Aufbau und Konfiguration von Internet-Multimedia-Konferenzen über Verbindungen niedriger Bandbreite

Diplomarbeit von
Christian Zahl (Matrikelnr. 12 66 80)

Betreut durch Prof. Adam Wolisz (TU Berlin)
und Prof. Henning Schulzrinne (GMD Fokus Berlin /
Columbia University, New York)

Berlin, den 25. Februar 1997

Fachgebiet Telekommunikationsnetze
Institut für Offene Kommunikationsysteme
Fachbereich Informatik (FB 13)
Technische Universität Berlin

Forschungsgebiet für Offene Kommunikation
GMD Fokus Berlin

Computer Science Devision of the
Columbia University, New York

Inhaltsverzeichnis

Kapitel 1

Einleitung

Seit mehr als drei Jahren werden im INTERNET Multimedia-Konferenzen übertragen, wofür inzwischen fast ausschließlich das Real-Time Transport Protocol RTP [SCFJ96] Anwendung findet. Innerhalb des ISO OSI-Referenzmodells ist RTP auf der Transportebene angesiedelt und baut u.a. auf UDP [Pos80] (User Datagram Protocol) auf, welches wiederum auf dem Internet Protocol (IP) [Pos81] basiert.

Ziel dieser Arbeit ist es, per RTP transportierte Multimedia-Datenströme über Verbindungen mit niedriger Bandbreite zu übertragen, worunter insbesondere Modem- und ISDN-Verbindungen zu verstehen sind. Hierfür soll prinzipiell kein neues Übertragungsprotokoll entwickelt werden. Vielmehr soll es möglich sein, parallel zur Übertragung der RTP-Datenströme auch andere IP-Dienste wie e-mail, telnet, rlogin weiterhin nutzen zu können.

Um einem breiten Publikum Zugriff zu diesen Multimedia-Konferenzen zu ermöglichen, haben wir uns mit PPP-Verbindungen über ISDN-, insbesondere jedoch über Modemverbindungen beschäftigt.

Gliederung der Arbeit

In Kapitel 2 beschreiben wir zunächst die prinzipiellen Probleme, die bei der Übertragung von IP-Multicast-Datenströmen über PPP-Verbindungen entstehen.

Anschließend widmen wir uns dem zweiten Problemkreis, der Übertragung von RTP-Datenströmen über Verbindungen niedriger Bandbreite. Das Hauptproblem hierbei ist, daß die verfügbare Bandbreite in der Regel für die Übertragung einer Audio-Konferenz nicht ausreicht. Die Untersuchung dieses Problems ist Gegenstand von Kapitel 3.

Multimedia-Konferenzen werden im INTERNET mit Hilfe des Session Announcement Protocols (SAP) und Session Description Protocols (SDP) angekündigt. Potentiell interessierte Netzwerkteilnehmer können diesen Ankündigungen u.a. Zeitpunkt und Art einer Konferenz entnehmen. Da hierdurch im Prinzip eine verteilte globale Datenbank realisiert wird, müssen diese Ankündigungen periodisch verschickt werden. In Kapitel 4 analysieren wir, wie groß der Aufwand für die Übertragung dieser Ankündigungen ist. Wir entwerfen ein Verfahren, mit dem das aufkommende Datenvolumen erheblich gesenkt werden kann.

Weiterhin wird die Frage behandelt, wie groß die verfügbare Bandbreite einer benutzten Slow-Speed-Verbindung ist. Zur Beantwortung dieser Frage wird in Kapitel 5 ein Verfahren vorgestellt, mit dem diese Frage im laufenden Betrieb durch Messungen beantwortet werden kann. Dabei werden Faktoren, wie die Verwendung von Komprimierungsverfahren etc. behandelt.

Um die behandelten Probleme in der Praxis zu erproben, wurde eine Testumgebung implementiert, mit der die gemessenen Ergebnisse überprüft werden konnten. Die Beschreibung dieser Umgebung ist Ge-

genstand von Kapitel 6.

2. Teil der Arbeit

Zu Beginn der Arbeit stand der Einsatz von ISDN auf einer SUN-Workstation im Vordergrund. Zu diesem Zwecke wurde für eine SUN-SPARC-Station eine ISDN-Karte mit Software besorgt. Bedauerlicherweise stellte sich heraus, daß es mit dieser Software nicht möglich war, mit eigener Software die ISDN-Karte zu programmieren. Lediglich eine PPP-Implementierung gehörte zum Lieferumfang.

Deshalb griffen wir auf die Implementierung eines ISDN-Stacks der Universität Helsinki, Finnland zurück. Da sich diese derzeit in einem experimentellen Stadium befand, beschlossen wir, nach Rücksprache mit dem Betreuer dieser Arbeit (Henning Schulzrinne) sowie dem Projektleiter in Finnland (Bengt Olof Shalin), den ISDN-Stack zum Teil neuzuentwickeln.

Das Ergebnis ist eine ISDN-Library, deren Entstehung in Kapitel 7 beschrieben ist.

Es folgt eine abschließende Zusammenfassung der Arbeit in Kapitel 8, in der die erzielten Ergebnisse bewertet werden.

Die Anhänge

Es schließen sich folgende Anhänge an:

- Anhang A ist eine formale Beschreibung des in Kapitel 4.4 vorgestellten Protokolls zur Komprimierung von SAP-Nachrichten. Es ist in englischer Sprache und Form eines Internet-Drafts gehalten, um einer breiten Masse von Benutzern als Diskussionsgrundlage zur Verfügung zu stehen.
- Anhang B stellt die formale Beschreibung des Kompressionsverfahren für SDP-Nachrichten dar. Es ist ebenfalls in englischer Sprache gehalten und hat ebenfalls den Aufbau eines Internet-Drafts.
- Die von uns implementierte Testumgebung, die die Teilnahme an Multimedia-Konferenzen im INTERNET über eine PPP-ähnliche Verbindung realisiert, ist im Anhang C enthalten. Dieser enthält auch die Implementierungen für die SDP- und SAP-Komprimierung.
- In Anhang D ist die Beschreibung der in Kapitel 7 vorgestellten *IsdnLib* enthalten. Die Funktionen sind dort einzeln, in der Art von Manualpages beschrieben.
- In Anhang E ist der Quellcode für die in Kapitel 7 beschriebene ISDN-Library enthalten.

Kapitel 2

Benutzung von IP Multicast

Im INTERNET besitzt jeder Rechner eine[1] 32-Bit Adresse[2]. Mittels dieser Adresse kann jeder Rechner im INTERNET mit jeden anderen Rechner in direkten Kontakt treten und Daten austauschen. Dieses ist für die traditionellen Anwendungen, wie ftp (Filetransfer zwischen zwei Rechnern) oder telnet (login auf einen entfernten Rechner) optimal geeignet.

Durch die Einführung von Multimedia-Anwendungen, wie die Übertragung von Audiodaten, entstand jedoch der Wunsch, diese Daten an mehrere Rechner gleichzeitig übertragen zu können. Damit die benötigte Bandbreite hierbei nicht proportional mit der Anzahl der potentiellen Empfänger steigt, wurden sogenannte Gruppenadressen eingeführt. Diese Gruppenadressen sind somit nicht einem, sondern mehreren Rechnern gleichzeitig zugeordnet. Verwendet werden hierzu IP-Class-D-Adressen (auch Multicast-Adressen genannt). Die nötigen Erweiterungen und ein zusätzliches Kontrollprotokoll sind in [Dee89] beschrieben.

Es ist somit möglich, daß ein von einem Rechner ausgesandtes Datenpaket gleichzeitig von hunderten oder tausenden anderen Rechnern gleichzeitig empfangen werden kann, ohne daß die benötigte Bandbreite hierdurch steigt! Ein wesentlicher innovativer Aspekt liegt dabei in der Tatsache, daß diese Gruppen nicht statisch sind. Es existiert keine Konfigurationsdatei oder ähnliches, in der definiert wurde, welche Rechner zu welcher Gruppe gehören. Diese Zuordnung wird dynamisch von jedem Rechner selbst verwaltet. Mit Hilfe von IGMP [Dee89], dem „Internet Group Management Protocol“ wird Multicast fähigen Routern mitgeteilt, welche Multicast Gruppen in einem Netzsegment verwendet wird. Mittels geeigneter Protokolle, die zwischen den Routern ausgetauscht werden ist es möglich, die IP-Datagramme in den jeweiligen Netzsegmenten zu duplizieren. Verwendet werden hierzu PIM [DEF+96] (Protocol Independent Multicast) oder DVMRP [DPW88] (Distance Vector Multicast Routing Protocol).

Dieses IP-Multicast Verfahren wird nun verwendet, um Konferenzen im INTERNET zu realisieren. Somit kann beispielsweise ein Vortrag an einer Universität in den USA gehalten, und auf der ganzen Welt verfolgt werden, wenn dieser mittels IP-Multicast im INTERNET übertragen wird.

Derzeit ist es noch ein wenig problematisch diese IP-Multicast-Pakete im INTERNET zu übertragen, da dies einige Änderungen in den Routing-Implementierungen erforderlich macht. Aus diesem Grunde wurde das sogenannte MBONE (Multicast Backbone), ein virtuelles IP-Multicast-Netz im INTERNET eingerichtet. Hierbei werden die IP-Multicast Pakete zwischen zwei Teilnetzen über einen sogenannten Tunnel ausgetauscht. Innerhalb dieser Netze werden die Daten wieder als gewöhnliche IP-Multicast-Pakete verschickt. Für einen Rechner, der diese IP-Multicast-Pakete versendet oder empfängt besteht somit keine Abhängigkeit zu dem verwendeten Routing-Verfahren. Sollten in Zukunft die Router in der Lage sein, das Routing selber ausführen zu können, wird nach und nach das MBONE als solches

[1] Ein Rechner kann auch mehrere Adressen besitzen („multi-homed host“), jedoch ist dies für die weitere Betrachtung nicht von Bedeutung

[2] Dies gilt für IPv4[Pos81]. Derzeit wird an der Einführung von IPv6 [DH96] gearbeitet, bei dem die Adressen länger sind.

verschwinden, ohne daß die Rechner oder Benutzer irgend etwas davon merken.

Nachdem das Umfeld der Multimedia-Konferenzen im INTERNET beschrieben wurde, wollen wir nun die Probleme herausarbeiten, die bei der temporären Koppelung des Internets mit einem externen Rechner (wie etwa von zu Hause) bei der Übertragung von Multimedia Daten entstehen[3].

2.1 SLIP und PPP Verbindungen

Wenn ein Rechner temporär mit dem INTERNET gekoppelt wird, so wird in der Regel das Verbindungsprotokoll SLIP [Rom88] (Serial Line IP) oder PPP [Sim94a] (Point to Point Protocol) verwendet. SLIP wird in der Zukunft immer weniger Verwendung finden, da dieses keinerlei flexible Konfigurationen zuläßt. PPP hingegen definiert ein flexibles und erweiterbares Verfahren, um neue Übertragungs- und Kontrollprotokolle hinzuzufügen. Wir werden uns aus diesem Grunde ausschließlich mit PPP-Verbindungen beschäftigen.

