Kapitel 4

Block D: Applikationsschicht

4.1 Das Chattool

Ziel ist die Programmierung eines verteilten ICQ-artigen ChatTools. Teilnehmer auf verschiedenen Rechnern sollen miteinander mittels Tastatur kommunizieren können, die Texte der einzelnen Gruppenmitglieder sollen jeweils auf allen anderen Maschinen lesbar sein und umgekehrt. Hierfür wird ein vollvermaschtes Netz von TCP-basierten Vollduplexverbindungen zwischen allen Teilnehmern errichtet wie in Bild 4.1 gezeigt.

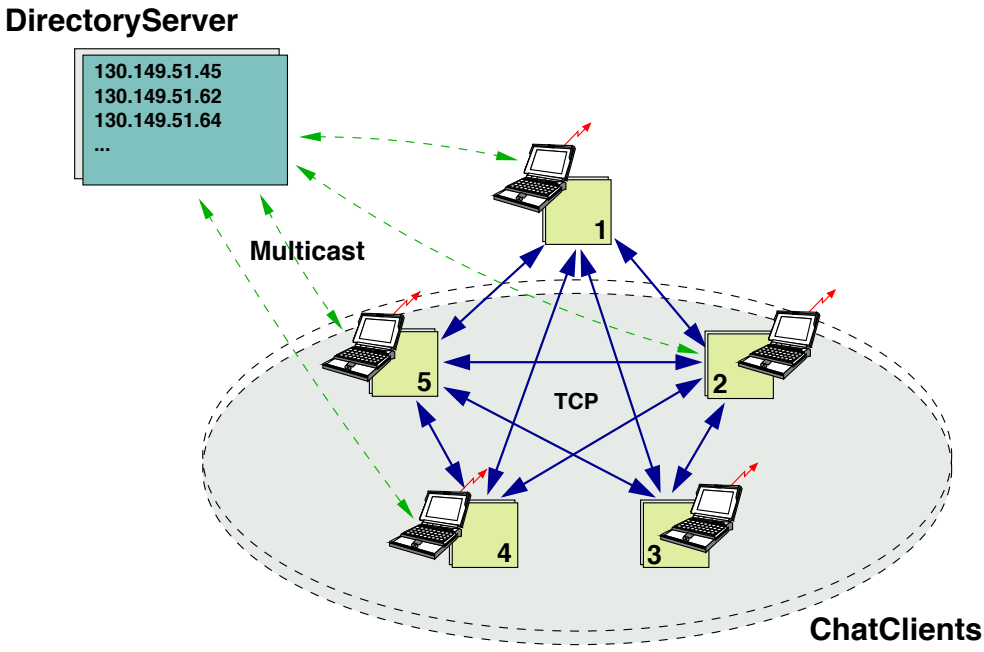


Abbildung 4.1: ChatTool

Ein neu hinzukommender Teilnehmer muß zunächst alle bereits teilnehmenden Gruppenmitglieder kennen lernen. Dieser Service wird bei klassischen Lösungen wie ICQ mittels zentraler Server erledigt. In der hier angestrebten Lösung soll ein verteilter Dienst entstehen: zur Kommunikation wird Multicast verwendet, alle Teilnehmer senden und empfangen Kontrollpakete

auf einer dedizierten Multicastadresse:

225.2.0.1:5020

Da Multicast verwendet wird, erhalten alle Teilnehmer die gleichen Informationen. Auf dieser Broadcastfähigkeit aufsetzend, wird der Suchdienst für den Server realisiert; durch Versenden eines speziellen Serversuchpaketes kann eine Station bei Aufnahme der Kommunikation den Server ausfindig machen und die entsprechende Liste aller Teilnehmer erhalten.
Wie entsteht aber ein primärer Server? Um zu vermeiden, daß ein spezieller Server im Netz laufen muß (dies würde der Idee des verteilten Systems widersprechen), sollen die Clients untereinander einen virtuellen Server auswählen können, der die Verwaltung der Teilnehmerliste übernimmt. Dies erfolgt unter Benutzung eines Verfahrens, das im übrigen dem im allseits bekannten Windowsnetzwerk Verwendung findenden SMB ähnelt. Hierbei wählen alle Teilnehmer innerhalb einer LAN-Broadcastdomain (hier: einer Multicast-LAN-Domain) einen Host aus, der alle "Shares", also die angebotenen File- und Druckdienste aller Windowsrechner im Netzwerk sammelt. Ähnlich wird der ChatServer alle IP-Adressen der Teilnehmer sammeln und verfügbar machen, er soll im folgenden als Master bezeichnet werden. Es ist nur natürlich, daß eine Station die allererste sein muß oder daß mehrere Stationen bei Verlust des Masters einen neuen Master wählen müssen. Das hierfür benutzte Verfahren arbeitet folgendermaßen:

1. Alle Stationen wählen eine gleichverteilte Zufallszahl zwischen 1 und 65535. Dieser Wert wird im folgenden als `OS_level` bezeichnet.

2. Bei Verlust des Masters oder wenn eine Station bei Betreten der Session keinen Master findet und eine Neuwahl forciert, tauschen die einzelnen Stationen ihre gewählten Werte `OS_level` miteinander aus: die Station mit dem größten Wert gewinnt die Wahl und übernimmt daraufhin die Aufgabe des Masters.

Anzumerken sei noch, daß der Ansatz des Aufsetzens auf Multicast natürlich auch ein Sicherheitsrisiko darstellt, da die Kommunikation durch einen Dritten abgehört werden kann (allerdings nur die Kontrollkommunikation) und dieser die Verbindung stören kann durch Absetzen gefälschter Pakete (Denial of Service - DoS). Wir werden in dieser Lehrveranstaltung aber auf diese Problematik nicht eingehen können. Zu den Begrifflichkeiten: (a) eine als Server arbeitende Station wird als *Master* bezeichnet, (b) ein nicht als Server arbeitender Client wird als *Slave* bezeichnet.

