In diesem Kapitel:
- *Was ist ein Multicast Socket?*
- *Arbeiten mit Multicast Sockets*
- *einfache Beispiele*

13

Multicast Sockets

13.1. Was ist ein Multicast Socket?

Alle Sockets, die wir bisher kennen, sind *unicast* Sockets. Diese erlauben eine Punkt-zu-Punkt Verbindung, also eine Kommunikation mit zwei klar definierten Endpunkten, einem Sender und einem Empfänger. Eine 1:1 Kommunikation ist die übliche Art der Kommunikation.

Allerdings gibt es auch eine 1:N Kommunikation, zum Beispiel wenn ein Politiker zu seinem "Volk" spricht. In diesem Falle wird eine ganze Gruppe angesprochen. Diese Art der Kommunikation kenne Sie auch vom Radio und dem Fernsehen.

Im Marketing ist der grosse Trend hin zu einem 1:1 Marketing: jedes Individuum soll möglichst direkt und individuell angesprochen werden. Die Informatik Support Systeme für diese neue Art des Marketings bezeichnet man als CRM = Customer Relations Management Systeme. Den gleichen Trend sehen sie auch im Fernsehen : das digitale Fernsehen soll auf jede Person zugeschnitten werden können.

Broadcasting wird zwar von TCP/IP unterstützt. Aber die Möglichkeiten sind sehr limitiert. Insbesondere muss auch der Router Broadcasting unterstützen. Diese unterstützen in der Regel ein Broadcasting lediglich in lokalen Netzen oder Subnetzen. Der Grund liegt an der limitierten Bandbreite. Wenn jeder Lipps und Zimmermann dieser Welt seine MP3 und Urlaubsvideos über das Internet der Allgemeinheit senden würde, könnte das Internet sehr schnell zusammen brechen.

Wenn 1000 Internet Benutzer ein bestimmtes RealVideo oder RealAudio ansehen möchten, dann muss dieses an jeden der Benutzer übermittelt werden. Dies ist nicht eine sehr effiziente Art der Kommunikation.

Eine effizientere Art der Kommunikation ist die Nutzung der *connection trees* (Usenet News und CU-SeeMe). Dabei werden Informationen von einem Server auf weitere Server kopiert. Diese können die Daten weiter kopieren und an Kunden senden.

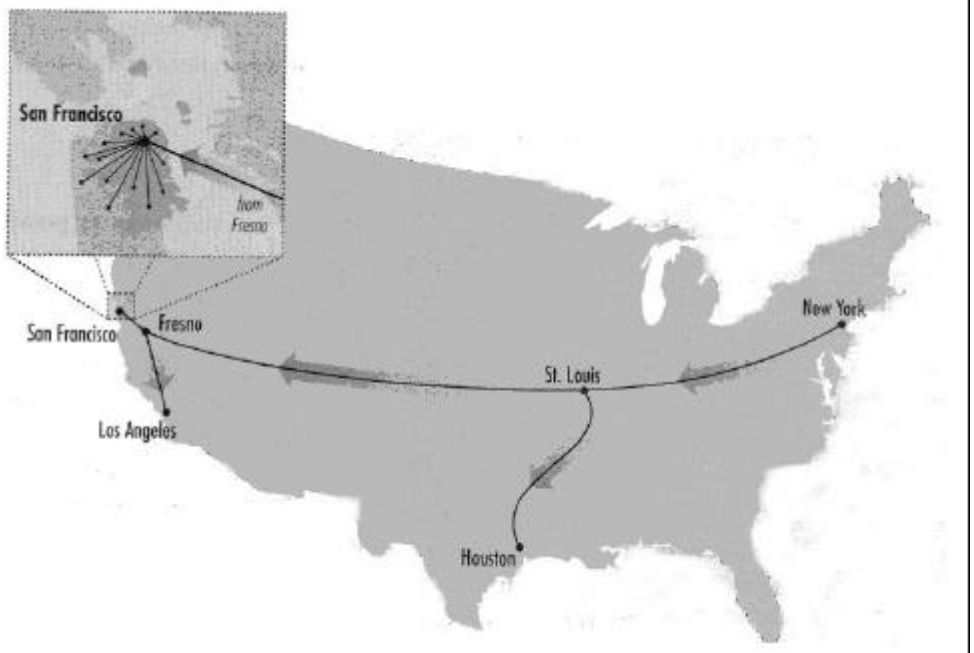
Jeder Kunde nutzt nun den nächstgelegenen Server. Dadurch wird der Kommunikationsaufwand reduziert, in der Regel. Kommt ein neuer Server hinzu, dann muss er sich an einer passenden Stelle in den Baum eintragen. Diese Wahl trifft der Router, wir brauchen uns nicht darum zu kümmern.

Multicasting erlaubt es, mehr als einen Kunden, aber eben nicht einfach jeden Internet Anwender, anzusprechen und mit diesen Kunden gleichzeitig (multicasting) zu kommunizieren.
Wir können also mit Hilfe von Multicasting ein "public meeting" organisieren.

