options Header: Gibt neben der Definition neuer Erweiterungsheadertypen die Möglichkeit, IPv6 mit neuer Funktionalität auszustatten. Die Verwendung von Destination Options Headers hat den Vorteil, den knappen Vorrat an freien Header-Typ-Nummern nicht unnötig zu verkleinern.

Hop-by-hop Options Header: Dieser Header hat dasselbe Format wie der Destination Options Header, muß aber im Gegensatz zu diesem von jedem

Router am Weg interpretiert werden. Er wird z.B. dazu benutzt, die *Jumbo Payload Option* zu transportieren und den zu passierenden Routern damit mitzuteilen, daß die Paketlänge die Darstellungsmöglichkeiten des 16 Bit langen Payload-Length-Feldes des Haupt-IPv6-Headers übersteigt.

2.3 Adreßarchitektur

Die Adreßarchitektur des Internet-Protokolls Version 6 wird in [Hinden 96a] definiert.

2.3.1 Adreßtypen

IPv6-Adressen gehören zu einer der drei Typen:

Unicast bezeichnet Punkt-zu-Punkt-Adressen. Eine Unicast-Adresse identifiziert genau eine Schnittstelle aus in ihrem Gültigkeitsbereich. Ein Paket, das an eine Unicast-Adresse gesendet wird, wird normalerweise an der damit bezeichneten Schnittstelle zugestellt.

Multicast identifiziert eine Gruppe von Schnittstellen. Ein Paket, das an eine Multicast-Adresse gesendet wird, wird normalerweise an alle Mitglieder der Gruppe ausgeliefert.

Anycast identifiziert ebenfalls eine Gruppe von Schnittstellen. Im Gegensatz zu Multicast-Adressen werden Pakete nur an das Mitglied der Gruppe ausgeliefert, das gemäß der Entfernungsmessung der verwendeten *Routing-Protokolle* (siehe Abschnitt 3.2.2) am nächsten liegt.

Es gibt keine Broadcast-Adressen. Diese Funktion wird durch Multicast-Adressen ersetzt. „Die“ Broadcast-Adresse entspricht der *All-Nodes*-Multicast-Adresse.

2.3.2 Adressierungsmodell

IPv6-Adressen werden den Netzschnittstellen eines Knotens zugeordnet, nicht dem Knoten selbst. Jede Unicast-Adresse einer Schnittstelle eines Knotens identifiziert diesen Knoten eindeutig.

Eine IPv6-Unicast-Adresse bezieht sich immer auf eine einzelne Schnittstelle. Einer Schnittstelle können mehrere Adressen beliebigen Typs zugeordnet sein. Dieses Modell hat zwei Ausnahmen:

1. Eine einzelne Adresse kann mehreren physikalischen Schnittstellen zugeordnet werden, wenn diese der Netzwerkschicht als eine Schnittstelle präsentiert werden. Eine Anwendung dafür ist z.B. die Lastverteilung über mehrere Schnittstellen.

2. Router können Schnittstellen ohne IP-Adresse besitzen. Diese Schnittstellen können in IPv6-Headern nicht als Quelle oder Ziel angegeben werden.

2.3.3 Textuelle Repräsentation

Die aus 128 Bits bestehenden IPv6-Adressen werden als acht 16-Bit Integerzahlen, getrennt durch Doppelpunkte, geschrieben. Jede Integerzahl wird durch 4 hexadezimale Ziffern repräsentiert. Führende Nullen können weggelassen werden. Beispiel:

`123:4567:89AB:CDEF:123:4567:89AB:CDEF`

Ist eine Folge dieser Integerzahlen Null, so kann sie durch `::` abgekürzt werden. Eine solche Abkürzung darf nur einmal pro Adresse angewendet werden. Beispiel:

`123:4567:0:0:0:0:89AB:CDEF` kann durch
`123:4567::89AB:CDEF` abgekürzt werden.

In IPv6-Adressen, die aus IPv4-Adressen durch Voranstellen von 96 Nullbits gebildet werden, sogenannten IPv4-kompatiblen IPv6-Adressen (siehe Abschnitt 3.1.2), können die letzten 32 Bits in dezimaler Schreibweise belassen werden. Beispiel:

`::129.187.214.42`

2.3.4 Aufteilung des Adreßraums

Gegenwärtig sind nur etwa 30 Prozent des Adreßraums bestimmten Zwecken zugeordnet. Der Rest kann in kommenden Jahren für heute noch unbekannte Dienste verwendet werden.

Routertabellen bestehen aus Präfixen der Länge zwischen 1 und 128 Bits. Alle Knoten müssen bestimmte Spezialadressen und -präfixe besonders behandeln können. Dazu zählen Multicast-, Loopback-, Site-Local- und Link-Local-Adressen. Tabelle 2.1 zeigt den derzeitigen Stand der Adreßzuordnungen.

Spezielle Adressen

Die Adresse `0:0:0:0:0:0:0:0` (oder vereinfacht `::`) ist die *unspezifizierte Adresse*. Sie kann z.B. als Absenderadresse eines Knotens verwendet werden, dessen Adresse(n) noch nicht konfiguriert wurde(n).

An die *Loopback-Adresse* `0:0:0:0:0:0:0:1` (oder vereinfacht `::1`) kann ein Knoten an sich selbst adressierte Pakete senden. Sie kann weder als Quell- noch als Zieladresse von Paketen dienen, die den Knoten verlassen.

Site-Local-Adressen

Site-Local- (ortslokale) Adressen erfüllen etwa denselben Zweck wie die Adressen zur Bildung privater Netze des Internet-Protokolls Version 4 nach [Rekhter 94], nämlich innerhalb einer nicht an das globale Internet angeschlossenen Organisation eindeutige Adressen zu vergeben, ohne dafür global eindeutige Adressen verwenden zu müssen. Site-Local-Adressen sind nur innerhalb einer Organisation eindeutig und können nicht im globalen Internet geroutet werden.

Zuordnung	Präfix (binär)
reserviert	0000 0000
nicht zugeordnet	0000 0001
reserviert für NSAP	0000 001
reserviert für IPX	0000 010
nicht zugeordnet	0000 011
nicht zugeordnet	0000 1
nicht zugeordnet	0001
aggregierbare Unicast-Adressen	001
providerbasierte Unicast-Adressen	010
nicht zugeordnet	011
reserviert für geographiebasierte Unicast-Adressen	100
nicht zugeordnet	101
nicht zugeordnet	110
nicht zugeordnet	1110
nicht zugeordnet	1111 0
nicht zugeordnet	1111 10
nicht zugeordnet	1111 110
nicht zugeordnet	1111 1110 0
Link-Local-Adressen	1111 1110 10
Site-Local-Adressen	1111 1110 11
Multicast-Adressen	1111 1111

Tabelle 2.1: Aufteilung des IPv6-Adreßraums

Link-Local-Adressen

Rechner, die weder mit einer Site-Local- noch mit einer global gültigen Adresse konfiguriert wurden, können über *Link-Local-* (verbindungslokale) Adressen kommunizieren, jedoch nur dann, wenn sie am selben lokalen Netz angeschlossen sind. Link-Local-Adressen werden nicht geroutet.

Providerbasierte globale Unicast-Adressen

Während IPv4-Adressen in verschiedene Netzwerkklassen A, B, C, etc. unterteilt werden, gibt es bei IPv6-Adressen eine derartige statische Trennung von Netzwerk- und Hostanteil nicht. Die Wegewahl wird von Routern aufgrund von Adreßpräfixen getroffen, die beliebige Längen zwischen 1 und 128 Bits haben können.

Eines der größten Probleme des alten Internet-Protokolls ist der Umfang der Routing-Tabellen in großen Backbone Routern und den damit verbundenen Leistungseinbußen. Diesem Problem versucht man im neuen Internet-Protokoll durch eine wohldefinierte Adressierungshierarchie zu begegnen, die unter anderem folgenden Anforderungen genügt [Estrin 94]:

- Die Adressierung muß topologisch signifikante Adreßzuordnungen unterstützen.

- Die Adressierungsstruktur sollte so wenige Vorbedingungen wie möglich an die Anzahl der Hierarchieebenen stellen, die an verschiedenen Stellen der Hierarchie unterschiedlich sein kann. Die einzelnen Ebenen dürfen nicht statisch mit bestimmten Feldern der Adresse verbunden sein.

- Hierarchische Adreßzuordnung soll nicht hierarchisches Routing bedingen.

Bis zur Definition eines weiteren Formats, das im nächsten Abschnitt vorgestellt wird, wurde erwartet, daß die *providerbasierte* Adreßzuordnung breite Verwendung finden wird. Ihr Format nach [Rekhter 97a] ist in Tabelle 2.2 zu sehen.

3 Bits	5 Bits	n Bits	56-n Bits	64 Bits
010	Registry ID	Provider ID	Subscriber ID	Intra-Subscriber

Tabelle 2.2: Providerbasiertes Unicast-Adreßformat

Registry ID: Für die Adreßvergabe zuständige Registratur:

Regionale Registratur	Wert (binär)
Multiregional (IANA)	10000
RIPE NCC (Europa)	01000
INTERNIC (Nord Amerika)	11000
APNIC (Asien/ Pazifik)	00100

Provider ID: Die Registratur teilt den von ihr verwalteten Diensteanbietern Adreßblöcke und eindeutige IDs zu.

Subscriber ID: Die Struktur und Aufteilung dieser Nummern ist Sache der einzelnen Diensteanbieter. Er kann seine Kunden z.B. nach topologischen Kriterien in Gruppen einteilen.

Intra-Subscriber: Die letzten 64 Bits stehen dem einzelnen Endbenutzer zur Verfügung. Die Autoren schlagen die in Tabelle 2.3 dargestellte Aufteilung vor:

64 Bits	16 Bits	48 Bits
Subscriber Prefix	Subnet ID	Interface ID

Tabelle 2.3: Adreßaufteilung bei Endbenutzern

Damit kann der Endbenutzer die Möglichkeiten zur Autokonfiguration der Interface IDs durch linkspezifische Adressen, wie z.B. den 48 Bit IEEE-802 MAC-Adressen, nutzen. Reichen die durch 16 Bits möglichen 65535 Subnetze nicht aus, können an solche Großunternehmen eigene Provider IDs vergeben werden, um zusätzlichen lokalen Adreßraum zu gewinnen.

Aggregierbare globale Unicast-Adressen

Erste Erfahrungen mit dem *6bone*-Testnetz (siehe Kapitel 4) zeigten, daß das providerbasierte Adressierungsschema auf Dauer zu einem zu starken Anwachsen der Routing-Tabellen führen würde.

Es wurden mehrere Vorschläge für ein neues Adressierungsschema diskutiert, zu nennen sind [O'Dell 97] und eine Analyse dazu in [Zhang 97]. Zentrale Konzepte dieses Vorschlags sind u.a.:

- strenge Trennung von öffentlicher und privater Topologie
- strenge Trennung von Identität und Anschlußort eines Systems
- globale, site-spezifische und endsystem-spezifische Adreßelemente
- stark hierarchische Gruppenbildung in der Netztopologie

Der momentan letzte Beitrag zu der noch nicht abgeschlossenen Diskussion ist [Hinden 97d]. Hier wird das Format für *aggregierbare globale Unicast-Adressen* definiert, das zusätzlich zur *providerbasierten* eine *exchange-basierte* Aggregation unterstützt. Die Kombination beider soll effizientes Routing ermöglichen. Eine Beschreibung dieser „Exchanges“ genannten Organisationen, die Vermittlungsdienste zwischen herkömmlichen Providern anbieten und auch selbst als Provider gegenüber Endbenutzern auftreten, wurde im genannten Dokument angekündigt, ist zum gegenwärtigen Zeitpunkt aber noch nicht verfügbar. Tabelle 2.4 zeigt ihr Adreßformat.

3 Bits	13 Bits	32 Bits	16 Bits	64 Bits
FP	TLA	NLA*	SLA*	Interface ID

Tabelle 2.4: Format aggregierbarer globaler Unicast-Adressen

FP: *Format Prefix* `001`

TLA: Der *Top-Level Aggregator* ist die oberste Ebene in der Routinghierarchie. TLAs werden an Organisationen vergeben, die öffentliche Transit-Netztopologien unterhalten.

NLA*: Der/Die *Next-Level Aggregator(s)* werden von den TLAs zur Bildung einer Adressierungshierarchie und zur Identifikation von Sites verwendet.

SLA*: Der/Die *Site-Level Aggregator(s)* werden von individuellen Organisationen zum Aufbau einer lokalen Adressierungshierarchie und zur Identifikation von Subnetzen verwendet.

Interface ID: Die Interface IDs sind in diesem Format 64 Bits lang und müssen nach dem IEEE EUI-64 Format [IEEE 97] gebildet werden.

Die Regeln zur Vergabe der TLAs und NLAs sind ausführlich in [Hinden 97c] dargelegt.

2.3.5 Adreßauflösung

Das Domain Name System (DNS) wird dazu verwendet, Rechnernamen sowohl zu IPv6-Adressen als auch zu IPv4-Adressen aufzulösen. IPv4-Adressen werden weiterhin als „A“-Resource Records dargestellt. Für IPv6-Adressen wurde in [Thomson 96a] der neue „AAAA“-Resource Recordtyp eingeführt. Duale IPv6/IPv4-Knoten werden mit Einträgen beider Recordtypen im DNS geführt. Ihre Resolverbibliotheken müssen beide Recordtypen unterstützen.

Zur Adreßauflösung in umgekehrter Richtung, also um zu einer gegebenen IPv6-Adresse den Knotennamen zu erhalten, wurde am selben Ort die neue Domain `IP6.INT.` definiert. Eine IPv6-Adresse wird in dieser Domain von rechts nach links gelesen durch Nibbles (halbe Bytes) in hexadezimalen Ziffern repräsentiert.

2.4 Neighbor Discovery

[Narten 96] definiert das *Neighbor Discovery Protocol* für IPv6. Dieses Protokoll übernimmt u.a. die Aufgaben, die in IPv4 die Protokolle *Address Resolution Protocol*, *ICMP Router Discovery* und *ICMP Redirect* innehatten. Knoten verwenden das Protokoll, um

- die Link-Layer-Adressen benachbarter Knoten auf demselben Link zu ermitteln,
- ungültig gewordene Cacheeinträge zu löschen,
- die für sie zuständigen Router zu finden und
- darüber Buch zu führen, welche Nachbarn erreichbar sind. Wird ein Router oder der Weg dorthin unerreichbar, sucht der Knoten aktiv nach Alternativen.

2.5 Autokonfiguration

IPv6 definiert zwei Arten der automatischen Konfiguration: zustandslos (stateless) [Thomson 96b] und dynamisch (stateful) [Perkins 97a].

2.5.1 Stateless Autokonfiguration

Bei Anwendung der zustandslosen Autokonfiguration ist keinerlei manuelle Konfiguration von Hosts und nur minimale Konfiguration von Routern notwendig. Router senden in regelmäßigen Abständen Präfixe aus, die die zu einem Link gehörenden Subnetze identifizieren. Hosts bilden aus diesem Präfix und einem selbst generierten Interface Token, i.a. der 48 Bit langen MAC-Adresse, ihre IPv6-Adresse. Ist auf dem Link kein Router vorhanden, kann nur eine Link-Local-Adresse gebildet werden, die aber zur Kommunikation mit anderen Knoten auf demselben Link ausreicht.

2.5.2 Stateful Autokonfiguration

Im dynamischen Modell erhalten Hosts über die IPv6-Variante des Dynamic Host Configuration Protocol (DHCPv6) (siehe auch [Rieg 96]) ihre Adresse und andere Konfigurationsinformationen von einem Server, der vergebene Adressen in einer Datenbank speichert. Diese Art der Konfiguration gestattet Netzmanagern umfangreichere Eingriffsmöglichkeiten als die zustandslose Autokonfiguration.

[Hinden 97b] definiert ein Protokoll, das die Konfiguration und die Änderung der von Routern gesendeten Präfixe ähnlich einfach macht, wie die Kombination aus Neighbor Discovery und Autokonfiguration dies für Hosts bietet.

Kapitel 3

Migrationsverfahren

Die Umstellung von Millionen von Rechnern, die über das Internet-Protokoll Version 4 kommunizieren, auf das Internet-Protokoll Version 6 kann nicht auf einen Schlag geschehen. Vielmehr muß von einer Jahrzehnte dauernden Übergangszeit ausgegangen werden, während der beide Protokolle nebeneinander verwendet werden. In dieser Zeit müssen neu installierte IPv6-fähige Knoten kompatibel zu den Rechnern sein, die nur über einen IPv4-Protokollstack verfügen.

Die Migration verfolgt folgende Ziele:

inkrementeller Upgrade: IPv4-Knoten sollen einzeln und voneinander unabhängig umgerüstet werden können.

minimale Upgrade-Abhängigkeiten: Router können jederzeit ohne vorherige Maßnahmen IPv6-fähig gemacht werden. Die Vorbedingung für Host-Upgrades ist ein IPv6-fähiges DNS.

einfache Adressierung IPv6/IPv4-Knoten können ihre IPv4-Adresse zunächst behalten.

3.1 Migrationsmechanismen

Die „Internet Protocol Next Generation Working Group“ (IPngwg) der IETF hat mehrere Migrationsmechanismen erarbeitet, durch die diese Ziele erreicht werden sollen. Das zentrale Dokument dazu ist [Gilligan 96]. In diesem Kapitel werden die dort dargestellten Mechanismen sowie einige neuere Vorschläge diskutiert.

3.1.1 IPv6/IPv4-Knoten

Um die Kompatibilität zwischen nur-IPv4- und IPv6-fähigen Knoten zu gewährleisten, sollen neu zu installierende Systeme vollständige Implementierungen beider Protokollstacks besitzen. Diese IPv6/IPv4-Knoten genannten Rechner können direkt mit IPv4-Knoten über IPv4 und mit IPv6-Knoten über IPv6 kommunizieren. Abbildung 3.1 zeigt den schematischen Aufbau der Protokollschichten solcher IPv6/IPv4-Knoten.

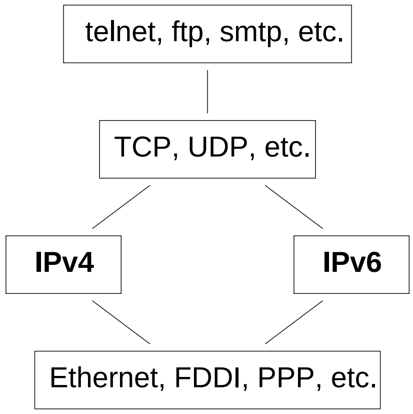

Abbildung 3.1: IPv6/IPv4-Knoten

Es sind folgende Typen von Hosts und Routern zu unterscheiden:

nur-IPv4-Knoten können nur über IPv4 kommunizieren.

IPv6/IPv4-Knoten besitzen beide Protokollstacks und können mit beiden Rechnertypen kommunizieren. Sie können mit zusätzlichen Mechanismen wie **IPv6-in-IPv4-Kapselung** (siehe Abschnitt 3.1.3) ausgestattet sein.

nur-IPv6-Knoten können nur über IPv6 kommunizieren.

3.1.2 Adreßkonfiguration

IPv6/IPv4-Knoten können sowohl mit IPv6-Adressen, als auch mit IPv4-Adressen ausgestattet sein. Beispielsweise können Rechner, die vor der Umstellung auf einen dualen Protokollstack bereits eine IPv4-Adresse hatten, diese weiterhin behalten. Eine IPv6-Adresse können diese Rechner unabhängig davon zusätzlich erhalten.

Ein IPv6/IPv4-Knoten kann die für den jeweiligen Protokollstack benötigte Adresse über die dafür üblichen Verfahren zur automatischen Adreßkonfiguration beziehen. Die IPv4-Adresse kann z.B. durch das Dynamic Host Configuration Protocol (DHCP) (siehe [Droms 97]) konfiguriert werden. Die IPv6-Adresse kann durch zustandslose oder dynamische Autokonfiguration (siehe Abschnitt 2.5) bezogen werden.

Soll ein Knoten die Fähigkeit zur automatischen IPv6-in-IPv4-Kapselung (siehe Abschnitt 3.1.3) erhalten, wird er mit einer *IPv4-kompatiblen IPv6-Adresse* konfiguriert. Tabelle 3.1 zeigt die Gestalt dieser Adressen.

80 bits	16 bits	32 bits
0:0:0:0:0	**0000**	IPv4-Adresse

Tabelle 3.1: IPv4-kompatible IPv6-Adressen

Alle 128 Bits einer IPv4-kompatiblen IPv6-Adresse dienen als IPv6-Adresse, während die niederwertigsten 32 Bits die IPv4-Adresse des Knotens bilden.

IPv4-kompatible Adressen können sowohl mit Verfahren zur automatischen Konfiguration von IPv4-Adressen erworben werden, als auch mit den IPv6-spezifischen zustandslosen oder dynamischen Mechanismen.

Diese Adressen sind nicht zu verwechseln mit *IPv4-mapped IPv6-Adressen*. Reine IPv4-Adressen können so als IPv6-Adressen z.B. in Datenstrukturen, die eine IPv6-Adresse erwarten, dargestellt werden. Tabelle 3.2 zeigt ihre Gestalt.

80 bits	16 bits	32 bits
0:0:0:0:0	**FFFF**	IPv4-Adresse

Tabelle 3.2: IPv4-mapped IPv6-Adressen

3.1.3 IPv6-in-IPv4-Kapselung

IPv6-in-IPv4-Kapselung, auch -Tunneln genannt, macht es möglich, bestehende und im Laufe der Zeit gewachsene IPv4-Infrastrukturen weiterzubetreiben und sie während der Umstellungsphase und danach für den Transport von IPv6-Paketen zu nutzen.

IPv6-in-IPv4-Kapselung bedeutet, der Knoten am Startpunkt eines Tunnels stellt vor die IPv6-Header der durch den Tunnel zu transportierenden Pakete IPv4-Header mit der Zieladresse des Tunnelendpunktes. Das gesamte IPv6-Paket inklusive des IPv6-Headers ist nun die Nutzlast des neu enstandenen IPv4-Paketes. Der Knoten am Endpunkt entfernt diese IPv4-Header und leitet die wiedergewonnenen IPv6-Pakete an das eigentliche Ziel weiter.

Abbildung 3.2 zeigt das Prinzip der Kapselung.

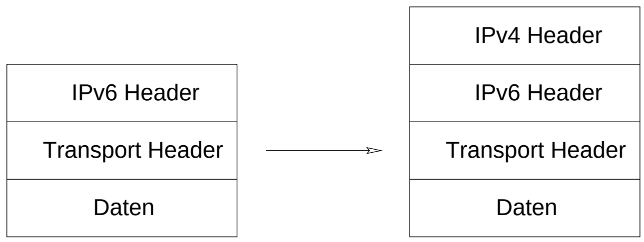

Abbildung 3.2: IPv6-in-IPv4-Kapselung

Vier Arten von Tunnels sind denkbar:

1. **Router zu Router**: Der Tunnel erstreckt sich über einen Teil des Ende-zu-Ende Pfades zwischen zwei IPv6/IPv4-Routern, die durch eine IPv4-Infrastruktur verbunden sind.

2. **Host zu Router**: Der Tunnel erstreckt sich über den ersten Teil des Ende-zu-Ende Pfades zwischen einem IPv6/IPv4-Host und einem IPv6/IPv4-Router.

3. **Host zu Host**: Der Tunnel erstreckt sich über den gesamten Ende-zu-Ende Pfad zwischen zwei IPv6/IPv4-Hosts.

4. **Router zu Host**: Der Tunnel erstreckt sich über den letzten Teil des Ende-zu-Ende Pfades zwischen einem IPv6/IPv4-Router und einem IPv6/IPv4-Host.

In den ersten beiden Fällen kann die IPv4-Adresse des Tunnelendpunktes nicht aus den zu transportierenden IPv6-Paketen gewonnen werden. Sie muß jeweils am Startpunkt konfiguriert werden. Man spricht hier von einem **konfigurierten Tunnel**.

In den beiden letzten Fällen ist der Endpunkt des Tunnels mit dem Ziel der Pakete identisch. Besitzt dieser Endpunkt eine IPv4-kompatible IPv6-Adresse, kann aus der eingebetteten IPv4-Adresse die IPv4-Adresse des Tunnelendpunktes automatisch gewonnen werden. Dieser Mechanismus wird **automatischer Tunnel** genannt.

Abbildung 3.3 zeigt einen automatischen und einen konfigurierten Tunnel.

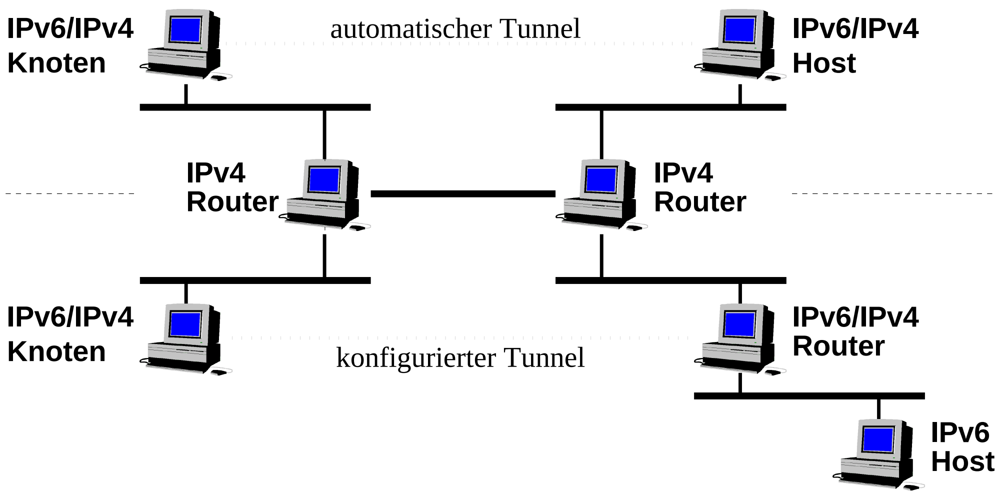

Abbildung 3.3: Automatische und konfigurierte Tunnel

In späteren Phasen der Migration, wenn in den meisten Routern das IPv4-Protokoll nicht mehr unterstützt wird, wäre es denkbar, daß die verbliebenen IPv4-Inseln einen inversen Mechanismus, die *IPv4-in-IPv6-Kapselung*, verwenden, um miteinander zu kommunizieren. Ein Entwurf zu einem Standard, der dies ermöglicht, ist noch nicht vorhanden.