2.1.1 IP-Multicast und PPP

Die jetzigen PPP-Implementierungen sind unserer Information nach nicht in der Lage, IP-Multicast-Pakete zu übertragen. Theoretisch betrachtet ist dies kein Problem, da es sich schließlich nur um IP-Pakete handelt. Da jedoch jede PPP-Verbindung einer Verbindung zwischen zwei Routern entspricht, entsteht hier das gleiche Problem wie bereits oben beschrieben.

Die eigentliche Schwierigkeit liegt nicht in der Übertragung der Multicast-Pakete an sich, sondern in der Entscheidung, welche Pakete zu übertragen sind. Angenommen in dem einen Teilnetz der PPP-Verbindung finden gerade mehrere IP-Multicast Konferenzen statt. Würden nun alle diese Pakete über die PPP-Verbindung geschickt werden, wäre diese sofort zu hunderten Prozent überlastet, da diese Verbindung in der Regel eine weit aus geringere Übertragungsrate zur Verfügung stellt, als die, die für die Konferenzen nötig wäre.

Aus diesem Grunde ist es wichtig zu wissen, welche Multicast Gruppen auf der jeweiligen Seite aktiv sind. Nur solche Gruppen, die auf beiden Seiten aktiv sind dürften über die PPP-Verbindung geroutet werden. Hierzu ist wiederum eines der bereits erwähnten Routing-Protokolle nötig, die jedoch bis jetzt nicht in PPP-Implementierungen realisiert wurden.

2.1.2 Warum kann IGMP nicht verwendet werden?

Man könnte sich fragen, warum man nicht einfach IGMP-Nachrichten über die PPP-Verbindung schicken kann. Wenn man dies täte, könnte man annehmen, daß beide Seiten der PPP-Verbindung wissen, welche Multicast-Gruppe auf jeder Seite aktiv ist. Solche Gruppen, die auf beiden Seiten benutzt werden, würden dann auch über die PPP-Verbindung gesendet werden.

Jedoch reicht dies nicht aus. Nehmen wir beispielsweise die Konfiguration in Abbildung 2.1 an. Zu irgendeinen Zeitpunkt t sendet der PPP-Router $P1$ eine `IGMP-REQUEST` Nachricht ins Netz. Jeder Rechner, der in der Lage ist IP-Multicast zu empfangen, empfängt daraufhin diese Nachricht. Für jede Multicast-Gruppe, in denen ein Rechner Mitglied ist, generiert er eine `IGMP-RESPONSE`. Diese wird nicht sofort, sondern erst nach einem zufälligen Timeout ins Netz gesendet (vgl. auch [Ste94]).

Nehmen wir an, der Timer vom Rechner A läuft zuerst ab, so daß dieser als erste seine Nachricht ins Netz schickt. Der-PPP Router $P1$ empfängt ebenfalls diese, und sendet sie an den anderen Router $P2$

[3] Es muß nicht unbedingt nur ein Rechner sein, der mit dem INTERNET gekoppelt wird, es kann sich auch um ein kleines Teilnetz handeln. Für die weitere Betrachtung spielt das jedoch nur eine untergeordnete Rolle.

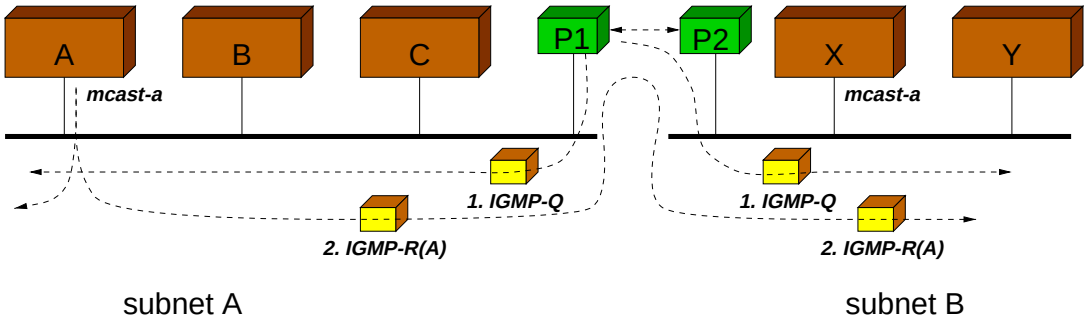


Abbildung 2.1: IGMP Nachrichten, wenn die PPP Router IGMP routen

weiter, der diese wiederum in das lokale Teilnetz auf seiner Seite sendet. Der Rechner X, der ebenfalls in der gleichen Multicast-Gruppe ist, empfängt diese Nachricht, und stoppt somit seinen Timer für das Aussenden der IGMP Nachricht. Er geht davon aus, daß alle Router das Paket ebenfalls empfangen haben, so daß er seine Mitgliedschaft nicht bekannt geben muß. Dies ist der normale Ablauf, so wie er in [Dee89] definiert wurde.

Durch dieses Verhalten können die beiden Router $P1$ und $P2$ nur feststellen, daß im Teilnetz A die Multicast-Adresse A verwendet wird. Keiner weiß jedoch, daß auch im Teilnetz B diese Gruppe benutzt wird. Aus diesem Grunde würden die Router auch keine IP-Multicast-Pakete über die PPP-Verbindung routen[4].

Das Problem könnte dadurch gelöst werden, daß die Router ein von dem anderen Router empfangenes `IGMP-REPORT` Paket nicht das lokale Netz schicken. Aus diesem Grunde würde dann der Timer von Rechner X ablaufen, so daß dieser ebenfalls einer `IGMP-REPORT` Nachricht sendet. Beide Router würden dann wissen, daß diese Adresse in beiden Netzen benutzt wird, und würden dann die IP-Pakete für diese Gruppe übertragen. Somit wäre das Problem der Übertragung von IP-Multicast-Paketen über PPP-Verbindungen bereits gelöst.

Das Problem besteht jedoch weiterhin! Wenn im benachbarten Teilnetz, das über die PPP-Verbindung verbunden ist, kein Rechner Mitglied dieser Multicastgruppe ist, gibt es auf die `IGMP-QUERY` nie eine Antwort aus dem Teilnetz A (vgl. Abbildung 2.2). Der Grund liegt darin, daß IGMP normalerweise nicht geroutet werden darf. Aus diesem Grunde wird der Router R aus dem Teilnetz A die IGMP-Nachricht auch nicht ins INTERNET schicken. Die PPP-Router würden somit immer annehmen, daß die Gruppe im benachbarten Teilnetz nicht benutzt wird, und würden somit auch keine Multicast-Pakete routen.

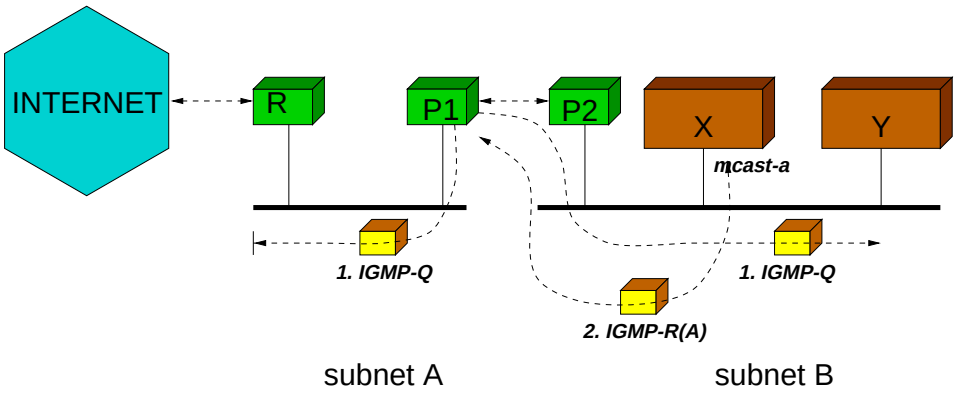


Abbildung 2.2: Problem mit IGMP und INTERNET

[4]In der Tat, nach einiger Zeit könnte es sein, daß der Timer von X vorher abläuft, so daß die Benutzung der Gruppe im Teilnetz B bekannt wäre. Wenn das Teilnetz A jedoch sehr groß ist, und mehrere Rechner Mitglied der Gruppe sind, kann das sehr lange dauern.

So oder so kommen wir zu dem Schluß, daß wir in jedem Fall ein entsprechendes Multicast-Routing-Protokoll benötigen, damit die PPP-Router wissen, welche Gruppen zu routen sind. Es gäbe nur eine temporäre Lösung. Wenn man weiß, daß das Teilnetz B keine anderen Netze enthält, dann könnte man die PPP-Verbindung als einseitige „Verlängerung“ des Netzwerkkabels von Teilnetz A verstehen. In diesem Fall, würden alle Multicast Pakete vom Teilnetz B zum Teilnetz A geschickt werden. Sobald der PPP-Router $R1$ ein solches Paket empfängt weiß er, daß auf der anderen Seite diese Gruppe in Benutzung ist. Er kann nun ebenfalls alle Multicast-Pakete für diese Gruppe über die PPP-Verbindung senden. Wenn für längere Zeit keine weiteren Multicast-Pakete aus dem Teilnetz B kommen, beendet er das Routing der Multicast-Pakete für diese Gruppe.

In unserer Testumgebung haben wir diese Probleme nicht weiter betrachtet. Die zu routenden Multicast-Adressen werden fest eingestellt.

Kapitel 3

Verwendung von RTP über PPP

3.1 Beschränkte Bandbreite

Im Gegensatz zu einem Ethernet ist die zur Verfügung stehende Bandbreite bei einer PPP Verbindung stark begrenzt. Da vor allem auch PCs zu Hause die Möglichkeit gegeben werden sollte, an Multimedia-Konferenzen im INTERNET teilnehmen zu können, haben wir insbesondere Modem und ISDN Verbindungen mit einem B-Kanal als physikalisches Übertragungsmedium angenommen.

Unsere Arbeit richtet sich somit in erster Linie an Verbindungen, deren physikalische Bandbreite zwischen 9,6 (einfache Modemverbindung) und 64 kbit/s (ein ISDN-B-Kanal) liegt. Die Überlegungen behalten auch bei höheren Bandbreiten ihre Richtigkeit.

3.2 Benötigte Bandbreite

Unabhängig von der Art der Übertragung, ist die zur Verfügung stehende Bandbreite stark beschränkt. Verwendet man eine ISDN-Verbindung, so stehen meist nur ein oder zwei B-Kanäle zur Verfügung. Dies entspricht einem Nettodurchsatz von 64 bzw. 128 kbit/s. Bei der Verwendung eines Modems sind heutzutage 28,8 kbit/s üblich.