13.1.1. Beispiel

Wenn eine Videoübertragung über das Internet aus New York an eine Gruppe nach San Francisco geschieht und zu einem späteren Zeitpunkt eine Gruppe in Los Angeles dazu

kommen möchte, dann werden die Router einen Kommunikationsknoten, zum Beispiel irgend wo in der San Francisco Bay Area oder südlich davon (Fresno zum Beispiel) auswählen und die Videodaten ab dort an die zwei Gruppen senden. Die transkontinentale Kommunikation über Nordamerika wird daher nicht dupliziert. Das Verfahren ist aber nur anwendbar, falls die zwei Gruppen das selbe Video anschauen.
Wenn nun auch noch eine Gruppe in Houston das Video sehen möchte, dann werden die



Router vermutlich einen weiteren Kommunikations-Verzweigungsknoten suchen, etwa in St. Louis, einigermassen in der Mitte der USA. Dann hätten wir folgenden Kommunikationsbaum:

1) New York - St. Louis
2) St. Louis - Houston
3) St. Louis - Fresno
4) Fresno - San Francisco
5) Fresno - Los Angeles

Insgesamt würden wir immer noch beachtlich viel Kommunikationsaufwand haben, aber viel weniger als wenn wir folgenden Baum hätten:

1) New York - Houston
2) New York - Los Angeles
3) New York - San Francisco

13.1.2. Mögliche Anwendungen

Naheliegende Anwendungen für ein Multicast System sind:

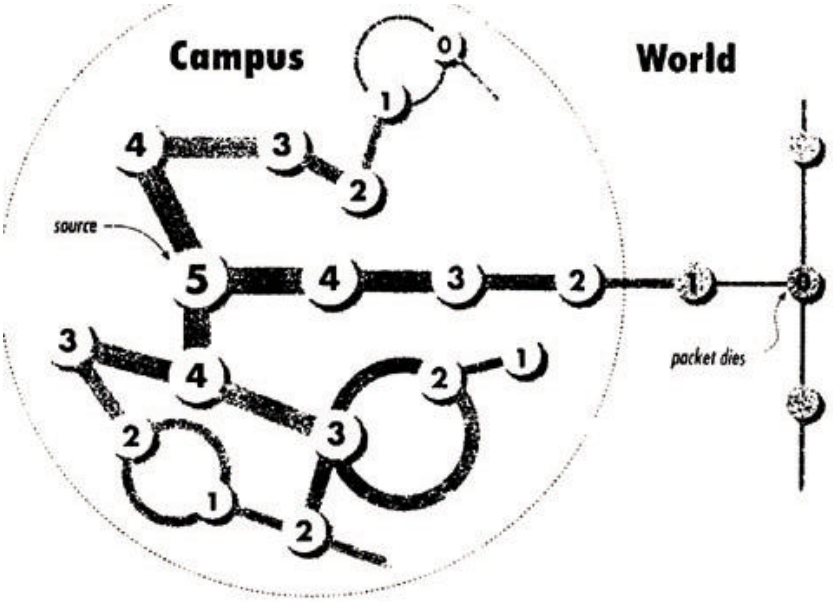
- Audio on Demand
- Video on Demand
- Multiplayer Games
- massiv parallele Berechnungen
- Multipersonen Konferenzsysteme; Multigruppen Konferenzsysteme
- effizienteres Internet Caching mit Hilfe eines weltweit tätigen Multicast Server Systems

13.2. Multicasting im Internet

Multicasting wurde so gut wie möglich in das Internet integriert. Die meiste Arbeit muss von den Routern gemacht werden und sollte transparent sein für die Anwendungen.

Eine Anwendung sendet ein Datagram Paket an eine multicast Adresse, analog zu einer Internet Adresse. Die Router sorgen dafür, dass das Paket an alle Hosts der Multicast Gruppe gesendet wird. Damit die Kommunikation problemlos funktioniert, müssen die Router also passend konfiguriert sein!

Bei Multicast Paketen muss man einen Headerparameter, den TTL (Time To Live) , spezifizieren. Der TTL Wert gibt Auskunft darüber, wie viele Router das Datagram besuchen darf, bevor es vernichtet wird.



Ein möglicher Wert für TTL ist 5, die Datagramme werden also maximal fünf Router weit transportiert.

13.2.1. Multicast Adressen und Gruppen

Eine Multicast Adresse ist die Adresse einer Gruppe von Hosts, einer Multicast Gruppe. Multicast Adressen sind IP Adressen im Adressbereich:

- 224.0.0.0 bis 239.255.255.255

Das heisst: (siehe Kapitel 2 (PowerPoint / PDF))

- Alle diese Adressen beginnen binär mit 1110.
  Byte 0 | Byte 1 | Byte 2 | Byte 3
  0111 **** **** ****
- diese Adressen werden als Class D Adressen bezeichnet

Wie jede IP Adresse besitzt auch eine Multicast Adresse einen Hostnamen.

- 224.0.1.1 heisst NTP.MCAST.NET und ist die Adresse des Network Time Protocol Distributed Service
- 224.0.0.1 ist ALL-SYSTEMS.MCAST.NET, eine Multicast Gruppe, welche Multicasting in lokalen Subnetzen unterstützt.
- zum Glück gibt es keine Multicast Adresse, welche Daten an alle Internet Hosts sendet.

- alle Adressen, die mit 224 beginnen (224.0.0.0 bis 224.0.0.255) sind für Router Protokolle und andere low-level Aktivitäten reserviert (Gateway Discovery, Group Membership Reporting,...). Routers senden keine Datagramme an solche Zielhosts.

