# TCP/IP – Kommunikation im Internet

Martin Hüsser
mhuesser@myokay.net

Kantonsschule Wohlen
16. August 2000

## Über diese Facharbeit

Diese Arbeit beschäftigt sich mit den Protokollen IP und TCP und will dem Leser einen vertieften Einblick in den Aufbau und die Funktionsweise dieser für das Internet so wichtigen Protokolle vermitteln. Natürlich können nicht alle Aspekte dieses umfangreichen Themas angesprochen werden, was auch nicht sinnvoll wäre. In dieser Arbeit soll das wichtigste erläutert und ein Beispiel in Form eines Programmes gezeigt werden.

# Inhaltsverzeichnis

**1 Einführung**
1.1 Internet heute . . . . . 4
1.2 Grundlagen. . . . . 4
1.3 Sinn und Zweck von Protokollen . . . . . 5
1.4 Steuer- und Nutzdaten . . . . . 6

**2 Informationsfluss durch IP**
2.1 Das Internet Protokoll. . . . . 7
2.2 Die IP-Adresse . . . . . 8
2.3 DNS . . . . . 8
2.4 Der IP-Header. . . . . 9
2.5 Die Grenzen von IP. . . . . 11

**3 Sicherheit dank TCP**
3.1 Das Transport Control Protokoll. . . . . 12
3.2 Ein Paket im Paket. . . . . 13
3.3 Von Anwendung zu Anwendung. . . . . 14
3.4 Das Sicherheitskonzept von TCP. . . . . 15
3.5 Die Flusskontrolle. . . . . 15
3.6 Der TCP-Header. . . . . 16

**4 Beispielprogramm**
4.1 Bemerkungen zum Programm . . . . . 17
4.2 Das Winsock Interface (ws.pas). . . . . 18
4.3 EasyChat (main.pas) . . . . . 21
4.4 Dialogfenster. . . . . 30

**5 Anhang**
5.1 Screenshots . . . . . 35
5.2 Literaturverzeichnis. . . . . 36

# 1 Einführung

## 1.1 Internet heute

Heute sind bereits viele Millionen von Computern miteinander durch das Internet vernetzt. Was einst als militärisches Forschungsprojekt begann, ist heute aus unserem täglichen Leben kaum noch wegzudenken. Auf T-Shirts, auf Werbeplakaten, im Fernsehen... überall sieht man Internetadressen, die den Surfer auf Firmenseiten locken. Das Internet erlebte in den 90‘er Jahren einen Boom, mit dem die damaligen Entwickler nie gerechnet hatten. Eine ungeheure und unüberschaubare Anzahl von Webseiten mit nützlichen und unnützen Informationen aller Art sind über das Internet weltweit abrufbar. Meinungen werden in Newsgroups(1) ausgetauscht, Emails erreichen den Empfänger innert Sekunden und auch der Einkauf kann heute bequem von zu Hause erledigt werden. Sogar Autos und Häuser stehen in sogenannten Onlineauktionen zum Verkauf.

Doch wie funktioniert das alles überhaupt? Wie gelangen die Daten von einem Rechner zum anderen? Und kann man überhaupt sicher sein, dass die Daten an den richtigen Empfänger kommen und ihn überhaupt erreichen? Diese und weitere Fragen versucht diese Arbeit zu beantworten.

## 1.2 Grundlagen

Damit ein Computer Zugriff auf das Internet erhält, muss er sich meist zuerst mit Hilfe eines Modems oder ISDN Adapters über eine Telefonleitung zu einem ISP(2) einwählen. Sobald der Verbindungsaufbau abgeschlossen ist, kann der Computer durch eine Internetapplikation(3) Daten senden und empfangen. Wie und in welcher Form diese Daten gesendet und empfangen werden, wird in Kapitel 2 und 3 dieser Arbeit beschrieben. Wenn der Anwender z.B. einen Browser(4) startet, um eine Website von einem entfernten Server(5) herunterzuladen und anzuzeigen, dann sendet der Browser zuerst eine Anfrage

(1) Dienst zum Austausch von Meinungen und Wissen
(2) Internet Service Provider, ermöglicht den Zugang zum Intenet
(3) Ein Programm, das für das Internet geschrieben wurde
(4) zeigt Webseiten an, z.B. Netscape Communicator
(5) Ein Computer oder ein Programm, das Daten bereitstellt

an den ISP. Der ISP prüft nun, ob sich der Zielrechner mit dem Dokument, welches die Website enthält, in seinem Netzwerk befindet. Falls ja, dann kann der ISP die Anfrage direkt an den Zielrechner zustellen. Falls nicht, dann leitet er die Anfrage durch einen Router(1) an ein benachbartes, am Zielrechner näher liegendes Netzwerk weiter. Dies geht so lange, bis der ISP des Zielrechners erreicht ist und dieser die Anfrage zustellen kann. Das angeforderte Dokument gelangt dann auf dem gleichen Wege zurück zum anfragenden Rechner und zum Browser, der die Website dann anzeigt. Wie man bei *Abbildung 1* sehen kann, besteht zwischen den beiden Rechnern keine direkte Verbindung. Der Datenaustausch zwischen Ihnen geht durch viele verschiedene Netzwerke. Dies mag wie ein langwieriger Vorgang erscheinen, doch normalerweise dauert eine solche Anfrage dank Glasfaser- und Satellitentechnologie und der leistungsfähigen Router der grossen Internetanbieter, die den sogenannten Backbone des Internets darstellen, nur ein paar Zehntelssekunden.

```
Route-Verfolgung zu altavista.com [204.152.190.21]über maximal 30 Abschnitte:
 1   58 ms   54 ms   56 ms  pop-ls-8-1-1.freesurf.ch [194.230.2.200]  2   56 ms   54 ms
56 ms  anygateway.freesurf.ch [194.230.2.193]  3   61 ms   59 ms   60 ms  3.a3-
0.zur01d03.sunrise.ch [195.141.190.209]  4  104 ms  102 ms  102 ms  linx.s10-0-
0.zur01d03.sunrise.ch[195.141.213.110]  5  100 ms  100 ms  100 ms  linx.london.above.net
[195.66.224.76]  6  103 ms  100 ms  102 ms  core2-core1-oc48.lhr.above.net
[208.185.156.14]  7  167 ms  170 ms  168 ms  iad-lhr-stm4-2.iad.above.net
[216.200.127.69]  8  240 ms  238 ms  239 ms  sjc2-iad-oc48.sjc2.above.net
[216.200.127.26]  9  236 ms  239 ms  237 ms  208.184.102.10.above.net [208.184.102.10]
10  238 ms  238 ms  239 ms  head7.pla.mibh.net [128.177.255.21] 11  244 ms  241 ms  240
ms  altavista.com [204.152.190.21]
Route-Verfolgung beendet.
```

*Abbildung 1: Ausgabe des Windows Tools tracert.exe*

## 1.3 Sinn und Zweck von Protokollen

Protokolle legen fest, wie etwas zu geschehen hat. Wenn sich z.B. zwei hohe Staatsleute verschiedener Nationen treffen, dann halten sie sich auch an ein Protokoll welches ihnen vorschreibt, wie sie sich zu verhalten haben. Ohne Protokolle würde sich jeder so verhalten, wie es ihm beliebt. Viele solche Protokolle sind übrigens als ISO-Norm(2) niedergeschrieben.

(1) verbindet zwei Netzwerke miteinander
(2) International Standards Organization, www.iso.ch

Auch im Internet gibt es eine grosse Anzahl an Protokollen, von denen der Anwender normalerweise nichts mitbekommt. Aber nur dank ihnen ist die Kommunikation im Internet überhaupt möglich. Sie legen ganz klar fest, wie der Austausch von Daten zu erfolgen hat. Internetprotokolle sind in Tausenden von RFC(1) Dokumenten niedergeschrieben und können durch das Suchformular bei [4] angezeigt werden.

## 1.4 Steuer- und Nutzdaten

Alle Übertragungsprotokolle im Internet haben die Gemeinsamkeit, dass sie zwischen Steuer- und Nutzdaten unterscheiden. Steuerdaten sind in vielerlei Hinsicht erforderlich. Sie geben dem Empfänger weitere Informationen über die Art der Übertragung und der Nutzdaten. Es reicht nicht, wenn ein Rechner einfach nur die Nutzdaten verschickt. Er muss ihnen mindestens die Empfängeradresse hinzufügen, damit die Router wissen, wohin sie die Daten zustellen sollen. Vielleicht möchte der empfangende Rechner nach Erhalt eine Bestätigen zurücksenden. Dazu braucht er aber die Adresse des Absenders, die man also auch noch hinzufügen sollte. Auch eine Angabe über das Format der Nutzdaten wäre wünschenswert, damit der Empfänger weiss, ob es sich um die Daten einer Textdatei, eines Dokuments, einer Website, einer Bilddatei oder eines Programm handelt. Steuerdaten sind also unbedingt erforderlich. Es ist aber auch nicht effizient, wenn man zum Übertragen einer Textseite mit Nutzdaten zwei Textseiten Steuerinformationen braucht. Daher sollte man versuchen, möglichst viele Informationen auf kleinstem Platz unterzubringen.
Nun stellt sich nur noch die Frage, wie der Empfänger Steuer- und Nutzdaten voneinander unterscheidet. Dies wird in der Praxis so realisiert, dass der Datenstrom zuerst mit den Steuerinformationen (Header) beginnt und sich dort eine Angabe befindet, an welcher Position die Nutzdaten anfangen.