3.2.1 Berechnung der benötigten Bandbreite

Üblicherweise besteht der Anspruch, Konferenzen mit einer Qualität zu übertragen, die wenigstens der einer Telefonverbindung entspricht. Im INTERNET werden deshalb solche Konferenzen meistens als μ-law komprimierter PCM [Int92a] Datenstrom übertragen. Die Abtastrate beträgt hierbei 8 kHz, mit einer Auflösung von 13 Bits, die durch die μ-law Kompression auf 8 Bits komprimiert werden. Dies entspricht einem Datenvolumen B_{audio} von

$$\begin{aligned} B_{\text{audio}} &= 8000\text{samples/s} * 8\text{bit/sample} \\ &= 64\text{kbit/s} \end{aligned}$$

Zu dieser Datenmenge kommen noch diverse Header und Trailer der benutzten Protokolle im INTERNET hinzu. Dies sind:

- 12 Bytes (wenigstens) für den-RTP Header, der die Audiodaten einkappselt und das Format etc. beschreibt.
- 8 Bytes für den UDP-Header, der zu jedem RTP-Paket hinzukommt.

- 20 Bytes (in der Regel) für den IP-Header, damit die Netzwerkpakete verschickt und empfangen werden können.

Weiterhin werden meistens 320 Samples der PCM-Daten in einem RTP Paket versendet, was 20 ms Audio-Daten entspricht. Rechnet man die Protokoll-Header nun noch zu den eigentlichen Audio-Daten hinzu, ergibt sich die benötigte Bandbreite B zu:

$$\begin{aligned} B &= (320 + 12 + 8 + 20)\text{bytes}/40\text{ms} \\ &= 9000\text{bytes/s} \\ &= 72 \text{ kbit/s} \end{aligned}$$

Vergleicht man diese benötigte Bandbreite mit der zur Verfügung stehenden, so fällt sehr schnell auf, daß eine Modemverbindung oder ein einzelner ISDN-B-Kanal hierfür nicht ausreicht. Die Folge wären massive Verluste von Paketen:

$$\begin{aligned} L_{64\text{kbit/s}} &= (72\text{kbit/s} - 64\text{kbit/s})/72\text{kbit/s} \\ &= 11\% \\ L_{28,8\text{kbit/s}} &= (72\text{kbit/s} - 28,8\text{kbit/s})/72\text{kbit/s} \\ &= 60\% \end{aligned}$$

Derartig hohe Verlustzahlen sind unakzeptabel, zumal noch ein weiterer Overhead durch die verwendetet Übertragungstechnik selber dazu kommt. Die zu erwartenden Verlustzahlen erhöhen sich somit noch einmal. Desweiteren kommt zu dem reinen Datenstrom durch RTP noch die durch RTCP (Realtime Transport Control Protocol) erzeugte Datenmenge, die abhängig von der Anzahl der Teilnehmer an der Konferenz ist. Unabhängig davon kann es bei einer Konferenz durchaus passieren, daß zeitweise mehr als ein Sender aktiv ist (z.B. im Falle einer Unterbrechung eines Sprechers).

3.2.2 Verringerung der benötigten Bandbreite

Die einzige Lösung, die sich hierfür anbietet ist die, daß das verwendete Audioformat geändert werden müßte. Es müßte ein Format gewählt werden, bei dem die zur Verfügung stehende Bandbreite noch ausreicht. Hierfür existieren im Prinzip zwei mögliche Ansätze:

1. Der Sender der Audiodaten sendet diese bereits im „richtigen“ Format, so daß die verfügbare Bandbreite noch ausreicht.
2. Vor der Übertragung der Daten über den slow-speed-link werden diese entsprechend der verfügbaren Bandbreite umgewandelt.

Der erste Punkt hat den Vorteil, daß direkt beim Sender, also an der Stelle, an der die Daten generiert werden, die Konvertierung ansetzt. Jedoch kann es auf der anderen Seite nicht vernünftig sein, daß aufgrund eines Einzelnen oder weniger die Qualität für alle anderen Teilnehmer der Konferenz abnimmt. Der zweite Ansatz scheint daher der vernünftigere zu sein, da hier nur diejenigen, die über eine schlechte Anbindung zum INTERNET haben mit der schlechteren Qualität auskommen müssen. Nachteilig ist jedoch, daß ein derartiges Komprimierungsverfahren dann für jeden slow-speed-link erneut durchgeführt werden muß.

3.2.3 Methoden zur Verringerung der Bandbreite

Eine unserer Forderungen war es, daß bestehende Applikationen wie vat oder NeVoT[1] weiterhin ohne Modifikationen benutzt werden können. Aus diesem Grunde ergab sich das Ziel, den RTP-Datenstrom

[1] Vat und NeVoT sind Programme, die eine Audiokonferenz im INTERNET ermöglichen

während der laufenden Konferenz mittels eines RTP-Mixers zu konvertieren. Dieser sollte den eingehenden RTP-Datenstrom empfangen, und je nach verfügbarer Bandbreite eine Formatwandlung vornehmen.

Nehmen wir nun an, daß dieser Mixer das Format nach GSM [Eur] wandelt. Somit entstehen für 320 Bytes PCM μ-law Daten 66 Bytes GSM kodierte Daten. Betrachtet man nun die Menge an Bytes, die für die verschiedenen Protokoll-Header benötigt werden (20 + 8 + 12 = 40 Bytes), so fällt auf, daß das fast genau so viele sind, wie an Nutzdaten zu übertragen sind.

Es stellt sich somit die Frage, ob es nicht möglich ist, diesen Overhead drastisch zu vermindern. Einen ähnlichen Ansatz gibt es bereits für Komprimierung der IP/TCP Header [Jac90]. In diesem Bericht wurden die einzelnen Felder der beiden Protokoll-Header während einer Verbindung untersucht. Dabei stellte sich heraus, daß sich ein großer Teil der Felder über die Lebensdauer der Verbindung nicht ändern. Die restlichen Felder ändern sich zwar, jedoch immer in der gleichen Art und Weise. So wird beispielsweise die Sequenznummer in einem IP-Protokoll-Kopf meistens um eins erhöht. Es ist somit möglich, anstelle der Protokollköpfe nur Änderungen zu übertragen. Da die Änderungen in vielen Fällen vorhergesagt werden können, müssen noch nicht einmal die Änderungen, sondern nur Abweichungen von den Vorhersagen übermittelt werden.

Zusammenfassend ergab sich, daß anstelle der IP/TCP-Protokollköpfe mit (20 + 20) Bytes im Durchschnitt nur 2 Bytes übertragen werden mußten! Es galt somit zu untersuchen, inwieweit etwas ähnliches auch bei der Kombination IP/UDP/RTP möglich ist.

Glücklicherweise wurde uns diese Arbeit bereits durch Stephen Casner und Van Jacobson (der auch die IP/TCP Komprimierung entwickelte) abgenommen. Sie verfaßten bereits einen Internet Draft, in dem exakt dieses Problem beschrieben und eine Lösung gefunden wurde [CJ96]. Das dort entwickelte Verfahren war in der Lage, den Overhead von 40 Bytes (20 + 8 + 12) auf ebenfalls nur 2 Bytes im Durchschnitt zu komprimieren.

Kapitel 4

Verwendung von Session Announcements

Bisher haben wir immer von Multimedia-Konferenzen im INTERNET gesprochen. Eine wesentliche Frage blieb bisher jedoch unbeantwortet: Welche Konferenzen existieren überhaupt, zu welchen Zeiten finden sie statt, und wie es ist möglich daran teilzunehmen?

Diese, sagen wir „Programmzeitschrift“ wird ebenfalls mittels IP-Multicast im INTERNET verbreitet. Eine Konferenz wird dabei mit Hilfe des „Session Description Protocols“ (SDP [Han96b]) beschrieben. Diese Beschreibung beinhaltet Informationen wie:

- den Titel der Konferenz,
- die Beschreibung des Inhaltes,
- die Zeit, zu der die Konfernz abgehalten wird. Bei regelmäßig abgehaltenen Konferenzen (wie Vorlesungen), wird der Tonus beschrieben,
- die verwendeten Medien und jeweiligen Formate,
- die Multicast-Adresse und Portnummer, die für die Übertragung verwendet wird,
- sonstige spezifische Parameter.

Das Format ist dabei fest definiert, so daß entsprechende Programme diese Ankündigungen aussenden und auch empfangen können. „Sdr“ ist ein Beispiel für ein solches Programm.

Übermittelt werden diese Ankündigungen mittels IP-Multicast unter Verwendung des Session Announcement Protocols (SAP [Han96a]). Die Ankündigungen werden dabei periodisch an eine definierte Multicast-Adresse gesendet, so daß eine globale verteilte Datenbank entsteht.

4.1 Übertragung von SAP/SDP über PPP

Auch Rechner, die nur temporär mit dem INTERNET über derartige PPP-Verbindungen gekoppelt sind, benötigen den Zugriff auf die Informationen über die stattfindenden Konferenzen. Demzufolge müßten diese SAP/SDP Pakete ebenfalls über die PPP-Verbindung gesendet werden.

Wie bereits in Kapitel 2 beschrieben, besteht das erste Problem darin, daß SAP mittels IP-Multicast übertragen wird. Um diese Pakete übertragen zu können, gibt es zwei Möglichkeiten:

1. Die Übertragung von IP-Multicast wird direkt im PPP realisiert.

2. Die Announcements werden über einen IP-Unicast „Tunnel“ zwischen zwei Prozessen (einer auf jeder Seite der PPP-Verbindung) übertragen.

Bei der zweiten Variante stellt sich die Frage, wie das konfiguriert werden muß. Dieser Tunnel zwischen den beiden Prozessen muß durch irgendeine Konfiguration eingerichtet werden. Besteht die PPP-Verbindung jedoch nur recht selten oder immer für relativ kurze Zeit, so entsteht hierdurch ein zusätzlicher Aufwand.

Die erste Variante ermöglicht die Übertragung der Informationen ohne weitere Konfiguration. Es besteht in diesem Falle sogar die Möglichkeit, daß die PPP-Implementierungen hierzu ein spezielles Protokoll verwenden, mit dem diese Announcements übertragen werden können. Somit lassen sich auch Kompressionsverfahren einsetzen, um die benötigte Bandbreite zu reduzieren.

4.2 Welche Bandbreite wird für SAP/SDP benötigt?

Zu Beginn dieses Kapitels beschrieben wir bereits, daß die SDP-Ankündigungen mittels SAP periodisch neu verschickt werden. Um nun bewerten zu können, wie groß das dadurch entstehende Datenvolumen ist, untersuchten wir, wieviele Ankündigungen mit welcher Größe in welcher Zeit versendet wurden. Die Ergebnisse sind in der Tabelle 4.2 zusammengefaßt. Die Messungen ergaben, daß im Durchschnitt ca.

Zeitraum	Anzahl Announcements	Anzahl der versch. Announcements	Datenvolumen
6,5h	1364	26	ca. 800 kbyte

Tabelle 4.1: Messung der empfangenen SAP/SDP Pakete

270 bit/s für dieses Announcements benötigt werden. Die Announcements waren durchschnittlich 585 Bytes groß und wurden alle 7,5 Minuten erneut verschickt.

Auf dem ersten Blick scheint dies recht wenig zu sein. Es ist jedoch damit zu rechnen, daß in der nahen Zukunft die Benutzung von Mulimediadiensten im INTERNET zunehmen wird. Wichtig ist jedoch zu beachten, daß diese Durchschnittswerte nur die durchschnittliche Belastung zeigen. Da die Announcements jedoch teilweise bis zu 800 Bytes groß waren, entsteht hierdurch ein temporärer Engpaß.