3.1.4 Header Translation

Die bislang beschriebenen Migrationsmechanismen erfordern Rechner mit dualem IPv6/IPv4-Protokollstack. [Nordmark 97] beschreibt einen Mechanismus, der es reinen IPv6-Hosts erlaubt, mit reinen IPv4-Hosts mittels eines *Header Translators* zu kommunizieren. In dem Entwurf werden dazu folgende Szenarios genannt:

- Ein neues Netz besteht aus Geräten, die nur IPv6 unterstützen. Auch die Router im Netz sind nicht in der Lage, IPv4-Pakete weiterzuleiten. Trotzdem möchten diese Geräte gelegentlich mit IPv4-Knoten außerhalb des Netzes kommunizieren.

- Einem bestehenden Netz wird eine größere Anzahl IPv6-fähiger Rechner hinzugefügt. In diesen neuen Hosts sind möglicherweise beide Protokollstacks implementiert, jedoch kann nicht jedem von ihnen eine permanente IPv4-Adresse zugewiesen werden, da dazu der in diesem Netz zur Verfügung stehende globale IPv4-Adreßraum nicht ausreicht.

Um die Probleme in beiden Fällen zu lösen, wird vorgeschlagen, daß an den betroffenen IPv6-Host durch ein erweitertes DHCP eine temporäre IPv4-Adresse vergeben wird, die von diesen als IPv4-kompatible IPv6-Adresse benutzt wird. Die Pakete werden durch einen zustandslosen Header Translator geschickt, der die Paket-Header zwischen IPv4 und IPv6 und die darin enthaltenen Adressen zwischen IPv4-Adressen und IPv4-kompatiblen oder -mapped IPv6-Adressen übersetzt.

Dieser erste Entwurf wird noch diskutiert. Implementierungen solcher Übersetzer sind erst in einigen Monaten zu erwarten.

3.2 Szenarios

Zur Veranschaulichung der Migrationsmechanismen und Herleitung der in Kapitel 5 untersuchten Managementanforderungen für die Verwaltung von IPv6-Netzen und bei der Migration werden im folgenden einige typische Szenarios der Migration von IPv4 zu IPv6 entwickelt.

Allen Szenarios ist gemeinsam, daß auf den zu migrierenden Rechnern zuerst eine IPv6/IPv4-fähige Betriebssystemversion und die benötigten IPv6/IPv4-fähigen Netzanwendungen installiert werden müssen. Außerdem muß ein Domain Name System eingerichtet werden, in das die IPv6-Adressen der umzustellenden Rechner entweder statisch eingetragen sind oder dynamisch etwa durch DHCPv6-Mitteilungen geliefert werden können (siehe auch Abschnitt 5.3.1).

3.2.1 Migration eines einzelnen Hosts

Dieser Abschnitt behandelt den Fall eines einzelnen IPv4-fähigen Hosts, der zusätzlich die Möglichkeit erhalten soll, über das Internet-Protokoll Version 6 mit anderen Rechnern zu kommunizieren. Dafür ergeben sich mehrere mögliche Szenarios:

1. *Im LAN des Hosts befindet sich ein IPv6-Router. Dieser Router ist für zustandslose Autokonfiguration eingerichtet oder ein DHCPv6-Agent ist zur dynamischen Autokonfiguration im LAN verfügbar.*

   Nach einem Neustart des Betriebssystems wird neben der *Link-Local*-Adresse mindestens auch eine *Site-Local-* wenn nicht eine global gültige IPv6-Adresse automatisch konfiguriert und der Host kann mit den Knoten in seinem LAN und, je nach Konfiguration des Routers, darüber hinaus über IPv6 kommunizieren.

2. *Im LAN des Hosts befindet sich ein IPv6-Router. Dieser Router ist nicht für zustandslose Autokonfiguration eingerichtet und es steht auch kein DHCPv6-Agent zur dynamischen Autokonfiguration im LAN zur Verfügung.*

   Dies ist möglich, sollte aber nicht vorkommen. Da nur die *Link-Local*-Adresse automatisch generiert wurde, kann der Host ohne manuellen Eingriff nur mit den Knoten in seinem LAN über IPv6 kommunizieren. Um mit Knoten jenseits des Routers Verbindung aufnehmen zu können, muß eine *Site-Local*- oder global gültige IPv6-Adresse manuell konfiguriert werden.

3. *Im LAN des Hosts befindet sich kein IPv6-Router, aber ein IPv4-Router.*

   Der Host kann ohne besondere Vorkehrungen nur mit Knoten in seinem LAN über IPv6 kommunizieren.

   Erhält der Host manuell eine *IPv4-kompatible IPv6-Adresse*, kann er mit solchen Hosts jenseits des IPv4-Routers, die ebenfalls IPv4-kompatible IPv6-Adressen besitzen, in beide Richtungen über IPv6 kommunizieren (siehe auch Abschnitt 3.1.3).

   Erhält der Host manuell eine IPv6-Adresse und wird manuell ein *konfigurierter Tunnel* zu einem IPv6/IPv4-Router eingerichtet, kann dieser Host, je nach Konfiguration des IPv6/IPv4-Routers, mit Knoten außerhalb seines LANs über IPv6 kommunizieren.

3.2.2 Migration eines Routers

Nun wird die Migration eines IPv4-Routers hin zu einem Router mit dualem IPv6/IPv4-Protokollstack untersucht. Auch hierfür gibt es mehrere Szenarios:

1. *Zwischen dem zu migrierenden Router und dem nächsten erreichbaren IPv6-Router befinden sich ein oder mehrere nur zur Kommunikation über IPv4 fähige Router.*

   Dieser Fall ähnelt dem letzten Szenario des vorherigen Abschnitts. Der Router kann nur dann IPv6-Pakete aus seinem LAN nach außerhalb und umgekehrt vermitteln, wenn manuell seine IPv6-Adressen und ein *konfigurierter Tunnel* zu einem IPv6/IPv4-Router außerhalb seines LANs eingerichtet wurde. Dieses Szenario trifft z.B. auf das in Kapitel 4 und Anhang A beschriebene Testnetz zu.

2. *Die nächsten IPv6-Router können direkt ohne Tunnel erreicht werden.*

3. *Einige benachbarte Router können direkt über IPv6 erreicht werden, zu anderen Routern muß über IPv4-Topologien getunnelt werden.*

Allen diesen Szenarien ist gemeinsam, daß mit zunehmender Komplexität der Topologie der Einsatz von für IPv6 modifizierten *Routing-Protokollen* zu empfehlen ist. Über Routing-Protokolle tauschen Router Informationen darüber aus, über welche Wege bestimmte Knoten erreichbar sind, und berechnen damit ihre Routing-Tabellen. Routing-Protokolle teilen sich in interne (RIPng, OSPF) und externe (IDRPv2, BGP-4+) auf, je nachdem, ob die benachbarten Router zum selben autonomen System gehören oder nicht.

3.2.3 Migration eines Subnetzes

Jetzt sollen alle Rechner eines Subnetzes, das über einen Router mit den übrigen Subnetzen einer Organisation verbunden ist, zur Kommunikation über IPv6 befähigt werden. Dies bedeutet, daß für den Router eines der im letzten Abschnitt und für alle Hosts des Subnetzes eines der in Abschnitt 3.2.1 beschriebenen Szenarios zutreffen wird.

Schon ab einer geringen Zahl von Hosts wird sich die Nutzung der Autokonfigurationsmöglichkeiten von IPv6 durch geringeren Konfigurationsaufwand auszahlen. Im günstigsten Fall beschränkt sich der Umstellungsaufwand auf ein Update des Betriebssystems und der Netzwerkzeuge aller beteiligten Rechner, das, abhängig vom verwendeten Betriebssystem, ebenfalls automatisiert stattfinden kann, und der Konfiguration des Routers.

3.2.4 Zusammenfassung

Sollen zum gegenwärtigen Zeitpunkt ein oder mehrere Hosts eines Netzes in die Lage versetzt werden, mit Hosts innerhalb und außerhalb des Netzes über IPv6 zu kommunizieren, kann im allgemeinen folgendermaßen vorgegangen werden:

1. Einrichten eines IPv6-fähigen Domain Name Systems
2. Einrichten eines dualen Routers und ggf. eines Tunnels zum nächsten IPv6-Router
3. ggf. Konfiguration eines Routing-Protokolls
4. Aktivieren der zustandslosen Autokonfiguration im Router
5. Update des Betriebssystems und der Netzwerkzeuge der zu migrierenden Hosts

Header Translation und DHCPv6 stehen momentan noch nicht zur Verfügung. Von den Routing-Protokollen sind nur RIPng, BGP-4+ und (selten) IDRPv2 im Einsatz und nicht auf allen Plattformen verfügbar.

3.3 Anwendungsentwicklung

Anwendungen für BSD-basierte Unix-Betriebssysteme wie auch einige Nicht-Unix-Systeme greifen auf Netzdienste meist über die Socket-Schnittstelle zu. In einigen Funktionen dieser Schnittstelle werden IP-Adressen für die Anwendung sichtbar. Wollen Anwendungen über das neue Internet-Protokoll kommunizieren, müssen sie mit 128 Bit langen IP-Adressen und, je nach Zweck der Anwendung, mit einigen der in IPv6 neu eingeführten Features wie Flußkennzeichnung und Priorität umgehen können.

Pläne der Hersteller von Werkzeugen wie Perl, Tcl/Tk, Java und ähnlichen, das Internet-Protokoll Version 6 zu unterstützen, sind gegenwärtig nicht bekannt. Die folgende Untersuchung beschränkt sich daher auf die in der Programmiersprache C notwendigen Änderungen.

Um auf die Funktionalität des neuen Internet-Protokolls zugreifen zu können, sind einige Erweiterungen der Socket-Schnittstelle notwendig. Grundlegende Erweiterungen wie eine neue Datenstruktur zur Aufnahme von IPv6-Adressen und zur Namensauflösung werden in [Gilligan 97] beschrieben. Zusätzliche Erweiterungen für den Zugriff auf Raw Sockets und die Erweiterungsheader schlägt [Stevens 97] vor.

Für die Änderungen der Schnittstelle waren mehrere Designüberlegungen maßgebend:

- Die Änderungen sollten quellcode- und binärkompatibel zur alten Programmierschnittstelle sein.
- Die Änderungen sollten möglichst gering ausfallen, um eine einfache Portierung bestehender Anwendungen zu gewährleisten.
- Anwendungen sollten über diese Schnittstelle sowohl mit IPv6- als auch mit IPv4-Hosts kommunizieren können, ohne den Typ des Hosts zu kennen.
- IPv6-Adressen in Datenstrukturen sollten auf 64-Bit-Grenzen angeordnet sein, um auf 64-Bit-Rechnerarchitekturen optimale Performanz zu erhalten.

Zur Veranschaulichung der grundlegenden Erweiterungen wird im folgenden ein kleines Beispiel zur Client-Server-Programmierung entwickelt und von der IPv4-Socket-Schnittstelle auf die erweiterte, nun IPv6-fähige Socket-Schnittstelle portiert. Die Auswahl der im Beispiel verwendeten Funktionen erhebt keinen Anspruch auf Vollständigkeit, versucht aber, alle wichtigen Funktionen zu berücksichtigen. Dieses Thema könnte z.B. in einem Systempraktikum ausführlich behandelt werden.

Das Beispiel besteht aus zwei Programmen, einem Server, der auf TCP-Verbindungswünsche wartet und empfangene Mitteilungen ausgibt, und einem Client, der Mitteilungen an den Server schicken kann. Die kompletten Quelltexte sind in Anhang B zu finden.

3.3.1 Einen Socket erzeugen

Ein Socket wird mit der Funktion `socket()` erzeugt. Die Aufrufschnittstelle lautet:

```
int socket(int family, int type, int protocol);
```

Der erste Parameter *family* gibt an, welche Protokollfamilie zur Kommunikation verwendet werden soll. Im Fall des Internet-Protokolls Version 4 ist hier `PF_INET` zu übergeben. Für IPv6 wurde die neue Familie `PF_INET6` definiert.

Der zweite Parameter *type* spezifiziert den Typ des Sockets. Hier stehen unter anderem unverändert `SOCK_STREAM` für zuverlässige, verbindungsorientierte und `SOCK_DGRAM` für unzuverlässige, verbindungslose Kommunikation zur Verfügung.

Mit dem dritten Parameter *protocol* wird das Transportprotokoll ausgewählt. Wird hier `0` angegeben, so benutzt die Funktion das jeweilige Default-Protokoll des Socket-Typs. Bei `SOCK_STREAM` ist dies TCP, bei `SOCK_DGRAM` UDP.

Der Funktionsaufruf liefert die Nummer des erzeugten Sockets als Ergebnis zurück.

Wurde in einem C-Programm ein IPv4/TCP-Socket bisher so erzeugt:

```
sock = socket(PF_INET, SOCK_STREAM, 0);
```

lautet der Funktionsaufruf für die Erzeugung eines IPv6/TCP-Sockets nun so:

```
sock = socket(PF_INET6, SOCK_STREAM, 0);
```

3.3.2 Einen Verbindungswunsch äußern

Hat ein Prozeß mittels `socket()` einen Socket erzeugt, kann er mit der Funktion `connect()` versuchen, eine Verbindung zu einem anderen Prozeß herzustellen. Die Schnittstelle lautet:

```
int connect(int socket, struct sockaddr *server_addr,
int addrlen);
```

Mit *socket* übergibt man die Nummer des Sockets.

server_addr enthält Adresse und Port des Serverprozesses. Die Struktur `sockaddr` ist dabei protokollunabhängig und je nach Protokoll gecastet von `sockaddr_in` oder `sockaddr_in6`. Letztere Struktur enthält zusätzlich ein 32 Bit langes Feld `sin6_flowinfo`, bestehend aus Flowlabel und Priorität. Inhalt und Interpretation dieses Feldes sind gegenwärtig noch nicht spezifiziert.

addrlen ist die Größe von *server_addr*.

Die `sockaddr_in6`-Struktur ist folgendermaßen definiert:

```
#include <netinet/in.h>

struct sockaddr_in6 {
    u_int16m_t      sin6_family;    /* AF_INET6 */
    u_int16m_t      sin6_port;      /* transport layer port # */
    u_int32m_t      sin6_flowinfo;  /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
};
```

3.3.3 Auf einen Verbindungswunsch warten

Der Server, der den mit `connect()` geäußerten Verbindungswunsch erfolgreich beantworten soll, muß diesen erwarten. Dies erreicht er z.B. mit den drei Funktionen `bind()`, `listen()` und `accept()`.

Mit `bind()` teilt man einem vorher mit `socket()` erzeugten Socket mit, zu welchem Port dieser gehört. Die Syntax lautet:

`int bind(int` *`socket`*`, struct sockaddr *`*`local_addr`*`, int` *`addrlen`*`)`

Die Parameter entsprechen denen der `connect()`-Funktion, mit dem Unterschied, daß der zweite Parameter *`local_addr`* die lokale Adresse enthält. Da ein Rechner mehrere Adressen besitzen kann, wird für IPv4 die symbolische Konstante `INADDR_ANY` zur Spezifikation einer „Wildcard"-Adresse angeboten. Die Funktion verbindet dann alle IP-Verbindungen mit dem angegebenen Port.

Da eine IPv6-Adresse durch eine Struktur `in6_addr` wie folgt dargestellt wird:

```
#include <netinet/in.h>

struct in6_addr {
    u_int8_t  s6_addr[16];      /* IPv6 address */
}
```

kann man in Deklarationen eine Variable solchen Typs zwar mit einer symbolischen Konstante initialisieren, einer zuvor deklarierten `in6_addr`-Struktur aber nicht deren Wert zuweisen. Es werden deshalb zwei Möglichkeiten zur Zuweisung einer „Wildcard"-Adresse angeboten:

1. Eine globale Variable `in6addr_any`:

   ```
   #include <netinet/in.h>:

   extern const struct in6_addr in6addr_any;
   ```

   die in Zuweisungen ähnlich der symbolischen Konstante `INADDR_ANY` verwendet werden kann:

   ```
   /* Die sockaddr_in6 Struktur mit Wildcards fuellen */
   server.sin6_family = AF_INET6;
   server.sin6_addr = in6addr_any;
   server.sin6_port = 0;

   /* Socket an IP-Adresse und Port binden */
   if (bind(sock, (struct sockaddr *) &server, sizeof(server))) {
     perror("bind");
     exit(1);
   }
   ```

und

2. eine symbolische Konstante `IN6ADDR_ANY_INIT`, die nur in Deklarationen zur Initialisierung dienen kann:

```
struct in6_addr anyaddr = IN6ADDR_ANY_INIT;
```

`listen()` versetzt den Socket in einen passiven Modus und richtet eine Warteschlange ein, die die Beantwortung mehrerer gleichzeitig eintreffender Verbindungswünsche gestattet. Die Syntax:

```
int listen(int socket, int queuelen);
```

`accept()` erzeugt für jeden Verbindungswunsch einen neuen Socket, über den die Datenübertragung abgewickelt werden kann und hält den ursprünglichen Socket zur Annahme von weiteren Verbindungswünschen bereit. Die Syntax:

```
int accept(int socket, struct sockaddr *client_addr,
    int addrlen);
```

Die Parameter entsprechen denen der `connect()`-Funktion.

3.3.4 Senden und Empfangen von Daten

Das Senden und das Empfangen von Daten mittels `write()` bzw. `read()` unterscheidet sich nicht von den unter IPv4 üblichen Verfahren. Der Vollständigkeit halber sind hier die Aufrufschnittstellen angegeben:

```
int write(int socket, char *data, int datalen);

int read(int socket, char *buf, int buflen);
```

3.3.5 Namensauflösung

Die Funktion `gethostbyname()` bleibt unverändert, wie auch die `hostent`-Struktur zu der die Funktion einen Zeiger zurückliefert. Bestehende Anwendungen bekommen beim Aufruf dieser Funktion weiterhin IPv4-Adressen (also A-Records, falls das DNS gefragt wird) zurückgeliefert. Zwei Änderungen erlauben die Unterstützung von IPv6-Adressen:

1. Eine neue Funktion `gethostbyname2()`:

   ```
   struct hostent *gethostbyname2(const char *name,
       int af);
   ```

   Der neu hinzugekommene Parameter *af* bezeichnet die gewünschte Adreßfamilie `AF_INET` oder `AF_INET6`.

2. Eine neue Resolver-Option `RES_USE_INET6`, die entweder als Umgebungsvariable (Bourne- und Kornshell: `export RES_OPTIONS=inet6`), in der Resolver-Konfigurationsdatei `/etc/resolv.conf` als `options inet6` oder in der Applikation mit

```
res_init();
_res.options |= RES_USE_INET6;
```

gesetzt werden kann.

Ist die Option `RES_USE_INET6` gesetzt, werden immer 128-Bit-Adressen zurückgeliefert, wenn nötig IPv6-mapped IPv4-Adressen.

Ist die Option `RES_USE_INET6` gesetzt, wird auch bei Aufruf der `gethostbyname()`-Funktion zuerst nach einem AAAA-Record gesucht, danach nach einem A-Record.

Kann eine Applikation keine 128-Bit-Adressen verarbeiten, darf diese Option nicht aktiviert werden.

3.3.6 Adreßauflösung

Die bestehende Funktion `gethostbyaddr()` benötigt bereits einen Parameter, der die gewünschte Adreßfamilie spezifiziert:

```
struct hostent *gethostbyaddr(const char *src, int len, int af);
```

und kann daher auch mit IPv6-Adressen arbeiten, wenn für *af* `AF_INET6` angegeben wird.

3.3.7 Protokollunabhängige Namensauflösung

Die in Arbeit befindliche Spezifikation des Institute of Electrical and Electronic Engineers (IEEE) POSIX 1003.1g definiert eine protokollunabhängige Funktion `getaddrinfo()` zur Übersetzung eines Rechnernamens in eine IP-Adresse. Die Beschreibung dieser Funktion wurde ebenfalls in [Gilligan 97] wiedergegeben:

```
int getaddrinfo(const char *hostname, const char *servname, const struct addrinfo *hints, struct addrinfo **res);
```

Im üblichen Client-Szenario, so auch im Beispiel IPv6/IPv4-TCP-Client, werden an die Funktion sowohl ein *hostname* als auch ein *servname* übergeben, die beide nicht `NULL` sind:

```
if (result = getaddrinfo(argv[1], argv[2], &hints, pai)) {
  fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
  exit(1);
}
```

`argv[1]` und `argv[2]` enthalten Namen und Port des Servers.

`hints` wurde zuvor mit Hinweisen auf den gewünschten Socket-Typ und weiteren Flags gefüllt:

```
memset(&hints, 0, sizeof(hints));
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_CANONNAME;
```

In `pai` wird ein Zeiger auf eine verkettete Liste von einer oder mehreren komplettierten `addrinfo`-Strukturen zurückgeliefert.

Die `addrinfo`-Struktur ist folgendermaßen definiert:

```
#include <sys/socket.h>
#include <netdb.h>

struct addrinfo {
  int     ai_flags;     /* AI_PASSIVE, AI_CANONNAME */
  int     ai_family;    /* PF_xxx */
  int     ai_socktype;  /* SOCK_xxx */
  int     ai_protocol;  /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
  size_t  ai_addrlen;   /* length of ai_addr */
  char   *ai_canonname; /* canonical name for hostname */
  struct sockaddr  *ai_addr; /* binary address */
  struct addrinfo  *ai_next; /* next structure in linked list */
};
```

Im Beispiel werden zur Verdeutlichung die einzelnen Feldinhalte ausgegeben.

Ist `ai` als `struct addrinfo` definiert, kann nach dem obigen Funktionsaufruf in dieser Weise ein Socket erzeugt werden:

```
if ((sock = socket(ai->ai_family, ai->ai_socktype,
                   ai->ai_protocol)) < 0) {
  printf("socket: %s(%d)\n", strerror(errno), errno);
  continue;
};
```

So kann ein Verbindungswunsch geäußert werden:

```
if (connect(sock, ai->ai_addr, ai->ai_addrlen) < 0) {
  printf("connect: %s(%d)\n", strerror(errno), errno);
  continue;
};
```

Im üblichen Server-Szenario wird die `getaddrinfo()`-Funktion ohne Angabe des Rechnernamens im ersten Parameter aufgerufen, was der Spezifikation einer „Wildcard"-Adresse entspricht:

```
if (result = getaddrinfo(NULL, port, &hints, pai)) {
  fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
  exit(1);
}
```

Die `bind()`-Funktion kann analog zur `connect`-Funktion verwendet werden:

```
if (bind(sock, ai->ai_addr, ai->ai_addrlen)) {
  perror("bind");
  exit(1);
}
```

3.3.8 Protokollunabhängige Adreßauflösung

Der POSIX 1003.1g-Entwurf enthält keine zu `getaddrinfo()` inverse Funktion zur Ermittlung von Host- und Portnamen für gegebene binäre Adressen und Ports. [Gilligan 97] definiert dazu die Funktion `getnameinfo()`:

```
int getnameinfo(const struct sockaddr *sa, size_t
salen, char *host, size_t hostlen, char *serv, size_t
servlen, int flags);
```

In *sa* wird eine `sockaddr`-Struktur übergeben, die die binären Host- und Portangaben enthält. In *host* bzw. *serv* wird der ermittelte Host- bzw. Portname zurückgeliefert, abhängig von in *flags* gesetzten Optionen. Im Beispiel wird die Funktion zweimal aufgerufen, um sowohl Namen als auch Nummern zu erhalten:

```
if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                         hbuf[0], sizeof(hbuf[0]),
                         sbuf[0], sizeof(sbuf[0]), 0)) {
  fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
  continue;
};
if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                         hbuf[1], sizeof(hbuf[1]),
                         sbuf[1], sizeof(sbuf[1]),
                         NI_NUMERICHOST | NI_NUMERICSERV)) {
  fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
  continue;
};
printf("ai_addr: %s.%s(%s.%s)\n\n",
       hbuf[0], sbuf[0], hbuf[1], sbuf[1]);
```

3.3.9 Adreßkonvertierung

Die Funktionen `inet_addr()` und `inet_ntoa()` zur Konvertierung von der Textform von IPv4-Adressen in deren Binärform und umgekehrt, erhalten Pendants, die auch IPv6-Adressen konvertieren können:

```
int inet_pton(int af, const char *src, void *dst);
```

und

```
const char *inet_ntop(int af, const void *src, char
*dst, size_t size);
```

Im Beispiel wurde der Aufruf der Funktion `inet_ntoa()`, die die übergebene Binäradresse im Klartext zurückgibt:

```
printf("offizieller Name: %s, IP-Adresse: %s\n",
       hostinfo->h_name, inet_ntoa(server.sin_addr));
```

in ihre IPv6-Form übersetzt:

```
if (inet_ntop(AF_INET6, &server.sin6_addr,
              str, INET6_ADDRSTRLEN) == NULL)
    strcpy(str, "[ungueltige Adresse]");

printf("offizieller Name: %s, IP-Adresse: %s\n",
       hostinfo->h_name, str);
```

3.3.10 Kompilieren der Beispiele

Da die Entwicklung der IPv6-fähigen Definitionsdateien und Bibliotheken noch nicht abgeschlossen ist, wurden die in den Beispielen verwendeten in Verzeichnissen `include` bzw. `lib` abgelegt. Die Präprozessoranweisungen zum Einbinden der benötigten Definitionsdateien in `v4_tcpclient.c` sollten unter Linux nach dem Installieren der zur neuen C-Bibliothek `libc`-Version 6 gehörenden Dateien auch für die Beispiele `v6_tcpclient.c` und `v6v4_tcpclient.c` ausreichen. Zusätzliche Bibliotheken müssen dann auch nicht mehr eingebunden werden.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

Bei den Server-Beispielen muß zusätzlich `sys/time.h` eingebunden werden. Die Beispiele werden durch Eingabe von `make` kompiliert.