Hinweis:
Für die Lösung des Problems ist grundsätzlich die Art der verwendeten Programmiersprache unwesentlich, es können C, C++, Java oder Perl verwendet werden. Es ist aber vorher sicherlich sinnvoll abzuklären, ob Euer Betreuer die Sprache auch beherrscht.

4.2 Die Zustandsmaschine des ChatTools

Wie jedes "richtige" Programm wollen auch wir eine Zustandsmaschine in das ChatTool einbauen, obwohl diese recht einfach erscheinen mag. Jedoch sei angemerkt, daß komplexere Programme nach den gleichen Prinzipien aufgebaut sind und analog funktionieren. Eine Übersicht über die Zustandsmaschine der Kontrollkommunikation zeigt die Abbildung 4.2 auf Seite 175. Es sei angemerkt, daß nicht alle Ereignisse für alle Zustände aus Gründen der Übersichtlichkeit dargestellt werden konnten. Beispielhaft sei hier die Neuwahl eines Masters

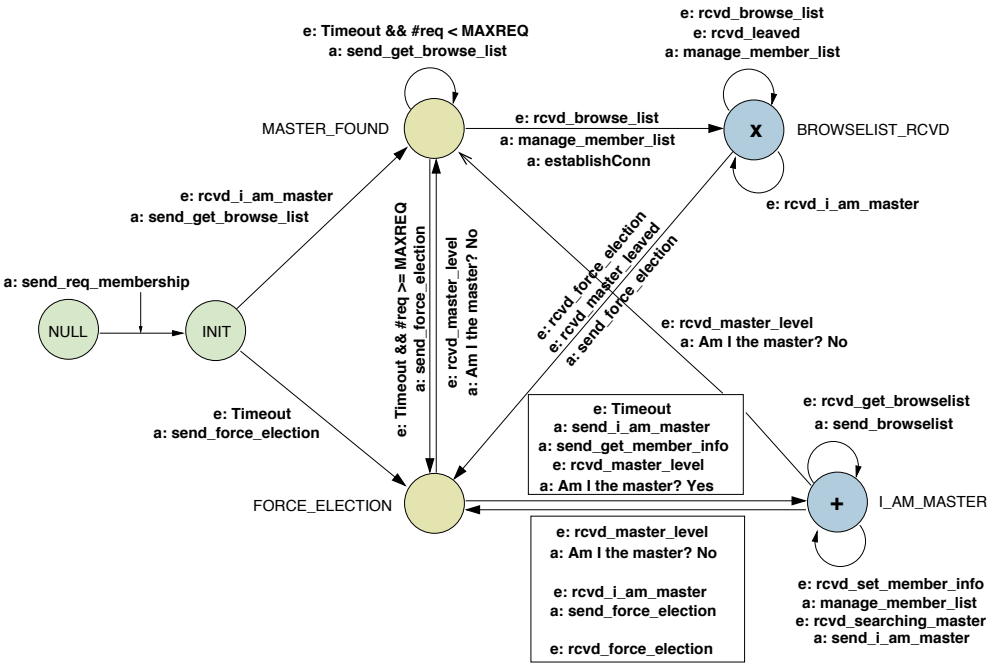


Abbildung 4.2: Zustandsmaschine Kontrollkommunikation

erwähnt, der aus jedem Zustand heraus aufgerufen werden kann und dann korrekterweise einen Wechsel in den Zustand Force_Election nach sich zieht.
Ereignisse werden durch ein vorangestelltes `e:`, Aktionen durch ein `a:` bezeichnet.

1. Der Zustand Null kennzeichnet den Start der Applikation, es wird die Multicast-Kontrollverbindung aufgebaut (siehe hierzu Abschnitt 4.6 auf Seite 183).

2. Nach der Initialisierung aller Komponenten versucht die Station einer eventuell bereits bestehenden Kommunikation beizutreten durch Absenden eines Request_Membership-Paketes, setzt einen Timer und wechselt in den Zustand Init. Im weiteren kann ein bereits arbeitender Master antworten, falls einer existiert. Sollte diese Station die erste überhaupt gestartete sein, so würde die Station in einem Deadlock festhängen. Durch das Setzen des Timers bei Betreten des Zustands Init und dessen Auslaufen wird dieser Fall verhindert. Da die Station nicht die notwendige Information besitzt, ob sie die erste und einzige, oder zeitgleich mit anderen Stationen gestartet war, versucht sie nun, eine Neuwahl des Masters zu initiieren durch Versenden eines Force_Election-Paketes und geht in den Zustand Force_Election über unter gleichzeitigem Neusetzen des globalen Timers.

3. Alternativ hierzu kann ein bereits arbeitender Master auf die Nachfrage nach Mitgliedschaft antworten mit einem Paket des Typs I_Am_Master, worauf die neu hinzukommende Station den Zustand Init verläßt und in den Zustand Master_Found wechselt. Somit wäre der Assoziationsvorgang eigentlich abgeschlossen, jedoch benötigt die Station noch die Liste aller Teilnehmer. Diese wird durch den Master verwaltet und kann durch ein Paket des Typs Get_BrowseList angefordert werden.

4. Bleiben wir noch einen Moment beim Zustand Master_Found, ehe wir zur Neuwahl

eines Masters zurückkehren: die Station erwartet nach der Anfrage bzgl. der Teilnehmerliste eine Antwort. Was sollte passieren, wenn keine Antwort eintrifft? Hierzu sollen Clients folgende Strategie anwenden: da wir eine Multicastkommunikation verwenden basierend auf UDP als Transportprotokoll können Pakete durch Pufferüberlauf verloren gehen. Daher ist der Verlust einer Anfrage nicht notwendigerweise ein Anzeichen dafür, daß der Master nicht mehr verfügbar ist. Schließlich antwortete er auf unser Request_MemberShip-Paket kurz zuvor! Ein Überlauf des Multicastpuffers beim Master kann zum Verlust unseres zweiten Paketes geführt haben. Daher soll ein Client die Anfrage bis zu einer Gesamtanzahl von MAXREQ Anfragen wiederholen, bevor er den Master als nicht mehr erreichbar annimmt. Dann jedoch sollte eine Neuwahl des Masters erfolgen, ausgelöst durch das Senden eines Force_Election-Paketes und Übergang in den Zustand Force_Election.