(1) Request For Comments, siehe auch [1]

# 2 Informationsfluss durch IP

## 2.1 Das Internet Protokoll

IP steht für “Internet Protocol“ und ist die Grundlage des Internets und aller anderen darauf aufbauenden Protokolle. Ob man ein Email senden, eine Datei herunterladen oder eine Website anzeigen lassen will, um IP kommt man im Internet nicht herum. Sämtlicher Datenverkehr wird über dieses Protokoll abgewickelt. Auch zeigt kein anderes Protokoll die dem Internet zugrunde liegende dezentrale Struktur so deutlich wie IP. Es erfüllt alle Anforderungen, um Daten im Internet von A nach B zu transportieren. Dazu verwendet das IP-Protokoll das Konzept des Pakettransports, wie es beispielsweise auch bei ISDN(1) eingesetzt wird. Wie man in *Abbildung 2* sieht, zerlegt es den zu sendenden Datenstrom in mehrere kleine Teilstücke, stellt die Steuerdaten als IP-Kopf (Header) voran und gibt das IP-Paket auf die Leitung. Davon merkt der Anwender zum Glück aber kaum etwas. Die installierte Software empfängt, generiert und verschickt die Pakete selbständig. Das einzige Element von IP das den meisten Anwendern bekannt sein dürfte, ist die IP-Adresse. Jeder Rechner der direkt mit dem Internet verbunden ist, hat eine eigene IP-Adresse.

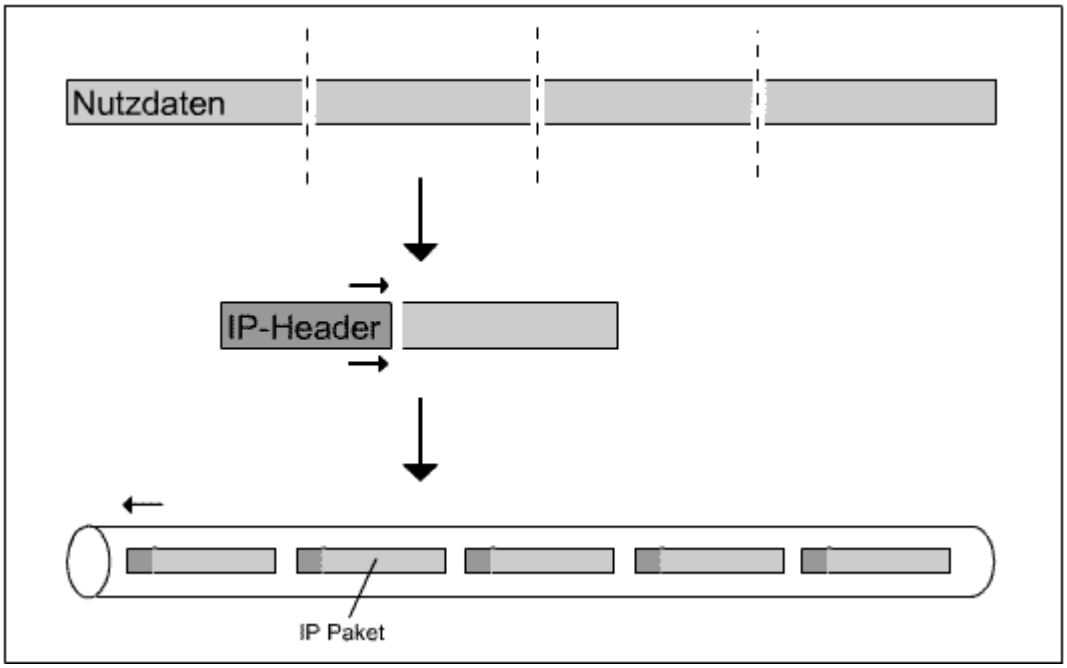


*Abbildung 2: Versand von IP Paketen*

(1) Integrated Services Digital Network, dienstintegrierendes digitales Netzwerk

## 2.2 Die IP-Adresse

Die IP-Adresse stellt im Internet eine Art Hausnummer für jeden angeschlossenen Rechner dar. Nur durch diese Nummer wissen die Router, wohin sie das Paket zustellen müssen. Ohne IP-Adresse läuft im Internet nichts. Im Gegensatz zu wirklichen Hausnummern weisen die meisten ISPs Ihren Kunden bei der Einwahl aber nur eine dynamische IP-Adresse zu, die bei jeder erneuten Einwahl wieder ändert. Wenn der Benutzer die Verbindung trennt, wird die Adresse augenblicklich wieder für andere Kunden freigegeben. Mit diesem System braucht der ISP nämlich nur so viele IP-Adressen für seine Kunden zu reservieren, wie maximal zur gleichen Zeit online sind. Dadurch können zwar wertvolle Adressen eingespart werden, jedoch ist durch die ständig wechselnde Adresse der Betrieb von Serversoftware, die von aussen ansprechbar sein muss, auf diesen Computern nur eingeschränkt möglich.
Hier ein Beispiel für eine typische IP-Adresse:

*194.230.245.21*

Wie man sieht, besteht eine gültige IP-Adresse aus vier Zahlen, die durch drei Punkte getrennt werden. Die Zahlen müssen aus dem Definitionsbereich $\mathbb{N}_0$ stammen und dürfen 255 nicht überschreiten. In dieser Form braucht die Adresse nur 4 Bytes an Platz (jede Zahl verbraucht 1 Byte) und bieten einen theoretischen Adressraum von über 4 Milliarden möglicher IP-Adressen. Leider sind es in Wirklichkeit weit weniger, da IP noch speziell reservierte Adressen vorsieht.

## 2.3 DNS

Wie gut die IP-Adresse im Internet auf Soft- und Hardwareseite auch funktionieren mag, für uns Menschen ist sie doch eher abstrakt und kaum zu merken. Am Anfang, als das Internet nur wenige hundert Rechner umfasste, gab es auf bestimmten Servern grosse Listen mit allen IP-Adressen aller an das Internet angeschlossenen Hosts(1).

(1) Rechner eines Netzwerkes

Damals mussten die wenigen Benutzer nur die Adresse eines dieser Server kennen, um die Liste herunterzuladen. In dieser Liste konnten sie dann die Adressen der anderen Rechner wie in einem Telefonbuch nachschlagen. Mit der Zeit wurden es aber immer mehr Rechner und die Listen schwollen immer mehr an. Dazu kam noch, dass eine heruntergeladene Liste bereits nach wenigen Tagen wieder veraltet war, da sich die IP-Adressen mancher Hosts durch Veränderungen in dessen Netzwerk mitverändern konnten. Bald wurde klar, dass das bestehende System nicht mehr ausreichen würde, und daher führte man das Domain Name System (DNS) ein.

Bei DNS wird die Liste der Hosts nicht mehr lokal gespeichert, vielmehr befindet sie sich auf dem DNS-Server des ISPs, dessen IP-Adresse dem Betriebssystem des Kunden beim Einwählvorgang bekannt gegeben wird oder alternativ auch auf Anwenderseite eingegeben werden kann. Wenn nun eine Internet-Applikation einen Server ansprechen will, aber nur dessen DNS-Adresse (z.B. www.altavista.com) kennt, muss zuerst eine Anfrage an den DNS-Server des ISPs gesendet werden. Nachdem der DNS-Server die Anfrage erhalten hat, durchsucht er seine lokalen Listen nach der IP-Adresse des Zielhosts und liefert sie bei Erfolg zurück. Da aber bereits viele Millionen von DNS-Adressen existieren und täglich neue dazukommen, kann nur ein kleiner Teil aller Adressen beim ISP lokal gespeichert werden. Deshalb kommt es oft vor, dass der ISP seinerseits eine Anfrage an einen übergeordneten DNS-Server schicken muss. Nach Erhalt der Adresse speichert er diese in einem Cache(1) ab, um bei einer erneuten Anfrage, was vor allem bei HTTP(2) ziemlich häufig vorkommt, schneller antworten zu können.