3.3.11 Testläufe der Programme

Der IPv4-Server wird mit `v4_tcpserver` gestartet. Er gibt dann die ihm zugeordnete Portnummer aus. Hier:

```
Zugeordneter Port: 2143
```

Ein IPv4-Client kann z.B. mit `v4_tcpclient pchegering2 2143 test` die Nachricht `test` an den Server schicken. Der Client gibt den mit `gethostbyaddr()` ermittelten offiziellen Namen des angegebenen Rechners aus:

```
offizieller Name: pchegering2.nm.informatik.uni-muenchen.de,\
IP-Adresse: 129.187.214.42
```

Der Server gibt die Nachricht und ihre Quelle aus:

```
Verbindungswunsch von Host pchegering2.nm.informatik.uni-muenchen.de\
(129.187.214.42), Port 2727
Mitteilung: test
Beende Verbindung
```

Da kein Rechnername gleichzeitig mit einer IPv6- und einer IPv4-Adresse im DNS eingetragen ist, wurde zu Testzwecken in `/etc/hosts` dieser Eintrag vorgenommen:

```
5f04:fb00:81bb:d600:d6:800:98b:a8d8     pc2
129.187.214.42                          pc2
```

Die Namensauflösung der C-Bibliothek Version 5 kommt mit der IPv6-Adresse von *pc2* erwartungsgemäß nicht zurecht:

```
$ v4_tcpclient pc2 2143 test
offizieller Name: pc2, IP-Adresse: 255.255.255.255
connect: Network is unreachable
```

Die neuen Funktionen im protokollunabhängigen `v6v4_tcpclient` dagegen schon. Dieser versucht zuerst, den Server über IPv6 zu erreichen, nachdem dies scheitert, gelingt es ihm über IPv4:

```
$ v6v4_tcpclient pc2 2143 test

Record 0:
ai_flags:      2(2)
ai_family:     inet6(10)
ai_socktype:   stream(1)
ai_protocol:   6(6)
ai_addrlen:    18(24)
ai_canonname:  pc2
ai_addr:       pc2.2143(5f04:fb00:81bb:d600:d6:800:98b:a8d8.2143)

Trying pc2.2143(5f04:fb00:81bb:d600:d6:800:98b:a8d8.2143)
connect: Connection refused(111)


Record 1:
ai_flags:      2(2)
ai_family:     inet(2)
ai_socktype:   stream(1)
ai_protocol:   6(6)
ai_addrlen:    10(16)
ai_canonname:  pc2
ai_addr:       pchegering2.nm.informatik.uni-muenchen.de.2143\
               (129.187.214.42.2143)

Trying pchegering2.nm.informatik.uni-muenchen.de.2143\
      (129.187.214.42.2143)
```

Die Funktionen `gethostbyname2()` und `gethostbyaddr()` der verwendeten Resolver-Bibliotheken sind offenbar fehlerhaft. Der `v6_tcpclient` meldet entweder `gethostbyaddr: Unknown host` oder wird vom Betriebssystem mit einem `Segmentation fault` beendet. Der `v6_tcpserver` meldet bei einer Verbindungsaufnahme mit dem protokollunabhängigen Client `gethostbyaddr: Unknown error`.

Die protokollunabhängigen Beispiele funktionieren als Server und Client korrekt. Startet man den `v6v4_tcpserver` unter einem IPv6-fähigen Betriebssystem, beantwortet er sowohl IPv6- als auch IPv4-Verbindungswünsche. Unterstützt das Betriebssystem IPv6 nicht, funktioniert der Server trotzdem, kann aber nur IPv4-Verbindungswünsche entgegennehmen.

Hier erreicht den Server eine von `v4_tcpclient` abgesandte Nachricht über IPv4 unter einem IPv6-fähigen Betriebssystem:

```
su_ai_addr:  pchegering2.nm.informatik.uni-muenchen.de.3682\
             (::ffff:129.187.214.42.3682)

Verbindung von pchegering2.nm.informatik.uni-muenchen.de.3682\
               (::ffff:129.187.214.42.3682)
Mitteilung: test
Beende Verbindung
```

Die IPv4-Adresse von *pchegering2* wird als IPv4-kompatible IPv6-Adresse ausgegeben.

Hier erreicht den Server eine von `v6v4_tcpclient` abgesandte Nachricht über IPv6:

```
su_ai_addr:  ipv6-localhost.3936(::1.3936)

Verbindung von ipv6-localhost.3936(::1.3936)
Mitteilung: test
Beende Verbindung
```

3.3.12 Bewertung

Es ist unproblematischer und portabler, für alle Namens- und Adreßauflösungen und -konvertierungen die Funktionen `getaddrinfo()` und `getnameinfo()` zu verwenden. Der Programmtext wird dadurch von Adreßfamilien unabhängig, und die Strukturen `sockaddr_in` und `sockaddr_in6` brauchen nicht benutzt werden. Der Code kann damit genauso in reinen IPv4- und reinen IPv6-, wie auch in gemischten Umgebungen eingesetzt werden.

Allerdings stellen ältere `libc`-Bibliotheken diese Funktionen nicht zur Verfügung. Auf derartigen Systemen müssen entweder die alten Funktionen aufgerufen werden, oder es muß eine Zusatzbibliothek wie die vom US Naval Research Laboratory (NRL) entwickelte `libinet6` eingebunden werden.

Kapitel 4

Das Testnetz

Um eine praktische Bewertung der verschiedenen in dieser Arbeit behandelten Migrationsaspekte möglich zu machen, sowie existierende IPv6-Software zu evaluieren, wurde am Institut für Informatik ein IPv6-Testnetz eingerichtet und mit dem *6bone*, einem weltweiten Testnetz zur Erprobung des Internet-Protokolls Version 6, verbunden.

Gegenwärtig existieren (Test-)IPv6-Implementierungen für die Betriebssysteme NetBSD, FreeBSD, BSDI, Linux, Windows 95, Windows NT, MacOS, Solaris, DEC, AIX, HP-UX, Novell, SCO und VMS. Implementierungen für Router bieten die Firmen: 3Com, Bay, cisco, DEC, Ipsilon, Merit, NTHU, Penril, Sumitomo und Telebit.

Das lokale IPv6-Testnetz des Instituts besteht derzeit aus den beiden Linux-PCs *terra.6bone.informatik.uni-muenchen.de*, der über IP Version 4 unter dem Namen *pchegering2.nm.informatik.uni-muenchen.de* erreichbar ist und *merkur.6bone.informatik.uni-muenchen.de*, dessen IPv4-Name *pchegering8.nm.informatik.uni-muenchen.de* ist.

terra bildet den Router zum *6bone*. Die Verbindung erfolgt über einen statischen v6-in-v4-Tunnel zu *6bone.join.uni-muenster.de*, einem Rechner des JOIN-Projekts der Universität Münster. Abbildung 4.1 zeigt den Aufbau des IPv6-Testnetzes am Institut.

Die Installation des lokalen Testnetzes wird in Anhang A ausführlich beschrieben. Das vorliegende Kapitel beschäftigt sich mit dem 6bone und dessen Verbindung mit dem lokalen Testnetz.

4.1 Überblick

Im März 1996 wurde während eines Treffens der Internet Engineering Task Force (IETF) in Los Angeles beschlossen, ein weltweites IPv6-Testnetz einzurichten. Ausgehend von den Gründungsmitgliedern in Europa, Japan und U.S.A. sollten sich im Lauf der Zeit möglichst viele Organisationen aus allen Teilen der Welt daran anschließen, um Erfahrungen mit neuentwickelten IPv6-Technologien sammeln und austauschen zu können. Als Name dieses Testnetzes wurde in Anlehnung an das bestehende Testnetz *MBONE* zur Entwicklung eines Multicast Backbones der Begriff *6bone* gewählt.

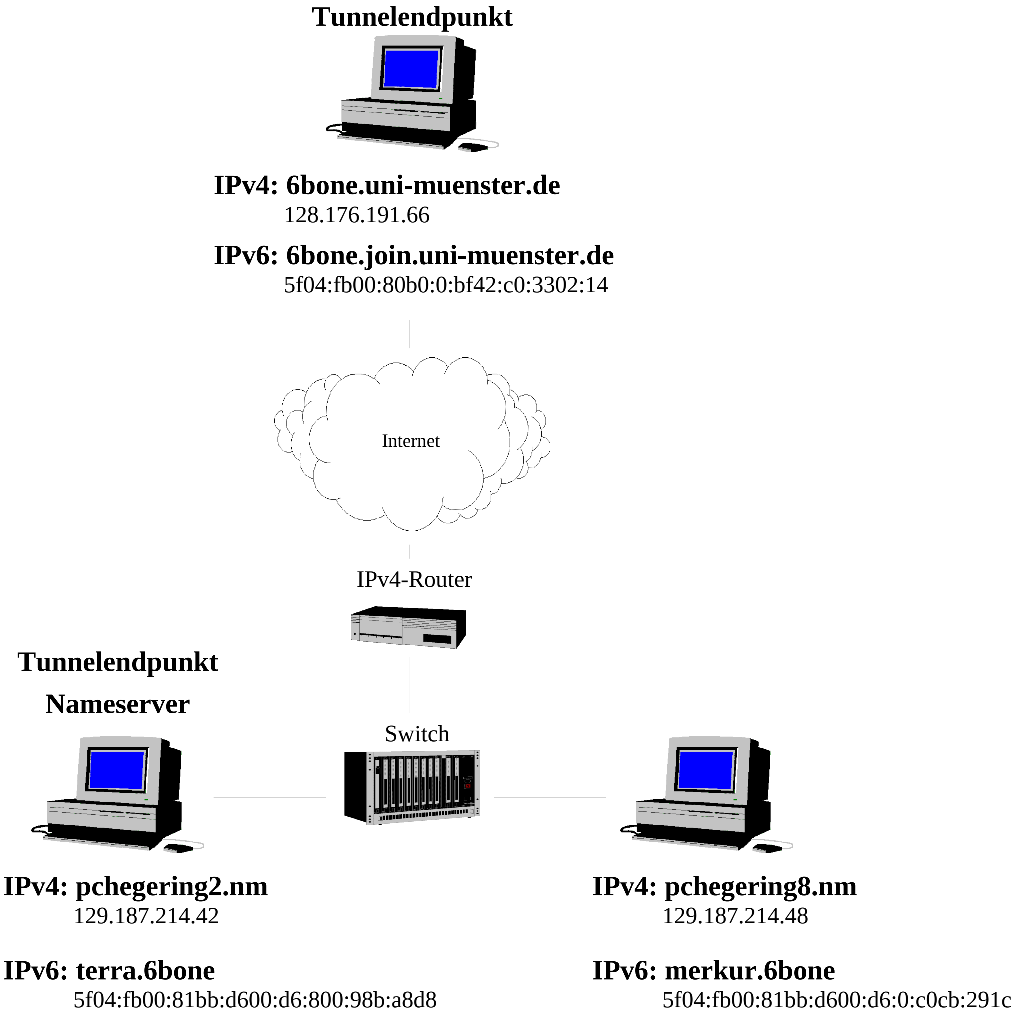

Abbildung 4.1: Das IPv6-Testnetz des Instituts

Die Ziele des 6bone beschreibt der Entwurf der 6bone-Charta [6bone 97] so:

- IPv6 Transport und Routing im globalen Internet zu entwickeln,
- RFC Dokumente zu erstellen, in denen Entwickler von IPv6-Technologien über ihre praktischen Erfahrungen mit den neuen Technologien berichten,
- Rückmeldungen zu verschiedenen IETF Aktivitäten bezüglich IPv6 zu liefern,
- Mechanismen und Prozeduren zu entwickeln, die die Migration zu IPv6 vereinfachen und
- Mechanismen und Prozeduren zum Austausch von globalen Routinginformationen zu entwickeln.

Im Juli 1997 waren Organisationen aus 27 Ländern an das 6bone angeschlossen, zumeist Universitäten, Forschungseinrichtungen, Hard- und Softwarehersteller. Die Topologie des Testnetzes unterteilt sich in die drei Ebenen Backbone-Router, Durchgangsrouter und Blattrouter. Die Router des Backbones verwenden in der Regel Routing-Protokolle wie RIPng, IDRPv6 oder BGP4+IPv6 zum Austausch von Routinginformationen, während die Router an den Blättern logisch zumeist über statische IPv6-in-IPv4-Tunnels an die Backbone- oder Durchgangsrouter angeschlossen sind. Abbildung 4.2 gibt einen Überblick über die Topologie des 6bone im Juli 1997. In Abbildung 4.3 sind die Verbindungen zwischen den Backbone-Routern zu sehen.

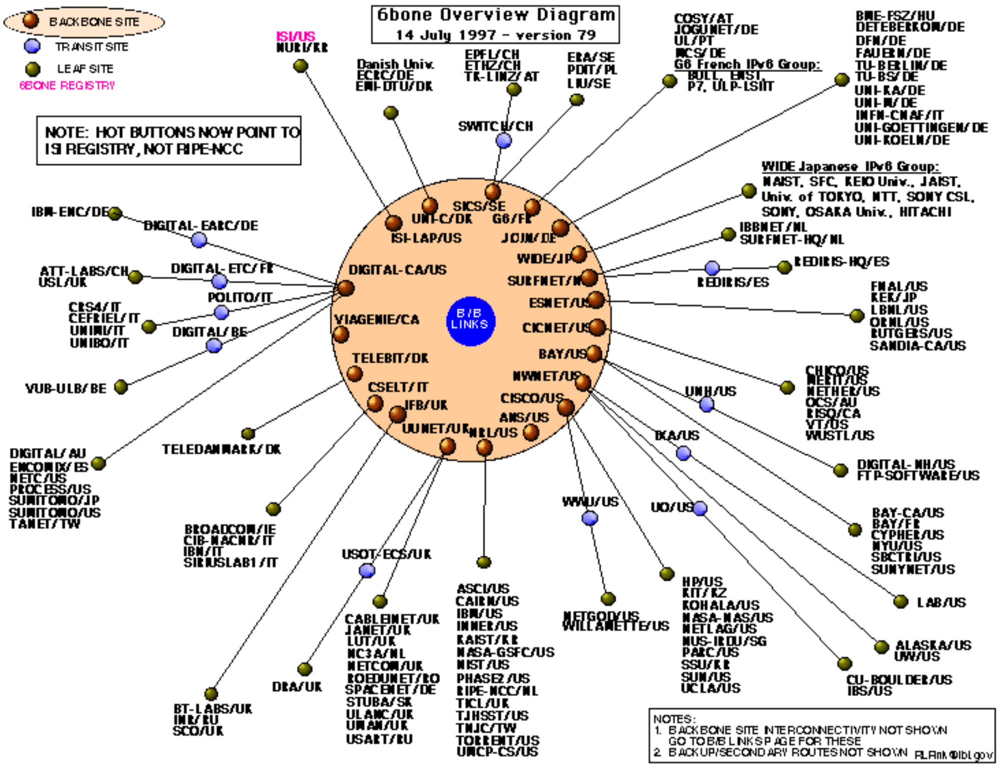

Abbildung 4.2: Das 6bone-Testnetz

4.2 Anmeldung der Teilnehmer

Um selbst am 6bone teilnehmen zu können, mußte zunächst ein geeigneter Anschlußpunkt gefunden werden. Da der Netzdiensteanbieter der Münchner Hochschulen, das Leibniz Rechenzentrum (LRZ), über den auch das Netz des Instituts für Informatik der Universität Zugang zum Internet hat, noch keine IPv6-fähigen Router betreibt, mußte ein IPv6-über-IPv4-Tunnel zu einem 6bone-Teilnehmer eingerichtet werden. Als Tunnelendpunkt bot sich die Universität Münster an, die im Rahmen des JOIN-Projektes (siehe [JOIN 97]), das die Einführung von

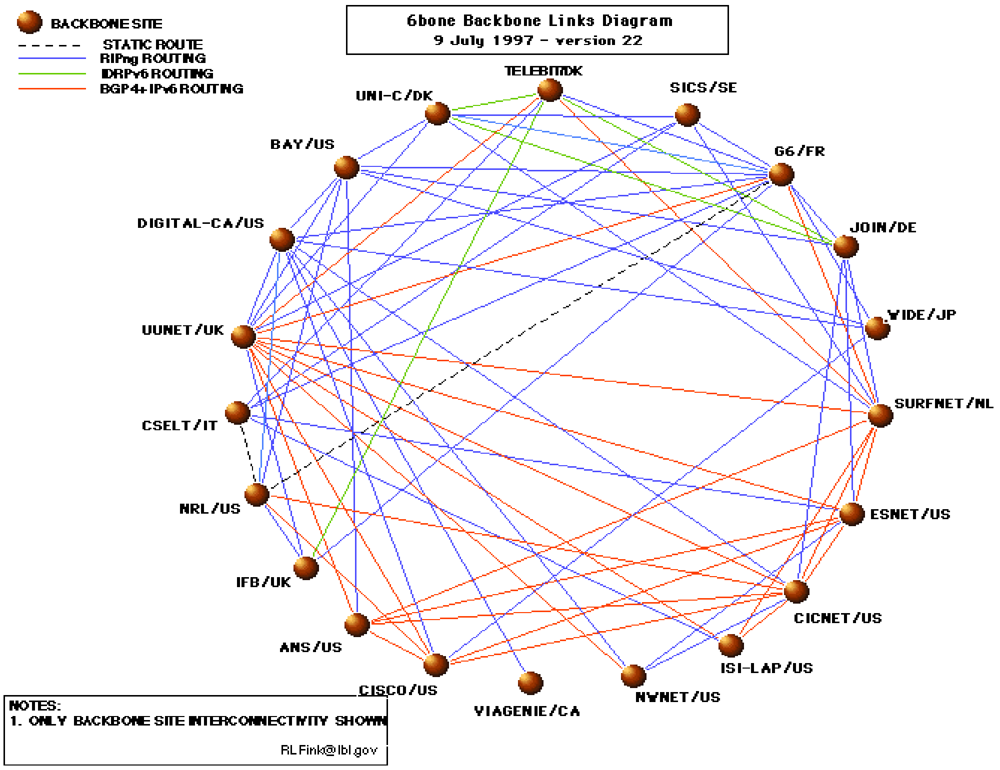

Abbildung 4.3: Die Backbone-Verbindungen des 6bone-Testnetzes

IPv6 im deutschen Forschungsnetz vorbereiten soll, IPv6-Backbone-Router betreibt.

Im 6bone werden IPv6-Adressen verwendet, die nach dem in [Hinden 96b] beschriebenen Plan berechnet wurden. Diese Adressen sind für Testzwecke bestimmt und nur temporär gültig. Das dort vorgestellte Adreßformat, das mit der provider-basierten Adreßaufteilung konsistent ist, zeigt Tabelle 4.1.

3 Bits	5 Bits	16 Bits	8 Bits	24 Bits	8 Bits	16 Bits	48 Bits
`010`	Registry-ID	Provider-ID	Res.	Subscriber-ID	Res.	Intra-Subscriber	
`010`	`11111`	ASN	`0`	IPv4-Net-Adr.	`0`	Subnet-Adr.	Intf.-ID

Tabelle 4.1: Format der Testadressen nach RFC 1897

Nach diesem Plan wurden die Adressen für die zu konfigurierenden Rechner *terra* und *merkur* ermittelt:

Präfix und Registry-ID: `01011111` ist hexadezimal `5f`

ASN: die Autonomous System Number des DFN ist 1275, hex.: `04fb`

Reserviert: `00`

IPv4-Netzwerk-Adresse: `129.187.214` ist hex.: `81bbd6`

Reserviert: `00`

Subnetz-Adresse: hier wurde `214` angegeben, also hex.: `00d6`

Interface-Adresse: Die 48 Bit langen IEEE-802 MAC-Adressen können hier verwendet werden:

terra: `08:00:09:8b:a8:d8`
merkur: `00:00:c0:cb:29:1c`

Dies alles hintereinandergeschrieben ergibt (in vereinfachter Schreibweise) die Adressen:

terra: `5f04:fb00:81bb:d600:d6:800:98b:a8d8`
merkur: `5f04:fb00:81bb:d600:d6:0:c0cb:291c`

Das 64 Bit lange Routing-Präfix für die Domain *6bone.informatik.uni-muenchen.de* ist `5f04:fb00:81bb:d600::/64`.

Die `IP6.INT.`-Subdomain für den Reverse-DNS-Eintrag ergibt sich als `0.0.6.d.b.b.1.8.0.0.b.f.4.0.f.5.IP6.INT.`

Als Tunnelendpunkt auf Münchner Seite wurde *pchegering2.nm.informatik.uni-muenchen.de* alias *terra.6bone.informatik.uni-muenchen.de* alias *tunnel.6bone.informatik.uni-muenchen.de* alias *ns.6bone.informatik.uni-muenchen.de* ausgewählt.

Dessen IPv4- und IPv6-Adresse sowie das Routing-Präfix, die `IP6.INT.`-Subdomain und der Primary Nameserver *ns.6bone.informatik.uni-muenchen.de* wurden den Mitarbeitern des JOIN-Projektes mitgeteilt, worauf diese den Tunnel aktivierten, die `IP6.INT.`-Subdomain an den Münchner Nameserver delegierten und in Münster einen Secondary Nameservice einrichteten.

Anschließend wurde ein Eintrag in die RIPE-Datenbank unter `ftp://ftp.ripe.net/ipv6/ip6rr/` als `UNI-M` in folgender Form vorgenommen:

```
site:           UNI-M, Institute of Computer Science, University
                of Munich
location:       Muenchen (Munich), Germany
loc-string:     48 08 23n 11 34 28e 530m
prefix:         5f04:fb00:81bb:d600::/64
ping:           5f04:fb00:81bb:d600:d6:800:98b:a8d8
tunnel:         129.187.214.42 128.176.191.66 JOIN
contact:        ipv6@nm.informatik.uni-muenchen.de
status:         operational since 24-Feb-1997
remark:         experimental
changed:        hoffmann@informatik.uni-muenchen.de 19970224
source:         RIPE
```

Diese Datenbank enthält Angaben über IP-Netzwerke in Europa, ihre AS-Nummern, Kontaktpersonen, Routing-Policies, etc., und soll Netzwerkadministratoren in ihrer Arbeit unterstützen. Die enthaltenen Daten sind per `ftp` oder das `whois`-Protokoll öffentlich zugänglich. Eine Beispielanfrage mit `whois`:

```
whois -h brind.isi.edu UNI-M
```

Diese FTP-basierte Registratur wurde im Verlauf der Diplomarbeit durch eine objektbasierte Datenbank ersetzt. Lesender Zugriff ist nun ausschließlich über das `whois`-Protokoll auf `whois.6bone.net` möglich. Schreibend wird mittels E-Mail an die automatische Mailbox `auto-dbm@ISI.EDU` zugegriffen.

In [de Groot 97] ist die Syntax dieses Datenbankobjektes beschrieben. Weitere Informationen sind unter [ISI 97] und [6bone 97] verfügbar.

Die Datenbank unterscheidet nun zwischen den Objekten *ipv6-site* und *person*. Letzteres dient zur Angabe einer Kontaktperson im *ipv6-site*-Objekt. Da bei der automatischen Konvertierung in die neue Form kein *person*-Objekt angelegt wurde, mußte zunächst ein solches durch eine E-Mail folgenden Inhalts an `auto-dbm@ISI.EDU` generiert werden:

```
person:       IPv6 Liste
address:      UNI-M
address:      Institute of Computer Science, University of Munich
address:      Oettingenstr. 67
address:      80538 Muenchen
address:      D
phone:        +49 89 21782168
e-mail:       ipv6@nm.informatik.uni-muenchen.de
nic-hdl:      AUTO-1
changed:      hoffmann@informatik.uni-muenchen.de 19970815
source:       6BONE
```

Durch die Zeile `nic-hdl: AUTO-1` wurde automatisch ein *NIC-Handle* mit dem Inhalt `IL1-6BONE` erzeugt.

Das *ipv6-site*-Objekt wurde zuletzt durch eine E-Mail mit dem folgenden Inhalt aktualisiert:

```
ipv6-site:    UNI-M
origin:       AS1275
descr:        UNI-M, Institute of Computer Science, University of Munich
descr:        Muenchen (Munich), Germany
location:     48 08 23 N 11 34 28 E 530m
country:      DE
prefix:       5F04:FB00:81BB:D600::/64
application: ping terra.6bone.informatik.uni-muenchen.de
application: ping merkur.6bone.informatik.uni-muenchen.de
tunnel:       IPv6 in IPv4 pchegering2.nm.informatik.uni-muenchen.de ->
              6bone.uni-muenster.de JOIN STATIC
contact:      IL1-6BONE
remarks:      experimental
remarks:      ipv6-site is operational since 24-Feb-1997
notify:       ipv6@nm.informatik.uni-muenchen.de
changed:      hoffmann@informatik.uni-muenchen.de 19970224
changed:      hoffmann@informatik.uni-muenchen.de 19970506
```

```
changed:     auto-dbm@ISI.EDU 19970608
changed:     hoffmann@informatik.uni-muenchen.de 19970623
changed:     hoffmann@informatik.uni-muenchen.de 19970709
changed:     hoffmann@informatik.uni-muenchen.de 19970815
source:      6BONE
```

Soll eine neue Kontaktperson vereinbart werden, ist zuerst ein neues *person*-Objekt nach obigem Schema anzulegen und danach das erzeugte *NIC-Handle* in das `contact:`-Feld des *ipv6-site*-Objekts einzusetzen.

Erfolgte Änderungen an den Objekten oder Fehler in den Angaben werden jeweils per E-Mail an die hinter `notify:` angegebene Adresse `ipv6@nm.informatik.uni-muenchen.de` gemeldet.

4.3 Verwendung in der Diplomarbeit

Der Anschluß des lokalen IPv6-Testnetzes an das 6bone diente in der Diplomarbeit dazu,

- Praxisbezug herzustellen,
- realistische Testbedingungen für das lokale IPv6-Testnetz zu schaffen,
- die Interoperabilität der Linux-IPv6-Implementierung mit anderen Implementierungen zu testen und
- eine Basis für zukünftige Versuche mit IPv6-Technologien am Institut zu schaffen.