5. Kehren wir zurück zum Zustand Force_Election: aufgrund des Aussendens eines Paketes vom Typ Force_Election müssen alle Stationen, die ein solches empfangen, sofort ihren gewählten `OS_level` an die Multicastverbindung senden. Ein solcher `OS_level` wird als gleichverteilte Zufallszahl zwischen 0 und 65535 gewählt. Erhält eine Station eine Zahl zugesandt, die größer ist als die selbst gewählte, so wird eine andere Station Master werden und der Empfänger scheidet durch Übergang in den Zustand Master_Found aus dem Wettbewerb aus. Ist der eigene Level größer als der empfangene, so geht die Station davon aus, daß sie selbst der nächste neue Master wird. Dies ist notwendig, da sie nicht wissen kann, ob noch weitere Stationen am Wettbewerb teilnehmen. Würde sie auf weitere Antwortpakete warten, so bliebe sie eventuell in einem Deadlock hängen. Da wir hier keinen Timer einsetzen wollen (die Abschätzung des Timerwertes hinge von der Anzahl der Teilnehmer, ihrem Aufenthaltsort im Netz und damit von ihrer Entfernung (als Delay) ab → schwierig zu schätzen!), geht die Station sofort in den Zustand I_Am_Master über. Empfängt sie einen höheren `OS_Level` später ohne daß eine weitere Neuwahl gestartet wurde, so geht sie in den Zustand Master_Found zurück und arbeitet als dedizierter Slave weiter.

   Konsequenterweise bleibt die Station mit dem höchsten `OS_level` im Zustand I_Am_Master und übernimmt die Verwaltung der Teilnehmer. Im nächsten Schritt fordert sie alle Clients auf, sich nochmals bei ihr zu melden, um die Clientliste neu aufzubauen. Ist die Liste komplett, so können beliebige Stationen die Liste abfragen.

6. Nach Empfang der Clientliste (auch als BrowseList bezeichnet), geht ein Client in den eigentlichen Datenmodus über (Zustand BrowseList_Rcvd). Hierfür müssen alle Datenverbindungen zu allen anderen Clients aufgebaut werden. Näheres hierzu im Abschnitt 4.7 auf Seite 185.

Nach der Betrachtung der Zustandsmaschine und des generellen Programmablaufs, wenden wir uns nun der eigentlichen Socketschnittstelle zu.

4.3 Die Socketschnittstelle

Die Socketschnittstelle stellt unter dem Betriebssystem UNIX eine Standardschnittstelle zwischen der Netzwerkschicht und der Applikationsschicht dar.Da die Socketschnittstelle (und ihre Funktionen wie open(), connect(), accept(), read(), write(), close(), ...) schon im Block B (siehe Kapitel Datensicherung) eingeführt wurde, werden hier nur noch einige Details und Hilfsfunktionen erläutert.

4.3.1 Socket-Adress-Struktur

Da die Socket-Schnittstelle für verschiedene Netzwerk-Protokolle (Internet, XNS, ..) konzipiert wurde, sind alle Funktionen, die Netzwerkadressen auswerten, so allgemein gehalten, daß sie zuerst die Protokoll-Familie auswerten und zusätzlich noch die Größe der Adress-Struktur als Parameter erwarten. Die allgemeine Adress-Struktur der Socketschnittstelle besteht daher auch nur aus einer (16-Bit, short int) Kennung der Protokoll-Familie, und einer festgelegten Anzahl von Adress-Bytes (hier 14 Bytes), die protokoll-spezifisch die Adresse repräsentieren.

```
struct sockaddr {
   u_short sa_family; /* address family */
   char sa_data[14];  /* up to 14 bytes of direct address */
}
```

Die Aufteilung der 14 allgemeinen Adress-Bytes in 6 IP-Adress-Bytes und 8 Füllbytes:

```
/* Structure describing an Internet (IP) socket address. */
struct sockaddr_in {
  short int sin_family;        /* Address family   */
  unsigned short int sin_port; /* Port number      */
  struct in_addr sin_addr;     /* Internet address */
  unsigned char __pad[8];      /* Pad to sizeof(struct sockaddr) */
};
```

Warum werden sechs Bytes für eine IP-Adresse benutzt?
Die `sockaddr_in`-Struktur bestand in früheren Implementationen aus verschiedenen Elementen, die eine IP-Adresse auf unterschiedliche Weise (Bytereihenfolge) darstellten.

```
/* Internet address. */
struct in_addr {
  unsigned long;
  int s_addr;
};
```

Heute besteht dieses Struktur nur noch aus dem einen Element s_addr, so daß eigenlich die Struktur unnötig ist, aber aus Kompatibilitätsgründen weiter verwendet wird (was sich auch in naher Zukanft nicht ändern wird). So kann man auf die als long_int dargestellte IP-Adresse nur über den Ausdruck sin_addr.s_addr zugreifen.

4.3.2 Daten-Repräsentation

Während das Phänomen, daß bei einigen Computersystemen Bytes nicht aus 8 Bits sondern aus 9, 11 oder 12 Bits bestehen, langsam verschwindet und daher vernachlässigt werden kann, bleibt ein anderes Problem weiterhin bestehen (und wird "gepflegt") : Verschiedene Computersysteme speichern bei Datenworten, die aus mehreren Bytes bestehen, die einzelnen Bytes in unterschiedlicher Reihenfolge. So kann ein 16-Bit Datenwort (short int) auf zwei verschiedene Arten gespeichert werden (siehe Bild 3.2), entweder mit dem niederwertigen Byte an der Startadresse (Little-Endian-Anordnung), oder mit dem höherwertigen Byte an der Startadresse (Big-Endian-Anordnung) .Die Anordnung eines 32-Bit-Datenwortes entspricht dabei (durch Aufteilung in zwei 16-Bit-Worte) der Anordnung zweier Bytes in einem 16-Bit-Wort.