Angenommen, zwei große Announcements mit 800 Bytes folgen dicht aufeinander. Werden diese, insgesamt 1600 Bytes (12,8 kbit), und nehmen wir weiter an, daß wir eine 28,8 kbit/s Modemverbindung benutzen, würde die Verbindung für ca. 0,5 Sekunden vollständig belegt sein. In dieser Zeit können keine Echtzeitdaten mehr übertragen werden, so daß es zu einem kurzen drop-out kommen würde. Natürlich ist es möglich die Daten verzögert abzuspielen, also die End-zu-End Verzögerung zu erhöhen. Dies mindert jedoch die Sprachqualität (bei Audio) bei einer bidirektionalen Verbindung extrem.

Da die uns zur Verfügung stehende Bandbreite ohnehin nicht ausreicht, die Multimedia-Daten in ihrer ursprünglichen Form zu übertragen und wir diese auf jeden Fall komprimieren müssen, sollten auch für andere Dienste kein einziges Byte verschenkt werden. Je mehr Bandbreite für die Übertragung der Multimediadaten zur Verfügung steht, desto besser ist die resultierende Qualität.

Aus diesem Grunde haben wir ein Verfahren entwickelt, mit dem die Session Announcements stark komprimiert werden können. Die Ergebnisse unserer Überlegungen sind in den nächsten Abschnitten aufgeführt.

4.3 Wie könnten SAP/SDP Announcements komprimiert werden?

Wie schon des öfteren beschrieben basiert SAP auf eine verteilte Datenbank. Jeder Rechner, der eine Session ankündigt muß diese immer wieder neu verschicken, damit alle Rechner im INTERNET diese

auch erhalten. Wenn eine Ankündigung seit einer bestimmten Zeit nicht mehr empfangen wurde bedeutet dies, daß diese auch nicht mehr aktuell ist. Programme wie `sdr` entfernen aus diesem Grunde Announcements, wenn sie seit mehr als eine halben Stunde[1] nicht mehr empfangen wurden.

Die Notwendigkeit, daß die Announcements regelmäßig, in ihrem vollen Umfang verschickt werden müssen, resultiert aus der Tatsache, daß wir eine globale verteilte Datenbank haben. Wird beispielsweise ein Rechner (oder genauer, ein Programm wie `sdr`) zu irgendeinem Zeitpunkt gestartet, besitzt er keine Informationen über aktuelle Announcements. Nur die Tatsache, daß diese immer wieder neu verschickt werden macht es möglich, daß dieser Rechner nach einigen Minuten dennoch das gesammte Wissen über die aktuellen Announcements erlangt hat.

Wenn wir uns unsere PPP-Verbindung betrachten stellen wir jedoch einen anderen Sachverhalt fest. Die PPP-Verbindung ist von ihrer Natur her immer eine direkte Verbindung zwischen zwei Rechnern. Somit ist es auch möglich, daß jede Seite sich „merkt“, was die andere weiß. Es ist also nicht nötig die Session Announcements immer wieder in ihrer vollen Länge zu übertragen. Es reicht aus

- entweder nur die Information zu übertragen, daß ein Announcement erneut empfangen wurde,
- oder nur das erste Announcements von A nach B zu schicken und das implizite Timeout durch eine explizite „DELETE“ Nachricht von A nach B zu realisieren.

Der erste Vorschlag macht eine einfache Form einer Kompression möglich, bei dem A nur feststellen muß, ob B die Ankündigung bereits erhalten hat. Die zweite Variante hat den Vorteil, daß nach einiger Zeit faktisch keine Daten für die Announcements mehr ausgetauscht werden müssen. Dafür ist der Aufwand ein wenig höher, da sichergestellt werden muß, daß B diese erste Ankündigung auch wirklich erhalten hat. Weiterhin muß sich dann B selber um das zeitgerechte Verschicken der Ankündigungen in seinem Teilnetz kümmern.

Wir haben uns der Einfachheit halber für die ersten Variante entschieden, und eine Komprimierung nach diesem Muster realisiert.

4.4 Komprimierung von SAP

Die generelle Idee ist hierbei, daß wir ein Client-Server-Verhältnis zwischen den beiden Stationen haben, die über die PPP-Verbindung gekoppelt sind, haben. Der Server der einen und der Client der anderen Station haben beide eine gemeinsame Liste von Kontexten. Jeder dieser Kontexte enthält die Informationen über exakt eine Ankündigung. Identifiziert werden diese Kontexte durch eine Kontext-ID.

4.4.1 Erzeugen eines Kontextes

Der Server ist diejenige Instanz, die einen Kontext erzeugt. Wenn eine PPP Verbindung aufgebaut wird, haben beide, Server und Client (auf jeder Seite) eine leere Listen von aktiven Kontexten. Empfängt ein Server nun ein SAP Paket, so durchsucht er seine Liste der Kontexte nach diesem Announcement[2]. Da diese leer ist findet er somit keinen Eintrag und erzeugt einen neuen Kontext. Dieser wird mit

[1] Die Aussage stimmt nicht ganz, die Zeit nach der ein „Implicit Timeout“ stattfindet wird über eine kompliziertere Formel bestimmt. Hierbei fließt der TLL-Wert, die Größe der Ankündigung und die Anzahl der momentan versendeten Ankündigungen mit ein. Er ist jedoch immer größer gleich einer halben Stunde, so daß wir uns hier auf diesen Wert beschränken.

[2] Zum Vergleichen gibt es mehrere Möglichkeiten. Zum einen kann der „Message Identifier Hash“ Eintrag des SAP Announcements verwendet werden. Wie wir aber später noch sehen werden, ist das hier beschriebene Verfahren nicht ausschließlich für SAP / SDP zu verwenden. Aus diesem Grunde haben wir uns für einen Vergleich der Nachrichten auf Byteebene entschieden. Somit sind wir unabhängig von den verwendeten Announcement-Protokoll.

- dem empfangenen SAP/SDP Paket und dessen Länge,
- der eindeutigen Kontext-ID für diesen Kontext,
- der IPv4-Adresse und UDP-Portnummer des Absenders,
- der IPv4-Adresse und UDP-Portnummer der Zieladresse (bei SAP/SDP fest definiert)

initialisiert. Da wir, wie bereits erwähnt, auch andere Announcement-Protokolle außer SAP/SDP mit diesem Verfahren komprimiert übertragen können, wird noch

- eine Identifikation des verwendeten Announcement-Protokolls (wie SAP) und
- des verwendeten Beschreibungsformates (wie SDP)

im Kontext abgelegt.

Nachdem der Server den Kontext angelegt hat, sendet er (fast) alle Daten aus dem Kontext in einer `ASSIGN` Nachricht zum Client. Der Client, der diese empfängt durchsucht daraufhin seine eigene Liste der Kontexte nach der ebenfalls übermittelten Kontext-ID. Findet es einen Eintrag mit der gleichen Kontext-ID, so wird dieser Kontext gelöscht.

Der Client erzeugt nun einen neuen Kontext und speichert ebenfalls, analog zum Server, alle Daten in diesem Kontext. Im Anschluß daran, sendet er auf seiner Seite des Netzes dieses Announcement.

4.4.2 Erneuter Empfang eines Announcements

Wenn der Server nach einiger Zeit das gleiche Announcement erneut empfängt, findet er es in seiner Liste mit den Kontexten. Er weiß daraufhin, daß der Client ebenfalls einen solchen Kontext besitzt. Anstatt das Announcement erneut zu übertragen, sendet der Server nur eine kurze `RETRANSMIT` Nachricht an den Client. In dieser ist nun nur noch die Kontext-ID enthalten, die zuvor vom Server für dieses Announcement vergeben wurde.

Wenn der Client diese Nachricht empfängt, sucht er mit Hilfe der Kontext-ID den richtigen Kontext. Nachdem dieser gefunden wurde, sendet der Client das Announcement, das im Kontext enthalten ist (vgl. Abbildung 4.1).

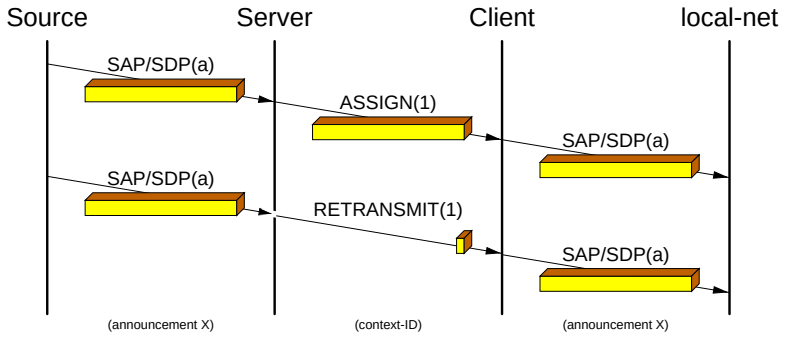


Abbildung 4.1: Komprimierte Übertragung der Session Announcements

4.4.3 Behandlung von Fehlersituationen

Es kann nun auch vorkommen, daß der Client eine `ASSIGN` Nachricht nicht erhält, da das Paket verloren ging (z.B. aufgrund eines CRC Fehlers von dem Link-Layer verworfen wurde). Der Client würde

somit nach einiger Zeit eine `RETRANSMIT` Nachricht mit einer Kontext-ID empfangen, für die er jedoch keinen Kontext findet.

Damit dieses Announcement dennoch auf der Seite des Clients bekannt wird, benötigen wir somit einen Mechanismus um diesen Fehler signalisieren zu können. Der Client sendet in diesem Fall eine `DELETED` Nachricht an den Server zurück.

Sobald dieser diese Nachricht empfängt, sucht er den Kontext anhand der ebenfalls übermittelten Kontext-ID. Wenn er diesen findet, generiert er daraufhin eine neue `ASSIGN` Nachricht für diesen Kontext, und sendet sie zum Client (Abbildung 4.2). Auf diesem Wege wird der Kontext im Client erneut angelegt, und der Fehler ist beseitigt.

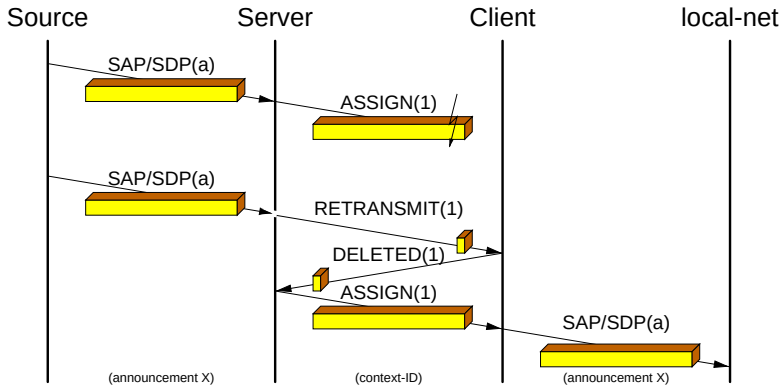


Abbildung 4.2: Synchronisation zwischen Client und Server nach Paketverlust

Das einzige Problem, was in diesem Fall noch bestünde wäre, wenn nun auch diese `DELETED` oder die `ASSIGN` Nachricht verloren geht (vgl. Abbildung 4.3). Der Server würde somit nicht über den Fehler informiert, oder der Client erhält nicht die neue Zuweisung, so daß dieses Announcement nicht auf der Seite des Client bekannt wäre. Da die Announcements jedoch periodisch neu verschickt werden, ergibt sich bei dem nächsten Announcement wiederum die gleiche Situation: Der Server sendet eine `RETRANSMIT` Nachricht, und der Client antwortet mit einer `DELETED` Nachricht. Die Wahrscheinlichkeit, daß gerade diese Pakete verloren gehen scheint sehr gering zu sein, so daß nach einiger Zeit auch dieses Announcement auf der Client-Seite bekannt wird.