13.2.2. Allgemein gültige Multicast Adressen

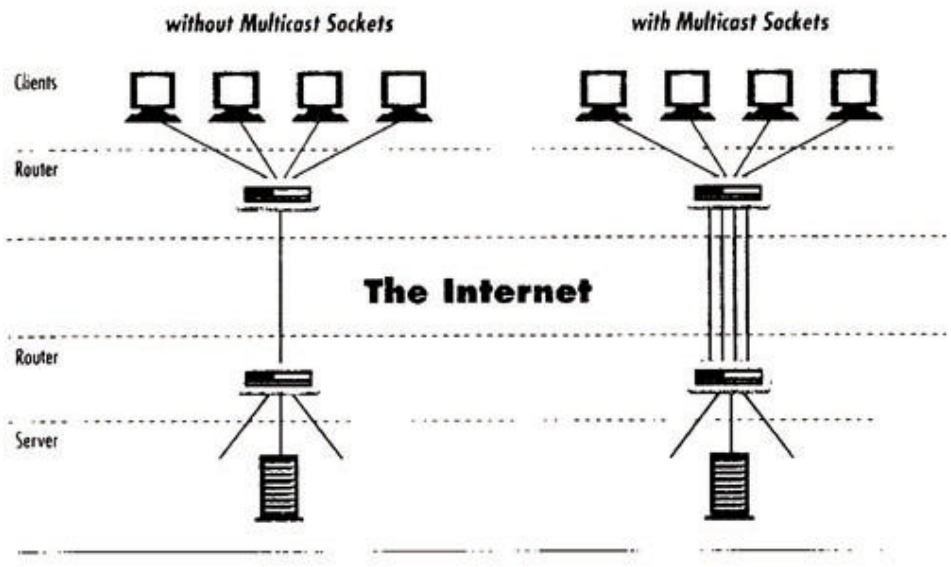
Domain Name	**IP Adresse**	**Einsatzmöglichkeit**
BASE-ADDRESS.MCAST.NET	224.0.0.0	reservierte Basisadresse. Diese Adresse wird keiner Multicast Gruppe zugeordnet
ALL-SYSTEMS.MCAST.NET	224.0.0.1	alle Systeme im lokalen Subnet
ALL-ROUTERS.MCAST.NET	224.0.0.2	alle Routers im lokalen Subnet
DVMRP.MCAST.NET	224.0.0.4	alle Distance Vector Multicast Routing Protocol Router. RFC1075 beschreibt eine alte Version. Eine neue Version finden Sie im Internet.
MOBILE-AGENT.MCAST.NET	224.0.0.12	Multicast Gruppe, die es einem Client erlaubt einen DHCP Server oder Relay Agent in einem lokalen Subnet zu finden.
PIM-ROUTERS.MCAST.NET	224.0.0.13	alle Protokoll unabhängigen Multicast Router im lokalen Subnet
RSVP-ECAPSULATION.MCAST.NET	224.0.0.14	RSVP-ENCAPSULATION : Resource reSerVation setup Protocol erlaubt einem Benutzer eine bestimmte Bandbreite des Internets im Voraus für ein Event zu reservieren.
NTP.MCAST.NET	224.0.1.1	Network Time Protocol
SGI-DOG.MCAST.NET	224.0.1.2	Silicon Graphics Dogfight Spiel
NSS.MCAST.NET	224.0.1.6	Name Service Server
AUDIONEWS.MCAST.NET	224.0.1.7	Audio News Multicast
SUB-NIS.MCAST.NET	224.0.1.8	Sun's NIS+ Information Service
MTP.MCAST.NET	224.0.1.9	Multicast Transport Protocol
IETF-1-LOW-AUDIO.MCAST.NET	224.0.1.10	Channel 1 low-quality Audio von IETF Meetings
IETF-1-AUDIO.MCAST.NET	224.0.1.11	Channel 1 high-quality Audio von IETF Meetings
IETF-1-VIDEO.MCAST.NET	224.0.1.12	Channel 1 Video von IETF Meetings
IETF-2-LOW-AUDIO.MCAST.NET	224.0.1.13	Channel 2 low-quality Audio von IETF Meetings
IETF-2-AUDIO.MCAST.NET	224.0.1.14	Channel 2 high-quality Audio von IETF Meetings
IETF-2-VIDEO.MCAST.NET	224.0.1.15	Channel 2 Video von IETF Meetings

MUSIC-SERVICE.MCAST.NET	224.0.1.16	MUSIC-SERVICE
SEANET-TELEMETRY.MCAST.NET	224.0.1.17	Telemetrie Daten für das SeaNet Projekt (Internet auf Schiffen)
SEANET-IMAGE.MCAST.NET	224.0.1.18	SeaNet Bilder
MLOADD.MCAST.NET	224.0.1.19	MLOADD misst den Traffic durch ein oder mehrere Netzwerke pro Sekunde
EXPERIMENT.MCAST.NET	224.0.1.20	für Experimente, die auf lokale Subnetze beschränkt sind
XINGTV.MCAST.NET	224.0.1.23	XING Technology's Streamworks TV Multicasting
MICROSOFT.MCAST.NET	224.0.1.24	wird von WINS Server verwendet, um weitere Server zu finden (WINS =Windows Internet Name Service)
NBC-PRO.MCAST.NET	224.0.1.25	NBC Professional News: live Video rund um die Uhr mit News für die Finanzmärkte (veraltet) http://www.desktop.nbc.com/pro/pro.html
NBC-PFN.MCAST.NET	224.0.1.26	NBC private Finanzinformationen http://www.desktop.nbc.com/pfn/pfn.html (veraltet)
MTRACE.MCAST.NET	224.0.1.32	Multicast Version von TRACEROUTE
weitere Gruppen	224.0.6.000 bis 224.0.6.127	Dieser Block ist für das ISIS Projekt reserviert http://www.stratus.com/ISIS/www.html (veraltet)
	224.0.9.000 bis 224.0.9.255	Dieser Block wird für das Internet Railroad Projekt verwendet, ein Projekt die Regierungen mit einer 45 MB / s Leitung um den Globus zu verknüpfen http://amsterdam.park.org/About/InternetRailroad/index.text.html
	224.2.0.0 bis 224.2.255.255	MBONE : diese Adressen sind reserviert für Multimedia Konferenzen
	224.2.2.2	Port 4000 dieser Adresse wird benutzt, um aktuell verfügbare MBONEs zu broadcasten. Mit dem X Windows Programm sd kann dies abgefragt werden.

13.2.3. Routers und Routing

Falls ein Server eine Meldung an mehrere Clients senden muss, dann haben wir folgende Möglichkeiten:

1. der Server sendet die Daten an den Router (Router 1); dieser sendet die Daten an einen Router (Router 2) eines lokalen Netzes, an dem die Clients hängen.
   Der zweite Router "verteilt" die Daten, das heisst die Daten werden vom zweiten Router an alle Clients gesendet
2. der Server sendet die Daten an Router 1; dieser sendet sie an lediglich einen Client



Version 2 verwendet offensichtlich wesentlich mehr Bandbreite des Internets. Video und Audio Applikationen rücken damit in en Bereich des Machbaren.