## 2.4 Der IP-Header

Der IP-Header enthält die in Kapitel 1.4 angesprochenen entscheidenden Steuerdaten, die unbedingt notwendig sind, damit ein Paket erfolgreich ausgeliefert werden kann. Nachfolgend sieht man in *Abbildung 3* eine Tabelle mit den einzelnen Feldern(3) des Headers und ihre Beschreibung:

(1) Zwischenspeicher für temporäre Daten
(2) Hypertext Transfer Protocol, zum Übertragen von Webseiten
(3) Informationselemente, können ganz unterschiedliche Grössen haben

<table>
<tr><td>Version (4bit)</td><td>HLEN (4bit)</td><td>Service Type (8bit)</td></tr>
<tr><td colspan="3">Total Length (16bit)</td></tr>
<tr><td colspan="3">Identification (16bit)</td></tr>
<tr><td>Flags (3bit)</td><td colspan="2">Fragment Offset (13bit)</td></tr>
<tr><td colspan="2">Time To Live (8bit)</td><td>Protocol (8bit)</td></tr>
<tr><td colspan="3">Header Checksum (16bit)</td></tr>
<tr><td colspan="3">Source IP Address (32bit)</td></tr>
<tr><td colspan="3">Destination IP Address (32bit)</td></tr>
</table>

*Abbildung 3: Die Felder des IP-Headers (IPv4)*

- Version: Gibt Auskunft über die Versionsnummer des IP-Headers (derzeit IPv4). Ohne dieses Feld wäre die spätere Einführung einer neuen Version, die mit IPv6 bereits in Sicht ist, nur schwer möglich.
- HLEN: Länge des Headers in DWords(1). Dieses Feld hat fast immer den Wert 5. IP kennt aber auch ein paar selten gebrauchte Zusatzoptionen, so dass der Header auch weit über 20 Bytes gross werden kann.
- Service Type: Hier kann die Priorität und die Übertragungsmethode des IP-Paketes eingestellt werden. In Wirklichkeit wird dieses Feld von den meisten Routern aber nicht beachtet.
- Total Length: Gibt die Grösse des gesamten IP-Pakets in Bytes an. Da für die Speicherung nur 16bit zur Verfügung stehen, sind IP-Pakete auf eine Maximalgrösse von 64KB(2) beschränkt.
- Identification: Identifiziert die Position des Pakets innerhalb des Datenstroms. Da IP-Pakete oft über mehrere unterschiedlich schnelle Routen zum Empfänger gelangen, kommt es oft vor, dass IP-Pakete sich gegenseitig „überholen“. Der Empfänger hätte ohne dieses Feld keine Möglichkeit, die (wie in *Abb. 2* gezeigt) zuvor zerstückelten und auf mehrere IP-Pakete aufgeteilten Nutzdaten wieder richtig zusammenzusetzen.

(1) 1 DWord = 2 Words = 4 Bytes
(2) 1KB = 1024 Bytes

- Flags: Enthält zwei Flags(1), welche bei der Fragmentierung eines IP-Paketes zum Einsatz kommen. Da im Internet viele verschiedene Netzwerk-technologien eingesetzt werden, kann es vorkommen, dass ein Paket unterwegs aufgrund seiner Grösse nicht durch ein Netzwerk passt und in einzelne Fragmente zerstückelt werden muss. Das erste Bit gibt dem Empfänger darüber Auskunft, ob das Paket das letzte Fragment ist oder noch weitere folgen. Mit dem zweiten Bit kann der Absender die Fragmentierung des Paketes verbieten.
- Fragment Offset: Gibt die Position des Paketes in der Fragmentkette an.
- Time To Live: Dieses Feld ist ein Zähler, der vom Absender auf 128 gesetzt und bei jedem Router dekrementiert(2) wird. Falls der Zähler auf null sinkt, wird das Paket verworfen. Dies soll die Lebensdauer eines Paketes begrenzen und verhindert, dass ein unzustellbares Paket auf ewig im Netz herumgeistert.
- Protocol: Gibt an, um welches höhere, auf IP aufsetzende Protokoll (z.B. TCP) es sich bei den Nutzdaten im IP-Paket handelt.
- Header Checksum: Prüfsumme des Headers, die bei jedem Router geprüft wird. Falls die Daten im Header unterwegs durch Übertragungsfehler verfälscht wurden, stimmt die Prüfsumme nicht mehr und der Router verwirft das Paket augenblicklich. Dies soll vor allem verhindern, dass Pakete mit veränderten IP-Adressen an einen falschen Empfänger gelangen.
- Source IP Address: IP-Adresse des Absenders
- Destination IP Address: IP-Adresse des Empfängers

## 2.5 Die Grenzen von IP

Obwohl man mit IP bereits ziemlich viel realisieren kann, gibt es doch auch einige Ansprüche, denen es nicht gerecht wird. Zum Beispiel lassen sich mit IP zwar Daten von einem Rechner zum anderen transportieren, aber wie entscheidet der empfangende Rechner, an welche Anwendung er die Daten liefern soll? Früher war dies kein Problem, da nie mehrere Anwendungen zur gleichen Zeit liefen.

(1) ein Flag entspricht einem Bit, es kann entweder 1 (true) oder 0 (false) sein
(2) um 1 verringern

Doch fast alle heutigen Betriebssysteme unterstützen Multitasking und können deshalb mit blossen IP-Paketen ohne höhere Protokolle nichts mehr anfangen. Eine viel wichtigere Frage ist aber, was mit den Paketen passiert, deren Nutzdaten auf dem Weg beschädigt wurden. IP sichert nämlich nur den Kopf mit einer Prüfsumme ab, nicht aber die Nutzdaten. Oder was passiert, wenn ein Paket unterwegs sogar ganz verloren geht? Die Antwort ist einfach: gar nichts! Der Absender merkt nicht, dass es nicht ausgeliefert wurde und der Empfänger bekommt einfach nichts. Zwar generieren Router spezielle IP-Pakete mit Fehlermeldungen(1), wenn sie ein Paket nicht ausliefern können, aber auch diese Fehlermeldungen können unterwegs verloren gehen. Und so bleibt der Absender letzten Endes völlig im unklaren, ob seine Daten korrekt angekommen sind oder nicht. Abhilfe schafft ein anderes Protokoll namens TCP, welches im folgenden Kapitel beschrieben wird.

# 3 Sicherheit dank TCP

## 3.1 Das Transport Control Protokoll

Im Internet gibt es unzählige Anwendungen, die auf einen sicheren Transport von Daten angewiesen sind. Dazu gehört das Versenden von Emails, das korrekte Anzeigen einer Website und vor allem sicherheitsrelevante Dienste wie Homebanking oder Online Shopping. Alle diese Anwendungen wären ohne TCP nicht möglich. Das TCP-Protokoll hat die Aufgabe, die korrekte Auslieferung von IP-Paketen und deren Inhalt sicherzustellen. Dazu verwendet es aber kein grundlegend neues Kommunikationskonzept, sondern baut auf IP auf und erweitert es so, dass auch ohne eine echte Verbindung zwischen Sender und Empfänger ein sicherer Kommunikationskanal geschaffen werden kann. Im Mittelpunkt von TCP steht aber nicht mehr die Datenkommunikation zwischen Hosts so wie bei IP, sondern zwischen Anwendungen.

(1) ICMP: ein weiteres Protokoll, das auf IP aufbaut

## 3.2 Ein Paket im Paket

Während IP vor allem auf der Reise eines Pakets von Router zu Router eine grosse Rolle spielt, kommt TCP nur beim Absender und beim Empfänger zum Vorschein. Für die Router ist das TCP-Protokoll transparent. Dieses Verhalten lässt sich sehr gut mit einem wirklichen Paket in unserem Postsystem vergleichen. Die Post sieht auch nur die äussere Hülle und merkt nichts davon, wenn ein Paket zweimal verpackt ist, d.h. in einem äusseren Paket (IP) ein weiteres Paket (TCP) liegt. Nur beim Absender, der das Paket verpackt und beim Empfänger, der es öffnet, kommt dieses zweite, innere Paket zum Vorschein. Wie man in *Abbildung 4* sehen kann, werden z.B. beim Versenden einer kleinen, komprimierten Datei im "World Wide Web" bis zu 5 Pakete benötigt, wobei immer ein inneres Paket als Nutzlast (Nutzdaten) von der Hülle (Header) eines übergeordneten Pakets umschlossen wird.

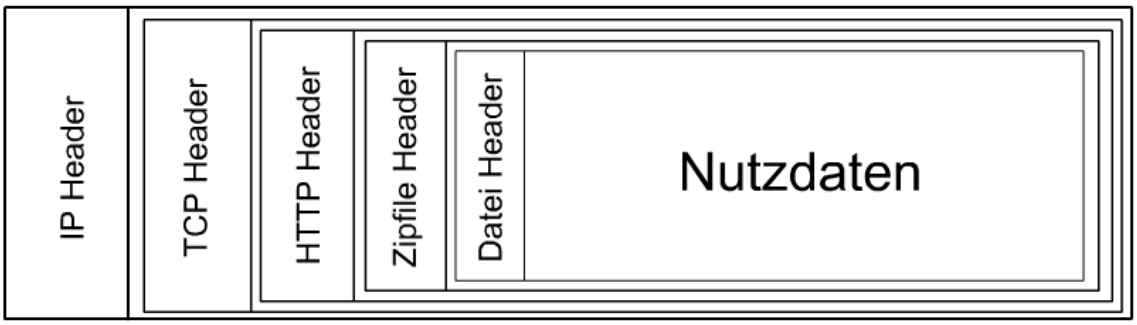


*Abbildung 4: Verschachtelung von Paketen*

Wie man sieht, kann durch diese Ansammlung von Protokoll-Headern das Verhältnis zwischen Steuer- und Nutzdaten ziemlich ungünstig ausfallen. Dies ist auch ein Grund, warum das TCP Protokoll, dessen 20 Bytes grosser Kopf alleine schon so gross ist wie ein IP-Header, nicht standardmässig im IP-Protokoll integriert ist. Es gibt im Internet nämlich auch Anwendungen wie z.B. Video- und Audio-Streaming(1), bei denen ein hoher Datendurchsatz und ein schneller Transport der Daten viel wichtiger ist als eine fehler- und lückenlose Auslieferung durch TCP. In diesen speziellen Fällen kommt das UDP(2) Protokoll zum Einsatz, das nur einen minimalen Header benötigt, um die

(1) z.B. Videokonferenzen und Audioübertragung in Echtzeit
(2) User Datagram Protocol

Kommunikation zwischen Anwendungen zu ermöglichen und auf die Sicherheitskonzepte von TCP komplett verzichtet.