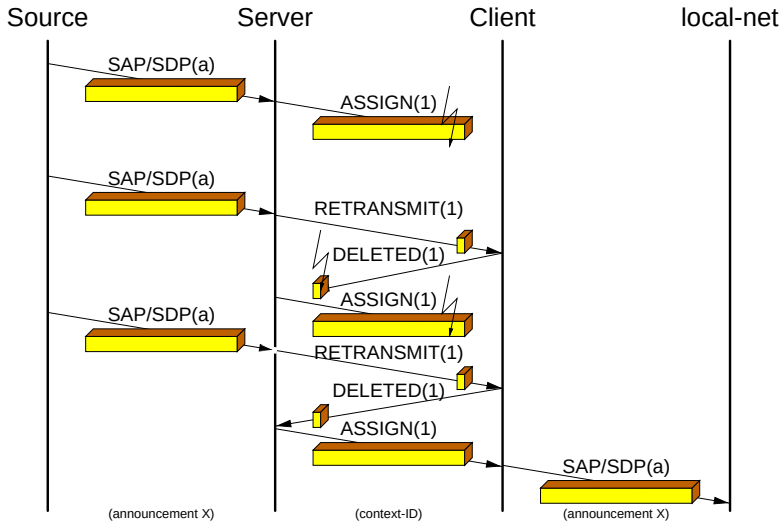


Abbildung 4.3: Synchronisation zwischen Client und Server nach zweifachem Paketverlust

Es gibt jedoch noch einen weitaus komplizierteren Fehlerfall. Angenommen ein Kontext (1) wurde

beim Server und Client richtig zugewiesen und benutzt. Wenn der Server nun entschied den Kontext zu löschen, weil seit gewisser Zeit keine weiteren Announcements (a) mehr eingegangen sind, wird somit auch die Kontext-ID frei. Da unser entwickeltes Verfahren keine explizite Nachrichten zum Löschen eines Kontextes beim Server verschickt, existiert der Kontext weiterhin beim Client.

Wenn der Server dieselbe Kontext-ID (1) für ein neues Announcement (b) vergibt, sendet er eine `ASSIGN` Nachricht zum Client. Wie oben bereits beschrieben, entsteht hierdurch kein Problem, da der Client beim Empfang dieser Nachricht den alten Kontext zuerst löscht.

Sollte jedoch diese `ASSIGN` Nachricht verloren gehen, entsteht ein Problem. Empfängt der Server das gleiche Announcement (b) erneut, so geht er von einem korrekten Kontext beim Client aus. Er sendet somit nur eine `RETRANSMIT` Nachricht zum Client. Dieser empfängt diese, durchsucht seine Kontextliste nach der Kontext-ID (1) und findet noch den alten Kontext. Der Client geht nun davon aus, daß das Announcement (a) erneut geschickt werden soll, und sendet somit das falsche, alte Announcement (a) in sein Teil des Netzes (siehe Abbildung 4.4). Es gibt keine Möglichkeit für ihn, diesen Fehler zu bemerken.

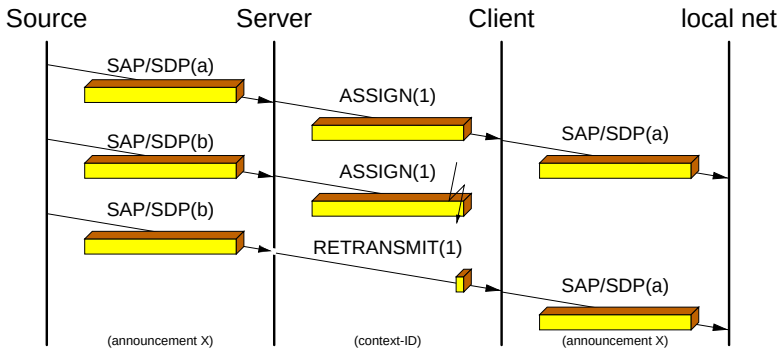


Abbildung 4.4: Aussenden des Announcements durch Paketverlust

Aus diesem Grunde bekommt jeder Kontext und jede Nachricht eine sogenannte „Genneration-ID“ hinzu. Diese ID wird vom Server verwaltet und jedesmal geändert (im Allgemeinen um eins erhöht), wenn die Kontext-ID neu vergeben wird. Empfängt nun der Client eine `RETRANSMIT` Nachricht, bei der die Generation-ID (2) nicht mit der im Kontext (1) übereinstimmt, weiß er, daß er mindestens eine `ASSIGN` Nachricht vom Server nicht erhalten hat. In diesem Fall löscht er den Kontext, und sendet eine `DELETED` Nachricht an den Server zurück (vgl. Abbildung 4.5). Es ergibt sich dann der gleiche Sachverhalt wie zuvor beschrieben.

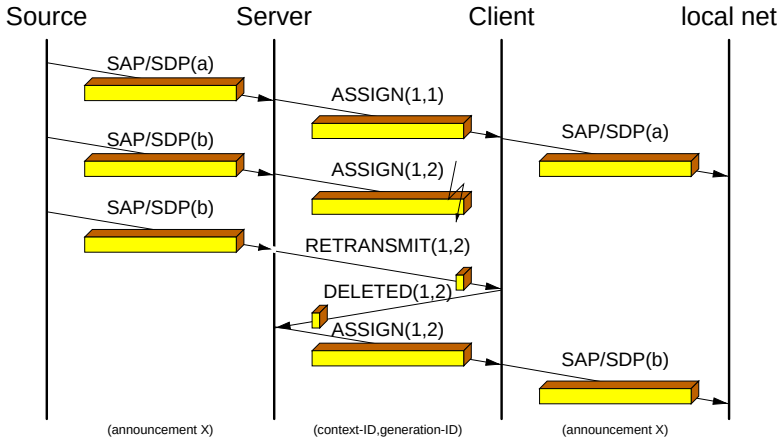


Abbildung 4.5: Korrektur durch Verwendung der Generation-ID

4.4.4 Implizites Timeout

Im Internet Draft für SAP [Han96a] ist angegeben, daß es ein implizites Timeout für Announcements gibt. Wenn ein Announcement seit einer bestimmten Zeit nicht mehr empfangen wurde, sollte das Announcement automatisch aus dem Verzeichnis der empfangenen Programme gelöscht werden.

Damit auch bei unserem Vorschlag ein Kontext nicht unendlich lange erhalten bleibt, sollte der Server zu jedem Kontext eine Timeout-Zeit angeben. Diese wird bei jedem Eintreffen eines Announcements neu aktualisiert. In Zeiten, in denen der Server nicht viel zu tun hat, kann dieser seine Liste mit den Kontexten durchsuchen, und alle Kontexte löschen, bei denen diese Zeit abgelaufen ist. Es gibt keine explizite Benachrichtigung des Clients. Durch die erneute Vergabe einer zuvor benutzten Kontext-ID, wird der alte Kontext beim Client durch den Empfang der `ASSIGN` Nachricht ohnehin gelöscht.

Dennoch hat auch der Client die Option, Kontexte, die seit einer gewissen Zeit nicht mehr aktualisiert wurden, selbstständig zu löschen. Auch hier gibt es keine Signalisierung zum Server. Sollte nun dennoch ein Announcement für diesen Kontext beim Server eingehen, so bekommt der Client eine `RETRANSMIT` Nachricht. Da dieser die Kontext-ID nicht in seiner Liste findet, greift die gleiche Fehlerbehandlung wie oben beschrieben. Der Client sendet daraufhin eine `DELETED` Nachricht zurück, die der Server mit einer neuen `ASSIGN` Nachricht beantworten wird. Es enstehen auch in diesem Fall keine Fehlinterpretationen, noch gehen irgendwelche Announcements verloren.

4.4.5 Caching Option bei temporären Verbindungsabbruch

Angenommen die PPP-Verbindung zwischen Client und Server ist eine Modemverbindung. Es kann dabei durchaus vorkommen, daß diese von den Modems aufgrund einer veränderten Leitungsqualität getrennt wird. Wird die Verbindung nun wieder aufgebaut, müßten eigentlich alle Kontexte beim Server und Client gelöscht werden. Es dauert dann also einige Minuten, bis die Kontexte wieder eingerichtet wurden, so daß in dieser Zeit die Daten nahezu unkomprimiert mittels `ASSIGN` Nachrichten versendet werden müssen.

Um diese Last zu verringern, können Client und Server die Option benutzen, die Kontexte nicht zu löschen, wenn sie sicher sind, daß der „Partner“ der war, mit dem sie gerade zuvor Kontakt hatten. Auf diese Weise würden nur die `RETRANSMIT` Nachrichten übertragen werden müssen.

Es gibt nun die folgenden vier Konstellationen:

1. Beide haben die Kontexte nicht gelöscht. In diesem Fall ist alles OK.

2. Der Client hat die Kontexte gelöscht, da inzwischen ein anderer Server Kontakt aufgenommen hatte. In diesem Fall würde der Client auf jede zuerst eintreffende `RETRANSMIT` Nachricht mit einer `DELETED` Nachricht antworten. Im Laufe der Zeit würden sich somit auch die Kontexte wieder synchronisieren. Wenn der Server „intelligent“ ist, bemerkt er die vielen `DELETED` Nachrichten, und kann dann, in eigener Verantwortung, ebenfalls seine Kontexte löschen. Es entfällt somit das anfängliche „Ping-Pong-Spiel“.

3. Der Server hat seinen Cache mit den Kontexten gelöscht. In diesem Fall sendet er für jedes, das erste Mal eingehende Announcement eine `ASSIGN` Nachricht. Die Liste der Kontexte wird beim Client somit langsam überschrieben[3].

4. Beide haben ihre Kontexte gelöscht. Dann fangen wir so an, wie immer, es gibt also keinerlei Probleme.

[3] Ein Problem ist jedoch die momentane Generation-ID. Wenn eine `ASSIGN` Nachricht verloren geht, und zufälligerweise die Generation-ID im Kontext die gleiche ist wie die in der folgenden `RETRANSMIT` Nachricht, sendet der Client falsche Daten. Es gilt in der Zukunft zu untersuchen, wie wahrscheinlich dieser Fall ist.

4.4.6 Verwendung eines neuen PPP-Protokolls

Wie bei der bisherigen Beschreibung bereits aufgefallen sein dürfte, haben wir immer von „Nachrichten“ gesprochen. Der Grund ist der, daß wir für unser Protokoll keine IP/UDP-Pakete benötigen. Es handelt sich um ein neues Network-Protocol für PPP, das als solches bei IANA (Internet Assigned Number Authority) registriert werden müßte.