Interoperabilitätstests beschränkten sich im wesentlichen auf Tests der Werkzeuge `ping`, `traceroute` und `ftp`. Mit der weiteren Entwicklung von Protokollspezifikationen und Implementierungen werden sich hier in nächster Zeit viele Möglichkeiten zu solchen Tests ergeben, etwa in den Bereichen Mobile Rechner, Real Audio und Routing-Protokolle.

Die Erreichbarkeit anderer 6bone-Teilnehmer war, entsprechend dem experimentellen Charakter des Netzes, wechselhaft. Einige Teilnehmer bieten über das WWW Erreichbarkeitsstatistiken an, gemessen mit `ping`, manchmal mit `traceroute`. Eine Liste solcher Statistiken ist z.B. unter [6bone 97] zu finden.

Hier ist ein `traceroute`-Lauf vom National Institute of Standards and Technology (NIST) zum Rechner *terra.6bone*:

```
           Traceroute Data from NIST to UNI-M

Traceroute to 5f04:fb00:81bb:d600:d6:800:98b:a8d8:
1   * * *
2  ipng9.ipng.nist.gov
     (::129.6.51.154)    3.091 ms *  3.088 ms
3  6bone-gw.ipv6.imag.fr
     (::129.88.26.1)    117.589 ms  109.420 ms  108.718 ms
4  6bone-core.ipv6.imag.fr
     (5f06:b500:8158:1a00:0:800:2bb9:f33d)  110.973 ms  115.862 ms\
```

```
                                                   117.082 ms
5  6bone.join.uni-muenster.de
     (5f04:fb00:80b0:0:bf42:c0:3302:14)    150.782 ms *  284.091 ms
6  terra.6bone.informatik.uni-muenchen.de
     (5f04:fb00:81bb:d600:d6:800:98b:a8d8)    168.747 ms *  217.907 ms
```

4.4 Zukunft des 6bone

In [Postel 97] wird ein Adreßvergabeplan vorgeschlagen, der mit dem neuen *aggregierbaren globalen Unicast-Adreßformat* (siehe Abschnitt 2.3.4) konsistent ist. In absehbarer Zeit wird es notwendig werden, die gegenwärtigen Adressen dementsprechend zu ändern.

Das 6bone wird nach Ansicht der IETF-Arbeitsgruppe IPngwg solange bestehen, bis IPv6-fähige Router weit genug entwickelt und verbreitet sind. Wenn das nur für frühe Test- und Entwicklungszwecke angelegte Netz aus Tunnels nicht mehr benötigt wird und Internetdiensteanbieter ihre Infrastruktur an die Anforderungen des neuen Internet-Protokolls anpassen, wird das 6bone schrittweise verschwinden.

Kapitel 5

IPv6-Managementanforderungen

Dieses Kapitel beschäftigt sich mit Netzmanagementaspekten, die in Zusammenhang mit der Migration zum Internet-Protokoll Version 6 und dem Betrieb eines Netzes, das dieses Protokoll verwendet, stehen.

Im Abschnitt 5.1 wird zunächst der Begriff „Management“ abgegrenzt und seine Bedeutung für Rechnernetze dargestellt. Die hier dargestellte Klassifizierungssystematik dient zur Gliederung dieses Kapitels.

Die Abschnitte 5.2, 5.3 und 5.4 bilden die Zeitachse des Managements. Die Abschnitte 5.3 und 5.4 behandeln außerdem dessen Funktionsbereiche.

Im Abschnitt 5.5 werden einige spezielle Netztypen genauer untersucht.

Standards zur Durchführung des Konfigurations-, Fehler- und Leistungsmanagements werden in Kapitel 6 vorgestellt.

5.1 Überblick

In [HeAb 93] wird Netz- bzw. Systemmanagement als die „Gesamtheit der Vorkehrungen und Aktivitäten zur Sicherstellung des effektiven und effizienten Einsatzes eines Rechnernetzes bzw. eines verteilten Systems“ definiert. Dazu gehören Verfahren sowie Hardware- und Softwarewerkzeuge zur Konfigurierung, Überwachung, Fehlerbeseitigung und Verwaltung des Netzes bzw. verteilten Systems. Managementaspekte eines verteilten Systems (Benutzerverwaltung, verteilte Dateisysteme, Rechenlastverteilung, etc.) sind für diese Arbeit nicht von Belang. Hier steht das Management von Ressourcen im Vordergrund, die der Kommunikation zwischen Rechensystemen dienen. Daher wird, entsprechend der Abgrenzung der Begriffe Netz- und Systemmanagement in [HeAb 93], im folgenden von Netzmanagement die Rede sein.

Management dient immer einem *Ziel*. Wegen der Breite des vorliegenden Themas wird das oben genannte allgemeine Ziel des Netzmanagements in den folgenden Abschnitten, die jeweils einzelne Aspekte wie DNS oder Routing behandeln, weiter verfeinert. Um dieses Ziel zu erreichen, muß man bestimmte *Aufgaben* erledigen, an die gewisse *Anforderungen* gestellt werden. Zur *Umsetzung* der Aufgaben stehen unterschiedliche Verfahren zur Verfügung, die sich verschiedener Werkzeuge bedienen.

Man kann den vielschichtigen Komplex Management in überschaubarere Einheiten zerlegen, z.B. in die folgenden *Dimensionen*:

Die zeitliche Dimension teilt den Managementprozeß in mehrere *Phasen* seines Lebenszyklusses: Planung, physikalische Erstinstallation, Erstkonfiguration, Betrieb und Änderung.

Die funktionale Dimension ordnet den *Funktionsbereichen* Konfiguration, Leistung, Fehler, Sicherheit und Abrechnung bestimmte Managementaufgaben zu.

Die Dimension der Szenarien unterscheidet verschiedene Management-*Disziplinen* nach ihren Zielobjekten: Netz, System, Anwendung, Dienst und Enterprise. Diese Arbeit beschäftigt sich nur mit dem Zielobjekt Netz.

Die Dimension der Anwendungsbereiche unterteilt sich in Daten, Sprache, Video, Multimedia und integrierte Dienste.

Die Dimension der Netztypen umfasst IN, VPN, Corporate Networks, WAN, MAN und LAN.

5.2 Planung

Nach [HeAb 93] ist die Aufgabe eines Netzplaners der iterative Prozess,

- die gegenwärtigen Netzcharakteristiken zu überwachen,
- an den Bereich des Netzes angrenzende Einschränkungen und Überlegungen zu verstehen,
- zukünftige Bedürfnisse und Technologieentwicklungen vorherzusehen,
- technische Möglichkeiten zu evaluieren,
- angemessene, konsistente und koordinierte Pläne auf lang-, mittel- und kurzfristiger Basis zu erarbeiten und
- Pläne aufgrund von aktuellen Implementierungen zu modifizieren,

um den Benutzern des Netzes preisgünstige und zeitgerechte Kommunikationsdienste anbieten zu können.

Netze sind demnach wichtige Subsysteme im Gesamtsystemkonzept einer Institution. Ihre Planung muß sich aber an den übergeordneten Unternehmenszielen orientieren und die übrigen DV-Versorgungsstrukturen und das technische Betriebskonzept für das Netz miteinbeziehen.

Dies läßt sich in mehrere Teilaufgaben zerlegen:

5.2.1 Anwendungsanalyse

Um zu erfahren, wozu ein Netz dienen soll, führt der Netzplaner eine Anwendungsanalyse durch. Sie beschreibt die qualitativen Informationsflüsse und die Anforderungen der Anwendungen, die von den Netzbenutzern betrieben werden.

Erwägt der Planer die Einführung des Internet-Protokolls Version 6 in ein bereits bestehenden Netz, muß er zur Bestandsaufnahme zunächst die folgenden Fragen beantworten:

- Welche Anwendungen werden wo eingesetzt?
- Auf welche Netzprotokolle sind diese Anwendungen angewiesen?
- Welche Protokolle sind in welchen Subnetzen im Einsatz?

Steht außerdem die Anschaffung neuer Anwendungen bzw. neuer Versionen von bereits im Einsatz befindlichen Anwendungen zur Debatte, sind zusätzlich Antworten auf diese Fragen zu finden:

- Ist die neue Software auf IPv6 angewiesen?
- Unterstützt die neue Software IPv6?
- Bietet die neue Software wichtige Leistungsmerkmale, die nur mit IPv6 zu nutzen sind?

5.2.2 Bedarfsschwerpunktanalyse

Ferner ist die räumliche Verteilung der Netzbenutzer, sowohl physikalisch als auch logisch, von Interesse. Der Planer muß feststellen, wo sich die Benutzer von IPv6 aufhalten, wie sie sich auf unterschiedliche Organisationseinheiten, Abteilungen, Gebäude, Produktionsstandorte, etc. verteilen, und wie diese Einheiten miteinander verbunden sind.

Ein Schwerpunkt liegt hier in der Analyse der Anordnung von Routern. Es muß geklärt werden, ob IPv6 während einer Übergangszeit durch größere IPv4-Topologien getunnelt werden soll, oder ob für bestimmte Verbindungen IPv6-fähige Router angeschafft werden müssen. Dabei ist zu berücksichtigen, daß das Routing beim Einsatz von Tunnels ohne größeren Administrationsaufwand nicht optimal sein kann und durch den zusätzlichen Overhead und durch mögliche Fragmentierung der Pakete auf den IPv4-Strecken Performanceverluste zu erwarten sind. Außerdem können IPv6-spezifische Dienste wie die Echtzeitbehandlung von Paketen von IPv4-Routern nicht erbracht werden.

Im Zusammenhang damit steht die Planung der Integration der IPv6-Dienste in das Domain Name System. Die Rechner sollen sinnvoll auf Domains und Subdomains aufgeteilt werden. Hier gibt es grundsätzlich zwei Möglichkeiten:

- Es werden eigene Subdomains gebildet, die nur IPv6-Adressen (AAAA-Records im DNS) enthalten. Dazu muß evtl. ein neuer, IPv6-fähiger Nameserver eingerichtet werden. Dies bietet sich etwa für Pilotprojekte an, oder wenn an bestehenden Nameservern, die keine AAAA-Records verwalten können, nicht ohne weiteres Änderungen vorgenommen werden können. Auf diese Weise wurde z.B. im institutseigenen IPv6-Testnetz verfahren (siehe Kapitel 4 und Anhang A.2.1).

- Die IPv6-fähigen Rechner werden voll in die bestehende Domainstruktur integeriert. Das bedeutet, daß nicht IPv6-fähige Nameserver erneuert werden müssen, um die zu den vorhandenen A-Records der IPv4-Adressen hinzukommenden AAAA-Records der IPv6-Adressen aufnehmen zu können. Für Rechner, in denen beide Protokollstacks implementiert sind und die über beide Protokolle kommunizieren können sollen, wird also für denselben Rechnernamen ein Eintrag je Recordtyp im DNS vorgenommen.

Weitere Managementaspekte des DNS werden in Abschnitt 5.3.1 diskutiert.

5.2.3 Bedarfsgrößenermittlung

Diese Überlegungen führen zur Bedarfsgrößenermittlung, mit der eine Aussage zu Informationstyp und Transaktionscharakteristik getroffen werden soll.

Informationstypen sind z.B. Texte, Binärdateien, Graphiken, digitalisierte Sprache, Audioübertragungen in niedriger und hoher Qualität, Videoübertragungen, etc. Diese stellen unterschiedliche Anforderungen an die Übertragungsbandbreite, die Echtzeitbehandlung und die Fehlerbehandlung. Tabelle 5.1 gibt einen Überblick darüber.

Informationstyp	Übertragungsrate	Echtzeit	Fehlerfreiheit
Textdialog	$10^1 - 10^2$ kbit/s	nein (2s)	ja
Graphikdialog	$10^1 - 10^3$ kbit/s	nein (2s)	ja
DFÜ	$10^1 - 10^5$ kbit/s	nein	ja
Sprache	$2.4 - 96$ kbit/s	ja	nein
Audio (geringe Qualität)	$2.4 - 96$ kbit/s	ja	nein
Audio (hohe Qualität)	$22 - 44$ kbit/s	ja	nein
Standbilder	10^3 kbit/s	nein	nein
Bewegtbilder	$10^4 - 10^5$ kbit/s	ja	nein

Tabelle 5.1: Informationstypen und ihre Anforderungen

Transaktionscharakteristik ist die mengenmäßige und zeitliche Verteilung der Interaktionen über das Netz, abhängig von Teilnehmeranzahl, Informationstypen und -mengen, etc. Ziel ist die Ermittlung der Verkehrslastanforderungen für die einzelnen Teilnehmer und das Netz. Dies kann durch Messung oder Schätzung geschehen.

Die von IPv6 gebotenen neuen Möglichkeiten wie Echtzeitübertragung, prioritätsgesteuertes Routing oder Bandbreitenreservierung spielen bei der Bedarfsgrößenermittlung eine Rolle. Sollen beispielsweise Videobilder in Echtzeit übertragen werden, müssen die Router dafür eingerichtet sein, die erforderliche Bandbreite für den gewünschten Zeitraum konstant zur Verfügung zu stellen. Das Tunneln über IPv4-Topologien verbietet sich in solchen Fällen, da IPv4-Router keine Echtzeitdienste anbieten.

5.2.4 Endgeräteanalyse

Die Analyse der Endgeräte beschränkt sich bei der Einführung von IPv6 auf die Feststellung von Typ, Anzahl und Anschlußort der vorhandenen IPv4-fähigen Geräte wie Hosts und Peripherie und den notwendigen IPv6-fähigen Neuanschaffungen. Dabei brauchen IPv4-fähige Geräte wie z.B. Drucker, die nur lokal mit anderen Geräten kommunizieren und aus zusätzlichen Leistungsmerkmalen von IPv6 keinen Nutzen ziehen, nicht ersetzt werden, solange in ihren Kommunikationspartnern beide Internet-Protokollstacks implementiert sind.

5.2.5 Analyse sonstiger Randbedingungen

Weitere Randbedingungen, die die Planung der Einführung von IPv6 beeinflussen können, sind:

- Investitionsschutz: Vorhandene nur IPv4-fähige Software soll evtl. weiterhin eingesetzt werden können.
- Technologische Entwicklung, Standardisierung: Wichtige Teilapekte des Internet-Protokolls Version 6 wie DNS-Erweiterungen, DHCPv6, Echtzeitübertragung, etc. befinden sich noch in der Entwicklung. Außerhalb von Forschungs- und Entwicklungsabteilungen ist der Einsatz des Protokolls evtl. zum jetzigen Zeitpunkt noch nicht sinnvoll.

5.2.6 Planung der Protokolleinführung

Ist dies alles geklärt, kann der Netzplaner mit der Planung der Protokolleinführung beginnen. Dies beinhaltet folgende Punkte:

- Überprüfen der Betriebsabläufe, Anpassung von Software an IPv6
- Planung der Einführungsschulung: bei Entwicklern von Netzsoftware, Bedienung neuer IPv6-Anwendungen
- Entwicklung eines HW/SW-Betriebskonzeptes: Managementwerkzeuge für die Verwaltung von DNS, DHCPv6, Routern, etc.
- Planung der Abnahme: Funktions-, Leistungsprüfung, Entwicklung von Testprogrammen
- Evtl. zunächst Pilotprojekt starten (z.B. Teilnahme am *6bone*)

5.3 Konfiguration

5.3.1 DNS

Planungsaspekte des DNS-Managements bei der Einführung von IPv6 in ein bestehendes Netz werden in Abschnitt 5.2.2 beleuchtet. Die Aufgaben des Netzmanagers in der Konfigurationsphase bestehen nach [Hauc 94] im

- Bereitstellen und Aktivieren von Nameservern und

- dem Anpassen der Nameserver- bzw. Zonenkonfiguration im Zuge von Aktualisierungen der Netzumgebung.

Beim Bereitstellen eines Nameservers muß festgelegt werden, welche Dienste dieser anbieten soll. Neben dem Service für die ihm zugeteilten Zonen, für das Loopback Interface und einem Cache Service kann der Nameserver Aufgaben des Sicherheitsmanagements übernehmen (siehe Abschnitt 5.4.3). Ferner muß die Zusammenarbeit von DNS und evtl. benutzten Mechanismen zur automatischen Konfiguration der Hosts gewährleistet sein.

[Rekhter 97b] schlägt einen Mechanismus für die Interaktion zwischen dem Dynamic Host Configuration Protocol (DHCP) und dem DNS vor, beschränkt sich dabei aber auf A- und PTR-Recordtypen. [Vixie 97] beschreibt eine neue UPDATE-Operation für DNS-Messages und [Eastlake 97b] einen Authentifizierungsmechanismus dazu.

Über die Realisierung dynamischer Updates des DNS bei Verwendung von zustandsloser Autokonfiguration von *Site-Local-* oder global gültigen Adressen liegen noch keine Untersuchungen vor.

[Harrington 97] schlägt ein eigenes Protokoll, das nicht das DNS benutzt, vor, um eine einfache Abbildung von automatisch konfigurierten *Link-Local*-Adressen auf Namen und umgekehrt anzubieten. Rechner, die daran teilnehmen wollen, können Anfragen an die Multicastgruppe `FF02::1:1` senden oder durch das Abhören dieser Adresse neue Namensangebote empfangen.

Der Entwurf schlägt ferner vor, die Namen, die *Link-Local*-Adressen zugeordnet sind, unter einer eigenen Domain, etwa *.link*, zu sammeln, damit nur lokal gültige Namen nicht mit global gültigen kollidieren.

Die Umsetzung dieses Entwurfs würde die Plug-und-Play-Eigenschaften von IPv6 um einen wesentlichen Punkt bereichern. Ohne automatische Namenszuordnung zu *Link-Local*-Adressen wäre die Benutzung dieser Adressen ein aussichtsloses Unterfangen. Ein manuell geführtes Adressen- und Namensregister wie `/etc/hosts` würde bald inkonsistent sein.

Bewertung

Die Eingabe von 16 Bytes langen Ziffern- und Buchstabenfolgen ist extrem fehleranfällig, ganz zu schweigen von den rückwärts zu lesenden, aus Nibbles bestehenden Reverse-Einträgen für die Adresse-zu-Name-Konvertierung. Für jede Art von IPv6-Adressen ist eine möglichst automatisierte Form der Konfiguration wünschenswert. Speziell für die Reverse-Einträge müssen geeignete Werkzeuge entwickelt werden.

5.3.2 Routing

Die Aufgaben beim Konfigurationsmanagement von IPv6-Routern und der Endsysteme im Zusammenhang mit dem Routing verringern sich gegenüber den in [HeAb 93] genannten um die Aufgaben, die Neighbor Discovery (Abschnitt 2.4) und Autokonfiguration (Abschnitt 2.5) übernehmen können: Die Konfiguration der Adresse(n) des/der Router in den Endsystemen. Dies entfällt sowohl bei der Erstkonfiguration als auch bei Änderungen.

Wenn nur selten Änderungen im Netz auftreten, können die Routing-Tabellen im Router auch manuell erstellt werden. Ansonsten empfiehlt sich der Einsatz von Routing-Protokollen wie RIPng, IDRPv6, BGP4+IPv6 oder OSPFv6. Auf die Konfigurationsaufgaben bei Verwendung dieser Protokolle wird hier nicht näher eingegangen, da die Entwicklung der Protokolle zum Teil noch nicht abgeschlossen ist und dies außerdem über den Rahmen dieser Arbeit hinausgehen würde.

In manchen Netzen ist der Einsatz von Filtern in den Routern erwünscht, um

- „Routing Loops“ und
- „Spoofing Attacks“ zu verhindern, oder einfach
- Pakete bestimmter Herkunft nicht ins Netz hereinzulassen.

Das Einstellen dieser Zugangsfilter entsprechend der Policy des Netzbetreibers gehört dann zu den Konfigurationsaufgaben.

5.4 Betrieb

Während des Betriebs eines IPv6-Netzes fallen, wie bei der Verwendung anderer Protokolle, verschiedene Aufgaben im Leistungs-, Fehler-, Sicherheits- und Abrechnungsmanagement an. Letzteres wird hier nicht weiter berücksichtigt, da IPv6 hier keine wesentlichen Neuerungen mit sich bringt.

5.4.1 Leistungsmanagement

Das Leistungsmanagement von IPv6-Routern umfaßt

- die Durchführung von Langzeitstatistiken und
- das Ermitteln optimaler Metriken,

um das Wegewahlverfahren optimieren zu können. Für diesen Zweck geeignete Managementobjekte stellt die IPv6-MIB zur Verfügung, die in Abschnitt 6.2.3 vorgestellt wird.

5.4.2 Fehlermanagement

Die Aufgaben des Fehlermanagements des Domain Name Systems sind:

- das Überwachen der laufenden Nameserverprozesse und
- die Korrektur der Konfiguration, falls Fehler aufgetreten sind.

Das Fehlermanagement von IPv6-Routern umfaßt die Überwachung

- des Netzzustands und der Erreichbarkeit der Zielnetze,
- der aktuellen Einträge in der Wegetabelle,

- des Belegungsgrads interner Puffer und der Auslastung des Prozessors und
- die entsprechenden Reaktionen darauf.

Managementobjekte, die die Erfüllung dieser Aufgaben unterstützen, werden in Kapitel 6 vorgestellt.

5.4.3 Sicherheitsmanagement

Verschiedene Eigenschaften von IPv6 erfordern ein Management von Schlüsseln zur Authentifizierung und Geheimhaltung. In [Eastlake 97a] werden DNS-Erweiterungen beschrieben, die die Distribution von öffentlichen Schlüsseln ermöglichen. Dies stellt zusätzliche Anforderungen an das Management des DNS.

Eine eingehende Untersuchung der Anforderungen an das IPv6-Sicherheitsmanagement könnte in weiterführenden Arbeiten vorgenommen werden.

5.5 Netztypen

5.5.1 Corporate Networks

Corporate Networks haben die Aufgabe, die Kommunikationsnetze verschiedener Organisationseinheiten wie Abteilungen, Tochtergesellschaften oder Produktionsstandorte eines Unternehmens so miteinander zu verbinden, daß die Ziele des Unternehmens erreicht werden können. Dabei haben die einzelnen Organisationseinheiten im allgemeinen unterschiedliche Bedürfnisse hinsichtlich der Art und des Umfangs der Kommunikation mit anderen Einheiten oder externen Partnern. Diese Kommunikationsbedürfnisse werden hauptsächlich von den jeweils eingesetzten Anwendungen und der beabsichtigten Nutzung der Informationen bestimmt.

Dieser Abschnitt analysiert den potentiellen Nutzen der Einführung des Internet-Protokolls Version 6 in unternehmensweite Netzwerke.

Analyse

Host-Kategorien Nach Y. Rekhter und B. Moskowitz in [Fleischman 94] kann man Hosts in Unternehmen in drei Kategorien einteilen:

- Hosts, die keinen Internetzugang benötigen,
- Hosts, die Zugang zu einer begrenzten Menge von Internetdiensten wie E-Mail, FTP und Netnews benötigen und
- Hosts, die unbegrenzten Zugang zum Internet brauchen.

Eine Produktionsabteilung etwa wird Zugriff auf Pläne benötigen, die in einer Entwicklungsabteilung erstellt wurden. Es dürfte aber kaum im Interesse des Unternehmens liegen, diese Pläne im globalen Internet öffentlich einsehbar zu machen.

Forschungsabteilungen brauchen evtl. die Möglichkeit, mit anderen Unternehmen oder Universitäten per E-Mail kommunizieren oder sonstige Daten austauschen zu können. Für Buchhaltungsabteilungen dagegen genügt vielleicht die E-Mail-Kommunikation innerhalb des Hauses.

Charakteristiken von Corporate Networks In [Fleischman 94] wird das Netz der Firma Boeing durch fünf Charakteristiken beschrieben, die als repräsentativ für andere große Unternehmensnetzwerke angesehen werden:

Verhältnis der Hosts mit Internetzugang zu Hosts ohne Internetzugang: Dieses Verhältnis liege je nach Unternehmen zwischen 1:1000 und 1:10000 oder mehr und zeige, daß nur ein geringer Anteil von Rechnern in Corporate Networks weltweit gültige Adressen benötige.

Router-zu-Host Verhältnis: Das Verhältnis liege hier typischerweise zwischen 100:1 und 600:1 oder mehr. Eine nur die Router betreffende Migration sei sehr viel einfacher durchzuführen als eine Migration aller Rechner.

Business Faktor: Rechner und Netze seien nur Werkzeuge, die von Anwendungen benutzt werden, um bestimmte Geschäftsziele zu erreichen. Benutzer kaufen Anwendungen, nicht Netztechnologie. Die Entscheidung zur Migration zu IPv6 hänge demnach davon ab, ob die benutzten Anwendungen auf IPv6 angewiesen seien.

Integrationsfaktor: In Unternehmensnetzen sei oft eine große Zahl unterschiedlicher Netzprotokollfamilien in Gebrauch. Die Einführung eines weiteren Protokolls würde zunächst nur mehr Arbeit und höhere Kosten verursachen.

Trägheitsfaktor: Es bestehe die natürliche Tendenz, das Internet Protokoll Version 4 weiterhin zu verwenden, u.a. aus folgenden Gründen:

- Abhängigkeiten der Anwendungen
- Widerstand gegen zusätzliche Protokollkomplexität
- Beschränktes Budget für die Anschaffung zusätzlicher Hard- und Software
- Zusätzliche Adreßmanagement- und Nameserverkosten
- Migrationskosten
- Support Kosten
- Schulungskosten

Gründe für eine Migration aus Unternehmenssicht [Fleischman 94] nennt vier Ursachen, die aus Unternehmenssicht für eine Migration sprechen und mögliche Reaktionen:

- *Viele externe Partner benutzen IPv6*: Kommunikation über Gateways
- *Das Internet benutzt nur noch IPv6*: Kommunikation über Gateways

- *Unternehmenswichtige Applikationen brauchen IPv6 (Echtzeitapplikationen, etc.)*: Upgrade des Netzes dort wo diese Funktionalität gebraucht wird

- *Oberes Management verlangt Migration*: Angemessene Reaktion

Zusammenfassung

Nach Meinung der Autoren hängt die Entscheidung zur Migration zu IPv6 davon ab, ob IPv6 für unternehmenswichtige Anwendungen essentiell ist. Falls die Migration dem Unternehmen nicht in signifikanter Weise zugute käme, sei sie wirtschaftlich nicht zu vertreten. Eine vollständige Umstellung aller Rechner auf IPv6 sei deshalb unrealistisch und die Kommunikationsmöglichkeit zwischen migrierten und nicht migrierten Rechner zwingend notwendig.