## 3.3 Von Anwendung zu Anwendung

Das Internet Protokoll bietet zur Adressierung des Empfängers nur die IP-Adresse des Hosts. Dies genügt bei den heutigen Betriebssystemen jedoch nicht, da auf jedem Host mehrere Serveranwendungen gleichzeitig laufen können. Damit der Absender dem Empfänger mitteilen kann, für welche Anwendung ein Datenpaket bestimmt ist, braucht es eine weitere Möglichkeit zur differenzierten Adressierung. Diese Aufgabe übernimmt das TCP Protokoll mit der Einführung der Ports. Ein gültiger Port ist eine Zahl zwischen 1 und 65535 und bietet dadurch theoretisch die Möglichkeit, auf einem einzigen Host 65‘535 Internetanwendungen gleichzeitig laufen zu lassen. Damit ein Anwender eine bestimmte Serverapplikation auf einem entfernten Host ansprechen kann, muss er nebst der IP-Adresse also auch noch die entsprechende Portnummer kennen. Die gängige Notation für IP-Adresse und Port, z.B. Port 80, sieht so aus:

*194.230.245.21:80*

Falls der Anwender die Portnummer nicht kennt, dann hat er keine Möglichkeit, eine Verbindung zu der betreffenden Applikation aufzubauen. Diese Portnummer kann man leider auch nicht ermitteln wie z.B. die IP-Adressen beim DNS Server. Auch der Zielhost selber gibt keine Auskunft über gültige Ports. Aber zum Glück besitzen die meisten Serverprogramme im Internet einen standardisierten Port, der immer gleich sein sollte. Für einen HTTP-Server wäre das z.B. Port 80 oder für FTP(1) Port 21. Eine Liste mit allen standardisierten Ports findet man bei [5]. Wenn man also eine Verbindung zum FTP-Server eines entfernten Hosts aufbauen will, kann man davon ausgehen, dass man seine Datenpakete mit der Anfrage an den Port 21 senden muss.

(1) File Transfer Protocol

## 3.4 Das Sicherheitskonzept von TCP

Die eigentliche Leistung des TCP-Protokolls ist aber nicht die Einführung der Ports, sondern die Fähigkeit, einen gesicherten Kommunikationskanal zum Zielhost aufzubauen. Das Schöne dabei ist, dass sich weder der Anwender noch der Programmierer von Internetapplikationen Gedanken zur Übertragungssicherheit machen müssen. Beide können davon ausgehen, dass das TCP-Protokoll die Daten auf der Gegenseite entweder korrekt abliefert oder bei Misslingen zumindest eine Fehlermeldung generiert. Zur Sicherung der Daten kennt TCP zwei Mechanismen: Zum einen versieht es den TCP-Header mit einer 16bit grossen Prüfsumme, die die Unverfälschtheit des vollständigen TCP-Pakets sicherstellen soll und zum anderen knüpft es zwischen Sender und Empfänger eine engere Beziehung als dies bei reinen IP-Paketen der Fall ist. Der Empfänger der Daten beschränkt sich bei TCP nicht mehr nur darauf, die Daten stillschweigend entgegenzunehmen, sondern bestätigt dem Sender seinerseits jedes korrekt empfangene Paket mit einem sogenannten Acknowledge. Falls der Empfänger ein fehlerhaftes oder überhaupt kein Paket erhält, erfolgt auch keine Bestätigung. In diesem Fall wird der Sender das Paket nach Erfolgen eines Timeouts(1) solange wiederholen, bis entweder eine Bestätigung eintrifft oder der Benutzer die Übertragung abbricht.

## 3.5 Die Flusskontrolle

Das TCP-Bestätigungssystem hat aber einen entscheidenden Nachteil: Durch die vielen Bestätigungspakete des Empfängers wird einerseits ein Teil der Bandbreite an reine Verwaltungsdaten verschenkt und andererseits verlangsamt es den Datenfluss. Tatsächlich würde die Übertragung einer Datei ein vielfaches länger dauern, wenn nach dem Senden jedes einzelnen Pakets zuerst auf eine Bestätigung der Gegenseite gewartet werden müsste. Deshalb verschickt TCP gleich eine gewisse Anzahl von Paketen im voraus, sozusagen als Vertrauensvorschuss, bevor auf Bestätigungen gewartet wird. Im schlimmsten Fall müsste der Sender bei Verlust aller Pakete alles noch einmal

(1) Ereignis nach Ablauf einer Zeitspanne

wiederholen. Im Erfolgsfall aber treffen alle Bestätigungspakete kurz nacheinander ein, wodurch sich das ganze Verfahren enorm beschleunigen lässt. Mit der Zeit passt die TCP-Protokollsoftware diesen Vorschuss, basierend auf Erfahrungswerten, dynamisch an die Übertragungsbegebenheiten an. Bei einer sehr guten Verbindung ohne packet loss(1) werden dann mit der Zeit immer mehr Pakete im voraus gesendet als bei einer schlechten Verbindung. Dieses System der sogenannten Flusskontrolle funktioniert sehr gut bei der Übertragung von grösseren Datenmengen. Bei sehr kurzweiligen Verbindungen mit kleinen Dateien, wie es im WWW bei der Übertragung von Webseiten häufig der Fall ist, wirkt sich dieses System leider nachteilig aus. Hier kommt der Datenfluss auch bei einer sehr guten Verbindung gar nicht erst auf volle Touren. Dieser Effekt lässt sich aber kaum vermeiden, wenn man auf eine gesicherte Verbindung nicht verzichten will.

## 3.6 Der TCP-Header

<table>
<tr><td colspan="3">Source Port (16bit)</td><td>Destination Port (16bit)</td></tr>
<tr><td colspan="4">Sequence-Number (32bit)</td></tr>
<tr><td colspan="4">Acknowledgment-Number (32bit)</td></tr>
<tr><td>HLEN</td><td>Reserved</td><td>Control Bits</td><td>Window (16bit)</td></tr>
<tr><td colspan="3">Checksum (16bit)</td><td>Urgent Pointer (16bit)</td></tr>
</table>

*Abbildung 5: TCP-Header*

- Source Port: Portnummer des Absenders.
- Destination Port: Portnummer des Empfängers.
- Sequence-Number: Gibt die Position der Nutzdaten dieses Pakets in Bytes bezogen auf den Anfang der TCP-Übertragung an. Ermöglicht dem Empfänger, die Daten korrekt in den Datenstrom einzuordnen.
- Acknowledgment-Number: Bestätigt dem Sender den Empfang des vorherigen Paketes und sagt ihm, welche Sequence-Number als nächstes erwartet wird.

(1) engl. Paketverlust

- HLEN (4bit): Grösse des TCP-Headers in DWords (meist 5).
- Reserved (6bit): Reserviert für zukünftige Nutzung.
- Control Bits (6bit): Gibt weitere Informationen wie z.B. den Status der Verbindung (RST,SYN,FIN) oder welche Felder des TCP-Headers belegt sind (ACK,URG).
- Window: Hier gibt der Empfänger die Grösse seines Empfangspuffers an, damit der Sender weiss, wieviele Pakete er gleichzeitig maximal senden darf, ohne den Empfänger zu überlasten.
- Checksum: Prüfsumme des TCP-Pakets. Wenn sie nicht stimmt, wird das Paket augenblicklich fallengelassen.
- Urgent Pointer: Wenn das URG Control Flag gesetzt ist, dann gibt dieser Zeiger die Position der normalen Daten hinter den dringenden Daten, welche sich am Anfang befinden, an.

# 4 Beispielprogramm

## 4.1 Bemerkungen zum Programm

Den folgenden Quelltext zur Anwendung “EasyChat“ wurde von mir während ca. einer Woche mit grösster Sorgfalt entwickelt. Fehler sind aber dennoch nicht ausgeschlossen. Auch habe ich versucht, einige Kommentare im Quelltext einzufügen, um die Lesbarkeit zu verbessern. Trotzdem sind aber schon einige Kenntnisse von Delphi und Object Pascal notwendig, um zu verstehen, wie das Programm funktioniert. Kernthema bilden hauptsächlich die Funktionen des Winsock Interface, die dank der nachfolgenden Unit Ws (für Winsock) sehr einfach anzusteuern sind. Doch die asynchrone Programmierung, die Prozeduren für das Design und das Konzept von Server und Client in einer Anwendung haben den Quellcode ziemlich schnell anschwellen lassen. Dem interessierten Leser sei deshalb empfohlen, einen Blick auf die beigelegte Diskette zu werfen. Dort befinden sich sämtliche Quelltexte des Programms, die man viel bequemer am PC und vielleicht sogar in der Delphi-IDE(1) betrachten kann.