Die Realität sieht natürlich immer wesentlich komplexer aus!

In Java werden uns aber die meisten Probleme abgenommen, sofern wir einen Multicast Socket kreieren. Die Haupteinschränkungen für Multicasting kommen jedoch von den Routern. Falls diese kein Multicasting erlauben, dann können wir gar nichts dagegen machen, ausser die Router umzukonfigurieren. Falls die Router Multicasting ausnutzen, dann kann auch der gesamte Traffic optimiert werden, durch geschicktes ausnutzen von Multicasting.

Der Support von Multicasting ist auch abhängig von der TCP/IP Implementation und vom verwendeten Betriebsystem. Windows NT und Windows 95/98 TCP Stacks unterstützen Multicasting. Beim Betriebssystem Solaris (SunOS 4.1.x) gab es Probleme : Multicasting funktionierte nicht. Wie die aktuelle Implementation von Solaris funktioniert ist mir leider nicht bekannt. MacTCP unterstützt kein Multicasting, aber es gibt auch eine Implementation von TCP (Open Transport), welche Multicasting auf den Macs implementiert.

13.3. Arbeiten mit Multicast Sockets

Schauen wir uns einmal die Beschreibung des APIs an:

13.3.1. java.net Class MulticastSocket

```
java.lang.Object
  |
  +--java.net.DatagramSocket
        |
        +--java.net.MulticastSocket
```

---

public class **MulticastSocket**
extends DatagramSocket
Die multicast Datagram Socket Klasse wird für das Senden und Empfangen von IP multicast Paketen eingesetzt. Ein MulticastSocket ist ein (UDP) DatagramSocket, mit zusätzlichen Fähigkeiten, zum joinen von"Gruppen" anderer multicast Hosts auf dem Internet.
Eine multicast Gruppe wird mit Hilfe der Class D IP Adressen spezifiziert, also IP Adressen im Bereich `224.0.0.1` bis `239.255.255.255`, inklusive, und einem Standard UDP Port. Man tritt einer multicast Gruppe bei, indem man zuerst ein MulticastSocket kreiert, mit der gewünschten Port Nummer. Dies geschieht mit Hilfe der Methode `joinGroup(InetAddress groupAddr)` :

```
// join a Multicast group and send the group salutations
...
byte[] msg = {'H', 'e', 'l', 'l', 'o'};
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket s = new MulticastSocket(6789);
s.joinGroup(group);
DatagramPacket hi = new DatagramPacket(msg, msg.length,
                            group, 6789);
s.send(hi);
// get their responses!
byte[] buf = new byte[1000];
DatagramPacket recv = new DatagramPacket(buf, buf.length);
s.receive(recv);
...
// OK, I'm done talking - leave the group...
s.leaveGroup(group);
```

Falls man eine Meldung an eine Multicast Gruppe sendet, dann empfangen **alle** Host dieser Gruppe diese Meldung. (sofern die time-to-live gross genug ist, das Paket also nicht vorher stirbt). Ein Socket muss nicht Mitglied einer Multicast Gruppe sein, um an eine Gruppe ein Datagramm senden zu können.
Falls der Socket Mitglied ist, dann empfängt er auch alle Meldungen, die an die Mitglieder der Gruppe (von Mitgliedern oder von aussen) gesendet werden. Ein Socket kündigt seine Mitgliedschaft bei einer Gruppe, indem er die leaveGroup(InetAddress addr) Methode aufruft.
Natürlich können mehrere MulticastSocket's sich gleichzeitig bei einer multicast Gruppe anmelden und damit alle Gruppen Datagrame erhalten.
Applets können (zurzeit) nicht mit Hilfe von Multicast Sockets arbeiten.
Seit:
JDK1.1

13.3.1.1. Konstructor Zusammenfassung

MulticastSocket() kreiert einen Multicast Socket.	
MulticastSocket(int port) kreiert einen Multicast Socket und bind ihn an einen spezifischen Port.	

13.3.1.2. Method Zusammenfassung

InetAddress	**getInterface**() Bestimmen der Adresse des Network Interface für die Multicast Pakete.
int	**getTimeToLive**() Liefert die Standard time-to-live (TTL) für Multicast Pakete dieses Sockets..
byte	**getTTL**() **veraltet.** neu: *getTimeToLive Methode.*
void	**joinGroup**(InetAddress mcastaddr) einer Multicast Gruppe beitreten .Diese Methode hängt zusammen mit setInterface.
void	**leaveGroup**(InetAddress mcastaddr) eine Multicast Gruppe verlassen.
void	**send**(DatagramPacket p, byte ttl) Sendet ein Datagramm Paket zu einer Destination, mit einem TTL (time- to-live) welches sich von der Standard TTL des Sockets unterscheidet..
void	**setInterface**(InetAddress inf) Setzen des Netzwerk Interfaces, falls möglich (siehe unten).
void	**setTimeToLive**(int ttl) Setzt die Standard time-to-live (TTL) für diesen Socket..
void	**setTTL**(byte ttl) **veraltet.** *neu: setTimeToLive Methode l.*

13.3.1.3. Methoden, die von java.net.DatagramSocket geerbt werden

close, connect, disconnect, getInetAddress, getLocalAddress, getLocalPort, getPort, getReceiveBufferSize, getSendBufferSize, getSoTimeout, receive, send, setReceiveBufferSize, setSendBufferSize, setSoTimeout

13.3.1.4. Methoden, die von java.lang.Object geerbt werden

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

13.3.1.5. Konstruktor Detail

MulticastSocket

```
public MulticastSocket()
                throws IOException
```

Kreiere einen Multicast Socket.
Falls ein Security Manager vorhanden ist, dann wird dessen `checkListen` Methode zuerst ausgeführt, mit 0 als Argument, um sicher zu sein, dass diese Operation erlaubt ist.
Allenfalls resultiert eine SecurityException.

Throws:

SecurityException - falls ein Security Manager installiert ist und dessen `checkListen` Methode die Operation nicht erlaubt.

Siehe auch:

`SecurityManager.checkListen(int)`

MulticastSocket

```
public MulticastSocket(int port)
                throws IOException
```

Kreiert einen Multicast Socket und bindet ihn an einen spezifischen Port.
Falls ein Security Manager installiert ist, wird zuest dessen `checkListen` Methode ausgeführt, mit dem Port als Argument, um zu prüfen, ob die Operation auch erlaubt ist.Allenfalls resultiert daraus eine SecurityException.

Parameter:

`port` - zu benützender Port

Throws:

SecurityException - falls der Security Manager mit Hilfe seiner `checkListen` Methode festgestellt hat, dass die Operation nicht erlaubt ist.

Siehe auch:

`SecurityManager.checkListen(int)`

13.3.1.6. Methoden Detail

setTTL

```
public void setTTL(byte ttl)
      throws IOException
```

veraltet. *neu: setTimeToLive.*
Setzt die Standard time-to-live (TTL) für Multicast Pakete dieses Sockets. Die TTL setzt die IP time-to-live für `DatagramPackets`, welche an eine MulticastGroup gesendet werden:diese gibt an, wie viele "hops" das Paket auf dem Netzwerk befördert wird, bevor es stirbt.
ttl ist eine **unsigned** 8-bit Grösse, und **muss** im Bereich `0 <= ttl <= 0xFF sein`.
Parameter:
`ttl` - die time-to-live

---

setTimeToLive

```
public void setTimeToLive(int ttl)
          throws IOException
```

Setzt die Standard time-to-live (TTL) für Multicast Pakete dieses Sockets. Die TTL setzt die IP time-to-live für `DatagramPackets`, welche an eine MulticastGroup gesendet werden:diese gibt an, wie viele "hops" das Paket auf dem Netzwerk befördert wird, bevor es stirbt.
ttl ist eine **unsigned** 8-bit Grösse, und **muss** im Bereich `0 <= ttl <= 0xFF sein`, sonst wird eine Exception geworfen
Parameters:
`ttl` - the time-to-live

---

getTTL

```
public byte getTTL()
throws IOException
```

veraltet. *neu getTimeToLiveMethode.*
Liefert die Standard time-to-live für Multicast Pakete dieses Socket.

---

getTimeToLive

```
public int getTimeToLive()
      throws IOException
```

Liefert die Standard time-to-live für Multicast Pakete dieses Socket.

---

joinGroup

```
public void joinGroup(InetAddress mcastaddr)
   throws IOException
```

Einer Multicast Gruppe beitreten.Diese Methode hängt mit der `setInterface` Methode zusammen.
Falls es einen Security Manager gibt, dann wird zuerst die `checkMulticast` Methode mit der `mcastaddr` als Argument aufgerufen.
Parameter:
`mcastaddr` - Multicast Adresse , der beigetreten werden soll.
Throws:
IOException - falls die Adresse keine Multicast Adresse ist oder der Beitritt verboten ist.
SecurityException - falls ein Security Manager installiert und dessen `checkMulticast` Methode den Beitritt zur Multicast Gruppe verbietet.
Siehe Auch:
`SecurityManager.checkMulticast(InetAddress)`

---

leaveGroup

```
public void leaveGroup(InetAddress mcastaddr)
 throws IOException
```

verlassen einer Multicast Gruppe. Falls `setInterface` eingesetzt wurde, kann sich das Verhalten ändern.
Falls ein Security Manager eingesetzt wird, dann wird zuerst dessen `checkMulticast` Methode ausgeführt, mit `mcastaddr` als Argument.
Parameter:
`mcastaddr` - die Multicast Adresse, die verlassen werden soll
Throws:
IOException - im Falle eines Fehlers beim Verlassen, oder falls die Adresse keine Multicast Adresse ist.
SecurityException - falls ein Security Manager installiert ist und dessen `checkMulticast` Methode die Operation nicht erlaubt.
Siehe Auch:
`SecurityManager.checkMulticast(InetAddress)`

---

setInterface

```
public void setInterface(InetAddress inf)
throws SocketException
```

Setzt das Multicast Netzwerk Interface

---

getInterface

```
public InetAddress getInterface()
       throws SocketException
```

bestimmt das Interface einer Multicast Adresse (siehe unten)

```
 public void send(DatagramPacket p,
byte ttl)
          throws IOException
```

Sendet ein Datagramm Paket zu einer Destination, mit einer TTL (time- to-live) die sich von der Standard TTL für diesen Socket unterscheidet. In der Regel sollte TTL für einen Socket fix gesetzt werden!. Die Methode hat keinen Einfluss auf die Standard TTL für diesen Socket. `setInterface` kann einen direkten Einfluss auf TTL haben.

Fall ein Security Manager installiert wurde, dann werden erst dessen Security Checks durchgeführt.

Erster Test: falls `p.getAddress().isMulticastAddress()` true liefert, dann wird die Methode `checkMulticast` des Security Managers aufgerufen, mit `p.getAddress()` und `ttl` als Argumente. Falls dies false liefert, dann wird die Methode `checkConnect` method des Security Managers aufgerufen, mit `p.getAddress().getHostAddress()` und `p.getPort()`als Argumente. Jeder Aufruf einer Security Manager Methode kann in einer SecurityException enden, falls die Operation nicht erlaubt ist..