5.5.2 VLANs

Ein Virtual Local Area Network (VLAN) ist eine Spezialform eines LANs, das Arbeitsgruppen unterstützt. Durch softwarekonfigurierbare Switches wird erreicht, daß einzelne LAN-Segmente bestimmten Arbeitsgruppen zugeordnet werden können.

Nachteile dieses Verfahrens sind der Bedarf an einer speziellen Switchinfrastruktur und deren Administration auf den Schichten 2 und 3. [Boudec 97] zeigt, daß sich mit Hilfe des Internet-Protokolls Version 6 auf Schicht 3 ohne spezielle Hardware flexible Broadcastbereiche und damit Arbeitsgruppenzugehörigkeiten realisieren lassen. Die Features, die dabei benutzt werden, sind Multicast-Adressierung, mobile Rechner, DHCPv6, Authentifizierung und Verschlüsselung.

Wie in Abschnitt 2.3 dargestellt, gibt es bei IPv6 nicht wie bei IPv4 *eine* Broadcast-Adresse, sondern eine ganze Hierarchie von Multicastadressen (beginnend mit dem Präfix `FF`). Eine Arbeitsgruppe kann dadurch gebildet werden, indem eine dieser Multicastadressen allen zu der Arbeitsgruppe gehörenden Rechnern zugeteilt wird. Da ein Rechner auf mehrere Multicastadressen gleichzeitig hören kann, kann er auch mehreren Arbeitsgruppen gleichzeitig angehören.

Die Verwaltung der Arbeitsgruppen soll mit Hilfe des Dynamic Host Configuration Protocol (DHCPv6) erfolgen. [Boudec 97] schlägt dazu eine neue DHCP-Extension vor, die *DHCP Workgroup Address Extension*. Die Managementanforderungen von DHCP werden z.B. in [Rieg 96] beschrieben. Version 6 von DHCP befindet sich noch in der Entwicklung.

Der Entwurf diskutiert auch Verfahrensweisen bei mobilen Teilnehmern von Arbeitsgruppen, bezieht sich dabei aber auf einen Entwurf, der gemäß den Regeln für Internet-Drafts abgelaufen ist, weil er innerhalb eines halben Jahres nicht erneuert wurde. Daher wird dieser Aspekt hier nicht weiter verfolgt, sondern auf den nächsten Abschnitt verwiesen, in dem es speziell um mobile Rechner geht.

5.5.3 Mobile Rechner

Bereits für das Internet-Protokoll Version 4 wurde in [Perkins 96] ein Protokoll zur Unterstützung mobiler Rechner vorgestellt. Das Verfahren beruht im wesentlichen auf der Kapselung der Pakete auf dem Weg von einem Homeagent im Heimatnetz des mobilen Knoten zu dessen Care-Of Adresse im fremden Netz. Dieser Ansatz ist nicht ganz unproblematisch, wie z.B. Tests im Rahmen eines Fortgeschrittenenpraktikums [Hoff 96] gezeigt haben. So ist unter anderem die Wegewahl nicht optimal und Firewalls und die Sicherheitsvorkehrungen von Routern können die Verwendung von mobilen Rechnern in bestimmten Netzen unmöglich machen.

In [Perkins 97b] wird ein Protokoll vorgeschlagen, das auf dem Konzept der Trennung zwischen einem festen Identifikator, der eindeutig bezeichnet, *welcher* Knoten gemeint ist, und einer temporären Adresse, die angibt, *wo* sich der Knoten aufhält, beruht.

Die Protokolle der Transportschicht, TCP und UDP, benutzen den Identifikator, um einen Knoten zu spezifizieren. Im IPv6-Header wird, falls bekannt, die topologisch signifikante temporäre Adresse als Zieladresse eingetragen, der Identifikator wird in einem Destination Options Header mitgesandt. Dazu wird in den mit dem mobilen Knoten (MN) kommunizierenden Knoten und in einem primären Adreßauflöser im Heimatnetz des mobilen Knoten eine *Address Mapping Table* (AMT) geführt, die die gegenwärtig gültige Zuordnung von Identifikator zu temporärer Adresse enthält.

Protokollüberblick

Befindet sich der mobile Knoten im Heimatnetz, so ist seine IPv6-Adresse gleich seinem Identifikator. Wird er in einem fremden Netz angeschlossen, erhält er durch ein Protokoll wie DHCPv6 eine temporäre Adresse. Sein Identifikator bleibt dabei unverändert. Der MN sendet an seinen primären Adreßauflöser ein AMT Update Request Paket, das die ihm zugeordnete temporäre Adresse enthält. Diese Adresse trägt der Adreßauflöser nach einer Authentifizierung in seine AMT ein.

Erreicht ein an den mobilen Knoten gerichtetes Paket den Adreßauflöser im Heimatnetz, wird es durch Tunneln an den MN weitergeleitet, was den MN wiederum veranlaßt, einen AMT Update Request an den Knoten zu senden, von dem das Paket ursprünglich ausgesendet wurde. Dadurch lernt dieser die temporäre Adresse des MN kennen, trägt sie in seine AMT ein und kann von diesem Zeitpunkt an mit dem MN auf optimaler Route ohne Umweg über das Heimatnetz des MN und ohne Kapselung der Pakete kommunizieren.

Beginnt ein mobiler Knoten im fremden Netz die Konversation, setzt er seine temporäre Adresse als Quelladresse im IPv6-Header ein, seinen Identifikator (Heimatadresse) übermittelt er in der Quelladress-Option des Destination Options Header an den Zielknoten.

Zur Authentifizierung müssen die AMT Update Request Pakete mit Authentifizierungsheadern (siehe Abschnitt 2.2) versehen sein.

Managementanforderungen

Aus der Protokollbeschreibung lassen sich zwei Managementanforderungen ableiten: das Management der Authentifizierungsschlüssel, das in Abschnitt 5.4.3 erwähnt wird und das Management der an fremde mobile Rechner zu vergebenden temporären Adressen. Da dies in der Regel automatisch geschehen dürfte, sei hier auf die momentan noch nicht vorhandene Literatur zum in Entwicklung befindlichen Dynamic Host Configuration Protocol für IPv6 verwiesen.

Kapitel 6

Managementstandards

Einige der im letzten Kapitel besprochenen Managementaufgaben erfordern den Zugriff auf die dafür relevanten Verwaltungsinformationen. Im Fehler-, Leistungs- und Konfigurationsmanagement werden Statusinformationen über die verwalteten Ressourcen benötigt. Aufgetretene Fehler sollen gemeldet, protokolliert und behoben werden. Die Auslastung von Netzkomponenten und anderen Geräten soll überwacht und auf Überlastungen soll angemessen reagiert werden. Und das alles sollte mit Geräten unterschiedlicher Hersteller funktionieren und an einem Ort mit Hilfe von in einer Managementplattform integrierter Managementwerkzeuge zu erledigen sein.

Die meisten herstellerübergreifenden Lösungen basieren auf dem Internet-Managementkonzept, das sich wegen seiner Einfachheit und Implementierungsfreundlichkeit durchgesetzt hat. Dieses Konzept, das Internet Standard Network Management Framework, besteht aus drei Hauptkomponenten:

- einem Informationsmodell, das die Struktur der Management Information (SMI) definiert,
- der MIB-II (Management Information Base Version 2), dem Kernstück der Managementobjekte für Internetprotokolle und
- zwei Versionen des Simple Network Management Protocol (SNMP) zum Zugriff auf die Managementobjekte, beschrieben in [Schoffstall 90] (SNMPv1) und [Case 96c] (SNMPv2).

In diesem Kapitel werden das bestehende Informationsmodell und die Management Information Base, sowie die für IPv6 relevanten Änderungen beschrieben.

6.1 Das Informationsmodell (SMI)

Die Aufgabe des Informationsmodells besteht darin, geeignete Mechanismen zur Beschreibung und Benennung der im Agenten zu haltenden und über das Managementprotokoll abzufragenden Managementinformation vorzugeben. Diese Struktur der Managementinformation wird für SNMP Version 1 in [McCloghrie 90], für Version 2 in [Case 96a] festgelegt.

6.1.1 Das existierende Informationsmodell

Die Managementinformation wird in einer Baumstruktur, dem Internet-Registrierungsbaum angeordnet. Jeder Knoten dieses Baumes besteht aus einem Namen und einer Nummer. Den vollständigen, eindeutigen Namen eines Objektes erhält man durch Verfolgen des Pfades von der Wurzel des Baumes zum Objekt.

Die Managementinformation wird mit Hilfe einer Untermenge der Abstract Syntax Notation One (ASN.1) beschrieben. Diese ASN.1-Makros beschreiben folgende Aspekte:[1]

- den Namen und Objektidentifikator des Knotens,
- die Syntax der Managementinformation als ASN.1-Datentyp,
- Zugriffsinformation (not-accessible, read-only, read-write, read-create)
- Statusinformation (current, deprecated, obsolete),
- Erläuterung der enthaltenen Information und
- den Defaultwert der Objektvariablen.

Es gibt zwei Klassen von Knoten:

- Die *Strukturierungsknoten* dienen der Strukturierung der MIB. Ein Beispiel für die absolute Angabe der IP-Gruppe:

```
ip OBJECT IDENTIFIER ::=
    { iso(1) org(3) dod(6) internet(1) mgmt(2) mib-2(1) 4 }
```

- Die *Objekttypen* sind immer Blätter des Baumes und enthalten die eigentliche Managementinformation. Ein Beispiel aus der IP-Gruppe:

```
ipForwarding OBJECT-TYPE
  SYNTAX  INTEGER {
      forwarding(1),    -- acting as a router
      notForwarding(2)  -- NOT acting as a router
    }
  MAX-ACCESS  read-write
  STATUS      current
  DESCRIPTION
      "The indication of whether this entity is acting as an IP
      router in respect to the forwarding of datagrams received
      by, but not addressed to, this entity.  IP routers forward
      datagrams.  IP hosts do not (except those source-routed via
      the host)."
::= { ip 1 }
```

Abbildung 6.1 zeigt in einem Ausschnitt aus dem Registrierungsbaum den Pfad zum Managementobjekt `ipForwarding` mit dem Objektidentifikator `.1.3.6.1.2.1.4.1`.

[1] SMI Version 2

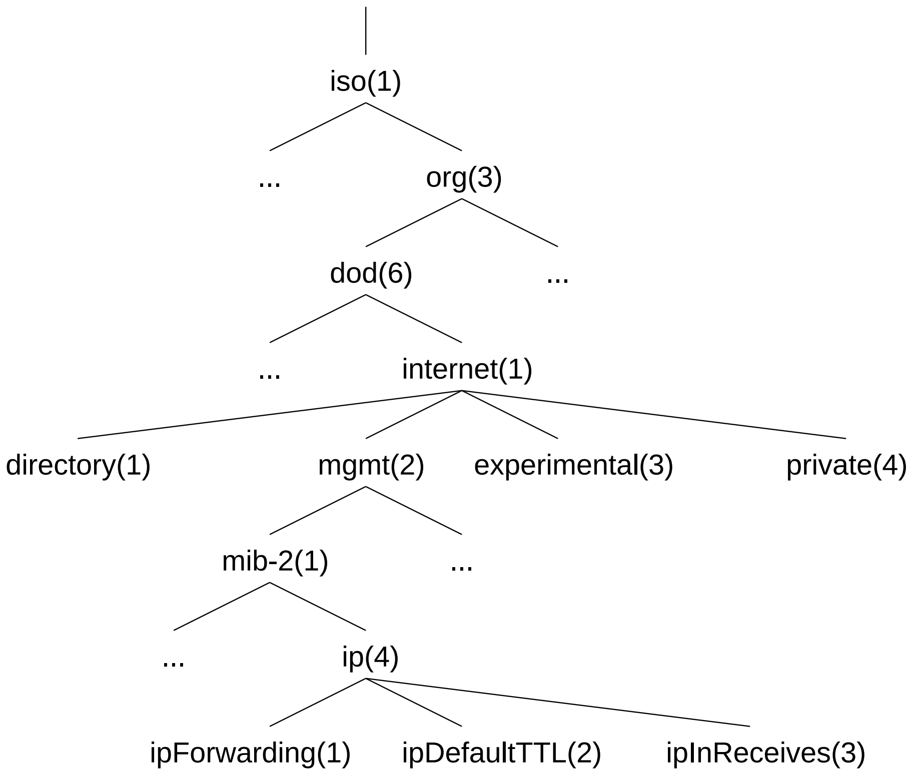

Abbildung 6.1: Ausschnitt aus dem Registrierungsbaum

Datentypen

- Das Informationsmodell für SNMPv1 stellt folgende Datentypen zur Beschreibung von Internetobjekten zur Verfügung:
  - primitive Datentypen: `INTEGER`, `OCTET STRING`, `OBJECT IDENTIFIER` und `NULL`
  - definierte Datentypen:

    NetworkAddress repräsentiert eine Auswahl aus mehreren Protokollfamilien. Derzeit steht nur die Internet Familie zur Auswahl.

    IpAddress dient zur Darstellung einer 32 Bit langen IPv4-Adresse und wird als `OCTET STRING` der Länge 4 in der Netzwerk Byteordnung repräsentiert.

    Counter stellt einen Zähler dar, der monoton bis zum Maximum von $2^{32} - 1$ wächst und danach wieder bei Null beginnt.

    Gauge ist ein Zähler, der hoch- und heruntergezählt und Werte zwischen 0 und $2^{32} - 1$ annehmen kann.

    TimeTicks ermöglicht die Beschreibung von Zeiträumen, wobei eine Einheit 1/100 Sekunde bedeutet.

    Opaque erlaubt die Angabe beliebiger ASN.1-Syntax.

  - zusammengesetzte Typen: `SEQUENCE <primitiver Datentyp>` oder `SEQUENCE OF <Listenkonstruktor>`; mit letzterem lassen sich Tabellen realisieren, wenn `<Listenkonstruktor>` eine `SEQUENCE <primitiver oder definierter Datentyp>` ist

- Das Informationsmodell für SNMPv2 fügt diesen die folgenden Typen hinzu:

  - einen weiteren primitiven Datentyp: `BITS`
  - und weitere definierte Datentypen:

    Integer32, Counter32, Gauge32 und Unsigned32
    entsprechen den von der SMI Version 1 bekannten Typen, benötigen aber nie mehr als 32 Bits zu ihrer Darstellung; Unsigned32 entspricht Gauge32

    Counter64 entspricht Counter und Counter32, benötigt nicht mehr als 64 Bits zur Darstellung und kann Werte zwischen 0 und $2^{64}-1$ annehmen

- Um eigene Typen, die Datentypen der SMI ähnlich sind, aber eine präzisere Semantik haben, einführen zu können, wird im SNMPv2-Framework das ASN.1 Makro `TEXTUAL-CONVENTION` verwendet. In [Case 96b] sind die grundlegenden textuellen Konventionen definiert. Zwei Beispiele daraus sind:

  - ```
    TruthValue ::= TEXTUAL-CONVENTION
      STATUS       current
      DESCRIPTION
        "Represents a boolean value."
      SYNTAX       INTEGER { true(1), false(2) }
    ```
  - ```
    MacAddress ::= TEXTUAL-CONVENTION
      DISPLAY-HINT "1x:"
      STATUS       current
      DESCRIPTION
        "Represents an 802 MAC address represented in the
        ‘canonical’ order defined by IEEE 802.1a, i.e., as if it
        were transmitted least significant bit first, even though
        802.5 (in contrast to other 802.x protocols) requires MAC
        addresses to be transmitted most significant bit first."
      SYNTAX       OCTET STRING (SIZE (6))
    ```

6.1.2 Änderungen am Informationsmodell

Das Informationsmodell für SNMPv2 bietet zur Repräsentation von IPv4-Adressen der Länge 32 Bit den Datentyp IpAddress an, definiert als OCTET STRING (SIZE (4)). Ein vergleichbarer Datentyp zur Darstellung einer 128 Bits langen IPv6-Adresse steht in diesem Informationsmodell nicht zur Verfügung.

Eine Möglichkeit wäre, SNMPv2 und dessen SMI um einen derartigen Datentyp zu erweitern. Eine andere wäre, diesen Datentyp erst in der Version 3 von SNMP, die derzeit diskutiert wird, einzubauen. Beide Lösungen haben den

Nachteil, daß die Benutzung der IPv6-MIBs erst nach Änderungen am Informationsmodell und am Protokoll möglich wäre. Bezieht man die Erfahrungen mit der schleppenden Verbreitung von SNMPv2 in die Überlegung mit ein, erscheint dies nicht sinnvoll.

Der Entwurf zur RSVP-MIB (siehe Abschnitt 6.2.9) repräsentiert IPv4- und IPv6-Adressen als `OCTET STRING (SIZE(4..16))`. Eine IPv6-Adresse benötigt 16 Bytes zu ihrer Darstellung, also könnte man den Typ `OCTET STRING (SIZE (16))` verwenden.

Eine praktische Lösung, die ohne Änderungen am Informationsmodell auskommt, ist die Vereinbarung des Datentyps der IPv6-Adresse als textuelle Konvention. Diesen Weg gehen auch die oben genannten Entwürfe für die IPv6-MIBs:

```
Ipv6Address ::= TEXTUAL-CONVENTION
     DISPLAY-HINT "2x:"
     STATUS       current
     DESCRIPTION
       "This data type is used to model IPv6 addresses.
        This is a binary string of 16 octets in network
        byte-order."
     SYNTAX       OCTET STRING (SIZE (16))
```

Weitere nützliche Typvereinbarungen werden für das Adreßpräfix und für das Adreßtoken gemacht:

```
Ipv6AddressPrefix ::= TEXTUAL-CONVENTION
     DISPLAY-HINT "2x:"
     STATUS       current
     DESCRIPTION
       "This data type is used to model IPv6 address
       prefixes. This is a binary string of up to 16
       octets in network byte-order."
     SYNTAX       OCTET STRING (SIZE (0..16))

Ipv6AddressToken ::= TEXTUAL-CONVENTION
     DISPLAY-HINT "2x:"
     STATUS       current
     DESCRIPTION
       "This data type is used to model IPv6 address
       tokens. This is a binary string of up to 8
       octets in network byte-order."
     SYNTAX      OCTET STRING (SIZE (0..8))
```

6.2 Die Management Information Base

Im Zusammenhang mit dem Internet-Protokoll Version 4 existieren verschiedene Managementstandards. Diese betreffen, der Einordnung des Internet-Protokolls entsprechend, Aspekte des Managements auf Schicht 3 und 4 des ISO-OSI-Architekturmodells. Dazu gehören die bereits genannte MIB-II und ihre Updates, die IP Forwarding Table MIB, die MIBs zu den Routing-Protokollen RIP-2 und BGP-4, zum Domain Name Service und die experimentelle MIB zum Resource Reservation Protocol (RSVP).

6.2.1 MIB-II

Die Standard Management Information Base (MIB) des Internets ist die MIB-II, veröffentlicht in [McCloghrie 91], Updates durch [McCloghrie 96a] und [Baker 97a] für die IP- und ICMP-Gruppe, [McCloghrie 96b] für die TCP-Gruppe und [McCloghrie 96c] für die UDP-Gruppe.

Die in der MIB-II enthaltenen Managementobjekte sind entweder für das Konfigurations-, Leistungs- oder das Fehlermanagement relevant (siehe auch die Abschnitte 5.3, 5.4.1 und 5.4.2). Die Objekte sind durch Einteilung in folgende Gruppen strukturiert:

System: allgemeine Konfigurationsinformation über den gesamten gemanagten Knoten

Interfaces: Schnittstelleninformation über die angeschlossenen Systeme

Address Translation: Information zur Adreßauflösung (wird nicht mehr verwendet)

IP, ICMP, TCP, UDP, EGP, SNMP: Information über die genannten Protokolle

Transmission: Information über spezifische Typen von Netzschnittstellen wie Ethernet, Token Ring, Loopback, etc.

6.2.2 IP Forwarding Table MIB

Die IP Forwarding Table MIB definiert Objekte, die das Management von Routen in IP-Netzen ermöglichen.

In [Baker 97a] wird das Objekt `ipRouteTable` aus der in [McCloghrie 91] definierten MIB-II, das bereits in [Baker 92] durch `ipForwardTable` ersetzt wurde, durch das Objekt `ipCidrRouteTable` ersetzt. Damit werden mehrfache nächste Hops, Policy Routing und Classless Inter-Domain Routing möglich.

Ferner gibt es ein Objekt `ipCidrRouteNumber`, das die Anzahl der gegenwärtig verwalteten Routen enthält.

6.2.3 IPv6-MIB

Das Internet-Protokolls Version 6 bietet, wie in Kapitel 2 dargestellt, einige neue Eigenschaften und Möglichkeiten, die sich auch in geänderten oder neuen Managementobjekten niederschlagen. So benötigt etwa eine IPv6-Adresse 128 Bits zu ihrer Darstellung.

Es existieren bereits Entwürfe für MIBs zu IPv6, diese sind [Haskin 97b] für die generelle IPv6-Gruppe, [Daniele 97a] für die TCP-Gruppe, [Daniele 97b] für die UDP-Gruppe und [Haskin 97a] für die ICMP-Gruppe.

Die Routing-MIB zu IPv6 ist im Entwurf für die generelle IPv6-Gruppe enthalten.

Eine MIB für IPv6 muß die für das Konfigurations-, Fehler- und Leistungsmanagement des Protokollbetriebs notwendigen Informationen bereitstellen. Sie muß die Änderungen im Vergleich zu IPv4 widerspiegeln und die neuen Leistungsmerkmale verfügbar machen.

Wie bei der Anwendungsentwicklung stand auch beim Design der IPv6-MIBs im Vordergrund, existierende IPv4-Implementierungen nicht verändern zu müssen. Stattdessen wurden nur dort Erweiterungen vorgenommen, wo dies unbedingt notwendig war.

Gemeinsam ist allen neuen Objekten der Entwürfe das vorangestellte Präfix `ipv6`.

IPv6-Gruppe

Die generelle *IPv6-Gruppe* [Haskin 97b] besteht aus sechs Tabellen:

ipv6IfTable enthält Informationen über die IPv6-Schnittstellen des Knotens:

Einige der Objekte sind gegenüber den entsprechenden Objekten der MIB-II aus [McCloghrie 91] bzw. [McCloghrie 94] umbenannt, haben aber etwa dieselbe Funktion, z.B. wurde `ifType` zu `ipv6IfLowerLayer` und `ifMtu` wurde zu `ipv6IfEffectiveMtu`.

Wegen der Autokonfigurationsmöglichkeiten von IPv6 gibt es in dieser Tabelle die neuen Objekttypen `ipv6IfToken` und `ipv6IfTokenLength`. Sie enthalten ein auf dem Link eindeutiges Adreßtoken bzw. seine Länge, das zusammen mit einem Präfix zu einer Schnittstellenadresse kombiniert werden kann.

`ipv6IfOperStatus`, das den gegenwärtigen Zustand der Schnittstelle anzeigt, kann einen Wert `tokenless(3)` annehmen, wenn der Schnittstelle kein Adreßtoken zugeordnet werden konnte, weil die *Duplicate Address Detection* (siehe Abschnitt 2.5) scheiterte.

Die Zählerobjekte der Schnittstellentabelle sind teilweise redundant zu den Zählern in der IP-Gruppe der MIB-II und daher nur in der folgenden `ipv6IfStatsTable` zu finden.

ipv6IfStatsTable liefert Statistiken zum Datenverkehr über die IPv6-Schnittstellen des Knotens:

Hier gibt es zusätzlich zu den Zählern in der MIB-II den Objekttyp `ipv6IfStatsInTooBigErrors`. Dieser Zähler gibt die Anzahl der Pakete wieder, die von Routern, die zu große IPv6-Pakete im Gegensatz zu IPv4-Paketen nicht fragmentieren, nicht weitergeleitet werden konnten (siehe auch Abschnitt 2.2).

ipv6AddrPrefixTable enthält Informationen über die Adreßpräfixe, die mit den IPv6-Schnittstellen verbunden sind:

Diese Tabelle hat keine Entsprechung in der MIB-II. Zu jedem eingetragenen Präfix enthält sie Informationen über die Dauer seiner Gültigkeit und ob es zur Autokonfiguration verwendet werden darf.

ipv6AddrTable enthält Adressierungsinformationen für die IPv6-Schnittstellen des Knotens:

Diese Tabelle entspricht der `ipAddrTable` der MIB-II, enthält aber zusätzliche Informationen über die Art, durch die die IPv6-Adressen automatisch konfiguriert wurden, darüber, ob die Adressen Anycast-Adressen oder andere sind und über den Status der Adresse (`preferred(1)`, `deprecated(2)`, `invalid(3)`, `inaccessible(4)`, `unknown(5)`).

Die Zuordnung der Adressen zu den Schnittstellen erfolgt durch `INDEX { ipv6IfIndex, ipv6AddrAddress }`, da eine Schnittstelle mit mehreren IPv6-Adressen konfiguriert sein kann.

ipv6RouteTable enthält je gültiger Unicast-Route einen Eintrag:

Da die Wegewahl eines IPv6-Pakets durch Vergleich von dessen Zieladresse mit den Routing-Präfixen vorgenommen wird, enthält die Tabelle statt einer `ip(Cidr)RouteMask` eine `ipv6RoutePfxLength`.