(1) Integrated Development Environment: Integrierte Entwicklungsumgebung

## 4.2 Das Winsock Interface (ws.pas)

Zur Ansteuerung der TCP/IP Protokollsoftware bietet Microsoft Windows dem Programmierer das Winsock Interface mit unzähligen Funktionen an. Eine umfangreiche Beschreibung aller Funktionen kann bei [6] gefunden werden. Da die wichtigsten dieser Funktionen aber komplizierte Strukturen als Übergabeparameter benötigen, ist es sinnvoll, wenn man zuerst einige Prozeduren zur einfacheren Ansteuerung erstellt. Zu diesem Zweck habe ich, bevor ich mit der Programmierung des Beispielprogramms angefangen habe, zuerst die Unit WS (für Winsock) geschrieben. Sie bietet einen wesentlich einfacheren Zugriff auf die wichtigsten Funktionen des Winsock APIs.

```
{*******************************************************}
{                                                       }
{    Ws: Unit zur Ansteuerung des Winsock Interface     }
{    Facharbeit: "TCP/IP: Kommunikation im Internet"    }
{                                                       }
{    Copyright (c)  1999-2000  by Martin Hüsser         }
{                                                       }
{*******************************************************}

unit Ws;
interface
uses Winsock;

  {Winsock Funktionen}
    function Winsock_Init : Boolean;
    function Winsock_Terminate : Boolean;

  {Socket Funktionen}
    function Socket_Open(SockNum:Word) : Boolean;
    function Socket_Close(SockNum:Word) : Boolean;
    function Socket_Connect(SockNum:Word;Port:Word;Addr:String) : Boolean;
    function Socket_Send(SockNum:Word; SendBuf:String) : Boolean;
    function Socket_Receive(SockNum:Word; var RecBuf:String) : Boolean;

  {Socket Funktionen für Server}
    function Socket_Bind(SockNum:Byte;Port:Word) : Boolean;
    function Socket_Listen(SockNum:Word;MaxNumOfConn:Integer) : Boolean;
    function Socket_Accept(SockNum,SockNum2:Byte) : Boolean;

  {Zusätzliche Funktionen}
    function Socket_Async(SockNum:Word;WndHandle:Integer;MsgNum:Integer;
      Flags:LongInt) : Boolean;

const
    MaxSockets = 100;
    FD_READ    = 01;
    FD_WRITE   = 02;
    FD_OOB     = 04;
    FD_ACCEPT  = 08;
    FD_CONNECT = 16;
    FD_CLOSE   = 32;
```

```
var
    MySock     : Array[0..MaxSockets] of TSocket;

implementation

{*** Winsock Funktionen *******************************************************}

{Winsock initialisieren}
function Winsock_Init : Boolean;
var
  WSAData : TWSAData;
  Cnt     : Integer;
begin
  for Cnt:=0 to MaxSockets do MySock[Cnt]:=0;
  if WSAStartUp($0101, WSAData)=0 then Winsock_Init:=True
  else Winsock_Init:=False;
end;

{Winsock beenden}
function Winsock_Terminate : Boolean;
begin
  if WSACleanUp=0 then Winsock_Terminate:=True
  else Winsock_Terminate:=False;
end;

{*** Socket Funktionen ********************************************************}

{Socket öffnen}
function Socket_Open(SockNum:Word) : Boolean;
begin
  Socket_Open:=False;
  if SockNum>MaxSockets then Exit;
  MySock[SockNum]:=Socket(AF_INET,Sock_Stream,0);
  if MySock[SockNum]<>Invalid_Socket then Socket_Open:=True
end;

{Socket schliessen}
function Socket_Close(SockNum:Word) : Boolean;
begin
  if CloseSocket(MySock[SockNum])=0 then Socket_Close:=True
  else Socket_Close:=False;
end;

{Socket verbinden}
function Socket_Connect(SockNum:Word;Port:Word;Addr:String) : Boolean;
var
  SockAddr : TSockAddrIn;
begin
  SockAddr.sin_family:=AF_INet;
  SockAddr.sin_port:=htons(Port);
  SockAddr.sin_zero:=#0#0#0#0#0#0#0#0;
  SockAddr.sin_addr.s_addr:=INet_Addr(@Addr[1]);

  if Connect(MySock[SockNum],SockAddr,SizeOf(TSockAddrIn))=0 then
    Socket_Connect:=True
  else Socket_Connect:=False;
end;

{Daten über Socket senden}
function Socket_Send(SockNum:Word; SendBuf:String) : Boolean;
begin
  if Send(MySock[SockNum],SendBuf[1],Length(SendBuf),0)<>Socket_Error then
    Socket_Send:=True
  else Socket_Send:=False;
end;
```

```
{Daten über Socket empfangen}
function Socket_Receive(SockNum:Word; var RecBuf:String) : Boolean;
var
  Count   : LongInt;
begin
  if ioctlsocket(MySock[SockNum],FIONREAD,Count)<>Socket_Error then
  if Count>0 then
  begin
    SetLength(RecBuf,Count);

    if Recv(MySock[SockNum],RecBuf[1],Count,0)<>Socket_Error then
    begin
      Socket_Receive:=True;
      Exit;
    end;
  end;

  RecBuf:='';
  Socket_Receive:=False;
end;

{*** Socket Funktionen für Server *********************************************}

{Socket an einen Port binden}
function Socket_Bind(SockNum:Byte;Port:Word) : Boolean;
var
  SockAddr : TSockAddrIn;
begin
  SockAddr.sin_family:=AF_INet;
  SockAddr.sin_port:=htons(Port);
  SockAddr.sin_zero:=#0#0#0#0#0#0#0#0;
  SockAddr.sin_addr.S_Addr:=INADDR_ANY;

  if Bind(MySock[SockNum],SockAddr,SizeOf(TSockAddrIn))=0 then
    Socket_Bind:=True
  else Socket_Bind:=False;
end;

{Serversocket-Parameter einstellen}
function Socket_Listen(SockNum:Word;MaxNumOfConn:Integer) : Boolean;
begin
  if Listen(MySock[SockNum],MaxNumOfConn)=Socket_Error then
    Socket_Listen:=False
  else Socket_Listen:=True;
end;

{Client andocken}
function Socket_Accept(SockNum,SockNum2:Byte) : Boolean;
begin
  MySock[SockNum2]:=Accept(MySock[SockNum],nil,nil);
  if LongInt(MySock[SockNum2])>9999 then Socket_Accept:=False
  else Socket_Accept:=True;
end;

{*** Zusätzliche Funktionen ***************************************************}

{Socket in asynchronen Modus schalten}
function Socket_Async(SockNum:Word;WndHandle:Integer;MsgNum:Integer;
  Flags:LongInt) : Boolean;
begin
  if WSAAsyncSelect(MySock[SockNum],WndHandle,MsgNum,Flags)=0
    then Socket_Async:=True
  else Socket_Async:=False;
end;end.
```