Little-Endian- und Big-Endian-Anordnung eines 16-Bit-Wortes und eines 32-Bit-Wortes
So verwenden Intel (80x86), DEC VAX, ... die Little-Endian-Anordnung, wogegen Motorola (68xxx), Sun Sparc und IBM-Groß rechner (IBM 370, ..) die Big-Endian-Anordnung verwenden.
Da ein Netzwerk (normalerweise) heterogene Rechner, also Rechner unterschiedlicher Architekturen (mit unterschiedlicher Datenrepräsentation (Bytefolge innerhalb eines Datenwortes)) miteinander verbinden kann bzw. soll, muß eine einheitliche Datenrepräsentation (Bytereihenfolge) der übertragenen Daten definiert werden.Im OSI-Referenz-Modell übernimmt die Schicht 6 (Präsentationsschicht) unter anderem diese Datenrepräsentation der Anwendungsdaten.
Beim Internet-Protokoll wird die Big-Endian-Anordnung verwendet.Dem Programmierer von Applikationen werden eine Reihe von Funktionen angeboten, die 16-Bit-Worte bzw. 32-Bit-Worte von der Netz-Repräsentation in die Repräsentation des lokalen Rechners (und umgekehrt) umformen (siehe unten). Unabhängig davon, ob der lokale Rechner eventuell die Big-Endian-Anordnung schon verwendet, sollte man diese Funktionen immer benutzen, um Daten über das Netzwerk zu übertragen, da eine Portierung auf andere Rechner sonst (fast) unmöglich ist. Auf Rechnern, die schon die Big-Endian-Anordnung schon verwenden, sind diese Funktionen Leer-Funktionen, d.h. sie liefern als Ergebnis die Daten unverändet zurück.Die IP-Portnummer und IP-Adresse müssen in der sockaddr_in-Struktur in Netz-Repräsentation gespeichert werden !!!

Net-to-Host-Short u_short ntohs(u_short netshort)
Wandelt die Netz-Repräsentation eines 16-Bit-Wortes (short int) in die Host-Repräsentation um.
Beispiel: printf("PortNr: %d\n", ntohs(in_addr.sin_port));

Net-to-Host-Long u_long ntohl(u_long netlong)
Wandelt die Netz-Repräsentation eines 32-Bit-Wortes (long int) in die Host-Repräsentation um.

Host-to-Net-Short. u_short htons(u_short hostshort)
Wandelt die Host-Repräsentation eines 16-Bit-Wortes (short int) in die Netz-Repräsentation um.
Beispiel: in_addr.sin_port = htons(13);

Host-to-Net-Long. u_long htonl(u_long hostlong)
Wandelt die Host-Repräsentation eines 32-Bit-Wortes (long int) in die Netz-Repräsentation um.

4.3.3 Adress-Umwandlung

Eine Internet-Adresse wird (wie schon im Kapitel Netzwerkschicht beschrieben) im punktierten Dezimalformat (z.B. 1.2.3.4) beschrieben. übertragen wird diese Adresse jedoch als 4-Byte-Wert (im allgemeinen als unsigned long integer bzw. als struct in_addr definiert). Für die nötige Adreß umwandlung zwischen der String-Darstellung und der 4-Byte-Darstellung werden dem Programmierer folgende zwei Funktionen zur Verfügung gestellt:

String-to-IN_addr

```
u_long inet_addr( char * string )
```

Konvertiert eine Internet-Adresse, die als String im punktierten Dezimalformat (z.B. “130.149.49.127“) übergeben wurde, in die 4-Byte-Darstellung (Netz-Repräsentation !).Konnte die Umwandlung nicht erfolgreich durchgeführt werden, wird der Wert 0xffffffff (als INADDR_NONE definiert) zurückgeliefert (was der Internet-(Broadcast-)Adresse 255.255.255.255 entspricht !).
Beispiel: u_long inaddr = inet_addr(“130.149.49.127”);

IN_addr-to-String

```
char * inet_ntoa( struct in_addr addr )
```

Konvertiert eine Internet-Adresse, die in der 4-Byte-Darstellung (Netz-Repräsentation !) übergeben wurde, in einen String, der die Adresse im punktierten Dezimalformat darstellt.Bei dieser Funktion gibt es keinen Fehlerfall, da jeder mögliche Wert von addr in einen String umgewandelt werden kann (“0.0.0.0” bis “255.255.255.255”).
Beispiel: printf(“Adr: %s\n“, inet_ntoa(in_addr.sin_addr));

4.3.4 Namens-Verwaltung

```
hostent * gethostbyname(char * hostname );
```

Die Funktion `gethostbyname` sucht zu einem vorgegebenen Hostnamen die dazugehörige IP-Adresse (bzw. Adressen). Zurückgeliefert wird entweder NULL (Null-Pointer), falls der Name unbekannt ist, oder einen Pointer auf die hostent-Struktur, in der alle zu diesem Host gespeicherten Namen und Adressen abgelegt sind.

```
struct hostent {
  char  *h_name;      /* official name of host */
  char **h_aliases;   /* alias list */
  int    h_addrtype;  /* host address type */
  int    h_length;    /* length of address */
  char **h_addr_list; /* list of addresses from name server */
#define h_addr h_addr_list[0] /* address, for backward compatiblity */
};
```