Eine detailierte Beschreibung unseres Protokolls ist in Anhang A enthalten. Diese ist in der Form eines Internet-Drafts aufgebaut, so daß diese recht schnell verbreitet werden könnte. Aus diesem Grunde wurde diese Spezifikation auch in englischer Sprache erstellt.

4.5 Komprimierung von SDP-Daten

Wenn man sich einmal eine SDP-Nachricht ansieht, stellt man fest, daß diese ausschließlich aus ASCII Text in Klartext besteht. Eine Internet Adresse, die sonst 4 Bytes groß ist, wird nun in „dotted decimal notation“ angegeben, und verbraucht damit bis zu 15 Bytes. Derartige Felder lassen sich natürlich stark komprimieren.

Wir entwickelten hierfür ebenfalls ein Verfahren, um SDP Nachrichten zu komprimieren. Die Komprimierungsrate ist dabei um Einiges besser, als die einfache, von Mark Handley [Han96b] vorgeschlagene Variante durch Benutzung von `gzip`.

An dieser Stelle wollen wir nicht genauer auf das Verfahren eingehen, da dies eine genaue Betrachtung jedes Feldes eines SDP Paketes voraussetzt. In Anhang B ist eine englischschprachige detailierte Spezifikation enthalten. Wie schon zuvor, wurde sie in Form eines Internet-Drafts gehalten, damit sie eine weite Verbreitung erfährt.

4.6 Zusammenfassung

In diesem Kapitel haben wir die Problematik beschrieben, die bei der Benutzung von SAP und SDP über Verbindungen niedriger Bandbreite entstehen. Wir haben Verfahren vorgestellt, die die benötigte Bandbreite stark reduziert. So muß für jedes wieder empfangene Announcement nicht das Announcement selber, zuzüglich der IP und UDP-Header gesendet zu werden, sondern nur noch 2 Bytes! Um auch die Bandbreite zu reduzieren, die bei der initialen Übertragung des Announcements, zum Aufbau des Kontextes nötig ist, komprimierten wir auch die eigentlichen SDP-Daten.

Das hier vorgestellte Verfahren ist im Prinzip auch für andere Announcement Protokolle verwendbar, es ist also nicht nur auf SAP und SDP beschränkt. Wir haben es allerdings nur mit SAP und SDP getestet.

In unserer Testumgebung, in der wir ein kleines Teilnetz mit dem INTERNET koppelten, haben wir auch diese Verfahren mit eingebaut und verwendet. Exakte Messungen des Gesamtgewinns, liegen derzeit noch nicht vor.

Der Quellcode für die Implementierungen ist in Anhang C enthalten.

Kapitel 5

Bandbreitenmessung einer PPP Verbindung

Ein wichtiger Parameter, der für die Entscheidung, welches komprimierte Audioformat noch über die Netzverbindung übertragen werden kann, sehr wichtig ist, ist die zur Verfügung stehende effektive Nutzbandbreite. Es gibt dabei zwei Ansätze, diesen Parameter zu erhalten:

1. Durch feste Angabe der Bandbreite, was beispielsweise bei einer ISDN B-Kanal Verbindung recht einfach geht.
2. Durch Messungen während der bestehenden Netzwerkverbindung.

Jede der beiden Varianten hat seine Vor- und Nachteile. Der erste Ansatz erfordert einen gewissen Aufwand für die Konfiguration. Wird beispielsweise ein Router benutzt wird, der dynamisch neue ISDN B-Kanäle auf- oder abbaut (je nach momentaner Auslastung), ist dies nicht so einfach zu benutzen. Die PPP-Protokolle müßte hierzu schon direkt die Information erhalten, wieviele Kanäle derzeit zur Verfügung stehen. Das Verfahren versagt aber dann, wenn es sich um ein Medium handelt, bei dem die Bandbreite von vornherein nicht feststeht, oder schwanken kann, wie etwa bei Modem-Verbindungen.

Die zweite Variante birgt die Gefahr der Fehlmessung, etwa bei Datenkompression. Weiterhin wäre es zu erwarten, daß ein zusätzlicher Aufwand nötig ist, um während einer laufenden Verbindung die Bandbreite messen zu können. Auf der anderen Seite können so dynamische Änderungen, wie die Anpassung der Anzahl der B-Kanäle oder die Verwendung eines anderen Übertragungsverfahren bei einer Modem Verbindung, erkannt werden.

In diesem Kapitel befassen wir uns mit dem zweiten Ansatz, der Messung der effektiv zur Verfügung stehenden Bandbreite einer PPP-Verbindung über eine Modemstrecke. Wir betrachten dabei die Einstellungen verschiedener Parameter, und versuchen jeweils eine Beziehung zwischen der gemessenen und tatsächlich vorhandenen Bandbreite herzustellen.

5.1 Wie kann die Bandbreite gemessen werden?

Für die Messung der Bandbreite verwenden wir ein Verfahren, das sich „Packet-Pair“ nennt [Kes94]. Es wird hierbei die Verzögerung gemessen, in der zwei direkt hintereinander gesendete Datenpakete beim Empfänger eintreffen. Diese spezielle Idee wurde von uns aus einer Arbeit abgeleitet, die sich mit der Bestimmung von Bottlenecks zwischen zwei entfernten Stationen befaßte [CP97].

Wir beschreiben das Verfahren hier nicht weiter, und verweisen stattdessen auf die beiden Arbeiten [Kes94, CP97].

5.2 Beschreibung der Versuchsumgebung

In den folgenden Abschnitten werden einige Messungen der Bandbreite über eine PPP-Verbindung vorgestellt. Diese Messungen wurden zwischen zwei UNIX Systemen (AIX und Linux) mittels einer Modemverbindung durchgeführt. Damit die Messungen so genau wie möglich sind und eventuell auftretende Störungen das Ergebnis nicht zu sehr negativ beeinflussen, wurde für die Modemverbindung eine Übertragungsrate von 4800 bit/s (V.32) verwendet.

Die Messungen wurden unter Verwendung des in [CP97] beschriebenen „Packet-Pair-Verfahrens“ durchgeführt. Gemessen wurde demzufolge die Zeit, die für den Empfang des zweiten Paketes benötigt wurde. Die Versuche wurden im 1-Sekunden Rhythmus 100 mal ausgeführt, und jeweils die effektive Bandbreite errechnet.

Sofern nicht anders beschrieben, wurden alle anderen Protokolle ausgeschaltet, es wird also ohne Datenkompression und Fehlerkorrektur gearbeitet:

- keine Verwendung von V.42 oder MNP 4 (Fehlerkorrektur)
- keine Verwendung von V.42bis oder MNP 5 (Datenkompression)

Auch für die PPP-Verbindung wurden alle Arten von Kompression ausgeschaltet:

- keine Address-Control-Compression
- keine Protocol-Field-Compression
- keine Van-Jacobson-Header-Compression
- keine Datenkompression

Die Geschwindigkeit zwischen dem Modem und dem jeweiligen Rechner betrug 38400 bit/s. Somit ist sichergestellt, daß diese Schnittstelle kein Engpaß darstellt.

Weiterhin wurde die Asynchron Control Character Map (ACCM) [Sim94b] auf den Wert `0x00000000` gesetzt, so daß keine Byte-Werte escaped werden müssen. Um eventuelle Optimierungen oder Kompressionen im Datenstrom weitgehend ausschließen zu können, wurde der Datenpuffer der zu übertragenen Pakete immer mit zufälligen Werten gefüllt

Das Programm auf der Empfängerseite empfängt die beiden Pakete und mißt die Zeit, die zwischen dem Eintreffen des ersten und zweiten Paketes vergeht. Mit Hilfe der Größe des empfangenen zweiten Paketes und der gemessenen Zeit, wird die resultierende effektive Bandbreite errechnet und ausgegeben. Ein Ausschnitt aus einem solchen Protokoll zeigt die folgende Liste:

```
...
10      3.200051        # 249.996 ms
11      3.077278        # 259.970 ms
12      3.200013        # 249.999 ms
13      3.200166        # 249.987 ms
14      3.199987        # 250.001 ms
15      3.199962        # 250.003 ms
16      3.077290        # 259.969 ms
17      3.200102        # 249.992 ms
18      3.200166        # 249.987 ms
19      3.200038        # 249.997 ms
20      3.200013        # 249.999 ms
...
```

Die erste Spalte gibt die Testnummer an, die zweite die errechnete effektive Bandbreite in kbit/s. Die dritte dient nur zur Kontrolle und enthält die gemessene Zeit in Millisekunden.

5.3 Theoretisch meßbare Bandbreite

Die oben beschriebene Modemverbindung arbeitet bei einer Übertragungsrate von 4800 bit/s (V.32 [Int93h]. V.32 stellt eine bitserielle synchrone Verbindung zwischen den Modems her. Damit Bytegrenzen erkannt werden können, und damit es auch Zeiten geben kann, in denen keine Nutzdaten übertragen werden, wird ein Start- Stopbit Verfahren nach V.14 [Int93o] verwendet. Jedes Byte wird durch ein Startbit eingeleitet und durch ein Stopbit beendet. Werden keine Nutzdaten übertragen, so werden fortlaufend Stopbits gesendet. Der Empfänger kann somit zwischen Ruhephase und Datenübertragung unterscheiden, und die Bytes aus dem Bitstrom zurückgewinnen.

Ein Byte besteht aus 8 Bits, demzufolge müßten

$$\begin{aligned} B &= (4800\text{bit/s}/8\text{bit/byte}) \\ &= 600\text{byte/s} \end{aligned}$$

übertragen werden können. Aufgrund des Start- Stopbit Verfahren werden jedoch 10 Datenbits für 8 Nutzdaten-Bits übertragen. Die somit für Nutzdaten effektiv zur Verfügung stehende Bandbreite B_{eff} ergibt somit zu

$$\begin{aligned} B_{\text{eff}} &= (4800\text{bit/s}/10\text{bit/byte}) \\ &= 480\text{byte/s} \\ &= 480 * 8\text{bit}_{\text{eff}}/\text{s} \\ &= 3840\text{bit}_{\text{eff}}/\text{s} \end{aligned}$$

Bei einer Bandbreitenmessung müßte somit einen Wert ergeben, der dem in der Gleichung 5.1 entspricht.

Es ergibt sich somit ein Overhead O von

$$\begin{aligned} O &= B - B_{\text{eff}} \\ &= 4800 - 3840 \\ &= 960 \\ O_{\text{rel}} &= O/B_{\text{eff}} \\ &= 960/3840 \\ &= 25\% \end{aligned}$$

, es werden also 25% mehr Bits übertragen, als an Nutzdaten anfallen.

5.3.1 Protokoll Overhead

Die von uns verwendete Modemverbindung wurde als Netzwerkverbindung mit PPP und IP verwendet. Somit kommen zu den eigentlichen Nutzdaten noch eine Reihe von Protokoll-Headern und Trailern hinzu, die ebenfalls beachtet werden müssen. Der Aufbau eines UDP-Datagramms, das über eine serielle PPP-Verbindung gesendet wird, ist in der Abbildung 5.1 dargestellt.

Erläuterung der einzelnen Ebenen:

- **UDP**
  Für unser Meßverfahren verwenden wir UDP-Pakete. Ein UDP-Paket hat einen festen 8 Byte Header, der bei jedem Paket mit übertragen wird.

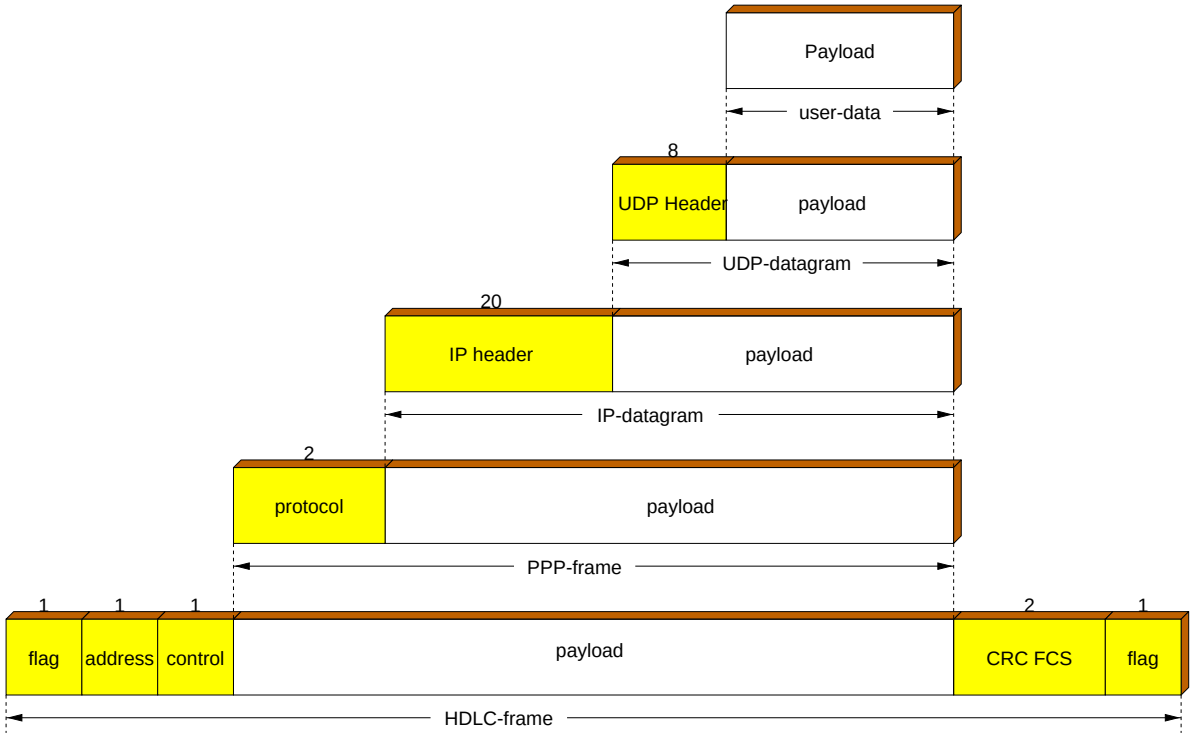


Abbildung 5.1: Aufbau eines PPP-Frames

- **IP**
  Die UDP-Pakete werden wiederum in einem IP Paket versendet, welches einen (im allgemeinen) fixen Header einer Länge von 20 Bytes besitzt.

- **PPP**
  Das verwendete Link-Layer-Protokoll PPP[Sim94a] hat seinerseits ein eigenes Paketformat. Es besteht aus einem 2 Byte Protokoll-Feld, den Nutzdaten (das von uns zu versendende IP-Paket) und einem 2 Byte CRC Feld.

- **HDLC**
  Auf unterster Ebene wird das PPP Paket in einem HDLC-Frame über die serielle Verbindung gesendet [Sim94b]. Jeder HDLC-Frame wird durch ein 1 Byte langes Flag eingekapselt. Danach folgt ein 1 Byte langes Address-Feld gefolgt von einem 1 Byte Control-Feld. Diese HDLC-Kapselung wird jedoch nur dann verwendet, wenn die Daten über eine serielle Schnittstelle (auch simuliert wie im ISDN mittels V.120 [Int92b]) gesendet werden.

  Die Verwendung des Flags kann je nach Implementierung ein wenig variieren. Werden zwei PPP-Pakete direkt hintereinander gesendet, Können die beiden Flags zwischen den beiden Paketen (am Ende des ersten und am Anfang des zweiten) durch eins ersetzt werden. Obwohl die von uns verwendete PPP-Implementierung[1] dies unterstützt, wurden jedoch immer beide Flags gesendet.

5.3.2 Zu erwartende Bandbreite

Bei der Verwendung von PPP kann es vorkommen, daß einige Bytewerte „escaped“ werden müssen (siehe 5.5). Auch wenn die `ACCM` auf `0x00000000` eingestellt wurde, werden weiterhin Bytewerte, die dem Wert eines `FLAG` und eines `ESC` entsprechen, escaped. Somit entsteht ein weiterer Overhead, so

[1] Wir verwendeten die Implementierung `PPP-2.2`, die als Sourcecode der Linux 2.0 Distribution beiliegt.

daß sich die Anzahl der zu versendenden Bytes $N_{esc}(n)$ um die Auftrittswahrscheinlichkeit der beiden Werte erhöht.

$$\begin{aligned} N_{\text{esc}}(n) &= n + n * (2/256) \\ &= n * (1 + 2/256) \\ &= 1,007812 * n \end{aligned}$$

Die reale Größe in Bytes eines zu übertragenden Datenpaketes über UDP ist somit erheblich größer:

$$\begin{aligned} N(n) &= N_{\text{HDLC-FLAGS}} + N_{\text{esc}}(N_{\text{UDP}} + N_{\text{IP}} + N_{\text{PPP}} + N_{\text{HDLC8}} + n) \\ &= 2 + 1,007812 * (8 + 20 + 2 + 4 + n) \\ &= 2 + 1,007812 * (34 + n) \end{aligned} \tag{5.1}$$

(5.2)

, die Summe aus den Overheads der einzelnen Protokollebenen und den eigentlichen Nutzdaten (n), erhöht um den Overhead für das eventuelle Escapen, zuzüglich der das Frame einschließenden Flags.

Aufgrund der Übertragung der Daten nach V.14 (Start-Stopbit), ergibt sich somit ein Bitstrom mit folgender Größe:

$$\begin{aligned} N_{\text{bit}}(n) &= N(n) * 10\text{bit/byte} \\ &= (2 + 1,007812 * (34 + n)) * 10\text{bit/byte} \\ &= 20 + 10,07812 * (34 + n) \end{aligned}$$

Die effektive Nutzdatenbandbreite $B_{\text{eff}}(n)$ für UDP-Pakete mit n Bytes Nutzdaten, ergibt sich aus der Differenz der realen Bandbreite B und der durch den Overhead verschwendeten Bandbreite $B_{\text{O}}(n)$:

$$B_{\text{eff}}(n) = B - B_{\text{O}}(n)$$

Die Bandbreite $B_{\text{O}}(n)$, die durch den Overhead belegt wird ergibt sich aus dem Produkt der realen Bandbreite B und dem relativen Overhead $O_{\text{rel}}(n)$:

$$B_{\text{O}}(n) = B * O_{\text{rel}}(n)$$

Der relative Overhead $O_{\text{rel}}(n)$ ergibt sich aus dem Overhead $O_{\text{bit}}(n)$ in Bits und den tatsächlich übertragenen Bits $N_{\text{bit}}(n)$:

$$O_{\text{rel}}(n) = O_{\text{bit}}(n)/N_{\text{bit}}(n)$$

Der Overhead in Bits $O_{\text{bit}}(n)$ ergibt sich aus den tatsächlich übertragenen Bits $N_{\text{bit}}(n)$ und der Anzahl der Bits der Nutzdaten ($8 * n$):

$$O_{\text{bit}}(n) = N_{\text{bit}}(n) - 8 * n$$

Durch Einsetzen der Teilformeln ergibt sich folgende Gleichung für $B_{\text{eff}}(n)$:

$$\begin{aligned} B_{\text{eff}}(n) &= B - (B * ((N_{\text{bit}}(n) - 8 * n)/N_{\text{bit}}(n))) \\ &= B - (B * (((20 + 10,07812 * (34 + n)) - 8 * n)/(20 + 10,07812 * (34 + n)))) \end{aligned} \tag{5.3}$$

Mit Hilfe dieser Gleichung können wir nun die zu erwartenden Bandbreite in Abhängigkeit von der Nutzdatengröße errechnen. Die sich ergebenen Werte sind in der Abbildung 5.2 graphisch dargestellt.

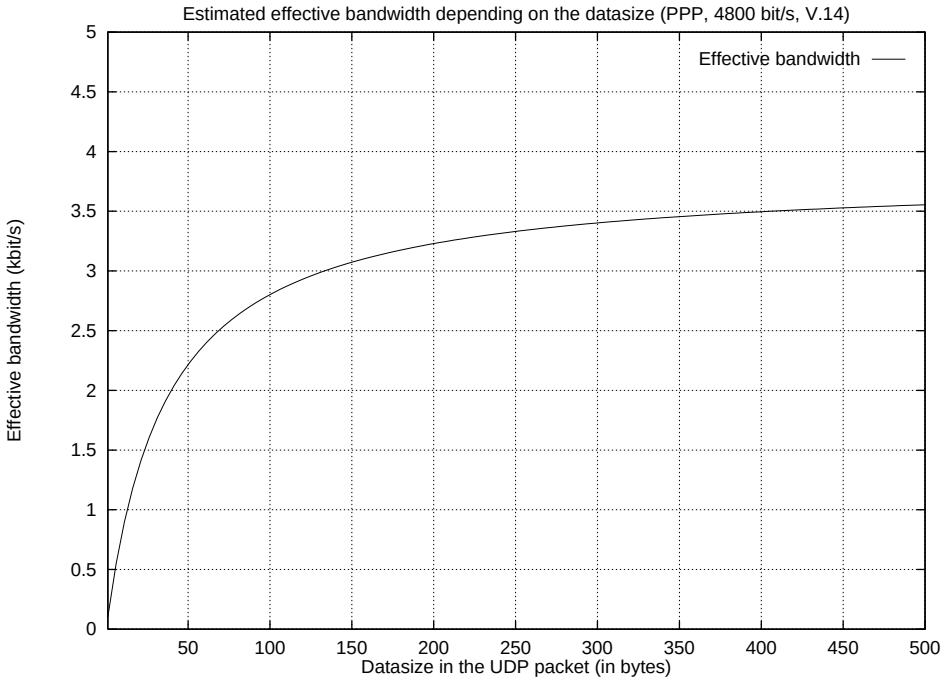


Abbildung 5.2: Theoretisch meßbare Bandbreite bei verschieden großen Datenpaketen