## 4.3 EasyChat (main.pas)

Das Beispielprogramm EasyChat ermöglicht es dem Anwender, mit bis zu 100 Leuten gleichzeitig über das Internet zu reden. EasyChat ist aber nicht nur ein Chat-Client, sondern auch ein Chatserver. Man ist also nicht auf einen äusseren Dienst angewiesen, sondern kann auf jedem Internet-PC dieses Programm im Servermodus laufen lassen. Den andern Teilnemern muss man dann nur noch die IP-Adresse des Servers bekannt geben, z.B. via Email, und es kann losgehen. (Wer seine eigene IP-Adresse nicht kennt, kann dies mit dem Windowstool “winipcfg.exe“ herausfinden)

```
{*******************************************************}
{                                                       }
{    EasyChat:   Chat-Client/Server                     }
{    Facharbeit: "TCP/IP: Kommunikation im Internet"    }
{                                                       }
{    Copyright (c)  2000  by Martin Hüsser              }
{                                                       }
{*******************************************************}
unit main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Menus, ComCtrls,
  ToolWin, ExtCtrls, ImgList, Ws;

const
  {Fensternachricht für SockComm Routine}
  WM_SOCKCOMM = WM_USER+1;

  {Message wird bei folgenden Socketereignissen ausgelöst:}
  FD_CLIENT   = FD_READ+FD_CONNECT+FD_CLOSE;
  FD_SERVER   = FD_CLIENT+FD_ACCEPT;

type
  TMainForm = class(TForm)
    {Controls}
    MainMenu: TMainMenu;
    Verbindung: TMenuItem;
    Verbinden: TMenuItem;
    Trennen: TMenuItem;
    N1: TMenuItem;
    Beenden: TMenuItem;
    Bearbeiten: TMenuItem;
    Ausschneiden: TMenuItem;
    Kopieren: TMenuItem;
    Einfuegen: TMenuItem;
    Loeschen: TMenuItem;
    Info: TMenuItem;
    ToolBar: TToolBar;
    ConnectButton: TToolButton;
    StatusBar: TStatusBar;
    SendMsgPanel: TPanel;
    SendMsgEdit: TEdit;
    Chat: TMenuItem;
```

```
    Kicken: TMenuItem;
    TimeOutTimer: TTimer;
    N3: TMenuItem;
    Aufzeichnen: TMenuItem;
    AufzchngSaveDlg: TSaveDialog;
    Servermodus: TMenuItem;
    ToolbarIconList: TImageList;
    ServerButton: TToolButton;
    DisconnectButton: TToolButton;
    ToolButton4: TToolButton;
    CutButton: TToolButton;
    CopyButton: TToolButton;
    PasteButton: TToolButton;
    ToolButton8: TToolButton;
    Autor: TMenuItem;
    Fluestern: TMenuItem;
    KickenButton: TToolButton;
    FluesternButton: TToolButton;
    ChatMemo: TMemo;

    {Initialisierung und Finalisierung}
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);

    {Design}
    procedure FormResize(Sender: TObject);
    procedure AufzeichnenClick(Sender: TObject);
    procedure BeendenClick(Sender: TObject);

    {Dialoge}
    procedure AutorClick(Sender: TObject);
    procedure FluesternClick(Sender: TObject);
    procedure KickenClick(Sender: TObject);

    {Werkzeugleiste}
    procedure ConnectButtonClick(Sender: TObject);
    procedure DisconnectButtonClick(Sender: TObject);
    procedure ServerButtonClick(Sender: TObject);
    procedure KickenButtonClick(Sender: TObject);
    procedure FluesternButtonClick(Sender: TObject);

    {Menu-Bearbeiten}
    procedure AusschneidenClick(Sender: TObject);
    procedure KopierenClick(Sender: TObject);
    procedure EinfuegenClick(Sender: TObject);
    procedure LoeschenClick(Sender: TObject);

    {Verbindung}
    procedure VerbindenClick(Sender: TObject);
    procedure ServermodusClick(Sender: TObject);

    {Kommunikation}
    procedure SendMsgEditKeyPress(Sender: TObject; var Key: Char);
    procedure TrennenClick(Sender: TObject);

    {Ereignisbehandlung für Socket-Nachrichten}
    procedure SockComm(var MsgInfo : TMessage); message WM_SOCKCOMM;

  end;

var
  MainForm      : TMainForm;

  {Steuervariablen}
  ServerMode    : Boolean;
```

```
  NumOfConn     : Integer;
  {Variablen für Serverkommandos}
  ServerCmd,
  ServerParam   : String;

  {Speicherpläzte für Benutzernamen}
  Nickname      : String;
  NickTable     : Array[2..MaxSockets] of String;

implementation

{Dialoge einbinden}
uses verbinden, info, server, kicken, fluestern;

{$R *.DFM}

{*** Interne Tools ************************************************************}

{Sucht einen freien Socket}
function FindFreeSock : Integer;
var
  Cnt : Integer;
begin
  for Cnt:=2 to MaxSockets do
    if MySock[Cnt]=0 then Break;
  FindFreeSock:=Cnt;
end;

{Liefert die zum Handle in WParam passende Socketnummer}
function GetSockNr(WParam : Integer) : Integer;
var
  Cnt : Integer;
begin
  for Cnt:=0 to MaxSockets do
    if MySock[Cnt]=WParam then Break;
  if MySock[Cnt]=WParam then GetSockNr:=Cnt
  else GetSockNr:=-1;
end;

{Interpretiert Steuernachrichten}
procedure ServerProcessMsg(var Buf : String; SockNum : Integer);
var
  Index : Integer;
begin
  ServerCmd:='';
  ServerParam:='';

  {Steuernachricht herausfiltern}
  while Pos('¦',Buf)>0 do
  begin
    ServerCmd:=UpCase(Buf[1]);
    Index:=Pos('¦',Buf);
    ServerParam:=Copy(Buf,2,Index-2);
    Delete(Buf,1,Index);
  end;

  {Nickname-Tabelle aktualisieren}
  if Pos('>',Buf)>0 then NickTable[SockNum]:=Copy(Buf,2,Pos('>',Buf)-2);

end;

{*** Initialisierung und Finalisierung ****************************************}

{Konstruktor}
procedure TMainForm.FormCreate(Sender: TObject);
```

```
begin
  {Winsock initialisieren}
  Winsock_Init;

  NumOfConn:=0;
  ServerMode:=False;
end;

{Destruktor}
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  {Sind noch Verbindungen offen?}
  if NumOfConn>0 then TrennenClick(MainForm);

  {Winsock beenden}
  Winsock_Terminate;
end;

{*** Form-Design ***************************************************************}

{Passt die EditBox-Eingabezeile der Fenstergrösse an}
procedure TMainForm.FormResize(Sender: TObject);
begin
  SendMsgEdit.Width:=MainForm.Width-8;
end;

{Terminiert das Programm bei Klick auf Menu-Beenden}
procedure TMainForm.BeendenClick(Sender: TObject);
begin
  Application.Terminate;
end;

{Setzt oder löscht das Häckchen beim Untermenu Aufzeichnung}
procedure TMainForm.AufzeichnenClick(Sender: TObject);
begin
  Aufzeichnen.Checked:=not(Aufzeichnen.Checked);
end;

{*** Dialoge *******************************************************************}

{Zeigt ein Dialogfenster mit Informationen an}
procedure TMainForm.AutorClick(Sender: TObject);
begin
  InfoDlg.ShowModal;
end;

{Zeigt den Flüster-Dialog an}
procedure TMainForm.FluesternClick(Sender: TObject);
var
  Cnt : Integer;
begin
  {Wurde der Cancle-Knopf gedrückt?}
  if FluesternDlg.ShowModal=mrCancel then Exit;

  if ServerMode then
  begin
    {Sende Flüsternachricht an die betreffenden Clients}
    for Cnt:=2 to MaxSockets do
      if NickTable[Cnt]=FluesternUser then
        Socket_Send(Cnt,FluesternMsg);
    ChatMemo.Lines.Add('(1)'+FluesternMsg);
  end
  else begin
    {Sende Flüsterkommando an Server}
    Socket_Send(0,'F'+FluesternUser+'¦'+FluesternMsg);
```

```
    ChatMemo.Lines.Add(FluesternMsg);
  end;
end;

{Zeigt den Kicken-Dialog an}
procedure TMainForm.KickenClick(Sender: TObject);
var
  Cnt : Integer;
begin
  if KickenDlg.ShowModal=mrCancel then Exit;
  for Cnt:=2 to MaxSockets do
    if NickTable[Cnt]=KickUser then
    begin
      {Client kicken!}
      Socket_Close(Cnt);
      MySock[Cnt]:=0;
      Dec(NumOfConn);
      StatusBar.Panels[0].Text:=
        IntToStr(Cnt)+' Benutzer eingeloggt';
    end;
end;

{*** Werkzeugleiste ************************************************************}

procedure TMainForm.ConnectButtonClick(Sender: TObject);
begin
  VerbindenClick(MainForm);
end;

procedure TMainForm.ServerButtonClick(Sender: TObject);
begin
  ServermodusClick(MainForm);
end;

procedure TMainForm.DisconnectButtonClick(Sender: TObject);
begin
  TrennenClick(MainForm);
end;

procedure TMainForm.KickenButtonClick(Sender: TObject);
begin
  KickenClick(MainForm);
end;

procedure TMainForm.FluesternButtonClick(Sender: TObject);
begin
  FluesternClick(MainForm);
end;

{*** Menu: Bearbeiten **********************************************************}

{Schneidet den selektierten Text aus und kopiert ihn in die Zwischenablage}
procedure TMainForm.AusschneidenClick(Sender: TObject);
begin
  if MainForm.ActiveControl=ChatMemo then
    ChatMemo.CopyToClipboard;
  if MainForm.ActiveControl=SendMsgEdit then
    SendMsgEdit.CutToClipboard;
end;

{Kopiert den selektierten Text in die Zwischenablage}
procedure TMainForm.KopierenClick(Sender: TObject);
begin
  if MainForm.ActiveControl=ChatMemo then
    ChatMemo.CopyToClipboard;
```

```
  if MainForm.ActiveControl=SendMsgEdit then
    SendMsgEdit.CopyToClipboard;
end;

{Fügt den Text aus der Zwischenablage an der aktuellen Cursorposition ein}
procedure TMainForm.EinfuegenClick(Sender: TObject);
begin
  if MainForm.ActiveControl=SendMsgEdit then
    SendMsgEdit.PasteFromClipboard;
end;

{Löscht den selektierten Text}
procedure TMainForm.LoeschenClick(Sender: TObject);
begin
  if MainForm.ActiveControl=SendMsgEdit then
    SendMsgEdit.ClearSelection;
end;

{*** Verbinden ****************************************************************}

{Verbindet den Client mit einem Server}
procedure TMainForm.VerbindenClick(Sender: TObject);
begin
  {Verbindungsdaten einholen}
  if VerbindenDlg.ShowModal=mrCancel then Exit;
  Nickname:=ClientNick;

  {Verbindung aufbauen}
  Socket_Open(0);
  Socket_Async(0,Application.MainForm.Handle,WM_SOCKCOMM,FD_CLIENT);
  Socket_Connect(0,ClientPort,ClientAddr);

  {Design}
  StatusBar.Panels[0].Text:='Verbindungsaufbau...';
  Verbinden.Enabled:=False;
  ConnectButton.Enabled:=False;
  ServerModus.Enabled:=False;
  ServerButton.Enabled:=False;
  Trennen.Enabled:=True;
  DisconnectButton.Enabled:=True;
end;

{Schaltet in den Servermodus}
procedure TMainForm.ServermodusClick(Sender: TObject);
begin
  {Serverinformationen einholen}
  if ServerDlg.ShowModal=mrCancel then Exit;
  Nickname:=ServerNick;

  {Server auf Empfang schalten}
  Socket_Open(1);
  Socket_Async(1,Application.MainForm.Handle,WM_SOCKCOMM,FD_SERVER);
  Socket_Bind(1,ServerPort);
  Socket_Listen(1,5);
  ServerMode:=True;

  {Design}
  Servermodus.Enabled:=False;
  ServerButton.Enabled:=False;
  Verbinden.Enabled:=False;
  ConnectButton.Enabled:=False;
  Trennen.Enabled:=True;
  DisconnectButton.Enabled:=True;
  StatusBar.Panels[0].Text:='0 Benutzer eingeloggt';
end;
```

```
{*** Kommunikation ************************************************************}

{Sendet eine Nachricht an den Server bzw. an die verbundenen Clients}
procedure SendMsg(ExcludeSock : Integer; Msg : String);
var
  Cnt : Integer;
begin
  {Nachricht an Server senden}
  if Servermode=False then Socket_Send(0,Msg)

  {Nachricht an die Clients senden}
  else begin
    for Cnt:=2 to MaxSockets do
      if Cnt<>ExcludeSock then
        if MySock[Cnt]<>0 then
          Socket_Send(Cnt, Msg);
  end;
end;

{Übergibt beim Drücken der Entertaste die eingegebene Meldung an SendMsg}
procedure TMainForm.SendMsgEditKeyPress(Sender: TObject; var Key: Char);
var
   Buf : String;
begin
  {Wenn keine Verbindung, Eingabe unterdrücken}
  if NumOfConn=0 then Key:=#0;

  {Wurde Enter gedrückt?}
  if Key=#13 then
  begin
    if SendMsgEdit.Text='' then Exit;

    {Steuerzeichen herausfiltern}
    Buf:=SendMsgEdit.Text;
    while Pos('¦',Buf)>0 do
      Delete(Buf,Pos('¦',Buf),1);

    {Nicknamen voranstellen}
    Buf:='<'+Nickname+'> '+Buf;

    {Nachricht senden}
    SendMsg(0,Buf);

    {Design}
    if ServerMode then Buf:='(1)'+Buf;
    ChatMemo.Lines.Add(Buf);
    Key:=#0;
    SendMsgEdit.Text:='';
  end;
end;

{Trennt die Verbindung mit dem Server bzw. den Clients}
procedure TMainForm.TrennenClick(Sender: TObject);
var
  Cnt : Integer;
begin
  {Sockets schliessen}
  if ServerMode=False then begin Socket_Close(0); MySock[0]:=0; end
  else begin
    for Cnt:=2 to MaxSockets do
    if MySock[Cnt]<>0 then
    begin
      Socket_Close(Cnt);
      MySock[Cnt]:=0;
```

```
    end;
    Socket_Close(1);
  end;
  NumOfConn:=0;

  {Design}
  SendMsgEdit.Text:='';
  StatusBar.Panels[0].Text:='Verbindung getrennt';

  {Aufzeichnung speichern?}
  if Aufzeichnen.Checked then
    if AufzchngSaveDlg.Execute then
      ChatMemo.Lines.SaveToFile(AufzchngSaveDlg.FileName);

  {Design}
  ChatMemo.Clear;
  Verbinden.Enabled:=True;
  ConnectButton.Enabled:=True;
  Servermodus.Enabled:=True;
  ServerButton.Enabled:=True;
  Trennen.Enabled:=False;
  DisconnectButton.Enabled:=False;
  Fluestern.Enabled:=False;
  Kicken.Enabled:=False;
  Servermode:=False;
end;

{*** Socket-Ereignisbehandlungsroutine ****************************************}

{Ereignisbehandlungsroutine für sämtliche geöffneten Sockets}
procedure TMainForm.SockComm(var MsgInfo : TMessage);
var
  Buf      : String;
  SockNum,
  FreeSock,
  Cnt      : Integer;
begin
  {Nummer des Sockets herausfinden, der die Fensternachricht ausgeöst hat}
  SockNum:=GetSockNr(MsgInfo.WParam);
  if SockNum=-1 then Exit;

  {Clientmodus}
  if SockNum=0 then
  begin
    case MsgInfo.LParam of
      FD_CONNECT : {Client ist verbunden}
        begin
          StatusBar.Panels[0].Text:='Verbunden';
          NumOfConn:=1;
          Fluestern.Enabled:=True;
          FluesternButton.Enabled:=True;
        end;
      FD_READ : {Client empfängt Daten vom Server}
        begin
          {Daten entgegennehmen und anzeigen}
          Socket_Receive(0,Buf);
          ChatMemo.Lines.Add(Buf);
        end;
      FD_CLOSE : {Server hat die Verbindung geschlossen}
        begin
          {Socket schliessen}
          TrennenClick(MainForm);
          Fluestern.Enabled:=False;
          FluesternButton.Enabled:=False;
        end;
```

```
  end;
end;

{Servermodus: Serversocket}
if SockNum=1 then
begin
  case MsgInfo.LParam of
    FD_ACCEPT : {Ein Client dockt an}
      begin
        {Client andocken und neuen Socket zuweisen}
        FreeSock:=FindFreeSock;
        Socket_Accept(1,FreeSock);
        NickTable[FreeSock]:='';

        {Maximale Anzahl Benuter darf nicht überschritten werden}
        if NumOfConn=MaxUser then
        begin
          Socket_Close(FreeSock);
          MySock[FreeSock]:=0;
          Exit;
        end;

        {Design}
        Inc(NumOfConn);
        StatusBar.Panels[0].Text:=IntToStr(NumOfConn)+' Benutzer eingeloggt';
        Kicken.Enabled:=True;
        KickenButton.Enabled:=True;
        Fluestern.Enabled:=True;
        FluesternButton.Enabled:=True;
      end;
  end;
end;

{Servermodus}
if SockNum>1 then
begin
  case MsgInfo.LParam of
    FD_CLOSE : {Ein Client hat die Verbindung getrennt}
      begin
        {Socket schliessen}
        Socket_Close(SockNum);
        MySock[SockNum]:=0;
        Dec(NumOfConn);

        {Design}
        StatusBar.Panels[0].Text:=IntToStr(NumOfConn)+' Benutzer eingeloggt';
        if NumOfConn=0 then
        begin
          Kicken.Enabled:=False;
          KickenButton.Enabled:=False;
          Fluestern.Enabled:=False;
          FluesternButton.Enabled:=False;
        end;
      end;
    FD_READ : {Server empfängt die Daten eines Clients}
      begin
        {Daten entgegennehmen}
        Socket_Receive(SockNum,Buf);
        {verarbeiten}
        ServerProcessMsg(Buf,SockNum);
        {Flüster-Nachricht?}
        if ServerCmd='F' then
        begin
          if ServerParam=Nickname then
            ChatMemo.Lines.Add('('+IntToStr(SockNum)+')'+Buf)
```

```
            else for Cnt:=2 to MaxSockets do
              if Cnt<>SockNum then
                if NickTable[Cnt]=ServerParam then
                  Socket_Send(Cnt,Buf);
            Exit;
          end;
          {an die anderen Clients weiterleiten}
          SendMsg(SockNum,Buf);
          {und anzeigen}
          ChatMemo.Lines.Add('('+IntToStr(SockNum)+')'+Buf);
        end;
    end;
  end;
end;

end.
```