Die beiden wichtigsten (meist-benutzten) Elemente der hostent-Struktur sind zum einen das h_addr-Element, welches die “offizielle” IP-Adresse des Hosts enthält, und das h_name-Element, welches den “offiziellen”-Namen enthält.
Obwohl die hostent-Datenstruktur so allgemein gehalten ist, daß nicht nur IP-Adressen verwaltet werden können, liefert die gethostbyname()-Funktion nur IP-Adressen, welche schon in der Netz-Repräsentation vorliegen, d.h. ohne Konvertierung durch die htonl()-Funktion in die sockaddr_in-Struktur geschrieben werden.
Beispiel: `hostent * hp = gethostbyname( ■ftat76.ee■);`

4.3.5 Service-(Port-)Verwaltung

```
servent * getservbyname( char * servname, char * protocol );
```

Transportprotokolle nutzen zum (De-)Multiplexen verschiedener Services Portnummern. Die Funktion `getservbyname` sucht zu einem vorgegebenen Service-Namen (echo, daytime, ...) und einem vorgegebenen Protokoll (tcp, udp, icmp, ..) die IP-Portnummer, unter der dieser Service mit diesem Protokoll zu erreichen ist.

```
struct servent {
  char  *s_name;    /* official service name */
  char **s_aliases; /* alias list */
  int    s_port;    /* port */
  char  *s_proto;   /* fsprotocol to use */
};
```

Zurückgeliefert wird entweder NULL (Null-Pointer), falls der Service nicht unter diesem (oder unter keinem) Protokoll zu erreichen ist, oder einen Pointer auf die servent-Struktur, in der die Portnummer und alle zu diesem Service gespeicherten Namen abgelegt sind. Die äquivalente Funktion hierzu heiß `getservbyport`.
Beispiel: `servent * sp = getservbyname( ■echo■,■udp■ );`

4.4 Über Transportprotokolle, Paketformate und anderes ...

Bevor wir fortfahren, sollen kurz einige Eigenschaften von Multicast angesprochen und das Paketformat einschließlich der verwendeten Datenrepräsentation besprochen werden. Zunächst zu letzterem:

4.5 Das Paketformat

Der Header Um die Programmierung einfach zu halten, wird ein allen Paketen gemeinsames generisches Paketformat für Daten- und Kontrollpakete benutzt. Dieses besteht aus einem 32bit langen Header und einem optionalen Datenteil. (Zusätzlich wurde auch die Einführung von Optionen vorgesehen, jedoch bisher nicht in der Musterlösung realisiert, vielleicht wird die beste Studentenlösung übernommen. ;) Der allgemeine Paketaufbau ist in Bild 4.3 auf Seite 181 dargestellt. Der Header beinhaltet die Version des Protokolles, derzeit immer 1, den Typ des Paketes wie angegeben in Tabelle 4.1, ein Flag, daß mögliche Headeroptionen signalisiert, eine Sequenznummer im Wertebereich null bis 255 und die Länge des optionalen Datenteiles. Ist dieser Wert zu null angegeben, so ist kein Datenteil vorhanden! Die Repräsentation der Daten unterteilt sich abhängig von Header oder Datenteil: ersterer wird als 32bit-Wert in network-byte-order versendet. Nach Aufbau des Headers (bsp. als Bitfeld in C), muß dieses mittels `htonl()` entsprechend umgewandelt werden!

Datenteil Die maximale Größe des Datenteiles ist auf $2^{16} = 65536$ bytes begrenzt. Pakete mit Datenanteilen stellen die Pakettypen Data, Master_Level zur Übertragung des Wertes `OS_level` und Browse_List dar:

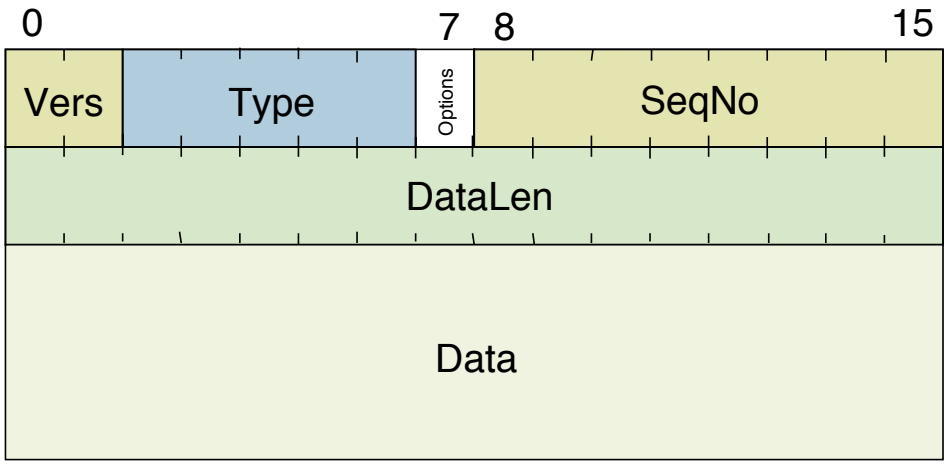


Abbildung 4.3: Das Paketformat

- Für den Datenteil in einem Textdatenpaket wird die einfachste Form der Datenrepräsentation und Übertragung im Internet verwendet, ASCII, d.h. es ist keine Transformation notwendig.
- Der `OS_level` wird als 32bit-Wert in network-byte-order versendet.
- Für die Browselist ist folgendes Format vorgesehen: jeder Host wird durch einen separaten Eintrag in ASCII-basierter dotted-notation repräsentiert, wobei gilt: jeder Eintrag wird in einem separaten Paket versendet, zuerst wird die Anzahl aller Einträge des Masters codiert, was dem Host die Möglichkeit der Überprüfung der Vollständigkeit seiner Liste gibt. Anschließend folgt der Eintrag des Indexes des