`ipv6RoutePolicy` entscheidet bei mehreren unterschiedlichen Routen zum selben Ziel über den für einen bestimmten Wert des Prioritätsfeldes im IPv6-Header (siehe Abschnitt 2.1) zu wählenden Weg:

```
ipv6RoutePolicy OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
     "The general set of conditions that would cause the
     selection of one multipath route (set of next hops
     for a given destination) is referred to as 'policy'.
     Unless the mechanism indicated by ipv6RouteProtocol
     specified otherwise, the policy specifier is the
     Priority field of the IPv6 packet header.  The
     encoding of IPv6 Priority is specified by
     the following convention:

       0 - uncharacterized traffic
       1 - 'filler' traffic (e.g., netnews)
       2 - unattended data transfer (e.g., email)
       3 - reserved
       4 - attended bulk transfer (e.g., FTP, NFS)
       5 - reserved
       6 - interactive traffic (e.g., telnet, X)
       7 - internet control traffic (e.g., routing
           protocols, SNMP)

     Protocols defining 'policy' otherwise must either
     define a set of values which are valid for
     this  object  or  must  implement  an  integer-
     instanced  policy table for which this object's
     value acts as an index."
    ::= { ipv6RouteEntry 8 }
```

In `ipv6RouteProtocol` werden IPv6-fähige Routing-Protokolle wie RIPng und IDRP berücksichtigt.

ipv6NetToMediaTable enthält die Übersetzungen von IPv6- nach physikalischen Adressen:

Diese Tabelle entspricht der `atTable` aus der MIB-II, enthält jedoch zusätzliche Informationen zum Typ der Zuordnung (dynamisch, statisch) der IPv6-Adresse zur physischen Medienadresse und zur Erreichbarkeit der Schnittstelle. Außerdem läßt sich die Verbindung der beiden Adressen durch Setzen von `ipv6NetToMediaValid` auf `false(2)` trennen.

TCP- und UDP-Gruppen

Da die Verwendung von IPv6 statt IPv4 für eine TCP- oder UDP-Implementierung weitgehend unsichtbar bleibt, können auch die meisten Objekte der *TCP- und UDP-Gruppen* ([Daniele 97a] und [Daniele 97b]) weiterverwendet werden. Dies trifft auf alle Zähler zu, was bedeutet, daß Implementierungen dieselben Zähler für IPv6 und IPv4 verwenden können.

Eine Ausnahme bilden `tcpConnTable` und `udpTable` für Verbindungen zwischen bzw. zu IPv6-Endpunkten, da dort IPv6-Adressen sichtbar werden. Außerdem muß ein Schnittstellenindex in die Tabelleneinträge eingebaut werden, da die 4-Tupel (`ipv6TcpConnLocalAddress`, `ipv6TcpConnLocalPort`, `ipv6TcpConnRemAddress`, `ipv6TcpConnRemPort`) bzw. die 2-Tupel(`ipv6UdpLocalAddress`, `ipv6UdpLocalPort`) nicht unbedingt eindeutig sein müssen:

```
ipv6TcpConnEntry OBJECT-TYPE
   SYNTAX      Ipv6TcpConnEntry
   MAX-ACCESS  not-accessible
   STATUS      current
   DESCRIPTION
    "A conceptual row of the ipv6TcpConnTable containing
    information about a particular current TCP connection.
    Each row of this table is transient, in that it ceases to
    exist when (or soon after) the connection makes the transition
    to the CLOSED state.

    Note that conceptual rows in this table require an additional
    index object compared to tcpConnTable, since IPv6 addresses
    are not guaranteed to be unique on the managed node."
   INDEX   { ipv6TcpConnLocalAddress,
             ipv6TcpConnLocalPort,
             ipv6TcpConnRemAddress,
             ipv6TcpConnRemPort,
             ipv6TcpConnIfIndex }
   ::= { ipv6TcpConnTable 1 }
```

ICMP-Gruppe

Die *ICMP-Gruppe* [Haskin 97a] bietet nun pro Schnittstelle eine ICMPv6 Statistik Tabelle.

Neu sind die Zähler der neuen ICMPv6-Mitteilungen (siehe [Conta 96] und [Narten 96]):

- `ipv6IfIcmpInPktTooBigs` zählt die empfangenen „Packet Too Big Messages“
- `ipv6IfIcmpInRouterSolicits` zählt die empfangenen „Router Solicitation Messages“
- `ipv6IfIcmpInRouterAdvertisements` zählt die empfangenen „Router Advertisement Messages“
- `ipv6IfIcmpInNeighborSolicits` zählt die empfangenen „Neighbor Solicitation Messages“
- `ipv6IfIcmpInNeighborAdvertisements` zählt die empfangenen „Neighbor Advertisement Messages“
- `ipv6IfIcmpInGroupMembQueries`, `ipv6IfIcmpInGroupMembResponses` und `ipv6IfIcmpInGroupMembReductions` zählen die jeweiligen empfangenen „Group Membership Messages“

Die entsprechenden ... Out...-Zähler zählen die gesendeten Mitteilungen.

6.2.4 RIP-2 MIB

[Malkin 94] definiert Objekte zum Management des *Routing Information Protocols Version 2*. Diese Objekte umfassen

- die globalen Zähler `rip2GlobalRouteChanges` und `rip2GlobalQueries`,
- schnittstellenspezifische Statistiken in der `rip2IfStatTable`,
- schnittstellenspezifische Konfigurationsinformationen in der `rip2IfConfTable` und
- eine Peer Tabelle `rip2PeerTable`, die Angaben zu aktiven Router-Partnern enthält.

Sie können bei der Fehlersuche von Nutzen sein und erlauben die Konfiguration verschiedener Parameter.

6.2.5 BGP-4-MIB

[Burruss 96] ist ein Update zu [Willis 94] und definiert Managementobjekte zur Konfiguration und Fehlerbehandlung einer Implementierung des *Border Gateway Protocols Version 4*. Die MIB ist aufgeteilt in

- die Skalare `bgpVersion` zur Angabe der unterstützten Protokollversionen, `bgpLocalAs`, der die lokale ASN wiedergibt und `bgpIdentifier`, der die ID des lokalen Systems enthält,
- die BGP Peer Tabelle `bgpPeerTable`, die Informationen über Zustand und Aktivität von Verbindungen mit BGP-Partnern enthält,

- die BGP Received Path Attribute Tabelle `bgpRcvdPathAttrTable`, die empfangene Attribute von Partnern enthält, die BGP Version 3 oder niedriger einsetzen und
- die BGP-4 Received Path Attribute Tabelle `bgp4PathAttrTable`, die empfangene Attribute von Partnern enthält, die BGP Version 4 einsetzen.

6.2.6 MIBs für modifizierte Routing-Protokolle

Die MIBs für die modifizierten Routing-Protokolle RIPng, BGP-4+, IDRPv2 und OSPF für IPv6 müssen an die IPv6-Anforderungen angepaßt werden. Betroffen von Syntaxänderungen sind hier alle Objekte, die IP-Adressen, Schnittstelleninformationen, Routing- und Routing-Präfixinformationen enthalten.

In IPv6 neu hinzugekommene Leistungsmerkmale wie das Routing abhängig von Prioritätsangaben in den Paket-Headern muß sich auch in den Managementobjekten der modifizierten Routing-Protokollen widerspiegeln. Für die Reservierung von Bandbreiten in den Routern wird es nötig sein, daß die Routing-Protokolle auch Informationen darüber austauschen, welche Bandbreiten auf den einzelnen Wegen zur Verfügung stehen. Diese Informationen sollten auch in den MIBs erscheinen.

6.2.7 Existierende DNS-MIBs

Die für Nameserver relevanten Managementobjekte sind in [Austein 94a], die für Resolver in [Austein 94b] definiert.

- Die Server-Objekte wurden in folgende Gruppen aufgeteilt:

  Configuration: ermöglicht das Lesen und Schreiben eines Parameters zur Rekursion und das Rücksetzen des Servers,

  Counter: stellt verschiedene Zähler zur Verfügung, die z.B. die Anzahlen der erfolgreich beantworteten, nicht beantworteten oder an andere Server weitergeleiteten Anfragen enthalten,

  Optional Counter: stellt weitere Zähler für Systeme zur Verfügung, die nach der Herkunft der Anfragen unterscheiden (Anfrage vom Nameserver Host selbst, von Hosts aus einer in einer Zugriffsliste definierten Gruppe, andere Hosts) und

  Zone: Diese Gruppe enthält in der Tabelle `dnsServZoneTable` Informationen zur Zonenkonfiguration.

- Die Resolver-Objekte sind in folgende Gruppen aufgeteilt:

  Configuration: enthält Angaben über die Art der Resolution (rekursiv, iterativ), CNAME-Limits, eine Tabelle `dnsResConfigSbeltTable` mit Einträgen zu Nameservern, die benutzt wird, falls der Resolver nicht vollständig konfiguriert ist und ermöglicht das Rücksetzen des Resolvers,

  Counter: enthält verschiedene Zähler, ähnlich denen des Servers,

Lame Delegation: stellt Informationen über „lahme Delegationen“ zur Verfügung.

Cache: Die Tabelle `dnsResCacheRRTable` enthält Informationen über momentan im Cache befindliche Resource Records.

Negative Cache: Enthält Angaben und Einstellmöglichkeiten über das Caching autoritativer Fehler.

Optional Counter: Enthält weitere Zähler verschiedener Arten von Anfragen und Antworten.

6.2.8 Änderungen an den DNS-MIBs

Recordtypen tauchen in den MIBs nur als Integer auf, deren Bedeutung in DNS-Spezifikationen oder -erweiterungen festgelegt werden kann. Daher sind wegen den neuen Recordtypen keine Änderungen an den beiden MIBs nötig.

In Objekten, in denen der Datentyp `IpAddress` verwendet wird, könnte dieser durch `Ipv6Address` ersetzt werden. Enthält das Objekt eine IPv4-Adresse, könnte sie dann als IPv6-mapped IPv4-Adresse dargestellt werden.

6.2.9 RSVP-MIB

Mit [Baker 97b] liegt ein Entwurf einer Management Information Base für das *Resource Reservation Protocol* vor. Die Objekte wurden folgendermaßen aufgeteilt:

Session Statistics Table: stellt Statistiken zur Anzahl der Sender und Empfänger je Sitzung bereit,

Session Sender Table: enthält Informationen über die gültigen PATH Mitteilungen, die der Knoten empfängt,

Reservation Requests Received Table: enthält Informationen über die empfangenen gültigen RESV Mitteilungen,

Reservation Requests Forwarded Table: enthält eine Liste der gültigen RESV Mitteilungen, die der Knoten an seine stromaufwärts gelegenen Nachbarn schickt,

RSVP Interface Attributes Table: enthält RSVP-spezifische Informationen zu den Schnittstellen,

RSVP Neighbor Table: enthält eine Liste der Nachbarn, von denen der RSVP Prozeß momentan Mitteilungen empfängt.

Das Internet-Protokoll Version 6 wird in dem Entwurf bereits berücksichtigt. So enthalten die ersten vier Tabellen Angaben über den Typ der Session, z.B. das Objekt `rsvpResvType` der Reservation Requests Received Table:

```
rsvpResvType OBJECT-TYPE
    SYNTAX      SessionType
    MAX-ACCESS  read-create
```

```
    STATUS      current
    DESCRIPTION
       "The type of session (IP4, IP6, IP6  with  flow
       information, etc)."
   ::= { rsvpResvEntry 2 }
```

Die Tabellen Session Sender, Reservation Requests Received und Reservation Requests Forwarded enthalten Objekte mit der Flow ID der IPv6-Session, z.B.:

```
rsvpSenderFlowId OBJECT-TYPE
    SYNTAX      INTEGER (0..16777215)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "The flow ID that  this  sender  is  using,  if
       this  is  an IPv6 session."
   ::= { rsvpSenderEntry 10 }
```

Die ersten vier Tabellen enthalten ferner Elemente, die das verwendete Internet-Protokoll wiedergeben, z.B.:

```
rsvpResvFwdProtocol OBJECT-TYPE
    SYNTAX      Protocol
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "The IP Protocol used by a session. for  secure
       sessions, this indicates IP Security.  This ob-
       ject may not be changed when the value  of  the
       RowStatus object is 'active'."
   ::= { rsvpResvFwdEntry 7 }
```

IP-Adressen werden in diesem Entwurf mit `OCTET STRING (SIZE(4..16))` durch eine Zeichenkette variabler Länge dargestellt, die sowohl IPv4-Adressen von 4 Bytes Länge aufnehmen kann, als auch IPv6-Adressen von 16 Bytes Länge.

6.3 Bewertung

Die Entscheidung, für die Darstellung von IPv6-Adressen keinen neuen Datentyp einzuführen, sondern, wie in der IPv6-MIB, dafür eine textuelle Konvention oder, wie in der RSVP-MIB, die Darstellung als `OCTET STRING` zu verwenden, kommt einer schnelleren Verbreitung der neuen MIBs zugute. In Weiterentwicklungen der SMI und von SNMP sollten die IPv6-spezifischen Datentypen jedoch berücksichtigt werden.

Die Eigenschaft der IPv6-TCP- und UDP-Gruppen, jeweils separate Tabellen für IPv6-Verbindungen zu definieren, alle übrigen Zähler aber aus der MIB-II zu beziehen, ist zwar für eine Übergangszeit eine praktische Lösung, reißt aber zusammengehörende Objekte auseinander. Hier sollte auf Dauer eine Integration der IPv4- und der IPv6-spezifischen Objekte in jeweils eine einzige TCP- bzw. UDP-Gruppe erfolgen.

Kapitel 7

Zusammenfassung

Die Diplomarbeit befaßte sich mit Migrationsverfahren vom Internet-Protokoll Version 4 hin zu Version 6 und arbeitete die Managementanforderungen bei der Migration und im Betrieb von IPv6-Netzen heraus.

Um die von der IETF vorgeschlagenen Migrationsmechanismen zu veranschaulichen, wurden mehrere typische Migrationsszenarios entwickelt. Diese Szenarios können, in Verbindung mit technischer Dokumentation der zu migrierenden Systeme, als Leitfaden bei der Umstellung einzelner Knoten und Subnetze dienen.

Anhand der Entwicklung und Portierung praktischer Beispiele wurde ferner untersucht, wie sich die Migration auf die Anwendungsentwicklung auswirkt. Dabei wurde festgestellt, daß es vorteilhaft ist, mit Hilfe dafür vorgesehener Mechanismen Anwendungen möglichst protokollunabhängig zu gestalten.

Am Institut wurde ein IPv6-Testnetz eingerichtet und mit dem weltweiten IPv6-Testnetz *6bone* verbunden. Dabei gewonnene Erfahrungen zeigen, daß für den sicheren und effizienten Umgang mit dem neuen Protokoll z.B. im Zusammenhang mit der Konfiguration von DNS und Routing weitestgehende Automation erwünscht und zum Teil auch bereits vorhanden ist. Des weiteren ist die Entwicklung und der Einsatz von geeigneten Managementwerkzeugen notwendig.

Bei der Untersuchung der Managementanforderungen wurde für die Phase der Planung zur Einführung von IPv6 festgestellt, daß die im Netz zu benutzenden Anwendungen und der wirtschaftliche Nutzen die zentralen Entscheidungskriterien für oder gegen die Protokolleinführung und bei der Wahl der Netztopologie sind.

Für die Phasen Konfiguration und Betrieb ergibt sich die Notwendigkeit der Entwicklung neuer oder der Erweiterung bestehender Managementstandards. Der Anfang wurde mit den IPv6- und RSVP-MIB-Modulen bereits gemacht, die Entwicklung von MIB-Modulen für das Management der Routing-Protokolle und des DNS steht noch aus.

Kapitel 8

Ausblick

Die Entwicklung des Internet-Protokolls Version 6 und damit zusammenhängender Protokolle und Verfahren ist noch nicht abgeschlossen.

- Mehrere IPv6-Zusatzprotokolle befinden sich erst im Entwurfsstadium. Dies trifft z.B. auf das Dynamic Host Configuration Protocol für IPv6, die Protokolle für dynamische Updates des Domain Name Systems, zur Header Translation, zur Unterstützung mobiler Rechner und virtueller LANs zu.

- Es gibt noch kaum Erfahrungen mit der Implementierung von Anwendungen, die die neuen Eigenschaften von IPv6, wie z.B. das prioritätsgesteuerte Routing, die Datenübertragung in Echtzeit oder die Übertragung von authentifizierten oder verschlüsselten Paketen nutzen.

- Es fehlen für die Managementanforderungen des neuen Protokolls geeignete Managementwerkzeuge, sowie die Management Information Bases für die für IPv6 modifizierten Routing-Protokolle und das für IPv6 und das Schlüsselmanagement erweiterte Domain Name System. Außerdem sind noch keine Implementierungen der neuen MIBs in Endsystemen bekannt.

Die genannten Punkte bieten vielfältige weiterführende Forschungsgelegenheiten. Das im Rahmen der Diplomarbeit eingerichtete IPv6-Testnetz kann dabei für praktische Projekte im Zusammenhang mit IPv6 verwendet werden. Versuche können z.B. zu den Themenbereichen

- Autokonfiguration, Schlüsselmanagement, DHCPv6,
- Echtzeitanwendungen, RSVP, prioritätsgesteuertes Routing,
- Mobile Rechner,
- Performance-Messungen,
- Entwicklung von MIBs für Routing-Protokolle,
- MIB-Implementierungen auf Linux-PCs oder
- Entwicklung von Managementanwendungen

unternommen werden.

Anhang A

Installation des IPv6-Testnetzes

In diesem Anhang wird die Einrichtung der am IPv6-Testnetz des Instituts beteiligten Rechner beschrieben. Im ersten Teil geht es um die Installation der Betriebssystemsoftware und die zum Testbetrieb notwendigen Änderungen daran. Der zweite Teil behandelt speziell die Installation und Konfiguration der IPv6-Software.

Eine Abbildung des lokalen Testnetzes und die Beschreibung seines Anschlusses an das 6bone ist in Kapitel 4 zu finden.

Die Namen *pchegering2* und *pchegering8* bzw. *terra* und *merkur* werden im folgenden je nach Kontext verwendet, erstere, wenn von IPv4, letztere, wenn von IPv6 die Rede ist.

A.1 Installation der Debian-Distribution

Auf beiden Rechnern ist das Betriebssystem Linux in der Kernelversion 2.1.43 installiert. Die Basis bildet die Debian-Distribution Version 1.3. Dabei bootet *pchegering8* nur den Betriebssystemkern von der kleinen lokalen Festplatte, das Dateisystem wird komplett per NFS von *pchegering2* bezogen.

A.1.1 Der Rechner *pchegering2*

Zunächst wurde zusätzlich zur bereits vorhandenen Festplatte, auf der DOS und eine Slackware-Linux-Distribution installiert sind, eine zweite Platte eingebaut. Diese wurde in die drei Partitionen:

- sdb1, gemountet unter /mnt/sdb1, für eigene Installationen,
- sdb2, der Swap-Partition und
- sdb3, gemountet unter /, für die Debian-Installation

aufgeteilt.

Die Installation der Debian-Distribution erfolgte weitgehend wie von deren Installationsskript vorgesehen. Notwendige Änderungen sind unten aufgeführt.

Nachdem auf sdb3 das Basissystem von den zuvor erstellten Disketten eingerichtet war, konnte man mit dem Befehl `dselect` die gewünschten Pakete auswählen. Als Installationsquelle wurde zuerst *ftp.uni-erlangen.de*, ein Mirror von *ftp.debian.org*, verwendet. Der Mirror scheint jedoch zeitweise inkonsistent zu sein, so daß für Updates der Originalserver *ftp.debian.org* angegeben wurde.

Auf Updates durch „unstable"-Pakete, neue Softwareversionen, die von den Debian-Entwicklern noch nicht als stabil angesehen werden, wurde nach einigen daraus resultierenden Inkonsistenzen verzichtet.

Änderungen am installierten Debian-System

Um ein lauffähiges System zu erhalten und Software, die nicht in der Debian-Distribution enthalten ist, kompilieren, installieren und verwenden zu können, waren einige Änderungen am Debian-System notwendig,

1. Die Betriebssystemkerne („Kernel") können aus ungeklärten Gründen nicht von der root-Partition des Debian-Systems gestartet werden. Der „Linux-Lader" LILO weigert sich, diese von der zweiten Platte zu laden.

   Die startbaren Kernel sind deshalb im Wurzelverzeichnis der auf der ersten Platte installierten älteren Slackware-Distribution, die unter `/mnt/sda3` gemountet ist, abgelegt.

   Die Konfigurationsdatei für LILO ist `/mnt/sda3/etc/lilo.conf`. Sollen neue Kernel installiert werden, muß LILO mit `lilo -r /mnt/sda3` aufgerufen werden.

2. Der Start des X-Servers `XF86_S3` führte trotz verschiedener Konfigurationsversuche von `/etc/X11/XF86Config` immer zum Rechnerabsturz, aus dem der Rechner nur durch Drücken der Reset-Taste wieder zum Leben erweckt werden konnte. Wurde zuvor jedoch das Programm `SuperProbe` ausgeführt, ließ sich der X-Server problemlos starten. Die Ursache für dieses Verhalten konnte nicht geklärt werden.

   Das für das Starten des Displaymanagers `xdm` verantwortliche Skript `/etc/init.d/xdm` wurde daher um einen Aufruf von `SuperProbe` vor dem Aufruf von `xdm` (was zum Start des X-Servers führt) ergänzt.

3. Die Verzeichnisse `/usr/include/linux` und `/usr/include/asm` wurden vom Installationsskript als normale Verzeichnisse eingerichtet. Dies steht im Widerspruch zum Linux-Dateisystemstandard [FsStnd] und führt beim Kompilieren von systemnaher Software dazu, daß möglicherweise inkompatible C-Headerdateien eingebunden werden, falls ein anderer Betriebssystemkern verwendet wird, als der, zu dem die in diesen Verzeichnissen enthaltenen Headerdateien gehören.

   Die genannten Verzeichnisse wurden deshalb gelöscht und durch symbolische Links auf `/usr/src/linux/include/linux` bzw. `/usr/src/linux/include/asm` ersetzt.

Installation von Nicht-Debian-Software

Software, die nicht zur Debian-Distribution gehört, wurde im Verzeichnisbaum `/usr/local` (ein symbolischer Link auf `/mnt/sdb1/local`) installiert. Somit ist gewährleistet, daß eigene Programme und nicht stabile Testversionen nicht mit der Distribution vermischt werden, was Debian-Updates vereinfacht.

1. Die zum Betrieb von IP Version 6 notwendigen Netzwerkzeuge sind in `/usr/local/bin` und `/usr/local/sbin` abgelegt. Beim Systemstart wird in `/etc/init.d/boot` und `/etc/init.d/netbase` das Vorhandensein der Pseudodatei `/proc/net/ipv6_route` geprüft und, falls sie existiert, die Suchpfade `/usr/local/sbin`, `/usr/local/bin` und `/usr/local/usr/bin` vor den übrigen Suchpfad gestellt.

   Somit werden nur dann die IPv6-Werkzeuge verwendet, wenn ein IPv6-fähiger Kernel läuft.

2. IPv6-Bibliotheken sind in `/usr/local/lib` installiert.

3. Die Konfigurationsdateien des Nameservers befinden sich in `/var/named`.

4. Gepackte Programmpakete sind unter `/usr/local/Sources` abgelegt.

5. Sie werden in `/usr/local/src` ausgepackt und kompiliert. `/usr/local/src/linux` ist ein symbolischer Link auf die momentan verwendete Kernelversion.

A.1.2 Der Rechner *pchegering8*

Die lokale Platte ist unter `/mnt/local_hda2` gemountet. Auf ihr befinden sich neben allen Kernels, mit denen der Rechner gestartet werden kann, auch die Konfigurationsdatei für den LILO.

Das root-Dateisystem wird per NFS von *pchegering2* bezogen. Die Dateisystemstruktur für *pchegering8* wurde nach der Anleitung des NFS-Root-Client Mini-Howto [Maor 96] eingerichtet.

1. Die startbaren Kernel befinden sich in `/mnt/local_hda2`. Die Konfigurationsdatei für LILO ist `/mnt/local_hda2/etc/lilo.conf`. Sollen neue Kernel installiert werden, muß LILO mit `/mnt/local_hda2/sbin/lilo -r /mnt/local_hda2` aufgerufen werden.

2. Neue Module werden auf *pchegering2* mit `make modules_install` installiert und anschließend mit `cp -a /lib/modules/<Versionsnummer> /clients/pchegering8/lib/modules` dem Rechner *pchegering8* zur Verfügung gestellt.

3. Die Mount-Tabelle auf *pchegering8* sieht so aus:

```
129.187.214.42:/ on / type nfs (rw,addr=129.187.214.42)
129.187.214.42:/bin on /bin type nfs (rw,addr=129.187.214.42)
129.187.214.42:/usr on /usr type nfs (rw,addr=129.187.214.42)
129.187.214.42:/sbin on /sbin type nfs\
                                (rw,addr=129.187.214.42)
129.187.214.42:/home on /home type nfs\
```

```
                                        (rw,addr=129.187.214.42)
129.187.214.42:/lib on /lib type nfs (rw,addr=129.187.214.42)
129.187.214.42:/clients/etc on /server/etc type nfs\
                                        (rw,addr=129.187.214.42)
129.187.214.42:/clients/var on /server/var type nfs\
                                        (rw,addr=129.187.214.42)
129.187.214.42:/mnt/sdb1 on /mnt/sdb1 type nfs\
                                        (rw,addr=129.187.214.42)
129.187.214.42:/mnt/sda3 on /mnt/sda3 type nfs\
                                        (rw,addr=129.187.214.42)
129.187.214.42:/mnt/sda7 on /mnt/sda7 type nfs\
                                        (rw,addr=129.187.214.42)
/dev/hda2 on /mnt/local_hda2 type ext2 (rw)
none on /proc type proc (rw)
```

4. Der Eintrag zum Starten des Kernels 2.1.43 in `/mnt/local_hda2/etc/lilo.conf`:

```
# LILO configuration file
#
# Start LILO global section
boot = /dev/hda
delay = 30
vga = ask    # force sane state
ramdisk = 0     # paranoia setting
# End LILO global section
# Linux bootable partition config begins

image = /zImage.2.1.43
  label = 2.1.43.nfsroot
  append = "ether=9,0x300,eth1 root=/dev/nfs \
    nfsroot=129.187.214.42:/clients/pchegering8 \
    nfsaddrs=129.187.214.48:129.187.214.42:129.187.214.254: \
    255.255.255.0:pchegering8:eth0:none"
  read-only
```

A.2 Installation der IPv6-Software

Zum Betrieb des Internet Protokolls Version 6 müssen ein IPv6-fähiger Betriebssystemkern („Kernel“) und zusätzliche Werkzeuge installiert werden, die gegenwärtig von mehreren Personen an verschiedenen Orten erstellt werden. Die Entwicklung dieser Software wurde über einen Zeitraum von ca. sechs Monaten mitverfolgt und ist zum Zeitpunkt der Niederschrift dieser Dokumentation noch nicht abgeschlossen. Während dieser Zeit wurden mehrere Versionen des Kernels und der Werkzeuge mit wechselndem Erfolg installiert und getestet. Im folgenden ist die Installation der jeweils letzten verfügbaren und funktionsfähigen Versionen beschrieben.