## 4.4 Dialogfenster

```
{*****************************************************************************}
{    Verbinden Dialog                                                         }
{*****************************************************************************}

unit verbinden;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TVerbindenDlg = class(TForm)
    VerbindenBtn: TButton;
    CancelBtn: TButton;
    DlgBevel: TBevel;
    AddrEdit: TEdit;
    PortEdit: TEdit;
    AddrLabel: TLabel;
    PortLabel: TLabel;
    NickLabel: TLabel;
    NickEdit: TEdit;
    procedure CheckValues;
    procedure PortEditChange(Sender: TObject);
    procedure AddrEditChange(Sender: TObject);
    procedure NickEditChange(Sender: TObject);
end;

var
  VerbindenDlg: TVerbindenDlg;

  ClientPort  : Integer;
  ClientAddr,
  ClientNick  : String;

implementation

{$R *.DFM}

procedure TVerbindenDlg.PortEditChange(Sender: TObject);
begin
  CheckValues;
end;
```

```
procedure TVerbindenDlg.AddrEditChange(Sender: TObject);
begin
  CheckValues;
end;

procedure TVerbindenDlg.NickEditChange(Sender: TObject);
begin
  CheckValues;
end;

{Eingaben überprüfen}
procedure TVerbindenDlg.CheckValues;
var
  Buf,
  Buf2 : String;
  Cnt,
  Code,
  IChk,
  CPos : Integer;
begin
  VerbindenBtn.Enabled:=True;

  {PortCheck}
  Val(PortEdit.Text,ClientPort,Code);
  if (Code<>0)or(ClientPort<0)or(ClientPort>65535) then
    VerbindenBtn.Enabled:=False;

  {AddrCheck}
  Buf:=AddrEdit.Text+'.';
  for Cnt:=1 to 4 do
  begin
    CPos:=Pos('.',Buf);
    if CPos=0 then Break;
    Buf2:=Copy(Buf,1,CPos-1);
    Val(Buf2,IChk,Code);
    if (Code<>0)or(IChk<0)or(IChk>255) then Break;
    Delete(Buf,1,CPos);
    if Buf='' then Break;
  end;
  if (Cnt<4)or(Buf<>'') then VerbindenBtn.Enabled:=False;
  ClientAddr:=AddrEdit.Text;

  {NickCheck}
  if NickEdit.Text='' then VerbindenBtn.Enabled:=False;
  ClientNick:=NickEdit.Text;

end;

end.


{*****************************************************************************}
{    Server Dialog                                                            }
{*****************************************************************************}
unit server;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TServerDlg = class(TForm)
    OKBtn: TButton;
```

```
    CancelBtn: TButton;
    Bevel1: TBevel;
    PortLabel: TLabel;
    MaxConnLabel: TLabel;
    PortEdit: TEdit;
    MaxUserEdit: TEdit;
    NickLabel: TLabel;
    NickEdit: TEdit;
    procedure CheckValues;
    procedure PortEditChange(Sender: TObject);
    procedure MaxUserEditChange(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure NickEditChange(Sender: TObject);
end;

var
  ServerDlg  : TServerDlg;

  ServerPort : Integer;
  ServerNick : String;
  MaxUser    : Integer;

implementation

{$R *.DFM}

procedure TServerDlg.FormShow(Sender: TObject);
begin
  CheckValues;
end;

procedure TServerDlg.PortEditChange(Sender: TObject);
begin
  CheckValues;
end;

procedure TServerDlg.MaxUserEditChange(Sender: TObject);
begin
  CheckValues;
end;

procedure TServerDlg.NickEditChange(Sender: TObject);
begin
  CheckValues;
end;

{Eingaben überprüfen}
procedure TServerDlg.CheckValues;
var
  Code : Integer;
begin
  OKBtn.Enabled:=True;

  {PortCheck}
  Val(PortEdit.Text,ServerPort,Code);
  if (Code<>0)or(ServerPort<0)or(ServerPort>65535) then
    OKBtn.Enabled:=False;

  {MaxUserCheck}
  Val(MaxUserEdit.Text,MaxUser,Code);
  if (Code<>0)or(MaxUser<1)or(MaxUser>95) then
    OKBtn.Enabled:=False;

  {NickCheck}
  if NickEdit.Text='' then OKBtn.Enabled:=False;
```

```
  ServerNick:=NickEdit.Text;

end;

end.


{*****************************************************************************}
{    Flüstern Dialog                                                          }
{*****************************************************************************}
unit fluestern;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TFluesternDlg = class(TForm)
    FluesternBtn: TButton;
    CancelBtn: TButton;
    Bevel1: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    MsgEdit: TEdit;
    UserEdit: TEdit;
    procedure UserEditChange(Sender: TObject);
    procedure MsgEditChange(Sender: TObject);
    procedure CheckValues;
end;

var
  FluesternDlg  : TFluesternDlg;

  FluesternUser,
  FluesternMsg  : String;

implementation
uses main;

{$R *.DFM}

procedure TFluesternDlg.UserEditChange(Sender: TObject);
begin
  CheckValues;
end;

procedure TFluesternDlg.MsgEditChange(Sender: TObject);
begin
  CheckValues;
end;

{Eingaben überprüfen}
procedure TFluesternDlg.CheckValues;
begin
  FluesternBtn.Enabled:=True;

  {UserCheck}
  if UserEdit.Text='' then FluesternBtn.Enabled:=False;
  FluesternUser:=UserEdit.Text;

  {MsgCheck}
  if MsgEdit.Text='' then FluesternBtn.Enabled:=False;
  FluesternMsg:='['+Nickname+']'+MsgEdit.Text;
end;end.
```

```
{*****************************************************************************}
{    Kicken Dialog                                                            }
{*****************************************************************************}

unit kicken;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TKickenDlg = class(TForm)
    KickBtn: TButton;
    CancelBtn: TButton;
    Bevel1: TBevel;
    UserComboBox: TComboBox;
    UserLabel: TLabel;
    procedure FormShow(Sender: TObject);
    procedure UserComboBoxChange(Sender: TObject);
end;

var
  KickenDlg : TKickenDlg;

  KickUser  : String;

implementation
uses main,ws;

{$R *.DFM}

procedure TKickenDlg.FormShow(Sender: TObject);
var
  Cnt : Integer;
begin
  UserComboBox.Clear;
  for Cnt:=2 to MaxSockets do
    if MySock[Cnt]<>0 then
      UserComboBox.Items.Add('('+IntToStr(Cnt)+')'+NickTable[Cnt]);
  UserComboBox.ItemIndex:=-1;
end;

procedure TKickenDlg.UserComboBoxChange(Sender: TObject);
begin
  if UserComboBox.ItemIndex<0 then KickBtn.Enabled:=False
  else KickBtn.Enabled:=True;
  KickUser:=UserComboBox.Items[UserComboBox.ItemIndex];
  KickUser:=Copy(KickUser,Pos(')',KickUser)+1,Length(KickUser));
end;

end.
```