Pakettyp	ID-Wert
CTRL_PKT	0x01
SEARCHING_MASTER	0x02
MASTER_LEVEL	0x03
I_AM_MASTER	0x04
FORCE_ELECTION	0x05
DATA_PKT	0x06
LEAVE_GROUP	0x07
LEAVE_GROUP_MASTER	0x08
CHECK_MASTER	0x09
GET_MEMBER_INFO	0x10
SET_MEMBER_INFO	0x11
GET_BROWSE_LIST	0x12
BROWSE_LIST	0X13

Tabelle 4.1: Pakettypen

enthaltenen Eintrages (32bit-Zahl, network-byte-order), die Reihenfolge hat aber keine Bedeutung. Um die sich anschließenden ASCII-Character korrekt lesen zu können wird vor diesen noch die Länge des Strings einschließlich des Abschlußwertes '\0' mitgesendet. Anzumerken ist, daß natürlich die Übersendung der IP-Adresse als 32bit-Zahl selbst einfacher wäre, jedoch ist das Debuggen mit Klartextnachrichten mit weniger Schwierigkeiten verbunden. Einige Beispiele für Paketformate liefert Bild 4.4 auf Seite 187.

4.6 Multicast

Die Kontrollverbindung wird mittels Multicast realisiert. Bei Multicast handelt es sich um eine allgemeine Form des Broadcasts, wobei eine Gruppe ausgesuchter Rechner miteinander kommuniziert, ohne einzelne Datenverbindungen zwischen allen Beteiligten und damit ein voll vermaschtes Netz aufbauen zu müssen. Bevor auf die Programmierung von Multicast eingegangen wird, sollen kurz einige Anmerkungen zur Konfiguration eines Rechners für Multicast unter Linux gemacht werden.
Multicast nutzt einen speziellen Bereich aller IP-Adressen: Class D. Für diese gilt, daß die ersten 4 bits im höchstwertigen Byte den Wert 1110 aufweisen. Dies beschränkt den für Multicast nutzbaren Adreßraum auf 224.0.0.0 bis 239.255.255.255. Der IP-Multicastansatz umgeht das Prinzip der Eindeutigkeit einzelner Schichten dadurch, daß er eine Fähigkeit des LAN-Mediums für die Erbringung von Multicastdiensten voraussetzt, nämlich die Broadcastfähigkeit, die auch bei allen Ethernet-basierten LANs vorhanden ist. Es ist klar, daß bei Einsatz einer Technologie ohne Broadcastfähigkeit (wie bsp. ATM), IP nur mit speziellen zusätzlichen Lösungen Multicast bieten kann.
Will ein Rechner Multicastpakete versenden, so müssen die speziellen Multicastadressen auf Broadcastethernetadressen umgesetzt werden. Dies erfolgt wie für Unicastpakete ebenfalls über den ARP-Mechanismus, jedoch mit einer Änderung. Aus Douglas E. Comers "Internetworking with TCP/IP Vol.1":

> *To map an IP multicast address to the corresponding Ethernet multicast address, place the low-order 23 bits of the IP multicast address into the low-order 23 bits of the special Ethernet multicast address* $1:0:5e:00:00:00_{16}$.

Es sei darauf hingewiesen, daß das Mapping der Adressen aufgrund der Beschränkung der Bitanzahl auf 23 nicht ganz eindeutig ist, jedoch die Wahrscheinlichkeit einer Kollision verschiedener Multicastadressen durch Mapping auf gleiche Ethernetadressen recht gering ist.
Untersucht, ob sich die Multicastadressen 224.66.34.18, 224.2.34.18 und 224.130.34.18 bzgl. des Ethernetmappings überschneiden!
Neben der Abbildung einer Multicastadresse auf MAC-Adressen muß noch ein zweiter Mechanismus erwähnt werden: das *Internet Group Management Protocol (IGMP)*. Wie

bereits erwähnt, stützt sich IP Multicast auf die Broadcastfähigkeit des Mediums, ist damit aber auf die "LAN-Insel" beschränkt. Um verschiedene multicastfähige LANs zu verknüpfen, werden diese über Tunnelverbindungen aneinandergebunden. Das hieraus entstehende Multicast Netz wird als Multicast Backbone, oder auch kurz als MBONE bezeichnet. Tunnelendpunkte werden durch spezielle Daemons gebildet namens `mrouted`, wobei diese innerhalb des angeschlossenen LANs feststellen müssen, welche Multicastadressen derzeit benutzt werden, hierfür dient IGMP. Auf mehreren definierten Multicastadressen (bsp. 224.0.0.1) werden hierfür Statusinformationen ausgetauscht.

Multicast unter Linux Für die Konfiguration von Multicast unter Linux sind folgende Schritte notwendig: bei der Übersetzung des Kernels muß Unterstützung für IGMP eingeschaltet werden, ob dies bei dem aktuell laufenden Kernel tatsächlich der Fall ist, läßt sich mit dem Befehl `dmesg` feststellen. Dieser dumpt die jeweils letzten 4k, die der Kernel als Meldungen abgesetzt hat, darunter auch die Bootmessages. Sollten dort einmal andere Meldungen die Bootmessages überschrieben haben, so können diese auch in `/var/log/boot.msg` eingesehen werden. Eine Beispielausgabe zeigt dieser Abschnitt:

```
 Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
NET4: Unix domain sockets 1.0 for Linux NET4.0.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
```

Wichtig hierbei ist der Eintrag IGMP in der Hinweiszeile auf die unterstützten IP Protokolle. Zusätzlich muß noch die Route für Multicast gesetzt werden. Dies kann mittels des Befehles `route` geschehen:

```
erde: # route add -net 224.0.0.0 netmask 240.0.0.0 dev
eth0
```

Wieso wird als Netzmaske eigentlich 240.0.0.0 verwendet?