Parameter:

`p` - ist das Paket welches gesendet werden soll. Das Paket sollte die Destination Multicast ip Adresse und die Daten , die gesendet werden sollen, enthalten. Wie bereits erwähnt, braucht man nicht Mitglied der Multicast Gruppe zu sein, um Meldungen an die Gruppe senden zu können.

`ttl` - optionaler Parameter, per Default 1.

Throws:

IOException - falls ein Fehler beim Setzen von ttl auftrat.

SecurityException - falls ein Security Manager existiert und dessen Methode `checkMulticast` oder `checkConnect` Methode das Senden von Datagrammen verbietet

Siehe Auch:

`DatagramSocket.send(java.net.DatagramPacket)`,
`DatagramSocket.receive(java.net.DatagramPacket)`,
`SecurityManager.checkMulticast(java.net.InetAddress, byte)`,
`SecurityManager.checkConnect(java.lang.String, int)`

Soweit die Beschreibung des APIs.
Schauen wir uns einige Programm Fragmente und Programmbeispiele an:

13.4. Multicast Programmfragmente und Anwendungsprogramme

Im folgenden schauen wir uns mehrere Programmfragmente an, die grundsätzliche Programmstrukturen aufzeigen sollen. Anschliessend werden wir zwei voll funktionsfähige Multicast Programme kennen lernen (Sniffer und Sender).

13.4.1. Programmfragment : Multicast Konstruktor

```
//Titel:       MulticastKonstruktor
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Aufbau eines Konstruktor Programmteils
//1) Aufruf des Konstruktors
//Alternativen:
//falls mit einem Interface gearbeitet werden muss / kann,
//dann  muss auch noch das InetAddress Objekt angegeben werden
//Ein Beispiel dazu folgt noch.
package Beispiel13_1MulticastKonstruktor;

import java.net.*;
import java.io.*;


public class MulticastKonstruktor {
  public static void main(String[] args) {

    try {
      MulticastSocket ms = new MulticastSocket();
      System.out.println(ms);
    }
    catch (SocketException se) {
      System.err.println(se);
    }
    catch (IOException ioe) {
      System.err.println(ioe);
    }

  }

}
```

Das Programm macht offensichtlich nicht besonders viel sinnvolles.

Schauen wir uns den zweiten Konstruktor an:

13.4.2. Multicast Konstruktor mit Portangabe

```
//Titel:       Multicast Konstruktor mit Portangabe
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einfaches Programmfragment welches den
//Multicast Konstruktor mit Portangabe verwendet.
package Beispiel13_2MulticastKonstruktorMitPortAngabe;

import java.net.*;
import java.io.*;

public class MulticastPortConstructor {

  public static void main(String[] args) {

    try {
      MulticastSocket ms = new MulticastSocket(2048);
      System.out.println(ms);
    }
    catch (SocketException se) {
      System.err.println(se);
    }
    catch (IOException ie) {
      System.err.println(ie);
    }

  }

}
```

Bemerkung :
hier muss unbedingt die IOException abgefangen werden. Diese bedingt den Import des IO Packages.

13.4.3. setInterface(InetAddress interface) Programmfragment

```
MulticastSocket ms;
InetAddress ia;
try {
        ia = new InetAddress("meinHost.com");
        ms = new MulticastSocket(2048);
        ms.setInterface(ia);    // das Interface ist ein InetAddress Objekt!
        System.out.println(ms);
} catch (UnknownHostException ue) {
        System.err.println(ue);
```

```
}
```

Bemerkung:
um sicher zu sein, dass das Interface auch gesetzt wurde, sollte dies direkt nach dem Konstruktoraufruf geschehen, wie im obigen Programmfragment.

13.4.4. Programmfragment zu getInterface()

```
try {
        ms = MulticastSocket(2048);
        ia = ms.getInterface();
} catch (SocketException se) {
        System.err.println(ue);
}
```

13.4.5. Anwendungsprogramm MulticastSniffer

Das folgende Beispiel zeigt, wie ein Sniffer für Multicasting aussehen könnte.

```
//Titel:       Multicast Sniffer
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Aufruf des MulticastSniffers :
//java MulticastSniffer ALL-SYSTEMS.MCAST.NET 4000
//1) Initialisierungen
//2) Konstruktion des InetAddress Objektes mit Hilfe von args[0]¨
//3) Definition des Ports als args[1]¨
//4) Konstruktion des Multicast Sockets mit Port args[1]¨
//5) joinGoup(IntAddress)
//6) LOOP
//a) receive(DatagramPacket)
//b) copy DatagramPacket data
//c) print DatagramPacket data
//7) close()
package Beispiel13_3MulticastSniffer;

import sun.net.*;
import java.net.*;
import java.io.*;

public class MulticastSniffer {

  public static void main(String[] args) {
    System.out.println("Multicast Sniffer : Start main;");
    //1) Initialisierungen
    InetAddress ia = null;
    byte[] buffer = new byte[65509];
```

```
int port = 0;

//2) Konstruktion des InetAddress Objektes mit Hilfe von args[0]¨
try {
  try {
    ia = InetAddress.getByName(args[0]);
  }
  catch (UnknownHostException e)  {
    System.out.println("Fehler bei der Konstruktion des InetAddress Objektes");
    e.printStackTrace();
    //ia = InetAddressFactory.newInetAddress(args[0]);
    ia = InetAddress.getByName(args[0]);
  }
  //3) Definition des Ports als args[1]¨
  port = Integer.parseInt(args[1]);
}  // end try
catch (Exception e) {
  System.err.println(e);
  e.printStackTrace();
  System.err.println("Aufruf: java MulticastSniffer MulticastAdresse port");
  System.exit(1);
}
try {
  //4) Konstruktion des Multicast Sockets mit Port args[1]¨
  MulticastSocket ms = new MulticastSocket(port);
  //5) joinGoup(IntAddress)
  ms.joinGroup(ia);
  //6) LOOP
  while (true) {
    DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
    //a) receive(DatagramPacket)
    ms.receive(dp);
    //b) copy DatagramPacket data
    // alte Version
    //String s = new String(dp.getData(), 0, 0, dp.getLength());
    String s = new String(dp.getData());
    //c) print DatagramPacket data
    System.out.println(s);
  }
}
catch (SocketException se) {
  System.err.println(se);
  se.printStackTrace();
}
catch (IOException ie) {
  System.err.println(ie);
  ie.printStackTrace();
}
//7) close() : das soll der Garbage Collector tun!
System.out.println("Multicast Sniffer : Ende main.");
```

```
  }
}
```

Und hier eine Beispielausgabe:

```
java Beispiel13_3MulticastSniffer.MulticastSniffer ALL-SYSTEMS.MCAST.NET 4000

AppAccelerator(tm) 1.2.010 for Java (JDK 1.2), x86 version.
Copyright (c) 1997-1999 Inprise Corporation. All Rights Reserved.
Multicast Sniffer : Start main;
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
Hier sind Daten vom Multicast Socket Server
```

Dabei ist zu beachten, dass die Daten von folgendem Beispielsender versandt wurden:

13.4.6. Anwendungsprogramm Multicast Sender

```
//Titel:       MulticastSender
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Funktionen:
//1) Eingabe der InetAddresse auf der Kommandozeile
//2) Kreieren eines Multicast Sockets
//3) Aufbau der Multicast Kommunikation
//a) Definition des Sockets
//b) joinGroup
//c) send
//d) leaveGroup
//e) close()
// Aufruf Beispiel java MulticastSender ALL-SYSTEMS.MCAST.NET 4000
package Beispiel13_4MulticastSender;
import java.net.*;
import java.io.*;
//import sun.net.*;
public class MulticastSender {
  public static void main(String[] args) {

    System.out.println("MulticastSender : Start main;");
    InetAddress ia = null;
    int port = 0;
    String characters = "Hier sind Daten vom Multicast Socket Server\n";
    byte[] data = new byte[characters.length()];

//1) Eingabe der InetAddresse auf der Kommandozeile
    System.out.println("1) Eingabe der InetAddresse auf der Kommandozeile");
```

```
    try {
      try {
        ia = InetAddress.getByName(args[0]);
      }
      catch (UnknownHostException e)  {
        //ia = InetAddressFactory.newInetAddress(args[0]);
        System.out.println("Fehler beim Konstruieren des InetAddress Objektes");
      }
      port = Integer.parseInt(args[1]);
    }
    catch (Exception e)  {
      System.err.println(e);
      System.err.println("Aufruf: java MulticastSender MulticastAdresse port");
      System.exit(1);
    }
    // alte Form
    //characters.getBytes(0, characters.length(), data, 0);
    System.out.println("Übergabe der Daten an das Datagramm");
    data = characters.getBytes();
    DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);

//2) Kreieren eines Multicast Sockets
//3) Aufbau der Multicast Kommunikation
    try {
//a) Definition des Sockets
      System.out.println("a) Definition des Sockets");
      MulticastSocket ms = new MulticastSocket();
//b) joinGroup
      System.out.println("b) joinGroup");
      ms.joinGroup(ia);
      for (int i = 1; i < 10; i++) {
//c) send
        System.out.println("c) send Datagramm");
        ms.send(dp, (byte) 1);
      }
//d) leaveGroup
      System.out.println("d) leaveGroup");
      ms.leaveGroup(ia);
//e) close()
      System.out.println("e) close Multicast Socket");
      ms.close();
    }
    catch (SocketException se) {
      System.err.println(se);
      System.out.println(se.getMessage());
    }
    catch (IOException ie) {
      System.err.println(ie);
      System.out.println(ie.getMessage());    }
    System.out.println("Multicast Sender : Ende main.");
```

```
  }
}
```

Dieser Sender war für die Daten verantwortlich, die vom Sniffer empfangen wurden.
Hier die Beispielausgabe des Servers:

```
java Beispiel13_4MulticastSender.MulticastSender ALL-SYSTEMS.MCAST.NET 4000
AppAccelerator(tm) 1.2.010 for Java (JDK 1.2), x86 version.
Copyright (c) 1997-1999 Inprise Corporation. All Rights Reserved.
MulticastSender : Start main;
1) Eingabe der InetAddresse auf der Kommandozeile
Übergabe der Daten an das Datagramm
a) Definition des Sockets
b) joinGroup
c) send Datagramm
c) send Datagramm
c) send Datagramm
c) send Datagramm
c) send Datagramm
c) send Datagramm
c) send Datagramm
c) send Datagramm
c) send Datagramm
d) leaveGroup
e) close Multicast Socket
Multicast Sender : Ende main.
```

13.4.7. Ein Multicast Peer-to-Peer Chat Programm

Chat Programme sind zwar sehr beliebt, zumindest bei einigen Studenten. Aber in der Regel benötigen sie einen Chat Server, der dann die verschiedenen Chat Clients verwaltet.
Interessanter wäre eine Peer-to-Peer Lösung, also eine Chat Version, bei der jeder Client auch Server sein kann.
Dieser Server verwendet UDP (als Basis für das Multicasting), also eine unzuverlässige Kommunikation. Für ein Chat System sollte dies jedoch kein allzu grosses Handicap sein.

13.4.7.1. Prinzipieller Aufbau

```
package MulticastChat;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
public class MulticastChat implements Runnable, WindowListener, ActionListener {
//public MulticastChat (InetAddress group, int port)
//public synchronized void start () throws IOException
//protected void initNet () throws IOException {
//public synchronized void stop () throws IOException {
//public void windowOpened (WindowEvent event) {
//public void windowClosing (WindowEvent event) {
//public void actionPerformed (ActionEvent event) {
//protected synchronized void handleIOException (IOException ex) {
//public void run () {
//public static void main (String[] args) throws IOException {
}
```

Die start() Methode startet das Chat System.