# 5 Anhang

## 5.1 Screenshots

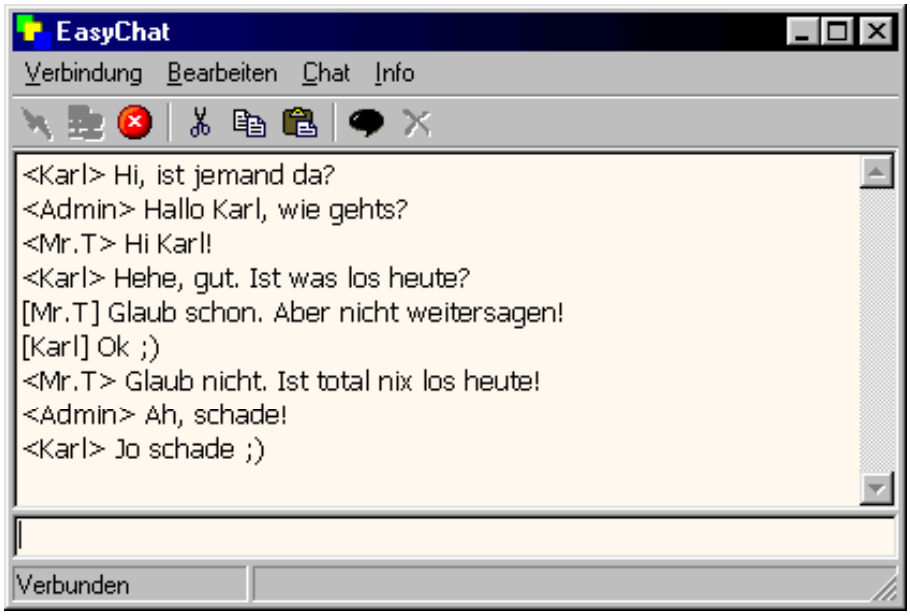


*Abbildung 6: EasyChat im Clientmodus.*

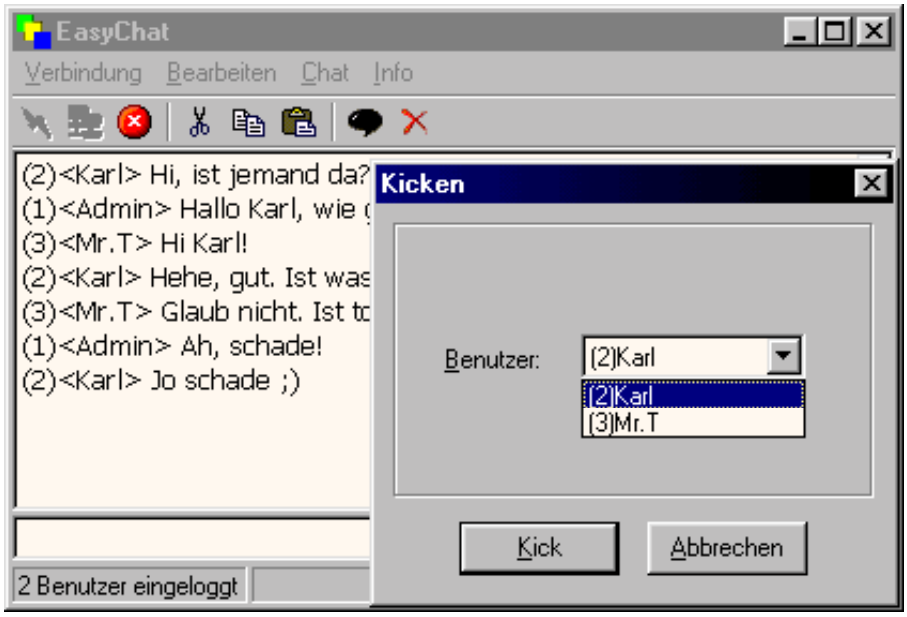


*Abbildung 7: EasyChat im Servermodus mit dem Kicken-Dialog*

## 5.2 Literaturverzeichnis

[1] Tischer Michael und Bruno Jennrich, *Internet intern: Technik & Programmierung*, Data Becker, 1997.

[2] Swan Tom, *Delphi 4 Bible*, IDG Books Worldwide Inc, 1998

[3] *Win32 Programmer's Reference* (win32.hlp), Microsoft, 1998

[4] http://www.nexor.com/info/rfc/index/rfc.htm?index/rfc.html

[5] http://src.doc.ic.ac.uk/computing/internet/rfc/rfc1700.txt

[6] ftp://ftp.microsoft.com/bussys/winsock/winsock2/wsapi22.doc

„Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe."

Ort: Datum: Unterschrift: