IPv6 in Java

Martin Pelzer, Moritz Vieth

Stand: 29.05.2003

Studienarbeit zur Vorlesung
„Verteilte und parallele Systeme 1“
bei Prof. Rudolf Berrendorf

Fachbereich Angewandte Informatik und Kommunikationstechnik
Fachhochschule Bonn-Rhein-Sieg

Inhaltsverzeichnis

1. Einleitung

Diese Studienarbeit wurde im Rahmen der Veranstaltung „Verteilte und parallele Systeme 1" im Sommersemester 2003 zur Erreichung des Leistungsnachweises angefertigt. Die Arbeit beschäftigt sich mit der Einbindung des neuen Internetprotokolls IPv6 in die Programmiersprache Java und den damit verbundenen Änderungen für den Programmierer.

Java unterstützte bisher nur IPv4. Durch die Einführung des neuen Protokolls wurde eine Überarbeitung der IP-Unterstützung in Java notwendig. Seit der Version 1.4 unterstützt Java nun auch IPv6. Die Veränderungen, die mit dieser Einbindung von IPv6 in Java einhergegangen sind, sollen durch diese Arbeit dargestellt und erläutert werden.

Um den Leser mit den Grundlagen der Thematik vertraut zu machen, bietet Kapitel zwei einen Überblick über das Protokoll IPv6. Hier wird zum Einen die Geschichte des Internet Protokolls von seiner Entstehung vor etwa 30 Jahren bis zur momentanen Einführung von IPv6 beschrieben. Zum Anderen bietet dieses Kapitel eine Übersicht über die Adressierung von IPv6-Adressen sowie über die wichtigsten Neuerungen und Änderungen im Vergleich zu IPv4.

Kapitel drei fasst kurz die Einbindung von IPv4 in Java zusammen und zeigt die dazu benötigten Klassen und deren Anwendung auf.

Das vierte Kapitel schließlich zeigt, wie IPv6 in Java benutzt wird und welche Änderungen es an den Klassen zur Einbindung des IP-Protokolls gegeben hat. Von besonderem Interesse ist hier die Klasse Inet6Adress. Außerdem beschreibt dieses Kapitel kurz Probleme, die unter Java bei der Verwendung von IPv6 mit einigen Betriebssystemen auftreten können.

Kapitel fünf fasst abschließend die Ausführungen dieser Arbeit noch einmal zusammen.

2. Einführung in IPv6

2.1 Was ist IPv6?

2.1.1 Geschichte des Internet Protokolls

Das Internet Protokoll entstand Mitte der siebziger Jahre des vergangenen Jahrhunderts. 1974 stellten die Wissenschaftler Vinton Cerf und Robert Kahn in den USA ein von ihnen entwickeltes netzübergreifendes Protokoll namens „TCP/IP" vor. Die Entwicklung eines einheitlichen Protokolls war nötig geworden, da die Vernetzung in Amerika, angestoßen 1969 durch das ARPA-Net, mehr und mehr zunahm und verschiedene Netze, die größtenteils auf verschiedenen Protokollen basierten, miteinander kommunizieren mussten [Sch].
Cerf und Kahn sahen TCP/IP anfangs als nur ein Protokoll, das sowohl die Adressierung der Rechner als auch die Kommunikation zwischen diesen regelte. Erst später wurde das Protokoll dann in seine heutigen Bestandteile TCP und IP aufgeteilt. Die beiden Wissenschaftler entwickelten bis Ende der siebziger Jahre mehrere Versionen ihres Protokolls bis im Jahr 1981 das bis heute genutzte IPv4 in [RFC791] standardisiert wurde.
Stellte der Adressraum von IPv4 mit 2^{32} Adressen 1981 noch eine wohl nie zu erreichende Größe dar, so führte der Internet-Boom der neunziger Jahre des 20. Jahrhunderts zu einem immens steigenden Bedarf an IP-Adressen und es zeichnete sich immer mehr ab, dass der Adressenvorrat, den IPv4 zur Verfügung stellt auf Dauer nicht ausreichen würde. Schon in den frühen neunziger Jahren wurde deshalb mit der Entwicklung eines neuen Protokolls, damals IPng (IP next generation) genannt, begonnen [laynet]. Im Jahr 1995 wurde IPv6 als Nachfolger für das alte Protokoll angenommen und in [RFC2460] standardisiert.
Betrachtet man heute, ca. acht Jahre nach der Standardisierung von IPv6, das Internet, so stellt man fest, dass die Verbreitung dieses Protokolls bisher nur sehr schleppend verlief. Die große Mehrheit der Knoten im Internet wird weiterhin über IPv4 adressiert. Die Abschwächung des Internet-Booms zu Beginn dieses Jahrzehnts ließ die Anzahl der IP-Adressen weniger stark wachsen als erwartet. Nichtsdestotrotz wird die Bedeutung von IPv6 über kurz oder lang zunehmen und irgendwann IPv4 als vorherrschendes Protokoll im Internet ablösen. Gründe für diese Annahme sind neben dem größeren Adressraum, der irgendwann benötigt werden wird, auch mehrere Neuerungen in IPv6, die das Protokoll gegenüber seinem Vorgänger komfortabler machen (siehe Kapitel 2.1.2). Ob IPv6 seinen Vorgänger jedoch komplett verdrängen wird, ist zur Zeit noch

nicht abzusehen, vor allem, da bereits einige Neuerungen aus IPV6 nach IPV4 rückportiert wurden.

2.1.2 Unterschiede zu IPv4

Der in der Öffentlichkeit wohl am besten bekannte Unterschied der beiden Protokolle ist der mit 2^{128} theoretisch möglichen Adressen um ein Vielfaches größerer Adressraum des neuen Protokolls. Aber neben diesem Punkt integriert der IPv6-Standard eine Reihe von Funktionalitäten, die im Laufe der letzten zwei Jahrzehnte entwickelt wurden aber nicht im IPv4-Standard enthalten sind oder dort nur optional angegeben werden.
Erstens ist in IPv6 ein „stateless host autoconfiguration"-Mechanismus vorhanden [RFC2462]. Dieser übernimmt die Aufgabe des in IPv4 nur optionalen DHCP-Servers, weist Rechnern also dynamisch eine IP-Adresse zu, wenn diese sich mit dem Netzwerk verbinden. Dieser Mechanismus wird heute beispielsweise beim Einwählen ins Internet genutzt. Während der Einwahl bekommt der Teilnehmer eine IP-Adresse zugewiesen [netBSD].
Zweitens wurde das Sicherheitstool IPsec, dass ebenfalls für IPv4 nur optional war, in IPv6 eingebaut. Somit ist gewährleistet, dass IPsec vorhanden ist, sobald man mit einem IPv6-Rechner kommuniziert. Bei IPv4 musste die Frage nach der Existenz von IPsec auf beiden miteinander kommunizierenden Systemen stets vorher geklärt werden, was ein unnötiges Kommunikationsaufkommen beim Aufbau der Verbindung bedeutet [netBSD].
Es gibt noch eine ganze Reihe nunmehr fest in den Standard eingebaute Mechanismen. Diese sollen hier aber nicht weiter erläutert werden.
Ein weiterer wichtiger Unterschied zwischen den beiden Protokollversionen sind die verschiedenen Adresstypen. IPv6 unterstützt Unicast-, Anycast- und Multicast-Adressen. Eine Unicast-Adresse spricht genau einen Rechner an, ist also der Standardfall in der Anwendung. Eine Anycast-Adresse richtet sich an mehrere Rechner. Allerdings wird die Nachricht nur einem, im Normalfall dem nächstliegenden Empfänger, zugestellt. Eine denkbare Anwendung für diese Art der Adressierung ist beispielsweise eine DNS-Anfrage. Der Client schickt diese per Anycast an alle ihm bekannten Nameserver. Der Client selber muss hierbei nicht wissen, welcher Nameserver am schnellsten zu erreichen ist. Multicast-Adressen schließlich adressieren mehrere Rechner, denen auch allen die Nachricht zugestellt wird. Multicast-Adressen übernehmen beispielsweise das im IPv4-Standard enthaltene Broadcast, welches es in alter Form in IPv6 nicht mehr gibt [hoff97][Sch].
Auch die Einteilung der Adressen hat sich mit der neuen Protokollversion geändert. IPv4-Adressen wurden in die Klassen A, B, C, D und E eingeteilt. Diese Klassen standen für Netze verschiedener Größen oder reservierten Adressbereiche für die Privatnutzung. In IPv6 wurde diese Klasseneinteilung wegen ihrer geringen Flexibilität ersetzt. Stattdessen

werden die Adressen nun nach bestimmten Präfixen in Adresstypen eingeteilt. Dadurch soll eine anwendungsorientiertere Hierarchie in den Adressraum gebracht werden (siehe Kapitel 2.1.3).
Als letzter Unterschied soll hier noch der IPv6-Header Erwähnung finden. Dieser wurde im Vergleich zu IPv4 deutlich verändert. Er hat nun eine feste Länge und ist deutlich kürzer geworden. Allerdings können zwischen dem Header und den Nutzdaten im Datagram noch mehrere sogenannte Erweiterungsheader eingefügt werden, die beispielsweise Informationen zur Fragmentierung enthalten können. Die Informationen dieser Erweiterungsheader waren bisher im IP-Header selbst enthalten, wurden aber nicht bei allen Paketen benötigt. Um diesem Umstand Rechnung zu tragen, enthält der eigentliche Header nur noch die bei jedem Paket notwendigem Felder während die restlichen Felder bei Bedarf in Erweiterungsheadern angefügt werden können [Sch].

2.1.3 Adressraum und Adressierung

Der Adressraum wurde bei IPv6 im Vergleich zu IPv4 erheblich erweitert. Es stehen nun theoretisch 2^{128} verschiedene Adressen zur Verfügung. Diese enorme Zahl wurde gewählt, um den Adressraum durch eine Hierarchie übersichtlicher gestalten zu können. IPv6-Adressen werden in Adresstypen geteilt. Das Kriterium, nach dem diese Einteilung geschieht, ist der Präfix der Adresse. So befinden sich zum Beispiel sämtliche Adressen, die in Binärdarstellung mit „100" beginnen, in einer geographischen Hierarchie, aus der sich ablesen lässt, auf welchem Kontinent oder in welchem Land sich der Anwender mit dieser IP-Adresse aufhält. Durch diese Hierarchierung des Adressraums wird die Zahl der praktisch zur Verfügung stehenden Adressen natürlich gesenkt, was aber aufgrund der Wahl des enorm großen Adressraums zu keinerlei Problemen führen dürfte.
128 Bit lange Adressen lassen sich nur schwer darstellen. Eine Darstellung wie bei IPv4 mit durch Punkte getrennten Dezimalzahlen, die jeweils acht Bit repräsentieren, würde bei IPv6 eine Folge von 16 Dezimalzahlen bedeuten. Um dieser unhandlichen Schreibweise entgegenzuwirken wurde für IPv6 eine neue Notation festgelegt. Die 128 Bit werden in 8 * 16 Bit zerlegt. Je 16 Bit werden durch eine vierstellige Hexadezimalzahl dargestellt. Um Verwechslungen mit der IPv4-Notation vorzubeugen werden diese Hexadezimalzahlen durch Doppelpunkte getrennt und nicht, wie bei IPv4, durch Punkte. Diese neue Notation ist zwar kürzer, aber trotzdem noch deutlich komplexer als die Schreibweise der IPv4-Adressen. Um die Notation weiter zu vereinfachen bedient man sich des Umstands, dass aufgrund der Hierarchierung des Adressraums die Wahrscheinlichkeit, eine längere Folge von Nullen in einer Adresse zu haben, relativ hoch ist. Dies kommt daher, dass der Zweig in der Hierarchie durch die ersten Bits der Adresse festgelegt wird während die Adressierung der Rechner in diesem Zweig bei den letzten Bits beginnt. Erst bei einer großen Anzahl an Rechnern, werden die zwischen diesen

beiden Teilen liegenden Bits aufgefüllt. Diese Folge von Nullen in der Mitte der Adresse darf in einer IPv6-Adresse einmalig durch zwei aufeinanderfolgende Doppelpunkte abgekürzt werden, so dass sich letztendlich bei den meisten Adressen eine einigermaßen übersichtliche und handhabbare Notation ergibt.
Ein großer Nachteil der neuen Notation ist allerdings die Verwendung von Doppelpunkten. War es bei IPv4-Adressen noch problemlos möglich, diese in die Adresszeile eines Browsers einzugeben, so ist dies bei IPv6-Adressen problematisch geworden. Ein Doppelpunkt wird von Browsern als Einleitung einer Portangabe interpretiert. Um hier wieder Eindeutigkeit herzustellen, legt [RFC2373] fest, dass IPv6-Adressen in eckigen Klammern geschrieben werden. Die meisten der heutigen Browser interpretieren in dieser Form geschriebene IPv6-Adressen, während die Angabe ohne Doppelpunkte zu einem Fehler führt.

2.1.4 Spezielle Adressen in IPv6

Wie bei IPv4 gibt es auch bei IPv6 spezielle Adressen und Adresstypen, die bestimmte vordefinierte Rechner oder Rechnermengen adressieren. In IPv6 sind dies die Unspecified address, die Loopback address, die IPv4-comaptible address sowie die IPv4-mapped address. Die Funktion dieser Adressen soll in diesem Kapitel veranschaulicht werden.
Die Unspecified address :: (also 128 Nullen) soll prinzipiell nicht genutzt werden. Ihr Verwendungsfeld liegt beispielsweise beim Booten eines Rechners, solange ihm noch keine IP zugewiesen wurde. Diese Adresse ist nur ein Platzhalter.
Die Loopback address ::1 (127 Nullen gefolgt von einer Eins) entspricht der IPv4-Loopback-Adresse 127.0.0.0.1.
Die IPv4-compatible address und die IPv4-mapped address stellen eine Möglichkeit der Kommunikation zwischen den beiden IP-Protokollen zur Verfügung. Bei beiden wird eine IPv6-Adresse angelegt und die IPv4-Adresse in die letzten 32 Bit der IPv6-Adresse geschrieben. Die restlichen 96 Bit werden bei der IPv4-compatible address komplett mit Nullen aufgefüllt, bei der IPv4-mapped address werden vor die IPv4-Adresse vier „F" gesetzt und dann die restlichen 92 Bit nach vorne mit Nullen aufgefüllt. Diese vier „F" dienen lediglich der Unterscheidung dieser beiden Adresstypen. Wieso es für die Kompatibilität zu IPv4 zwei verschiedene Adresstypen gibt, wird im nächsten Kapitel erläutert [hoff97].

2.2 Koexistenz von IPv4 und IPv6

Wie weiter oben bereits erwähnt verläuft die Einführung von IPv6 bisher nur sehr schleppend. Es wird auf unabsehbare Zeit eine Koexistenz von IPv4 und IPv6 im Internet geben. Daher war die Abwärtskompatibilität von IPv6 Grundvoraussetzung für eine

erfolgreiche Einführung. Die oben bereits beschriebenen IPv4-mapped- und IPv4-compatible-Adressen bilden die Grundlage für diese Abwärtskompatibilität. Durch diese beiden Adresstypen ist es möglich, IPv4-Adressen in IPv6-Adressen darzustellen und somit IPv4-Knoten mit in die Kommunikation einzubeziehen.

Möchte ein IPv6-Knoten mit einem IPv4-Knoten kommunizieren, so muss der IPv6-Knoten in der Lage sein, seine Adresse für den IPv4-Knoten verständlich angeben zu können. Hierfür werden IPv4-compatible Adressen genutzt. Da in diesen nur die letzten 32 Bit als eigentliche Adresse verwendet werden – die restlichen Bits sind ja alle Null – kann der IPv6-Knoten seine Adresse auch in IPv4-Form angeben. Diese Form kann der IPv4-Knoten verstehen. Für ihn ist es nicht sichtbar, dass er mit einem IPv6-Knoten kommuniziert. Bei IPv4-compatible Adressen muss allerdings sichergestellt sein, dass keine äquivalente IPv4-Adresse existiert, da es sonst zu Verwechslungen kommen kann. Der vorgezeichnete Weg zur Umstellung auf IPV6 ist hier also, alle IPV4-Knoten nach und nach umzurüsten und ihnen IPV4-compatible Adresse zu geben, die ihren alten IPv4-Adressen entsprechen.

IPv4-mapped Adressen entstehen dynamisch. Möchte ein IPv4-Knoten mit einem IPv6-Knoten kommunizieren, so schickt er in seiner Anfrage seine IPv4-Adresse mit. Der IPV6-Knoten „verpackt" diese dann in eine IPv4-mapped Adresse und ist somit in der Lage, sie wie eine IPv6-Adresse zu behandeln und zu verarbeiten. Ein Knoten mit dieser IPv4-mapped Adresse existiert also eigentlich gar nicht. Die Adresse dient lediglich dazu, dem IPv6-Knoten die Möglichkeit zu geben, die Adresse des IPv4-Knotens zu verarbeiten und zu verwalten, ohne dass er das alte Adressformat verstehen muss. Ein Knoten, der IPv4-Adressen in IPv4-mapped Adressen umwandeln kann, nennt man Dual Stack Node. Jeder IPv6-Server-Socket in Java ist eine Dual Stack Node. Der Programmierer muss hierzu keine gesonderten Optionen aktivieren.

Ein großes Problem bei der Koexistenz von IPv4 und IPv6 ist weniger die Abwärtskompatibilität von IPv6 als vielmehr die IPv4-Infrastruktur. Ein IPv4-Router kann keine IPv6-Pakete verstehen und wird diese folglich nicht verarbeiten. Es muss also erst eine gewisse Struktur aus IPv6-Komponenten – Dual Stack Nodes oder IPv6-Knoten, denen IPv4-compatible Adressen zugewiesen wurden - vorhanden sein um den Übertragungsweg für IPv6-Pakete sicherzustellen bevor eine sinnvolle Koexistenz stattfinden kann.

Um diesem Umstand Rechnung zu tragen, gibt es einen weiteren Mechanismus. Zwei weit entfernte IPv6-Knoten können durch das sogenannte Tunneln miteinander über IPV6 kommunizieren obwohl die dazwischenliegenden IPv4-Knoten dieses Protokoll nicht verstehen. Beim Tunneln werden die IPv6-Pakete in IPv4-Pakete verpackt um somit die kritischen Bereiche des Weges, auf denen kein IPv6 verstanden wird, passieren zu können. Der letzte IPv6-Knoten vor dem Tunnel erstellt ein neues IPv4-Paket mit der IP-Adresse des Knotens, an dem der Tunnel endet. Das eigentliche IPv6-Paket liegt

komplett als Payload in diesem neuen IPv4-Paket. Der Endpunkt des Tunnels entpackt das eigentliche IPv6-Paket und schickt es weiter. Anfangs- und Endknoten eines Tunnels müssen also IPV6-Knoten sein. Es ist denkbar, dass bei einer irgendwann erreichten weiten Verbreitung von IPv6 der Tunnelmechanismus umgekehrt wird und somit Tunnel für IPv4-Pakete entstehen.

Eine Koexistenz der beiden IP-Versionen ist durch das Design von IPv6 also grundsätzlich möglich. Hierbei stehen sogar verschiedene Möglichkeiten zur Verfügung. Trotzdem hält sich die Verbreitung von IPV6 – wie bereits oben erwähnt – bisher stark in Grenzen. Inwiefern die oben beschriebenen Mechanismen im Alltag effizient und reibungslos funktionieren, lässt sich also noch nicht sagen.

3. IPv4 in Java

IPv4 wird bereits seit Version 1.0 des JDK unterstützt. Die Klassen, in denen die IP-Unterstützung für Java implementiert ist, finden sich im Paket „java.net".
Die wichtigste Klasse heißt „InetAddress". Sie repräsentiert eine IP-Adresse in Java. Für diese Klasse existiert nur der Standardkonstruktor ohne Parameter. Die Zuweisung einer IP-Adresse zu einem Objekt der Klasse erfolgt über eine der folgenden drei Methoden.
Die Methode getByName(String Hostname) liefert zu einem String, der einen Hostnamen beschreibt, die zugehörige IP-Adresse als Objekt der Klasse InetAddress zurück. Diese Methode führt also eine DNS-Auflösung durch.
Da einem Host mehrere IP-Adressen zugeordnet sein können, gibt es die Methode getAllByName(String Hostname). Sie liefert ein Feld des Typs InetAddress zurück. Dieses Feld beinhaltet alle IP-Adressen, die dem als Parameter angegebenen Hostnamen zugeordnet sind. Auch hier kann der Hostname in einer der zwei oben beschriebenen Formen angegeben werden.
Als dritte Möglichkeit, eine IP-Adresse zu erhalten, existiert die Methode getLocalHost(), die die IP-Adresse des Rechners liefert, auf dem die Anwendung läuft. Diese Methode erwartet keine Parameter.
Alle drei Methoden werfen eine UnknownHostException, falls der als Parameter angegebene Host nicht existiert bzw. nicht gefunden werden kann. Falls ein SecurityManager aktiv ist und die von der Methode veranlasste Operation verbietet, werfen alle drei Methoden eine SecurityException.
Der bei zwei der drei Methoden erwartete Parameter Hostname kann in zwei Varianten angegeben werden. Sowohl eine textuelle Beschreibung wie beispielsweise „www.fh-bonn-rhein-sieg.de" als auch eine String-Darstellung der IP-Adresse, ist möglich. Im zweiten Fall wird nur auf die formale Korrektheit der Adresse geprüft! Eine Sicherstellung, dass ein Host mit dieser Adresse auch tatsächlich existiert, geschieht nicht. Die String-Darstellung einer IP-Adresse kann wiederum in verschiedenen Varianten geschehen. Die Zahl der Punkte zwischen den Zahlen darf zwischen null und drei liegen. Hat der String die Form „zahl.zahl.zahl.zahl", so wird der Parameter als IP-Adresse in gängiger Notation gewertet. Die Formen „zahl.zahl.zahl" sowie „zahl.zahl" dienen der einfacheren Handhabung von Klasse-B- bzw. Klasse-A-Netzen. Die jeweils letzte Zahl wird als 16-Bit bzw. 24 Bit-Angabe verstanden und der gesamte String dementsprechend in eine gängige IP-Adresse umgewandelt und weiterverarbeitet. Enthält der String eine Zahl, die nicht durch Punkte getrennt ist, so wird der Parameter wie gegeben weiterverarbeitet und keine Anpassung in Byte-Struktur vorgenommen.
Die Klasse InetAddress verfügt über einen Cache, der sowohl erfolgreiche als auch erfolglose DNS-Anfragen speichert. Dieser Cache dient zum Einen der

Performancesteigerung, zum Anderen aber auch dem Schutz vor DNS-Spoofing-Angriffen. Während erfolgreiche DNS-Anfragen standardmäßig unbegrenzt lange gespeichert werden, werden erfolglose Anfragen nur sehr kurz, standardmäßig 10 Sekunden, gespeichert und dann aus dem Cache entfernt. Sollte in einem speziellen Fall gewährleistet sein, dass eine DNS-Spoofing-Attacke unmöglich ist, so besteht für den Systemadministrator die Möglichkeit, die Speicherungszeit für erfolgreiche DNS-Anfragen herunterzusetzen.

Um eine InetAddress an einen Port zu binden, verwendet man in Java die Klasse InetSocketAddress. Dem Konstruktor dieser Klasse übergibt man entweder ein Objekt der Klasse InetAddress sowie eine Portnummer oder den Hostnamen als String und eine Portnummer. Im letzteren Fall wird die Adresse dann noch mittels DNS-Anfrage aufgelöst. Ist dies nicht möglich, so wir die Adresse auf „unresolved" gesetzt. Es ist auch möglich, den Konstruktor nur mit einer Portnummer aufzurufen. In diesem Fall wird die IP-Adresse automatisch auf die „Unspecified-address" (siehe Kapitel 2.1.4) gesetzt.

Über Klassen wie Socket, ServerSocket oder DatagrammSocket können diese Ports dann an Sockets gebunden werden. Hierdurch wird dann eine Kommunikation über diese Sockets ermöglicht. Diese ist aber nicht Bestandteil dieser Arbeit.

Obwohl Multicast im IPv4-Standard nicht explizit enthalten ist, implementiert Java Multicast bereits seit JDK 1.0. Dies geschieht über die Klasse MulticastSocket. Es können beliebig IP-Adressen hinzugefügt und entfernt werden [API].

4. IPv6 in Java

4.1 Unterstützung von Betriebsystemen

Bei einer solch systemnahen Technik wie IPv6 ist die beste Implementierung wertlos, wenn sie nicht auf dem Betriebssystem lauffähig ist. Daher wollen wir hier einen kleinen Blick darauf werfen, welche Betriebsysteme von IPv6 in Java unterstützt werden und bei welchen Systemen Probleme auftreten.
Die aktuelle Version von Java 2, Standard Edition (im folgenden J2SE), 1.4.1, unterstützt Solaris ab Version 8 und Linux ab Kernel-Version 2.1.2, dies entspricht RedHat 6.1.
Unter Linux ist es zu empfehlen, mindestens Kernel Version 2.4.0 zu verwenden, da IPv6 in diesen Versionen besser unterstützt wird. Auch ist darauf zu achten, dass ein Kernel verwendet wird, bei dem IPv6-Unterstützung aktiviert ist. Unter Umständen muss ein Kernel-Rebuild durchgeführt werden, um dies zu erreichen. Wichtig ist, hierbei zu erwähnen, dass bis Kernel-Version 2.4.2 die IPv6-Implementierung als „experimentell" gekennzeichnet ist. Außerdem ist es dabei notwendig, einige zusätzliche Netzwerkutilities zu installieren; darauf wird hier aber nicht weiter eingegangen.
Nicht unterstützt werden sämtliche Windows-Versionen. IPv6 ist in keiner Windows-Version standardmäßig eingebunden. Microsoft hat eine IPv6-Vorab-Technologie für Windows 2000 herausgebracht, diese jedoch unterstützt lediglich Seperate Stack Nodes; der Vorteil der Abwärtskompatibilität von IPv6 durch Dual Stack Nodes ist somit hinfällig. Bis zur Veröffentlichung von Service Pack 1 für Windows XP gab es keinen offiziellen Support seitens Microsoft für diese Technologie. Mit dem Service Pack 1 für Windows XP wurde jedoch eine vollständige, Dual-Stack-fähige Implementierung mit einhergehendem offiziellem Support veröffentlicht; J2SE wird diese in einer kommenden Version unterstützen [NetJava].

4.2 Änderungen an der Programmierung

Die Einführung und Nutzung von IPv6 in Java machten es nötig, bei der Implementierung der Netzwerkkommunikation in Java einige Änderungen durchzuführen, gleichzeitig durften diese aber nicht so ausfallen, dass die Kompatibilität zu früheren, nur IPv4 unterstützenden Versionen verloren geht. Wie diese Änderungen in der J2SE API aussehen und wie sie sich auswirken, wird in diesem Kapitel diskutiert.

4.2.1 Die Klasse InetAddress

Die Klasse InetAddress ist, wie bereits in Kapitel 3 erwähnt, eine der zentralen Klassen bei der Kommunikation mittels dem IP-Protokoll. Diese Klasse ist nun erweitert worden, um sowohl IPv4- als auch IPv6-Adressen unterstützen zu können. Zum einen wurden neue Methoden hinzugefügt, um zum Beispiel zu überprüfen, von welchem Typ eine Adresse ist, weiterhin wurden zwei neue Unterklassen geschaffen, Inet4Address und Inet6Address, die es ermöglichen sollen, zwischen den verschiedenen Adresstypen unterscheiden zu können, wo es nötig ist. Dies jedoch sollte – durch den Dual-Stack-Betrieb – im Normalfall nicht vonnöten sein. Wird ein Socket an die *unspecified address (::)* gebunden, so arbeitet er im Dual Stack Mode, er kann also Anfragen von sowohl IPv4- als auch IPv6-Hosts akzeptieren und bearbeiten; der Typ des Hosts bleibt dabei vollkommen transparent für die Anwendung. Lediglich wenn sie auf Funktionen oder Methoden mit protokoll-spezifischem Verhalten zugreifen muss, hat eine Anwendung die Möglichkeit, zwischen den verschiedenen Adresstypen zu unterscheiden.
Die neuen Methoden sind im einzelnen [API]:

`isAnyLocalAddress()`

Diese Methode prüft, ob die Adresse die Unspecified-Adresse ist. Der Rückgabewert ist boolean.

`isLoopbackAddress()`

Hier wird überprüft, ob die vorliegende Adresse eine Loopback-Adresse, also ::1 für IPv6 bzw. 127.0.0.1 für IPv4 ist. Der Rückgabewert ist ebenfalls boolean.

`IsLinkLocalAddress()`
`IsSiteLocalAddress()`

Diese Methoden dienen dazu, zu überprüfen, ob die vorliegende Adresse einen lokalen Typ hat, also nur Bedeutung in einem Subnetz bzw. einem separaten, lokalen Netzwerk und nicht global eindeutig ist. Auch hier wird ein Wert vom Typ boolean zurückgegeben.

`IsMCGlobal()`
`IsMCNodeLocal()`
`IsMCLinkLocal()`
`IsMCSiteLocal()`
`IsMCOrgLocal()`

Der Sinn dieser Methoden ist es, eine Multicast-Adresse nach ihrem Typ (Empfänger-Spektrum) zu überprüfen bzw. zu überprüfen, ob diese Adresse überhaupt eine Multicast-Adresse ist. Die Hierarchie der Multicast-Adressen, angefangen bei dem

Typ mit dem größten Empfängerspektrum, geht von Global über Org, Site, Link bis nach Node. Der Rückgabewert all dieser Methoden ist boolean.

`getCanonicalHostName()`

Diese Methode versucht, den *full qualified domain name (FQDN)*, der der zugrunde liegenden IP-Adresse zugeordnet ist, zu ermitteln und sie als String zurückzugeben. Ein FQDN ist eine vollständige Domain-Adresse; sie hat für jeden Domain-Level, vom Host bis zu Top-Level-Domain, einen Eintrag.
Diese Methode ist eine Best-Effort-Methode, das heißt, dass nicht garantiert werden kann, dass der FQDN ermittelt werden kann. Ein solcher Fall, in dem der FQDN nicht ermittelt werden kann, ist, wenn ein evtl. vorhandener SecurityManager der Anwendung eine Verbindung zu dem entsprechenden Host nicht gestattet. In diesem Fall wird die textuelle Darstellung der IP-Adresse zurückgegeben. [API]

`getByAddr(String host, byte[] addr)`

Diese statische Methode wird genutzt, um ein Objekt vom Typ InetAddress unter Angabe eines Hosts und einer IP-Adresse zu erzeugen. Der Host kann hierbei auch die textuelle Darstellung der IP-Adresse sein. Bei der Benutzung dieser Methode ist Vorsicht geboten, da nicht über einen Name Service validiert wird, ob die Host/IP-Zuordnung korrekt ist. Es ist also durchaus möglich, falsche „Paare" zu bilden, so dass bei der Kontaktierung eines Hosts über dessen Namen und über seine IP-Adresse verschiedene Adressaten angesprochen werden. Bei der textuellen Darstellung einer IPv6-Adresse muss diese entweder konform nach [RFC2732] oder [RFC2373] sein.
Das byte-Array muss für die Erzeugung einer IPv4-Adresse 4 Byte, für die Erzeugung einer Adresse nach IPv6 16 Byte lang sein. [API]

In der Klasse Inet6Address wurde zusätzlich zu den geerbten, an die Spezifikationen des IPv6-Protokolls angepassten Methoden, eine weitere Methode eingeführt, `isIPv4CompatibleAddress()`, in der überprüft wird, ob die vorliegende Adresse eine IPv4-kompatible Adresse (::w.x.y.z) ist.
Die neue Klasse Inet4Address repräsentiert, wie der Name schon sagt, eine Adresse nach dem IPv4-Protokoll. All ihre Methoden funktionieren nach den Spezifikationen für diese Adressen; somit unterscheidet sie sich ausschließlich im Namen und in den geerbten neuen Methoden der überarbeiteten Klasse InetAddress von der Klasse InetAdress früherer Versionen der J2SE.

4.2.2 Socket, ServerSocket und DatagramSocket

Aufgrund der Objektorientierung in Java ist es für die Klassen auf dem Socket-Level irrelevant, mit welcher speziellen Art von Adressen und Speicherstrukturen sie arbeiten. Somit sind auch keine neuen Klassen vonnöten. Sämtliche Socket-Optionen von IPv4 haben ihren Gegenpart in IPv6, somit war es lediglich nötig, die entsprechenden Methoden zu überladen, um auch IPv6 zu unterstützen.
Welcher Betriebsmodus (Dual Stack oder Seperate Stack) von den Sockets benutzt wird, hängt einerseits von dem gewählten Betriebssystem ab, da es Dual Stack-Betrieb unterstützen muss, damit dieser benutzt werden kann, andererseits von den Einstellungen des Benutzers, welcher Modus bevorzugt werden soll. Auf diese Einstellungsmöglichkeiten wird später in diesem Kapitel eingegangen.

4.2.3 MulticastSocket

Die Klasse MulticastSocket ist eine Erweiterung zur Klasse DatagramSocket; sie arbeitet demnach auch auf der Socket-Ebene. Auch hier sind dementsprechend lediglich die entsprechenden Methoden überladen worden, um IPv6-Socket-Optionen zu genügen.
Ein MulticastSocket ist ein UDP-Datagram Socket mit zusätzlichen Optionen, um sich Multicast-Gruppen anzuschließen oder sie zu verlassen [API]. In IPv4 wurde und wird ein Multicast-Interface durch eine IP-Adresse dargestellt, in Java kann es also durch ein Objekt der Klasse InetAddress repräsentiert und angesprochen werden. Um sich einem Multicast-Interface anzuschließen, existieren die Methoden `setInterface(InetAddress inf)` und `getInterface()`. Diese sind für IPv4 vollkommen ausreichend, und arbeiten auch weiterhin mit IPv6-MulticastSockets problemlos. Allerdings wurde beschlossen, dass dieses Interface in IPv6 durch einen Index beschrieben werden soll, wie in [RFC2553] spezifiziert, statt durch eine IP-Adresse (egal, welcher Version). Um diesem Konzept besser entsprechen zu können, wurde die Klasse NetworkInterface eingeführt. Ein solches Interface besteht aus einem Namen und einer Reihe von IP-Adressen, die diesem Interface zugeordnet sind [API]. Es gibt keinen Konstruktor für ein Objekt der Klasse NetworkInterface. Um ein Objekt dieses Typs zu erzeugen, existieren die Methoden `getByName(String name)` und `getByInetAddress(InetAddress addr)`. Die erste Methode sucht nach einem NetworkInterface mit dem entsprechenden Namen, die zweite sucht nach einem NetworkInterface, dass an die angegebene IP-Adresse gebunden ist. Sind mehrere Interfaces an diese IP gebunden, wird eines von diesen zurückgegeben. Welches, ist jedoch nicht definiert; dies ist abhängig von der zugrunde liegenden Systemkonfiguration, da die Methode, über die das Interface ermittelt wird, eine native Methode ist [code]. Die Klasse NetworkInterface beinhaltet noch zusätzliche Utility-

Methoden, um zum Beispiel die mit dem jeweiligen Interface verbundenen IP-Adressen aufzulisten. Darauf wird hier aber nicht weiter eingegangen.
Der Klasse MulticastSocket wurden mit der Einführung der Klasse NetworkInterface zusätzlich zu den überladenen, bereits existenten Methoden vier neue hinzugefügt. Dies sind einerseits die Methoden, um ein NetworkInterface zu setzen oder zu darauf zuzugreifen, `setNetworkInterface(NetworkInterface netIf)` und `getNetworkInterface()`, zum anderen zwei Methoden, die eine komplette Neuerung in der Java API darstellen. Mit Hilfe der Methoden `joinGroup(SocketAddress mcastaddr, NetworkInterface netIf)` und `leaveGroup(SocketAddress mcastaddr, NetworkInterface netIf)` ist es nun möglich, sich Multicast-Gruppen anzuschließen oder sie zu verlassen [Guide]. Beide Methoden konsultieren den SecurityManager derAnwendung, falls vorhanden, ob die gewünschte Operation gestattet ist. Ist dies nicht der Fall, wird eine SecurityException geworfen [API].

4.3 Netzwerkeinstellungen in Java

Mit der Einführung von IPv6 in Java wurden auch einige neue netzwerkspezifische Systemeinstellungen eingeführt, die hier nun erläutert werden.
Eine Gruppe dieser Einstellungen sind IP-spezifisch. Mit
`java.net.preferIPv4Stack`
lässt sich einstellen, dass der IPv4-Betrieb dem Dual Stack Mode vorgezogen wird. Auf einem System, dass IPv6 unterstützt, ist normalerweise der zugrundeliegende Socket einer Java-Applikation, die IP benutzt, ein IPv6-Socket [Guide]. Dies resultiert darin, dass ein Dual Stack Mode möglich ist, sprich es können sowohl IPv4- als auch IPv6-Adressen und somit Verbindungen behandelt werden (Hier darf allerdings nicht der Fehler begangen werden, eine Socket mit einem Knoten zu verwechseln. Diese Option ermöglicht nur den Dual Stack Betrieb, ob er nun ausgeführt wird, hängt vom System ab)
Wird die oben angegebene Einstellung auf true gesetzt, ist lediglich ein IPv4-Betrieb möglich; es können also nur Verbindungen zu anderen IPv4-Hosts aufgebaut bzw. von diesen angenommen werden. Ein IPv6-Betrieb ist demnach nicht mehr möglich.
Eine weitere Einstellungsmöglichkeit ist
`java.net.preferIpv6Addresses`.
Beim Dual-Stack-Betrieb, also wenn sowohl IPv4- als auch IPv6-Adressen behandelt werden können, werden normalerweise IPv4-Adressen bevorzugt. Diese Vorgehensweise wird angewandt, um zu garantieren, dass die Kompatibilität zu Anwendungen, die ausschließlich mit IPv4 arbeiten können oder auf die textuelle Darstellung der IPv4-Adressen (w.x.y.z) angewiesen sind, bestehen bleibt.

Wird diese Einstellung auf true gesetzt, kann erreicht werden, dass diese Bevorzugung umgekehrt wird, also vornehmlich IPv6-Adressen behandelt werden. Dies ist zum Beispiel sinnvoll, wenn eine Anwendung in einer Umgebung getestet und eingesetzt werden soll, in der sie mit IPv6-Diensten arbeitet [Guide].

Eine weitere Gruppe von Netzwerkeinstellungen, die mit der Einführung von IPv6 hinzugekommen sind, erlaubt es, die Einstellungen für die Namespace-Provider zu ändern. Dies soll aber nicht Bestandteil dieser Arbeit sein.

5. Zusammenfassung

Dieses Kapitel soll die wichtigsten Punkte dieser Seminararbeit zusammenfassen und somit einen kurzen Gesamtüberblick über die Thematik geben.
Unseres Erachtens ist die Implementierung von IPv6 in Java recht geschickt gelöst, da sehr auf Transparenz und damit auf Benutzerfreundlichkeit für den Programmierer geachtet wurde. Programme, die für alte JDK-Versionen und damit für IPv4 geschrieben wurden, sollten auch unter JDK 1.4 problemlos laufen. Grund hierfür ist die standardmäßige Unterstützung einer Dual Stack Node unter Java. In neuen Programmen kann durch die Benutzung der Konstruktoren der Unterklassen Inet6Address und Inet4Address aber auch explizit zwischen den beiden Adresstypen unterschieden werden. Ähnlich verhält es sich mit Sockets, welche die Klasse InetAddress benutzen. Da sämtliche Socket-Optionen in IPv4 auch in IPv6 enthalten sind, reichte hier das Überladen der Methoden aus. Der Programmierer kann also auch mit Sockets wie bisher weiterarbeiten.
Mit der Version 1.4 sind die Standards [RFC2373], [RFC2553] und [RFC 2732] vollständig in Java eingebunden. Somit bleiben keine Lücken in der Implementierung. Die oben beschriebenen Funktionen von IPv6 wie Tunneln oder der „stateless host autoconfiguration"-Mechanismus [RFC2462] spielen in Java insofern keine Rolle, als das dies Aufgaben des Betriebssystems sind. Daher gibt es zur Zeit auch noch Probleme mit einigen Betriebssystemen, die IPV6 noch nicht vollständig unterstützen.
Wie in der geschichtlichen Ausarbeitung beschrieben, ist IPv6 bisher nicht sehr weit verbreitet. Die Einbindung von IPv6 in Java war ein weiterer Schritt zur Verbreitung des neuen Protokolls. Dass ein Java-Programm, welches IPv6 benutzt, standardmäßig eine Dual Stack Node ist, wird sicherlich seinen Beitrag zur Durchsetzung von IPv6 leisten, garantiert aber gleichzeitig auch die weitere Unterstützung von IPv4.

6. Literatur

[API] SUN Microsystems, http://java.sun.com/j2se/1.4.1/docs/api/. *API-Dokumentation des J2SDK, Version 1.4.1_02.*

[code] SUN Microsystems, Source-Code des J2SDK, Version 1.4.1_02

[Guide] SUN Microsystems, http://java.sun.com/j2se/1.4.1/docs/guide/net/ipv6_guide. *Networking IPv6 User Guide for J2SDK/JRE 1.4.*

[hoff97] Robert Hoffmann, *Das Internet-Protokoll Version 6 - Migrationsverfahren und Managementanforderungen*. http://wwwhegering.informatik.tu-muenchen.de/common/Literatur /MNMPub/Diplomarbeiten/hoff97/HTML-Version/node1.html, München, 1997.

[jjoller] Hochschule für Technik Rapperswil, http://informatik.hsr.ch/Content/Gruppen/Doz/jjoller/ws2002ss2003/vs1/Folien/PPTKapitel004.pdf. *DNS Domain Name System & IP „Internet Protokoll".*

[Kassel] Universität Kassel, http://www.neuro.informatik.unikassel.de/Vorlesungen/SeminarRechnernetze/IPV4IPV6/seite12.htm. *Das IPv6 Adressformat.*

[laynet] Lay Network, http://www.laynetworks.com/users/webs/IPv6.htm. *IPv6 or IPng (IP Next generation).*

[netBSD] netBSD.org, http://www.netbsd.org/de/Documentation/network/ipv6/#diff_ipv4. *Eine kurze Geschichte von IPv6 und einige Hauptmerkmale.*

[NetJava] SUN Microsystems, http://developer.java.sun.com/developer/community/chat/JavaLive/2002/jl1203.html. *New Networking Support in J2SE v 1.4.*

[RFC2373] IETF, http://www.ietf.org/rfc/rfc2373.txt. *IP Version 6 Addressing Architecture.*

[RFC2460] IETF, http://www.ietf.org/rfc/rfc2460.txt. *Internet Protocol, Version 6 (IPv6) Specification.*

[RFC2462] IETF, http://www.ietf.org/rfc/rfc2462.txt. *IPv6 Stateless Address Autoconfiguration.*

[RFC2553] IETF, http://www.ietf.org/rfc/rfc2553.txt. *Basic Socket Interface Extensions for IPv6.*

[RFC2732] IETF, http://www.ietf.org/rfc/rfc2732.txt. *Format for Literal IPv6 Addresses in URL's.*

[RFC791] IETF, http://www.ietf.org/rfc/rfc791.txt. *Darpa Internet Program Protocol Specification.*

[Sch] Michael Schmitz: *IPv4 und IPv6*, www-i4.informatik.rwth-aachen.de/content/teaching/ proseminars/sub/2002_2003_ws_docs/ip.pdf, RWTH Aachen, Lehrstuhl für Informatik