Die Software ist in mehrere Pakete aufgeteilt:

Im Betriebssystemkern sind die Netzprotokollstacks implementiert. Der in der Debian-Distribution enthaltene Kernel Version 2.0.30 unterstützt IPv6 nicht. Dafür muß ein Entwickler-Kernel der 2.1.x-Serie kompiliert und installiert werden. Ein Kernel stellt zur Kompilierung und zum Betrieb bestimmte Mindestanforderungen an die verwendeten System- und sonstigen Hilfsprogramme. Diese Anforderungen konnten durch die Installation der Debian-Distribution Version 1.3 und der im folgenden beschriebenen IPv6-fähigen Software nicht ganz erfüllt werden. So mußte z.B. zusätzlich eine neue `mount`-Version installiert werden. Die Anforderungen des jeweiligen Kernels sind in dessen Sourcebaum immer in der Datei `Documentation/Changes` angegeben. Installiert ist:
`ftp://ftp.kernel.org/pub/linux/kernel/v2.1/linux-2.1.43.tar.gz`
Dieser Kernel ist nicht besonders stabil. Nach einigen Tagen Laufzeit lassen sich z.B. keinen neuen Prozesse mehr starten. Auch sind die für IPv6 notwendigen Systemaufrufe noch nicht vollständig implementiert. So fehlt z.B. die Möglichkeit, IPv6-Routen aus der Routing-Tabelle des Kernels zu löschen.

inet6-apps enthält `finger`, `fingerd`, `ftp`, `ftpd`, `inetd`, `ping` und `tftp`. Dieses Paket stellt außerdem die Bibliothek `libinet6` zur Verfügung, die einige IPv6-spezifische Funktionen enthält, die in die Standard-libc (bis einschließlich Version 5) von Linux noch nicht eingebaut sind. Installiert ist:
`ftp://ftp.inner.net/pub/ipv6/inet6-apps-0.24.tar.gz`

net-tools enthält die Programme `arp`, `ifconfig`, `rarp`, `netstat`, `route` und `hostname`. `dnsdomainname`, `ypdomainname`, `nisdomainname` und `domainname` werden jeweils als Link auf `hostname` installiert. Dieses und die meisten der folgenden Pakete benötigen zum Kompilieren entweder die `libinet6` aus den *inet6-apps* oder eine neuere Version der `glibc 2.1` (ab 970501) oder eine gepatchte Version der `glibc 2.0.2`. Installiert ist:
`ftp://ftp.cs-ipv6.lancs.ac.uk/pub/Code/Linux/Net_Tools/`
`net-tools-1.41.tar.gz`

inner-apps enthält ein Diagnoseprogramm `fdsniff`, einen Finger-Daemon `fingerd`, ein Firewallhilfsprogramm `ipfwdump` und `netd`, einen Ersatz für `inetd` und die `tcp_wrappers`. Installiert ist:
`ftp://ftp.inner.net/pub/ipv6/inner-apps-0.06.tar.gz`

libpcap ist ein API zur Entwicklung von Netzanalyseprogrammen. Installiert ist:
`ftp://ftp.inner.net/pub/ipv6/libpcap-0.3.1a2+ipv6-1.tar.gz`

telnet, traceroute, tcpdump werden als getrennte Pakete angeboten. `traceroute` und `tcpdump` benötigen die `libpcap`. Das `telnet`-Paket enthält auch einen `telnetd`-Daemon. Installiert sind:
`telnet.95.10.23.NE+ipv6-2.tar.gz`,
`tcpdump-3.3.1a2+ipv6-1.tar.gz` und

traceroute-1.4a4+ipv6-3.tar.gz,
jeweils aus ftp://ftp.inner.net/pub/ipv6.

radvd ist der *Router Advertisement Daemon* für IPv6. Installiert ist:
ftp://ftp.cityline.net/pub/systems/linux/network/ipv6/radvd/
radvd-0.4.tar.gz

A.2.1 Der Nameserver

Der in der Debian-Distribution Version 1.3 enthaltene Nameserver `bind-4.9.5-REL` ist in der Lage, mit IPv6 Adressen umzugehen. Debian plaziert seine Konfigurationsdateien in `/var/named` und gibt eine Aufteilung in mehrere Dateien vor. `/etc/named.boot` ist ein Link auf `../var/named/named.boot`, dessen Inhalt ist:

```
directory /var/named

; type          domain                  source          file
cache           .                                       named.root

; Zone boot information and daemon options are kept in other files

include boot.zones
include boot.options
```

Folgende Schritte wurden zu seiner Konfiguration unternommen:

1. ein Update des Root Caches `ftp://ftp.rs.internic.net/domain/named.root` nach `/var/named` kopiert,

2. in `boot.zones` die zu verwaltenden Zonen eingetragen:

```
primary 6bone.informatik.uni-muenchen.de db.6bone
primary 6.d.0.0.0.0.6.d.b.b.1.8.0.0.b.f.4.0.f.5.IP6.INT\
                                                  db.IP6.INT
```

3. in `boot.options` Optionen und loopback-Zonen eingetragen:

```
forwarders      129.187.214.135 129.187.214.20 129.187.10.25\
                129.187.16.1 193.196.32.1

; type    domain            source      file
primary   localhost                     named.local
primary   127.in-addr.arpa              named.rev-local

;; Custom configurations may appear below (will be preserved)
primary   ipv6-localhost                named.local.ipv6
primary
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.\
                        0.0.IP6.INT   named.rev-local.ipv6
```

4. `db.6bone` enthält:

```
@ IN  SOA ns.6bone.informatik.uni-muenchen.de.
    hostmaster.mailhost.6bone.informatik.uni-muenchen.de. (
        199707091      ; serial number (YYYYMMDDxx)
        10800   ; refresh time in seconds (3 hours)
        3600    ; retry time in seconds (1 hour)
        604800  ; 604800 expiration time in seconds (1 week)
        86400 ) ; 86400 minimum time in seconds (1 day)

  IN  NS        ns.6bone.informatik.uni-muenchen.de.
  IN  MX  10    mailhost.6bone.informatik.uni-muenchen.de.

ns       IN CNAME  pchegering2.nm.informatik.uni-muenchen.de.
mailhost IN CNAME  pchegering2.nm.informatik.uni-muenchen.de.

terra           IN  AAAA  5f04:fb00:81bb:d600:d6:800:98b:a8d8
pchegering2     IN  CNAME       terra
tunnel          IN  CNAME       terra
merkur          IN  AAAA  5f04:fb00:81bb:d600:d6:0:c0cb:291c
pchegering8     IN  CNAME       merkur
```

5. `db.IP6.INT` enthält:

```
@ IN  SOA 6bone.informatik.uni-muenchen.de.
   hostmaster.pchegering2.nm.informatik.uni-muenchen.de. (
   199707092 86400 10800 604800 86400 )
  IN  NS  ns.6bone.informatik.uni-muenchen.de.

8.d.8.a.b.8.9.0.0.0.8.0  IN  PTR \
                    terra.6bone.informatik.uni-muenchen.de.
c.1.9.2.b.c.0.c.0.0.0.0  IN  PTR \
                    merkur.6bone.informatik.uni-muenchen.de.
```

6. Die interessanteren Zeilen in den IPv6-loopback-Einträgen sind:

```
@       IN      AAAA    ::1
```

und Reverse:

```
1       IN      PTR     localhost.
```

A.2.2 Der Kernel

Der Kernel-Sourcebaum wurde unter `/usr/local/src` ausgepackt. Mit `make config`[1] wurden die benötigten Treiber ausgewählt. Für die IPv6-Unterstützung

[1] `make xconfig` funktioniert bei dieser Version nicht

muß mindestens auf die Fragen nach „Prompt for development and/or incomplete code/drivers“, „Networking support“, „TCP/IP networking“ und „The IPv6 protocol“ mit „y“ geantwortet werden. Die vollständigen Konfigurationen sind als `/usr/local/Sources/linux/config.pchegering{2,8}` abgelegt und können bei Bedarf als `.config` in den Kernel-Sourcebaum kopiert werden.

Nach Speichern der Konfiguration wurde der Kernel mit `make dep clean {b,}zImage`[2] `modules` übersetzt und das entstandene `arch/i386/boot/{b,}zImage` wie in Abschnitt A.1 beschrieben installiert.

A.2.3 inet6-apps

inet6-apps und die folgenden Pakete wurden unter `/usr/local/src/ipv6` ausgepackt.

Im `GNUmakefile.config` mußte die gewünschte Resolver-Bibliothek einkommentiert werden: Diese mit dem Nameserver `bind` gelieferte Bibliothek heisst `libresolv.a`, also war der entsprechende Abschnitt so zu ändern:

```
#LIBRESOLV=-lbind
#LIBRESOLV=
LIBRESOLV=-lresolv
```

Die Angabe

```
DESTDIR=/usr/inet6
```

wurde so belassen, da viele IPv6-Programme diesen Pfad benutzen. `/usr/inet6` ist hier ein Link nach `/usr/local`.

Ein mit `make` gestarteter Compilerlauf brachte ein paar Probleme ans Licht, die sich durch kleine Änderungen an einigen Makefiles der Programme beheben ließen. Im einzelnen sind dies:

`finger/GNUmakefile`: einzufügen war

```
CFLAGS+=-I/usr/include/db
```

`ftpd`: verlangte nach einem `yacc`, der nur als `byacc` installiert war. Lösung: Link angelegt

`inetd/GNUmakefile`: aus unerfindlichen Gründen mußte

```
CFLAGS+=-I/usr/include
```

eingefügt werden.

Nach diesen Änderungen konnte alles mit `make` übersetzt und mit `make install` installiert werden.

---

[2] Für *pchegering8* war `zImage` ausreichend, auf *pchegering2* mußte `bzImage` angegeben werden.

A.2.4 net-tools

Im `Makefile` waren zwei Kommentarzeichen zu entfernen und der Headerdatei- und der Bibliotheks-Suchpfad zu berichtigen, um die mit den `inet6-apps` installierte Bibliothek `libinet6` verwenden zu können:

```
COPTS = -O2 -Wall -g  -I/usr/inet6/include
LOPTS =
RESLIB = -L/usr/inet6/lib -linet6
```

Beim anschließenden `make config` wurden alle angebotenen Optionen mit ja beantwortet. Mit `make` wurde übersetzt und mit `make install` installiert. Da die Version 1.41 keine Test-, sondern eine reguläre Version ist, und eine Installation unter `/usr/local` Änderungen an vielen anderen Stellen notwendig gemacht hätte, wurde die im `Makefile` mögliche Angabe eines anderen `BASEDIR` unterlassen, so daß die neuen Programme die in der Debian-Distribution enthaltenen ersetzten.

A.2.5 inner-apps, libpcap, telnet, traceroute und tcpdump

Hier waren keine Änderungen nötig. `make` komplierte und `make install` installierte die Programme.

A.2.6 inetd

Um die neu installierten Daemons `ftpd`, `telnetd` und `fingerd` bei Bedarf vom ebenfalls neuen `inetd` starten lassen zu können, wurden in dessen Konfigurationsdatei `/etc/inetd.conf` die Einträge so geändert:

```
ftp    stream  tcp  nowait  root   /usr/local/bin/ftpd     ftpd -l
telnet stream  tcp  nowait  root   /usr/local/bin/telnetd  telnetd
finger stream  tcp  nowait  nobody /usr/local/bin/fingerd  fingerd
```

A.2.7 radvd

Um den *Router Advertisement Daemon* zu kompilieren und zu installieren, wurde mit `LIBS="-L/usr/inet6/lib"  ./configure -prefix=/usr/inet6` ein Makefile generiert, mit `make` übersetzt und mit `make install` installiert.

In der Konfigurationsdatei `/usr/inet6/etc/radvd.conf` wurde das zu sendende Präfix eingestellt:

```
interface eth0
{
        AdvSendAdvert on;
        prefix 5f04:fb00:81bb:d600:d6::0/80
        {
                AdvOnLink on;
                AdvAutonomous on;
        };
};
```

A.2.8 Die Bootskripten

Damit die Rechner gleich nach ihrem Start zu IPv6-Diensten fähig sind, mußten einige Startskripten geändert werden.

- `/etc/init.d/boot` ist das erste Skript, das vom `init`-Prozeß aufgerufen wird. An seinem Anfang wurde ein Test auf eine Pseudodatei im `proc`-Dateisystem eingefügt, die nur existiert, wenn ein IPv6-fähiger Kernel läuft. Existiert die Datei, werden vor den normalen Suchpfad weitere zu durchsuchende Verzeichnisse gestellt, in denen sich IPv6-Programme befinden:

```
if [ -f /proc/net/ipv6_route ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/local/usr/bin:\
        /sbin:/usr/bin:/bin:/usr/bin/X11:/usr/games:."
else
  PATH="/sbin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:\
        /usr/games:."
fi
```

- `/etc/init.d/netbase` startet und stoppt u.a. den Superserver `inetd`. Damit der neu installierte, IPv6-fähige `inetd` verwendet wird, wurde der Aufruf `/usr/sbin/inetd` überall durch `/usr/local/bin/inetd` ersetzt.

  Auf *terra* wurde außerdem ein Aufruf von `/usr/local/sbin/rc.inet6.pchegering2` eingefügt, wiederum abhängig von der Existenz von `/proc/net/ipv6_route`.

- `/usr/local/sbin/rc.inet6.pchegering2` enthält Befehle, die u.a. eine IPv6-Adresse und einen Tunnel zur Uni Münster konfigurieren. Dies ist nur auf *terra* notwendig, da dieser Rechner als Router fungiert. *merkur* wird mittels `radvd` (siehe Abschnitt A.2.7) automatisch zustandslos konfiguriert. Das Skript sieht so aus:

```
#!/bin/bash

set -x

PATH=/usr/local/bin:/usr/local/etc:/usr/bin:/bin:/etc
export PATH

# Das Routing-Praefix:
PREFIX=5f04:fb00:81bb:d600::/64

# Die IPv6-Adresse von pchegering2:
IPV6ADDRESS=5F04:FB00:81BB:D600:D6:800:98B:A8D8

# Die IPv4-Adresse des Tunnelendes in Muenster:
TUNNEL=128.176.191.66
```

```
# terra ist Router:
echo 1 > /proc/sys/net/ipv6/forwarding

# Dem eth0-Interface die IPv6-Adresse hinzufuegen:
/sbin/ifconfig eth0 add $IPV6ADDRESS

# Alles, was mit $PREFIX anfaengt, gehoert zum lokalen Netz:
/sbin/route -A inet6 add $PREFIX dev eth0

# Den Tunnel nach Muenster graben:
/sbin/ifconfig sit0 up
/sbin/ifconfig sit0 tunnel 0::$TUNNEL
/sbin/ifconfig sit1 up

# und einen Wegweiser aufstellen:
/sbin/route -A inet6 add 5f04::0/8 gw 0::$TUNNEL dev sit1

# Den Router Advertisement Daemon starten:
/usr/local/sbin/radvd
```

A.2.9 Multi-threaded Routing Toolkit

Um Tests mit dem Leibniz Rechenzentrum (LRZ) vorzubereiten, wurde zuletzt versucht, das Multi-threaded Routing Toolkit (MRT), das das Routing Protokoll RIPng unterstützt, zu installieren. Das Kompilieren der Quelltexte war nach einem ersten vergeblichen Versuch mit einer früheren Version schließlich erfolgreich, jedoch konnte eine passende Konfiguration mangels Dokumentation nicht vorgenommen werden.

Für spätere Versuche, evtl. mit einer stabileren Betriebssystemversion, ist mrtd auf pchegering2 als /usr/sbin/mrtd installiert, die Konfigurationsdatei (mit vergeblichen Konfigurationsversuchen) ist /etc/mrtd.conf, die Sourcen sind unter /usr/local/src/mrt-current-970807 abgelegt, Debuggingausgaben gehen nach /tmp/mrtd.debug.

Anhang B

Beispielprogramme

B.1 TCP-Client

B.1.1 IPv4-Version

```
/*
  IPv4-TCP-Client
  Beispiel zu Abschnitt 3.3 Anwendungsentwicklung
  */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

main(int argc, char *argv[])
{
        int sock;
        struct sockaddr_in server;
        struct hostent *hostinfo;
        char message[255];

        if (argc != 4) {
          printf("Syntax: %s Hostname Port Mitteilung\n", argv[0]);
          exit(1);
        }

        strcpy(message, argv[3]);

        /* Socket erzeugen */
        sock = socket(PF_INET, SOCK_STREAM, 0);
        if (sock < 0) {
          perror("socket");
          exit(1);
        }

        /* IPv4-Adresse zum angegebenen Rechnernamen holen */
```

```
        if ((hostinfo = gethostbyname(argv[1])) == NULL) {
          herror("gethostbyname");
          exit(1);
        }

        /* Die sockaddr_in Struktur fuellen */
        memcpy(&server.sin_addr, hostinfo->h_addr, hostinfo->h_length);
        server.sin_port = htons(atoi(argv[2]));
        server.sin_family = AF_INET;

        /* offiziellen Namen holen */
        if ((hostinfo = gethostbyaddr((char *) &server.sin_addr,
                                      sizeof(server.sin_addr),
                                      AF_INET)) == NULL) {
          herror("gethostbyaddr");
          exit(1);
        }

        printf("offizieller Name: %s, IP-Adresse: %s\n",
               hostinfo->h_name, inet_ntoa(server.sin_addr));

        /* Verbindung aufbauen */
        if (connect(sock, (struct sockaddr *) &server,
                    sizeof(server)) < 0) {
          perror("connect");
          exit(1);
        }

        /* und etwas darauf schreiben */
        if (write(sock, message, sizeof(message)) < 0)
          perror("write");

        close(sock);
}
```

B.1.2 IPv6-Version

```
/*
  IPv6-TCP-Client
  Beispiel zu Abschnitt 3.3 Anwendungsentwicklung
  */

#include <sys/types.h>
#include <netinet/in.h>
#include "include/socket.h"
#include "include/in6.h"
#include "include/netdb.h"
#include <stdio.h>
#include "include/resolv.h"

main(int argc, char *argv[])
{
        int sock;
        struct sockaddr_in6 server;
        struct hostent *hostinfo;
        char message[255];
        char *str;

        if (argc != 4) {
          printf("Syntax: %s Hostname Port Mitteilung\n", argv[0]);
          exit(1);
        }

        strcpy(message, argv[3]);

        /* Socket erzeugen */
        sock = socket(PF_INET6, SOCK_STREAM, 0);
        if (sock < 0) {
          perror("socket");
          exit(1);
        }

        /* IPv6-Adresse zum angegebenen Rechnernamen holen */
        res_init();
        _res.options |= RES_USE_INET6;

        hostinfo = gethostbyname2(argv[1], AF_INET6);
        if (hostinfo == 0) {
          herror("gethostbyname");
          exit(1);
        }

        /* Die sockaddr_in6 Struktur fuellen */
        memcpy(&server.sin6_addr, hostinfo->h_addr, hostinfo->h_length);
        server.sin6_port = htons(atoi(argv[2]));
        server.sin6_family = AF_INET6;

        /* offiziellen Namen holen */
```

```
        if ((hostinfo = gethostbyaddr((char *) &server.sin6_addr,
                                      sizeof(server.sin6_addr),
                                      AF_INET6)) == NULL) {
          herror("gethostbyaddr");
          exit(1);
        }

        if (inet_ntop(AF_INET6, &server.sin6_addr, str,
                      INET6_ADDRSTRLEN) == NULL)
          strcpy(str, "[ungueltige Adresse]");

        printf("offizieller Name: %s, IP-Adresse: %s\n",
               hostinfo->h_name, str);

        /* Verbindung aufbauen */
        if (connect(sock, (struct sockaddr *) &server,
                    sizeof(server)) < 0) {
          perror("connect");
          exit(1);
        }

        /* und etwas darauf schreiben */
        if (write(sock, message, sizeof(message)) < 0)
          perror("write");

        close(sock);
}
```

B.1.3 IPv6/IPv4-Version

```
/*
  Protokollunabhaengiger IPv6/IPv4-TCP-Client
  Beispiel zu Abschnitt 3.3 Anwendungsentwicklung
  */

#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "include/in6.h"
#include <netdb.h>
#include <stdio.h>
#include "include/resolv.h"
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include "include/support.h"

main(int argc, char *argv[])
{
  int sock, result, i = 0;
  struct addrinfo *ai = NULL;
  struct addrinfo **pai = &ai;
  struct addrinfo hints;
  char message[255];
  char *str;
  char hbuf[2][NI_MAXHOST], sbuf[2][NI_MAXSERV];

  memset(&hints, 0, sizeof(hints));
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_CANONNAME;

  if (argc != 4) {
    printf("Syntax: %s Hostname Port Mitteilung\n", argv[0]);
    exit(1);
  }

  strcpy(message, argv[3]);

  /* IP-Adresse(n) und andere Informationen
     zum angegebenen Rechnernamen und -port holen */
  if (result = getaddrinfo(argv[1], argv[2], &hints, pai)) {
    fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
    exit(1);
  }

  for (; ai; ai = ai->ai_next) {
    printf("\n\nRecord %d:\n", i++);
    printf("ai_flags:      %x(%d)\n", ai->ai_flags, ai->ai_flags);
    printf("ai_family:     %s(%d)\n", nrl_afnumtoname(ai->ai_family),
           ai->ai_family);
    printf("ai_socktype:   %s(%d)\n",
```

```
          nrl_socktypenumtoname(ai->ai_socktype), ai->ai_socktype);
    printf("ai_protocol:   %x(%d)\n", ai->ai_protocol, ai->ai_protocol);
    printf("ai_addrlen:    %x(%d)\n", ai->ai_addrlen, ai->ai_addrlen);
    printf("ai_canonname:  %s\n", ai->ai_canonname);
    if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                             hbuf[0], sizeof(hbuf[0]),
                             sbuf[0], sizeof(sbuf[0]), 0)) {
      fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
      continue;
    };
    if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                             hbuf[1], sizeof(hbuf[1]),
                             sbuf[1], sizeof(sbuf[1]),
                             NI_NUMERICHOST | NI_NUMERICSERV)) {
      fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
      continue;
    };
    printf("ai_addr:       %s.%s(%s.%s)\n\n", hbuf[0], sbuf[0],
                                              hbuf[1], sbuf[1]);

    fprintf(stderr, "Trying %s.%s(%s.%s)\n", hbuf[0], sbuf[0],
                                             hbuf[1], sbuf[1]);

    /* Socket erzeugen */
    if ((sock = socket(ai->ai_family, ai->ai_socktype,
                       ai->ai_protocol)) < 0) {
      printf("socket: %s(%d)\n", strerror(errno), errno);
      continue;
    };

    /* Verbindung aufbauen */
    if (connect(sock, ai->ai_addr, ai->ai_addrlen) < 0) {
      printf("connect: %s(%d)\n", strerror(errno), errno);
      continue;
    };

    /* und etwas darauf schreiben */
    if (write(sock, message, sizeof(message)) < 0)
      perror("write");
    close(sock);
    break;
  };
}
```

B.2 TCP-Server

B.2.1 IPv4-Version

```
/*
  IPv4-TCP-Server
  Beispiel zu Abschnitt 3.3 Anwendungsentwicklung
  */

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#define TRUE 1

main()
{
  int sock, server_length, client_length;
  struct sockaddr_in server, client;
  int msgsock;
  char buf[1024];
  int rval;
  fd_set ready;
  struct timeval to;
  struct hostent *hostinfo;

  /* Socket erzeugen */
  sock = socket(PF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("socket");
    exit(1);
  }

  /* Die sockaddr_in Struktur mit Wildcards fuellen */
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_port = 0;

  /* Socket an IP-Adresse und Port binden */
  if (bind(sock, (struct sockaddr *) &server, sizeof(server))) {
    perror("bind");
    exit(1);
  }

  /* Die zugeordnete Portnummer ermitteln und ausgeben */
  server_length = sizeof(server);
  if (getsockname(sock, (struct sockaddr *) &server, &server_length)) {
    perror("getsockname");
    exit(1);
  }
  printf("Zugeordneter Port: %d\n", ntohs(server.sin_port));
```

```
  fflush(stdout);

  /* Akzeptiere Verbindungswuensche */
  listen(sock, 5);
  do {
    FD_ZERO(&ready);
    FD_SET(sock, &ready);
    to.tv_sec = 5;
    if (select(sock + 1, &ready, 0, 0, &to) < 0) {
      perror("select");
      continue;
    }
    if (FD_ISSET(sock, &ready)) {
      client_length = sizeof(client);
      msgsock = accept(sock, (struct sockaddr *) &client, &client_length);
      if (msgsock == -1)
        perror("accept");
      else do {
        bzero(buf, sizeof(buf));
        if ((rval = read(msgsock, buf, 1024)) < 0)
          perror("read");
        else if (rval == 0)
          printf("Beende Verbindung\n");
        else {
          if (getpeername(msgsock, (struct sockaddr *) &client,
                          &client_length) < 0)
            perror("getpeername");
          else {
            if ((hostinfo = gethostbyaddr((char *) &client.sin_addr,
                                          sizeof(client.sin_addr),
                                          AF_INET)) == NULL)
              herror("gethostbyaddr");
            else {
              printf("\nVerbindungswunsch von Host %s (%s), Port %d\n",
                     hostinfo->h_name,
                     inet_ntoa (client.sin_addr),
                     ntohs (client.sin_port));
              printf("Mitteilung: %s\n", buf);
            }
          }
        }
      } while (rval > 0);
      close(msgsock);
    } else
      printf(".");
    fflush(stdout);
  } while (TRUE);
}
```

B.2.2 IPv6-Version

```
/*
  IPv6-TCP-Server
  Beispiel zu Abschnitt 3.3 Anwendungsentwicklung
  */

#include <sys/types.h>
#include "include/socket.h"
#include <netinet/in.h>
#include <sys/time.h>
#include "include/in6.h"
#include "include/netdb.h"
#include "include/resolv.h"
#include <stdio.h>
#define TRUE 1

main()
{
  int sock, server_length, client_length;
  struct sockaddr_in6 server, client;
  int msgsock;
  char buf[1024];
  int rval;
  fd_set ready;
  struct timeval to;
  struct hostent *hostinfo;
  char *str;

  /* Socket erzeugen */
  sock = socket(PF_INET6, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("socket");
    exit(1);
  }

  res_init();
  _res.options |= RES_USE_INET6;

  /* Die sockaddr_in6 Struktur mit Wildcards fuellen */
  server.sin6_family = AF_INET6;
  server.sin6_addr = in6addr_any;
  server.sin6_port = 0;

  /* Socket an IP-Adresse und Port binden */
  if (bind(sock, (struct sockaddr *) &server, sizeof(server))) {
    perror("bind");
    exit(1);
  }

  /* Die zugeordnete Portnummer ermitteln und ausgeben */
  server_length = sizeof(server);
  if (getsockname(sock, (struct sockaddr *) &server, &server_length)) {
```

```
    perror("getsockname");
    exit(1);
  }
  printf("Zugeordneter Port: %d\n", ntohs(server.sin6_port));
  fflush(stdout);

  /* Akzeptiere Verbindungswuensche */
  listen(sock, 5);
  do {
    FD_ZERO(&ready);
    FD_SET(sock, &ready);
    to.tv_sec = 5;
    if (select(sock + 1, &ready, 0, 0, &to) < 0) {
      perror("select");
      continue;
    }
    if (FD_ISSET(sock, &ready)) {
      client_length = sizeof(client);
      msgsock = accept(sock, (struct sockaddr *) &client, &client_length);
      if (msgsock == -1)
        perror("accept");
      else do {
        bzero(buf, sizeof(buf));
        if ((rval = read(msgsock, buf, 1024)) < 0)
          perror("read");
        else if (rval == 0)
          printf("Beende Verbindung\n");
        else {
          if (getpeername(msgsock, (struct sockaddr *) &client,
                          &client_length) < 0)
            perror("getpeername");
          else {
            if ((hostinfo = gethostbyaddr((char *) &client.sin6_addr,
                                          sizeof(client.sin6_addr),
                                          AF_INET6)) == NULL)
              herror("gethostbyaddr");
            else {
              if (inet_ntop (AF_INET6, &client.sin6_addr, str,
                             INET6_ADDRSTRLEN) == NULL)
                      strcpy(str, "[ungueltige Adresse]");
              printf("\nVerbindungswunsch von Host %s (%s), Port %d\n",
                      hostinfo->h_name,
                      str,
                      ntohs (client.sin6_port));
              printf("Mitteilung: %s\n", buf);
            }
          }
        }
      } while (rval > 0);
      close(msgsock);
    } else
      printf(".");
    fflush(stdout);
```

```
  } while (TRUE);
}
```

B.2.3 IPv6/IPv4-Version

```
/*
  Protokollunabhaengiger IPv6/IPv4-TCP-Server
  Beispiel zu Abschnitt 3.3 Anwendungsentwicklung
  */

#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "include/in6.h"
#include <netdb.h>
#include <stdio.h>
#include "include/resolv.h"
#include <errno.h>
#include <unistd.h>
#include <string.h>
/*#include "include/support.h"*/
#include <sys/time.h>
#include <sys/un.h>
#define TRUE 1


main()
{
  int msgsock, sval;
  int sock, result, i = 0;
  char buf[1024];
  int rval;
  fd_set ready;
  struct timeval to;
  char *str;
  struct addrinfo *ai = NULL;
  struct addrinfo **pai = &ai;
  struct addrinfo hints;
  char hbuf[2][NI_MAXHOST], sbuf[2][NI_MAXSERV];
  char *port = "12345";

  union sockaddr_union {
    struct sockaddr sa;
    struct sockaddr_in sin;
    struct sockaddr_in6 sin6;
    struct sockaddr_un sun;
    char __pad[128];
  } su;

  sval = sizeof(su);

  memset(&hints, 0, sizeof(hints));
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_CANONNAME | AI_PASSIVE;

  /* IP-Adresse(n) und andere Informationen
```

```
    zum angegebenen Rechnernamen und -port holen */
  if (result = getaddrinfo(NULL, port, &hints, pai)) {
    fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
    exit(1);
  }

  for (; ai; ai = ai->ai_next) {
    printf("\n\nRecord %d:\n", i++);
    printf("ai_flags:      %x(%d)\n", ai->ai_flags, ai->ai_flags);
    printf("ai_family:     %s(%d)\n", nrl_afnumtoname(ai->ai_family),
           ai->ai_family);
    printf("ai_socktype:   %s(%d)\n",
           nrl_socktypenumtoname(ai->ai_socktype), ai->ai_socktype);
    printf("ai_protocol:   %x(%d)\n", ai->ai_protocol, ai->ai_protocol);
    printf("ai_addrlen:    %x(%d)\n", ai->ai_addrlen, ai->ai_addrlen);
    printf("ai_canonname:  %s\n", ai->ai_canonname);
    if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                             hbuf[0], sizeof(hbuf[0]),
                             sbuf[0], sizeof(sbuf[0]), 0)) {
      fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
      continue;
    };
    if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                             hbuf[1], sizeof(hbuf[1]),
                             sbuf[1], sizeof(sbuf[1]),
                             NI_NUMERICHOST | NI_NUMERICSERV)) {
      fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
      continue;
    };
    printf("ai_addr:       %s.%s(%s.%s)\n\n", hbuf[0], sbuf[0],
                                              hbuf[1], sbuf[1]);

    fprintf(stderr, "Trying %s.%s(%s.%s)\n", hbuf[0], sbuf[0],
                                             hbuf[1], sbuf[1]);

    /* Socket erzeugen */
    if ((sock = socket(ai->ai_family, ai->ai_socktype,
                       ai->ai_protocol)) < 0) {
      printf("socket: %s(%d)\n", strerror(errno), errno);
      continue;
    };

    /* Socket an IP-Adresse und Port binden */
    if (bind(sock, ai->ai_addr, ai->ai_addrlen)) {
      perror("bind");
      exit(1);
    }

    /* Die zugeordnete Portnummer ermitteln und ausgeben */
    if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                             hbuf[0], sizeof(hbuf[0]),
                             sbuf[0], sizeof(sbuf[0]), 0)) {
      fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
```

```
    continue;
  };
  if (result = getnameinfo(ai->ai_addr, ai->ai_addrlen,
                           hbuf[1], sizeof(hbuf[1]),
                           sbuf[1], sizeof(sbuf[1]),
                           NI_NUMERICHOST | NI_NUMERICSERV)) {
    fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
    continue;
  };
  printf("ai_addr:       %s.%s(%s.%s)\n\n", hbuf[0], sbuf[0],
                                           hbuf[1], sbuf[1]);

  fprintf(stderr, "Trying %s.%s(%s.%s)\n", hbuf[0], sbuf[0],
                                          hbuf[1], sbuf[1]);

  fflush(stdout);

  /* Akzeptiere Verbindungswuensche */
  listen(sock, 5);
  do {
    FD_ZERO(&ready);
    FD_SET(sock, &ready);
    to.tv_sec = 5;
    if (select(sock + 1, &ready, 0, 0, &to) < 0) {
      perror("select");
      continue;
    }
    if (FD_ISSET(sock, &ready)) {
      msgsock = accept(sock, (struct sockaddr *)&su, &sval);
      if (msgsock == -1)
        perror("accept");
      else do {
        bzero(buf, sizeof(buf));
        if ((rval = read(msgsock, buf, 1024)) < 0)
          perror("read");
        else if (rval == 0)
          printf("Beende Verbindung\n");
        else {
          if (getpeername(msgsock, (struct sockaddr *)&su, &sval) < 0)
            perror("getpeername");
          else {
            if (result = getnameinfo((struct sockaddr *)&su, sval,
                                     hbuf[0], sizeof(hbuf[0]),
                                     sbuf[0], sizeof(sbuf[0]), 0)) {
              fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
              continue;
            };
            if (result = getnameinfo((struct sockaddr *)&su, sval,
                                     hbuf[1], sizeof(hbuf[1]),
                                     sbuf[1], sizeof(sbuf[1]),
                                     NI_NUMERICHOST | NI_NUMERICSERV)) {
              fprintf(stderr, "getnameinfo: %s\n", gai_strerror(result));
              continue;
```

```
              };
              printf("su_ai_addr:       %s.%s(%s.%s)\n\n",
                     hbuf[0], sbuf[0],
                     hbuf[1], sbuf[1]);

              fprintf(stderr, "Verbindung von %s.%s(%s.%s)\n",
                      hbuf[0], sbuf[0], hbuf[1], sbuf[1]);

              printf("Mitteilung: %s\n", buf);
            }
          }
        } while (rval > 0);
        close(msgsock);
      } else
        printf(".");
      fflush(stdout);
    } while (TRUE);
  }
}
```

Anhang C

Abkürzungsverzeichnis

A	
AMT	Address Mapping Table
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
B	
BGP	Border Gateway Protocol
C	
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CIDR	Classless Inter-Domain Routing
D	
DFN	Deutsches Forschungsnetz
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
F	
FTP	File Transfer Protocol
I	
IAB	Internet Architecture Board
IDRP	Inter-Domain Routing Protocol
IEEE	Institute of Electric and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IPX	Internetwork Packet Exchange
ISO	International Standardisation Organisation

L

LAN	Local Area Network
LILO	Linux Loader
LRZ	Leibniz Rechenzentrum

M

MAN	Metropolitan Area Network
MIB	Management Information Base
MN	Mobile Node
MRT	Multi-threaded Routing Toolkit

N

NCC	Network Coordination Centre
NFS	Network Filesystem
NIST	National Institute of Standards and Technology
NRL	US Naval Research Laboratory
NSAP	Network Service Access Point

O

OSI	Open System Interconnection
OSPF	Open Shortest Path First

R

RFC	Request for Comments
RIP	Routing Information Protocol
RIP-2	Routing Information Protocol Version 2
RIPE	Regional Internet Registry for Europe
RPC	Remote Procedure Call
RSVP	Resource Reservation Protocol

S

SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SNMPv1	Simple Network Management Protocol Version 1
SNMPv2	Simple Network Management Protocol Version 2
SOA	Start of Authority

T

TCP	Transmission Control Protocol

U

UDP	User Datagram Protocol

W

WAN	Wide Area Network

Literaturverzeichnis

[6bone 97] „WWW-Seiten des 6bone“, Juli 1997, http://www.6bone.net/.

[Austein 94a] R. Austein und J. Saperia, „RFC 1611: DNS Server MIB Extensions“, Technischer Bericht, IAB, Mai 1994, http://rfc.fh-koeln.de/rfc/html/rfc1611.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1600-1699/rfc1611.

[Austein 94b] R. Austein und J. Saperia, „RFC 1612: DNS Resolver MIB Extensions“, Technischer Bericht, IAB, Mai 1994, http://rfc.fh-koeln.de/rfc/html/rfc1612.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1600-1699/rfc1612.

[Bada 94] A. Badach, E. Hoffmann und O. Knauer, *High Speed Internetworking*, Addison-Wesley, 1994.

[Baker 92] F. Baker, „RFC 1354: IP Forwarding Table MIB“, Technischer Bericht, IAB, Juli 1992, http://rfc.fh-koeln.de/rfc/html/rfc1354.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1300-1399/rfc1354.

[Baker 97a] F. Baker, „RFC 2096: IP Forwarding Table MIB“, Technischer Bericht, IAB, Januar 1997, http://rfc.fh-koeln.de/rfc/html/rfc2096.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2096.

[Baker 97b] Fred Baker, John Krawczyk und A. Sastry, „RSVP Management Information Base“, Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-rsvp-mib-09.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-rsvp-mib-09.txt.

[Boudec 97] J. Le Boudec, T. Kurz und H. Einsiedler, „Realizing the Benefits of Virtual LANs by Using IPv6“, Internet Draft, IAB, 03 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-kurz-virtual-lans-benefits-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-kurz-virtual-lans-benefits-00.txt.

[Burruss 96] J. Burruss, Jodi Ito, Jeff Johnson, S. Willis und J. Chu, „Definitions of Managed Objects for the Fourth Version of Border Gateway Protocol (BGP-4)“, Internet Draft, IAB, 03 1996, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-idr-bgp4-mib-02.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-idr-bgp4-mib-02.txt.

[Case 96a] J. Case, K. McCloghrie, M. Rose und S. Waldbusser, „RFC 1902: Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)“, Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1902.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1902.

[Case 96b] J. Case, K. McCloghrie, M. Rose und S. Waldbusser, „RFC 1903: Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)“, Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1903.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1903.

[Case 96c] J. Case, K. McCloghrie, M. Rose und S. Waldbusser, „RFC 1905: Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)“, Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1905.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1905.

[Chen 97] E. Chen und John Stewart III, „Route Aggregation Tutorial“, Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-idr-aggregation-tutorial-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-idr-aggregation-tutorial-00.txt.

[Conta 96] A. Conta und S. Deering, „RFC 1885: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)“, Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1885.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1885.

[Daniele 97a] M. Daniele, „IP Version 6 Management Information Base for the Transmission Control Protocol“, Internet Draft, IAB, 06 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-ipv6-tcp-mib-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-ipv6-tcp-mib-00.txt.

[Daniele 97b] M. Daniele, „IP Version 6 Management Information Base for the User Datagram Protocol", Internet Draft, IAB, 06 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-ipv6-udp-mib-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-ipv6-udp-mib-00.txt.

[de Groot 97] G. de Groot und David Kessens, „A proposal for an IPv6 site database object", Internet Draft, IAB, 06 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ngtrans-6bone-registry-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ngtrans-6bone-registry-01.txt.

[Deering 96] S. Deering und R. Hinden, „RFC 1883: Internet Protocol, Version 6 (IPv6) Specification", Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1883.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1883.

[Droms 97] R. Droms, „RFC 2131: Dynamic Host Configuration Protocol", Technischer Bericht, IAB, April 1997, http://rfc.fh-koeln.de/rfc/html/rfc2131.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2100-2199/rfc2131.

[Eastlake 97a] D. Eastlake und C. Kaufman, „RFC 2065: Domain Name System Security Extensions", Technischer Bericht, IAB, Januar 1997, http://rfc.fh-koeln.de/rfc/html/rfc2065.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2065.

[Eastlake 97b] D. Eastlake, „RFC 2137: Secure Domain Name System Dynamic Update", Technischer Bericht, IAB, April 1997, http://rfc.fh-koeln.de/rfc/html/rfc2137.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2100-2199/rfc2137.

[Estrin 94] D. Estrin, T. Li und Y. Rekhter, „RFC 1668: Unified Routing Requirements for IPng", Technischer Bericht, IAB, August 1994, http://rfc.fh-koeln.de/rfc/html/rfc1668.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1600-1699/rfc1668.

[Fisc 96] S. Fischer und W. Müller, *Netzwerkprogrammierung unter Linux und Unix*, Hanser, 1996.

[Fleischman 94] E. Fleischman, „RFC 1687: A Large Corporate User's View of IPng“, Technischer Bericht, IAB, August 1994, http://rfc.fh-koeln.de/rfc/html/rfc1687.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1600-1699/rfc1687.

[FsStnd] „Linux Filesystem Standard Release 1.2“, März 1995, ftp://ftp.funet.fi/pub/Linux/doc/fsstnd.

[Gilligan 96] R. Gilligan und E. Nordmark, „RFC 1933: Transition Mechanisms for IPv6 Hosts and Routers“, Technischer Bericht, IAB, April 1996, http://rfc.fh-koeln.de/rfc/html/rfc1933.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1933.

[Gilligan 97] R. Gilligan, S. Thomson, J. Bound und W. Stevens, „RFC 2133: Basic Socket Interface Extensions for IPv6“, Technischer Bericht, IAB, April 1997, http://rfc.fh-koeln.de/rfc/html/rfc2133.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2100-2199/rfc2133.

[Harrington 97] D. Harrington, „Link Local Addressing and Name Resolution in IPv6“, Internet Draft, IAB, 01 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-linkname-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-linkname-01.txt.

[Haskin 97a] Dimitry Haskin und S. Onishi, „Management Information Base for IP Version 6: ICMPv6 Group“, Internet Draft, IAB, 03 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-ipv6-icmp-mib-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-ipv6-icmp-mib-01.txt.

[Haskin 97b] Dimitry Haskin und S. Onishi, „Management Information Base for IP Version 6: Textual Conventions and General Group“, Internet Draft, IAB, 06 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-ipv6-mib-02.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-ipv6-mib-02.txt.

[Hauc 94] H. Hauck, „Integration des Managements des Domain-Name-Services in die vorhandenen Managementumgebung des Leibniz-Rechenzentrums (LRZ)", Master's thesis, Technische Universität München, November 1994.

[HeAb 93] H.-G. Hegering und S. Abeck, *Integriertes Netz– und Systemmanagement*, Addison-Wesley, 1993.

[Hein 94] M. Hein und D. Griffiths, *SNMP Simple Network Management Protocol Version 2*, Thomson Publishing, 1994.

[Hinden 96a] R. Hinden und S. Deering, „RFC 1884: IP Version 6 Addressing Architecture", Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1884.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1884.

[Hinden 96b] R. Hinden und J. Postel, „RFC 1897: IPv6 Testing Address Allocation", Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1897.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1897.

[Hinden 97a] Bob Hinden und Steve Deering, „Internet Protocol, Version 6 (IPv6) Specification", Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-ipv6-spec-v2-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-ipv6-spec-v2-00.txt.

[Hinden 97b] Bob Hinden und M. Crawford, „Router Renumbering for IPv6", Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-router-renum-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-router-renum-01.txt.

[Hinden 97c] Bob Hinden und Michael O'Dell, „TLA and NLA Assignment Rules", Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-tla-assignment-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-tla-assignment-00.txt.

[Hinden 97d] Bob Hinden, Michael O'Dell und Steve Deering, „An IPv6 Aggregatable Global Unicast Address Format", Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-unicast-aggr-02.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-unicast-aggr-02.txt.

[Hoff 96] R. Hoffmann, „Weiterentwicklung der Testumgebung für die Erprobung von Protokollen zur Unterstützung mobiler Systeme“, Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, November 1996.

[Horn 94] T. Horn, *Einführung in die digitale Sprachsignalverarbeitung*, Centrum für Informations- und Sprachverarbeitung, LMU München, 1994.

[HS 96] M. Avitabile und G. Müller, „IP Next Generation: IPv6“, Ludwig-Maximilians-Universität und Technische Universität München, Institut für Informatik, Juni 1996, Hauptseminar.

[Huit 96a] C. Huitema, *IPv6 - The New Internet Protocol*, Prentice Hall, 1996.

[Huit 96b] C. Huitema, *Routing im Internet*, Prentice Hall, 1996.

[IEEE 97] IEEE, „Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority“, März 1997, http://standards.ieee.org/db/oui/tutorials/EUI64.html.

[ISI 97] „WWW-Seiten der 6bone-Registratur“, Juli 1997, http://www.isi.edu/~davidk/6bone/.

[JOIN 97] „WWW-Seiten des JOIN-Projektes an der Universität Münster“, Juli 1997, http://www.join.uni-muenster.de/JOIN/.

[Kern 90] B. Kernighan und D. Ritchie, *Programmieren in C*, Hanser, 1990.

[Kowa 94] W. Kowalk und M. Burke, *Rechnernetze*, Teubner, 1994.

[Malkin 94] G. Malkin und F. Baker, „RFC 1724: RIP Version 2 MIB Extension“, Technischer Bericht, IAB, November 1994, http://rfc.fh-koeln.de/rfc/html/rfc1724.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1700-1799/rfc1724.

[Maor 96] O. Maor, „Linux NFS-Root-Client Mini-Howto“, 1996, ftp://ftp.funet.fi/pub/Linux/doc/HOWTO/mini/NFS-Root-Client.

[McCloghrie 90] K. McCloghrie und M. Rose, „RFC 1155: Structure and Identification of Management Information for TCP/IP-based Internets“, Technischer Bericht, IAB, Mai 1990, http://rfc.fh-koeln.de/rfc/html/rfc1155.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1100-1199/rfc1155.

[McCloghrie 91] K. McCloghrie und M. Rose, „RFC 1213: Management Information Base for Network Management of TCP/IP-based internets: MIB-II“, Technischer Bericht, IAB, Maerz 1991, http://rfc.fh-koeln.de/rfc/html/rfc1213.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1200-1299/rfc1213.

[McCloghrie 94] K. McCloghrie und F. Kastenholz, „RFC 1573: Evolution of the Interfaces Group of MIB-II“, Technischer Bericht, IAB, Januar 1994, http://rfc.fh-koeln.de/rfc/html/rfc1573.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1500-1599/rfc1573.

[McCloghrie 96a] K. McCloghrie, „RFC 2011: SNMPv2 Management Information Base for the Internet Protocol using SMIv2“, Technischer Bericht, IAB, November 1996, http://rfc.fh-koeln.de/rfc/html/rfc2011.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2011.

[McCloghrie 96b] K. McCloghrie, „RFC 2012: SNMPv2 Management Information Base for the Transmission Control Protocol“, Technischer Bericht, IAB, November 1996, http://rfc.fh-koeln.de/rfc/html/rfc2012.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2012.

[McCloghrie 96c] K. McCloghrie, „RFC 2013: SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2“, Technischer Bericht, IAB, November 1996, http://rfc.fh-koeln.de/rfc/html/rfc2013.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2013.

[Narten 96] T. Narten, E. Nordmark und W. Simpson, „RFC 1970: Neighbor Discovery for IP Version 6 (IPv6)“, Technischer Bericht, IAB, August 1996, http://rfc.fh-koeln.de/rfc/html/rfc1970.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1970.

[Nordmark 97] Erik Nordmark, „Site prefixes in Neighbor Discovery“, Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ngtrans-header-trans-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ngtrans-header-trans-00.txt.

[O'Dell 97] Michael O'Dell, „GSE - An Alternate Addressing Architecture for IPv6“, Internet Draft, IAB, 02 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-gseaddr-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-gseaddr-00.txt.

[Perkins 96] C. Perkins, „RFC 2002: IP Mobility Support“, Technischer Bericht, IAB, Oktober 1996, http://rfc.fh-koeln.de/rfc/html/rfc2002.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2002.

[Perkins 97a] C. Perkins und J. Bound, „Dynamic Host Configuration Protocol for IPv6 (DHCPv6)“, Internet Draft, IAB, 05 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-dhc-dhcpv6-10.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-dhc-dhcpv6-10.txt.

[Perkins 97b] C. Perkins, F. Teraoka und D. Johnson, „Mobility Support in IPv6“, Internet Draft, IAB, 06 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-teraoka-ipv6-mobility-sup-04.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-teraoka-ipv6-mobility-sup-04.txt.

[Postel 97] J. Postel, Bob Fink und Bob Hinden, „IPv6 Testing Address Allocation“, Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-testv2-addralloc-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-testv2-addralloc-01.txt.

[Rekhter 94] Y. Rekhter, R. Moskowitz, D. Karrenberg und G. de Groot, „RFC 1597: Address Allocation for Private Internets“, Technischer Bericht, IAB, Maerz 1994, http://rfc.fh-koeln.de/rfc/html/rfc1597.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1500-1599/rfc1597.

[Rekhter 97a] Y. Rekhter, P. Lothberg, R. Hinden, S. Deering und J. Postel, „RFC 2073: An IPv6 Provider-Based Unicast Address Format“, Technischer Bericht, IAB, Januar 1997, http://rfc.fh-koeln.de/rfc/html/rfc2073.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2073.

[Rekhter 97b] Yakov Rekhter, „Interaction between DHCP and DNS“, Internet Draft, IAB, 05 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-dhc-dhcp-dns-04.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-dhc-dhcp-dns-04.txt.

[Rieg 96] G. Riegert, „Entwicklung einer Managementschnittstelle für DHCP-Server“, Master's thesis, Technische Universität München, August 1996.

[Rose 93] M. Rose, *Einführung in die Verwaltung von TCP-IP-Netzen*, Hanser und Prentice-Hall Int., 1993.

[Schoffstall 90] M. Schoffstall, M. Fedor, J. Davin und J. Case, „RFC 1157: A Simple Network Management Protocol (SNMP)“, Technischer Bericht, IAB, Mai 1990, http://rfc.fh-koeln.de/rfc/html/rfc1157.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1100-1199/rfc1157.

[Stainov 97] R. Stainov, *IPng – Das Internet Protokoll der nächsten Generation*, Thomson Publishing, 1997.

[Stevens 97] W. Stevens und M. Thomas, „Advanced Sockets API for IPv6“, Internet Draft, IAB, 07 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-stevens-advanced-api-04.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-stevens-advanced-api-04.txt.

[Thomson 96a] S. Thomson und C. Huitema, „RFC 1886: DNS Extensions to support IP version 6“, Technischer Bericht, IAB, Januar 1996, http://rfc.fh-koeln.de/rfc/html/rfc1886.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1886.

[Thomson 96b] S. Thomson und T. Narten, „RFC 1971: IPv6 Stateless Address Autoconfiguration“, Technischer Bericht, IAB, August 1996, http://rfc.fh-koeln.de/rfc/html/rfc1971.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1971.

[Vixie 97] P. Vixie, S. Thomson, Y. Rekhter und J. Bound, „RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE)“, Technischer Bericht, IAB, April 1997, http://rfc.fh-koeln.de/rfc/html/rfc2136.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2100-2199/rfc2136.

[Willis 94] S. Willis, J. Burruss und J. Chu, „RFC 1657: Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2“, Technischer Bericht, IAB, Juli 1994, http://rfc.fh-koeln.de/rfc/html/rfc1657.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1600-1699/rfc1657.

[Zhang 97] Lixia Zhang, J. Stewart, Thomas Narten und M. Crawford, „IPng Analysis of the GSE Proposal“, Internet Draft, IAB, 03 1997, gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipngwg-esd-analysis-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipngwg-esd-analysis-00.txt.