Multicastprogrammierung Unter Multicast kommt als Transportprotokoll ausschließlich UDP zum Einsatz. Daher wird ein Multivastsocket grundsätzlich genauso wie jeder andere UDP-Socket angesprochen. Zunächst muß also eine Socketstruktur im Kernel alloziert und initialisiert werden. Als Protokollfamilie wird hierbei das Internetprotokoll verwendet (`PF_INET`) und als Transportportprotokollfamilie UDP (`SOCK_DGRAM`). Da hier nur ein Transportprotokoll verfügbar ist, wird als Flag 0 übergeben, was die Benutzung des Standardprotokolles im Bereich `SOCK_DGRAM` bedeutet.

```
int fd;

  if ((fd = socket(PF_INET,SOCK_DGRAM,0)) < 0)
    {
      perror(socket);
      exit(1);
    }
```

Für Multicast steht abweichend von den bisher bekannten Adreßstrukturen ein spezielles Konstrukt zur Verfügung, das ausgefüllt durch einen Aufruf der Funktion `setsockopt` an den Kernel übergeben werden muß. Hierbei kann der Aufrufer die Multicastadresse ohne Port und das Interface definieren, auf dem der Host Mitglied der Gruppe werden soll. Für die Multicastadresse 224.45.2.1 sei hier ein Beispiel gezeigt:

```
#define MC_GROUP_ADDR "224.45.2.1"

  /* use setsockopt() to request that the kernel join a multicast group */
  mreq.imr_multiaddr.s_addr = inet_addr(MC_GROUP_ADDR);
  mreq.imr_interface.s_addr = INADDR_ANY;

  if (setsockopt(fd,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq)) < 0)
     perror("setsockopt");
```

Die Benutzung von `INADDR_ANY` signalisiert dem Kern, daß er das Standardmulticastinterface benutzen soll, also jenes, auf das der Routeneintrag für das Netz 224.0.0.0 weist (siehe den vorhergehenden Abschnitt ↑). Der Befehl `setsockopt` kann für eine ganze Reihe von speziellen Operationen auf einen Socket genutzt werden, es lassen sich auch die Puffergrößen anpassen oder die Wiederbenutzung einer Adresse erlauben (wir kommen hierauf gleich zurück). Für Multicast von Bedeutung sind die in Tabelle 4.2 aufgeführten Optionen und ihre Nutzbarkeit in `setsockopt` bzw. `getsockopt`. Im übrigen muß ein Host nicht Mitgleid einer Multicastgruppe sein, um an diese Pakete zu senden. Ausschließlich der Empfang benötigt die Mitgliedschaft. Mit der `IP_MULTICAST_LOOP`-Option werden abgesendete Multicastpakete intern durch den Kernel geloopt und an auf dem selben Rechner laufende Prozesse ebenfalls verteilt. Eine Anmerkung hierzu: die einzelnen Linuxkernel unterscheiden sich in diesen Implementationsdetails leider, so kann trotz eingeschaltetem Loop ein Prozeß u.U. aufgrund der falschen Kernelversion keine geloopten Pakete empfangen. Eine besondere Bedeutung kommt der Option

Option	setsockopt	getsockopt
`IP_MULTICAST_LOOP`	yes	yes
`IP_MULTICAST_TTL`	yes	yes
`IP_MULTICAST_IF`	yes	yes
`IP_ADD_MEMBERSHIP`	yes	no
`IP_DROP_MEMBERSHIP`	yes	no

Tabelle 4.2: Multicastspezifische Befehle für setsockopt

`IP_MULTICAST_TTL` zu. Sie steuert die "Reichweite" von Multicastpaketen. Da es keinen dedizierten Empfänger gibt, muß dafür gesorgt werden, daß Multicastpakete auch wieder sicher aus dem Netz entfernt werden und nicht ewig kreisen können. Bei TCP-Verbindungen dagegen, entfernt immer der Empfänger das Paket endgültig aus dem Netz. Multicast benutzt die Eigenschaft des TTL-Feldes im IP-Paket: jeder Router, den

das Datagramm passiert, erniedrigt den Wert des TTL-Feldes um eins. Erreicht der Wert des Feldes null, so verwirft der Router das Paket und sendet an den ursprünglichen Sender eine ICMP-Nachricht. Durch das explizite Setzen des TTL-Wertes kann eine Anwendung also die Reichweite ihrer Datagramme steuern: um den Verkehr innerhalb eines LANs einzugrenzen, muß TTL auf eins, für das DFN auf 15 und für eine weltweite Verbreitung auf 64 gestellt werden. Ein Wert von null beschränkt Multicastpakete ausschließlich auf den lokalen Host, eine einfache Form der Interprozeßkommunikation.

```
char ttl  = 1;
char loop = 0;

  if (setsockopt(fd, IPPROTO_IP, IP_MULTICAST_TTL, &ttl,
                                              sizeof(ttl)) < 0)
     perror("setsockopt IP_MULTICAST_TTL");

  if (setsockopt(fd, IPPROTO_IP, IP_MULTICAST_LOOP, &loop,
                                               sizeof(loop)) < 0)
     perror("setsockopt IP_MULTICAST_LOOP");
```

Zuletzt sei eine weitere Option angegeben mit der die gleichzeitige Benutzung der Socketadresse möglich wird:

```
  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char *) &reuse,
                                                  sizeof(reuse)) < 0)
     perror("setsockopt SO_REUSEADDR");
```

4.7 TCP und Datenverbindungen

Für die Versendung der eingegebenen Texte soll TCP genutzt werden. Die TCP-Implementierung bietet einen Vollduplexkanal an, das heißt, daß ein Stationenpaar (i, j) nur genau eine Verbindung öffnen muß, um Daten in beiden Richtungen austauschen zu können. Das bedeutet, daß eine Station ein "passive open" und die entsprechende Gegenstelle ein "active open" durchführen muß. Die Realisierung solcher Verbindungsaufbauten werden im Abschnitt x weiter unten besprochen. Für eine Stationsanzahl von N werden somit $\sum_{i=1}^{n} i$ Verbindungen benötigt, die einen Hin- und Rückkanal bereitstellen. Eine Station muß somit bei N Teilnehmern genau N-1 Verbindungen nutzen.