Mit initNet() wird die Netzwerkverbindung aufgebaut.
Für jede neuen Chat Partner wird ein neuer Thread gestartet.
Die Methode run() ist eine Endlosschleife, empfängt Meldungen und stellt diese dar.

Das Chat System kann als Standalone Applikation gestartet werden:
java MulticastChat <MulticastAdresse>:<Port>

Als Multicast Adresse kann man zum Beispiel die interne Adresse 239.1.2.3 verwenden.
Das System kann nicht als "untrusted Applet" gestartet werden. Der Security Manager wird eine Ausnahme werfen (Applet Adresse ist ungleich der Gruppen Adresse).

Hier das vollständige Programm:

13.4.7.2. Multicast Chat Programm

```
//Titel:        Multicast Chat
//Version:
//Copyright:    Copyright (c) 2000
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  ein einfaches Chat System, welches mit Hilfe von Peer-to-Peer
//Kommunikation und Multicasting aufgebaut ist
package MulticastChat;

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;

public class MulticastChat implements Runnable, WindowListener, ActionListener {
  protected InetAddress group;
  protected int port;

  public MulticastChat (InetAddress group, int port) {
    this.group = group;
    this.port = port;
    initAWT ();
  }

  protected Frame frame;
  protected TextArea output;
  protected TextField input;

  protected void initAWT () {
    frame = new Frame
      ("MulticastChat [" + group.getHostAddress () + ":" + port + "]");
    frame.addWindowListener (this);
    output = new TextArea ();
    output.setEditable (false);
    input = new TextField ();
    input.addActionListener (this);
```

```
  frame.setLayout (new BorderLayout ());
  frame.add (output, "Center");
  frame.add (input, "South");
  frame.pack ();
 }

 protected Thread listener;

 public synchronized void start () throws IOException {
  if (listener == null) {
   initNet ();
   listener = new Thread (this);
   listener.start ();
   frame.setVisible (true);
  }
 }

 protected MulticastSocket socket;
 protected DatagramPacket outgoing, incoming;

 protected void initNet () throws IOException {
  socket = new MulticastSocket (port);
  socket.setTimeToLive (1);
  socket.joinGroup (group);
  outgoing = new DatagramPacket (new byte[1], 1, group, port);
  incoming = new DatagramPacket (new byte[65508], 65508);
 }

 public synchronized void stop () throws IOException {
  frame.setVisible (false);
  if (listener != null) {
   listener.interrupt ();
   listener = null;
   try {
    socket.leaveGroup (group);
   } finally {
    socket.close ();
   }
  }
 }

 public void windowOpened (WindowEvent event) {
  input.requestFocus ();
 }

 public void windowClosing (WindowEvent event) {
  try {
   stop ();
  } catch (IOException ex) {
   ex.printStackTrace ();
```

```
  }
}
```

```
public void windowClosed (WindowEvent event) {}
public void windowIconified (WindowEvent event) {}
public void windowDeiconified (WindowEvent event) {}
public void windowActivated (WindowEvent event) {}
public void windowDeactivated (WindowEvent event) {}

public void actionPerformed (ActionEvent event) {
  try {
    byte[] utf = event.getActionCommand ().getBytes ("UTF8");
    outgoing.setData (utf);
    outgoing.setLength (utf.length);
    socket.send (outgoing);
    input.setText ("");
  } catch (IOException ex) {
    handleIOException (ex);
  }
}

protected synchronized void handleIOException (IOException ex) {
  if (listener != null) {
    output.append (ex + "\n");
    input.setVisible (false);
    frame.validate ();
    if (listener != Thread.currentThread ())
      listener.interrupt ();
    listener = null;
    try {
      socket.leaveGroup (group);
    } catch (IOException ignored) {
    }
    socket.close ();
  }
}

public void run () {
  try {
    while (!Thread.interrupted ()) {
      incoming.setLength (incoming.getData ().length);
      socket.receive (incoming);
      String message = new String
        (incoming.getData (), 0, incoming.getLength (), "UTF8");
      output.append (message + "\n");
    }
  } catch (IOException ex) {
    handleIOException (ex);
  }
}
```

```
  public static void main (String[] args) throws IOException {
    if ((args.length != 1) || (args[0].indexOf (":") < 0))
      throw new IllegalArgumentException
        ("Syntax: MulticastChat <group>:<port>");

    int idx = args[0].indexOf (":");
      InetAddress group = InetAddress.getByName (args[0].substring (0, idx));
    int port = Integer.parseInt (args[0].substring (idx + 1));

    MulticastChat chat = new MulticastChat (group, port);
    chat.start ();
  }
}
```

Das Programm könnte durch Kombination mit Unicast Kommunikation und einem Server in Bezug auf die Zuverlässigkeit verbessert werden (MixedCast Chat).

13.5. Anhang : TTL Werte

Reichweite des Multicastings	**TTL Wert**
local host	0
lokales Subnet	1
localer Campus : mehrere LAN's	16
Land Breitbandbackbone	32
landesweit	48
kontinental	64
Hosts mit hoher Bandbreite weltweit	128
alle Sites weltweit	255

13.6. Aufgaben

Sie haben vorne mehrere Programme kennen gelernt (Sniffer, Sender).

1) Verbessern Sie das Sender Programm so, dass es so lange sendet, wie Sie gerne möchten. Am Besten schaffen Sie dazu ein GUI mit einem Start und einem Abbruch Knopf.
2) Multicasting Chat:
   ändern Sie das Programm so ab, dass Sie auch einen Benutzernamen eingeben können un dieser wie ein Prompt mit übertragen und angezeigt wird.
   Alternativ dazu können Sie die Benutzer auch farbig in eine Liste eintragen und die Meldungen jeweils in der entsprechenden Farbe anzeigen