4.8 I/O Multiplexing unter Unix - select()

Wie in den vorhergehenden Abschnitten deutlich geworden ist, müssen verschiedene Verbindungen gleichzeitig aufgebaut und I/O-Events verarbeitet werden. Hinzu kommen die Standardfiledeskriptoren für Standardein- und ausgabe zum Lesen bzw. Ausgeben der Textdaten. Um nun nicht dauerhaft einzelne Deskriptoren abfragen (= pollen) zu müssen, bietet Unix den Systemaufruf `select`. Dieser erhält eine Liste aller relevanten Deskriptoren und legt den Prozeß schlafen, solange kein Ereignis eingetreten

ist (blockierender Systemaufruf). Tritt ein Ereignis auf, bsp. der Empfang eines Datenpaketes, so weckt der Scheduler des Linuxkernels den Prozeß wieder auf und teilt ihm mit, auf welchem Deskriptor (oder auf welchen Deskriptoren) ein Ereignis stattgefunden hat. Ein Beispiel ist im Programmgerüst zu finden, von dem ausgehend die Aufgabe gelöst werden kann.

4.9 Aufgabenstellung

Wie soll die Aufgabe nun gelöst werden? Ihr erhaltet als Hilfe ein kleines Programmgerüst, das bereits die auf `select` basierende Grundschleife enthält. Dieses muß aber nicht verwendet werden, andere Programmiersprachen dürfen ebenfalls verwendet werden! Als Referenzimplementierung läuft im Praktiumsnetzwerk ein ChatTool mit, daß zunächst die Aufgabe des Servers übernehmen wird und mit dem ihr eure Implementierung testen könnt. Folgende Teilaufgaben sind zu bewältigen:

1. **Standardein/-ausgabe und select()** Die Basisfunktionalität von select als Grundgerüst ist bereits gegeben. Erweitert das Programm derart, daß Daten von der Tastatur gelesen und auf den Bildschirm ausgegeben werden können.

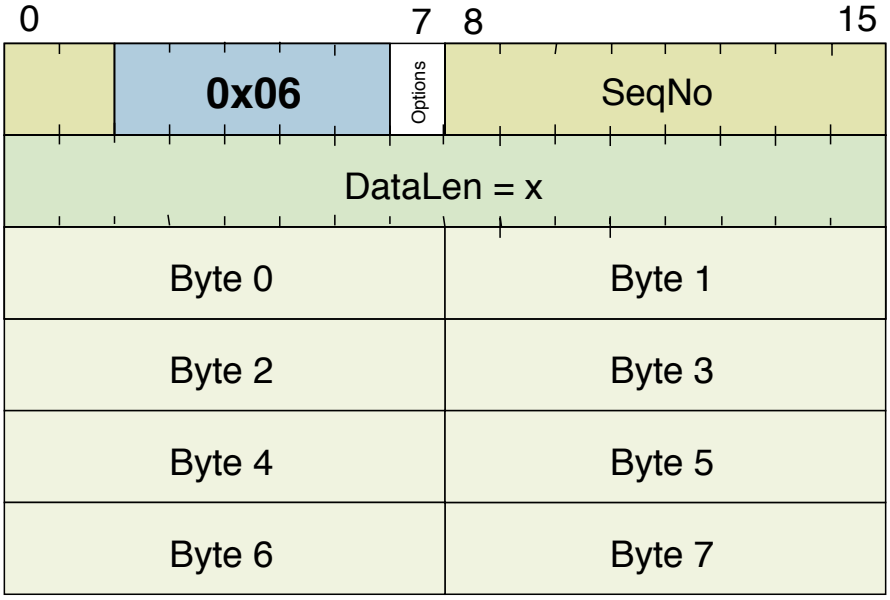
2. **Kontrollverbindung 1** Implementiert den ersten Teil der Zustandsmaschine (Clientfunktionalität). Euer Programm soll sich beim Referenzserver anmelden und die Browseliste empfangen können. Hierfür ist notwendig, die Multicastverbindung aufzubauen und deren Filedeskriptor mit in `select` aufzunehmen. Der Aufbau von Paketen und deren Handhabung einschließlich der Datenrepräsentation ist zu programmieren.

   Vergeßt nicht, daß der Server das Programm auffordert sich zu melden auf Nachfrage, emplementiert die Antwortfunktionalität.

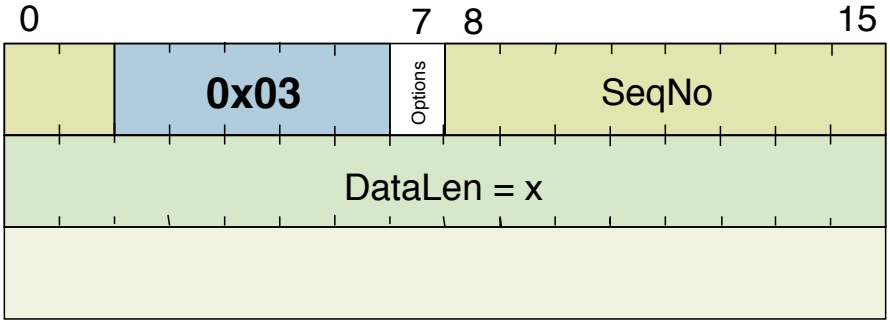
3. **Datenverbindungen** Nach dem Empfang der Browseliste müssen die als ASCII-String in dotted-notation empfangenen Peeradressen genutzt werden, um die noch nicht existenten Datenverbindungen aufzubauen. Um anderen den Aufbau einer Datenverbindung zu ermöglichen, muß ein "passive open" lokal erfolgen! Lest die Manpage zum Systemaufruf `accept`.

4. **Kontrollverbindung 2** Implementiert den zweiten Teil der Zustandsmaschine (Serverfunktionalität), so daß der Referenzclient abgeschaltet werden kann.

Datenpaket



Master–Levelpaket



Browselistpaket