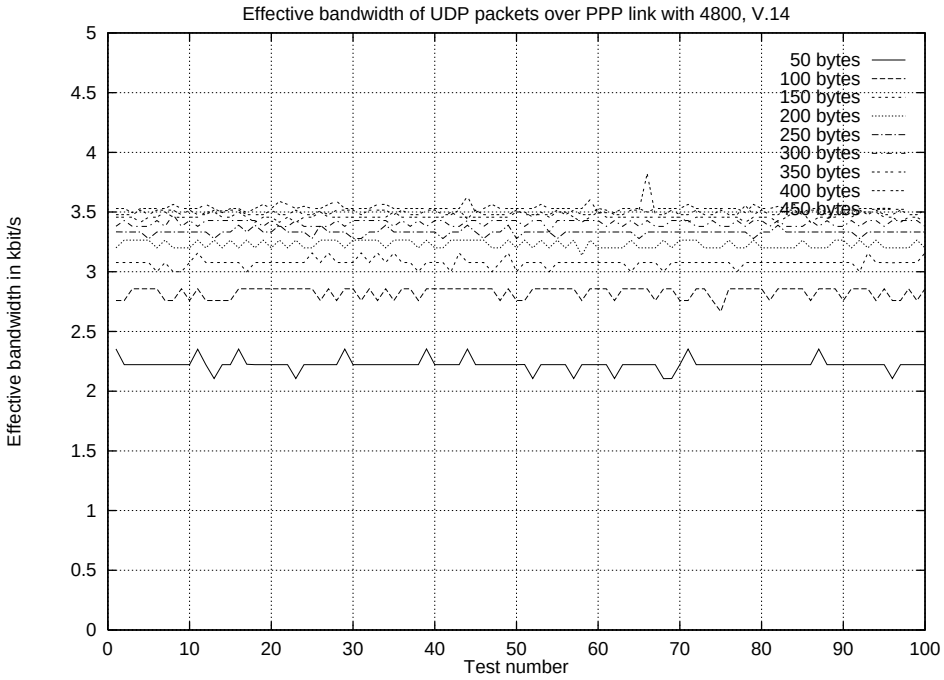


Abbildung 5.3: Gemessene effektive Bandbreite mit verschieden großen Datenpaketen

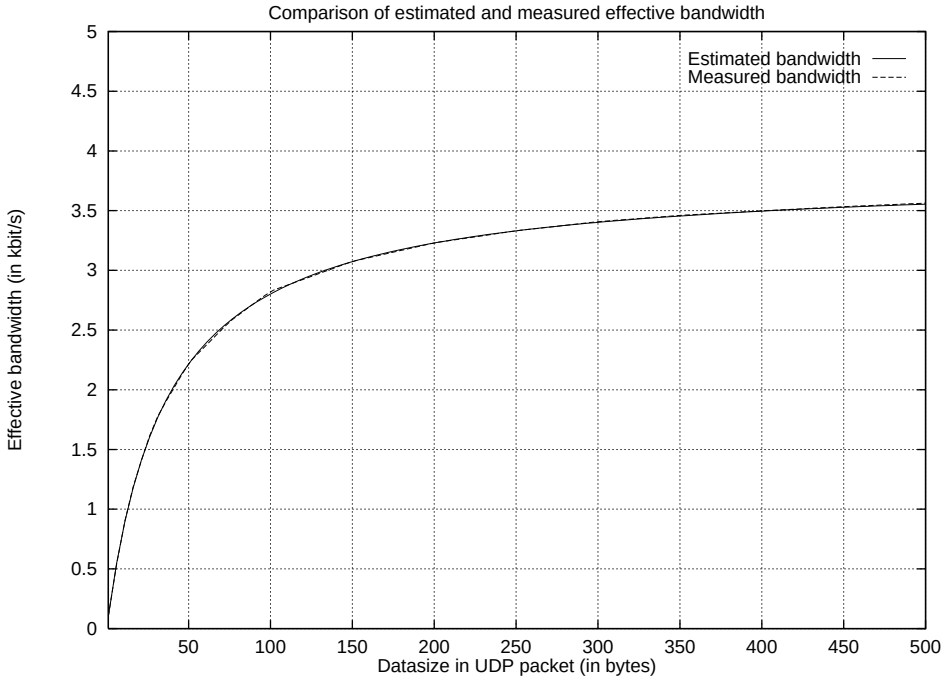


Abbildung 5.4: Vergleich der gemessenen mit der berechneten effektiven Bandbreite

5.4 Messungen mit verschiedenen Paketgrößen

Nachdem wir auf theoretischer Basis die effektive Bandbreite hergeleitet haben, gilt es diese durch reale Messungen zu überprüfen. Hierzu führten wir eine Reihe von Testmessungen über eine PPP-Verbindung durch, und variierten die Größe der Nutzdaten in den zu übertragenden Paketen. Die Ergebnisse sind in der Abbildung 5.3 graphisch dargestellt. Es ist deutlich zu erkennen, daß die effektive Bandbreite mit zunehmender Paketgröße steigt, und daß ihr Maximum bei etwas über 3,5 kbit/s liegt. Die ebenfalls deutlich sichtbaren diskreten Stufen sind auf den Escape-Mechanismus zurückzuführen.

Um eine genauere Aussage über die Qualität unserer theoretischen effektiven Bandbreite machen zu können, wurden die Messungen jeweils arithmetisch gemittelt, und der sich ergebende Graph mit dem aus der theoretischen Betrachtung zusammen dargestellt (siehe Abbildung 5.4).

Es ist eine deutliche Übereinstimmung der beiden Kurven zu erkennen, so daß davon auszugehen ist, daß unsere Überlegungen richtig waren. Um eine noch genauere Aussage treffen zu können, stellten wir die prozentualen Abweichung zwischen den gemessenen und berechneten Werten in der Abbildung 5.5 dar. Es zeigt sich, daß die Abweichung unterhalb von 1% liegt, wenn die Pakete mehr als ca. 10 Byte Nutzdaten fassen. Aber auch unterhalb dieser Größe liegt sie bei weniger als 1,5%. Das die Schwankungen mit zunehmender Datengröße in den Paketen abnimmt, läßt sich durch den Escape-Mechanismus erklären. Bei kleinen Paketen macht sich ein mehr zu escapendes Byte stärker bemerkbar, als bei größeren.

Es kann somit davon ausgegangen werden, daß die angegebene Formel eine sehr gute Annäherung an die tatsächliche Bandbreite darstellt.

5.5 Messungen bei verschiedenen ACCM-Werten

Bei der Übertragung von HDLC Frames über Modems besteht die Möglichkeit, daß einige Bytewerte nicht transparent transportiert werden können, wie z.B. `XON` (0x11), `XOFF` (0x13) oder das HDLC `FLAG` (0xFE). Um derartige Werte dennoch übertragen zu können, wird ein Escapemechanismus verwendet.

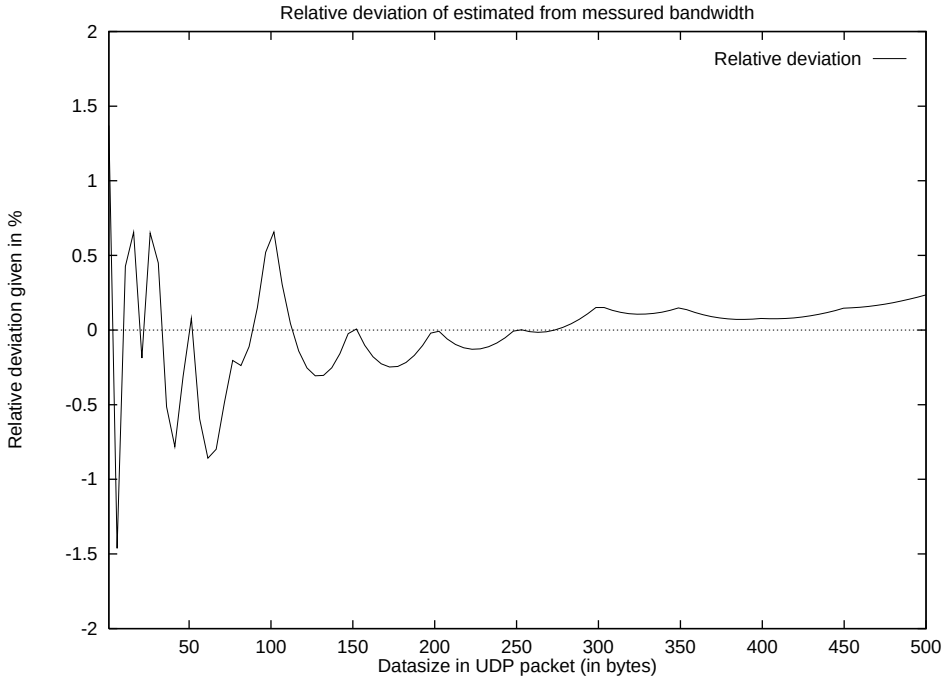


Abbildung 5.5: Vergleich der gemessenen mit der berechneten effektiven Bandbreite

Hierbei wird das zu „escapende“ Byte von einem ESC (0x7E) eingeleitet, dann wird das eigentliche Byte XOR 0x20 übertragen.

Jede der beiden Stationen hat eine sogenannte „Asynchron Control Character Map“ (ACCM [Sim94b]), die angibt, welche der unteren 32 Control-Character über diesen Mechanismus escaped werden müssen. Beim Aufbau einer PPP-Verbindung ist zum Anfang diese Map auf den Wert 0xFFFFFFFF eingestellt, jeder der 32 Control-Character von 0x00 bis 0x1F muß escaped werden. Diese ACCM kann jedoch mittels des „Link Control Protocol“ (LCP) [Sim94a] geändert werden und wird in den meisten Fällen auf 0x000A0000 (nur XON, XOFF, ESC und FLAG escapen) oder sogar auf 0x00000000 (nur ESC und FLAG escapen) eingestellt.

Diese Einstellung hat einen Einfluß auf die Messung der Bandbreite, da somit eine variable Anzahl von Bytes zum eigentlichen Frame hinzukommen (ein ESC für jedes zu escapende Zeichen). Um die Auswirkung der ACCM für die Bandbreitenmessung bewerten zu können, leiten wir uns wiederum die zu erwartende effektive Bandbreite mathematisch her, und werden diese anhand realer Messungen verifizieren.

In 5.3.2 haben wir bereits erwähnt, daß zwei Bytewerte in jedem Fall escaped werden müssen, unabhängig von der jeweiligen ACCM. Wir definierten die insgesamt durch das escapen zu übertragenden Bytes N_{esc} wie folgt:

$$N_{\text{esc}}(n) \quad = \quad n * (1 + 2/256)$$

$N_{\text{esc}}(n)$ gab die zu übertragenen Bytes für eine ACCM von 0x00000000 an, so daß nur die beiden Werte ESC und FLAG escaped werden müssen. Im allgemeinen läßt sich $N_{\text{esc}}(n)$ bei einer gegebenen ACCM wie folgt definieren:

$$N_{\text{esc}}(n, e) \quad = \quad n * (1 + e/256) \tag{5.4}$$

, wobei e die Anzahl der zu escapenden Bytewerte angibt. Der Wert ergibt sich aus der Anzahl der gesetzten Bits in der ACCM plus 2:

$$e \quad = \quad \text{BITS\_ON}(ACCM) + 2$$

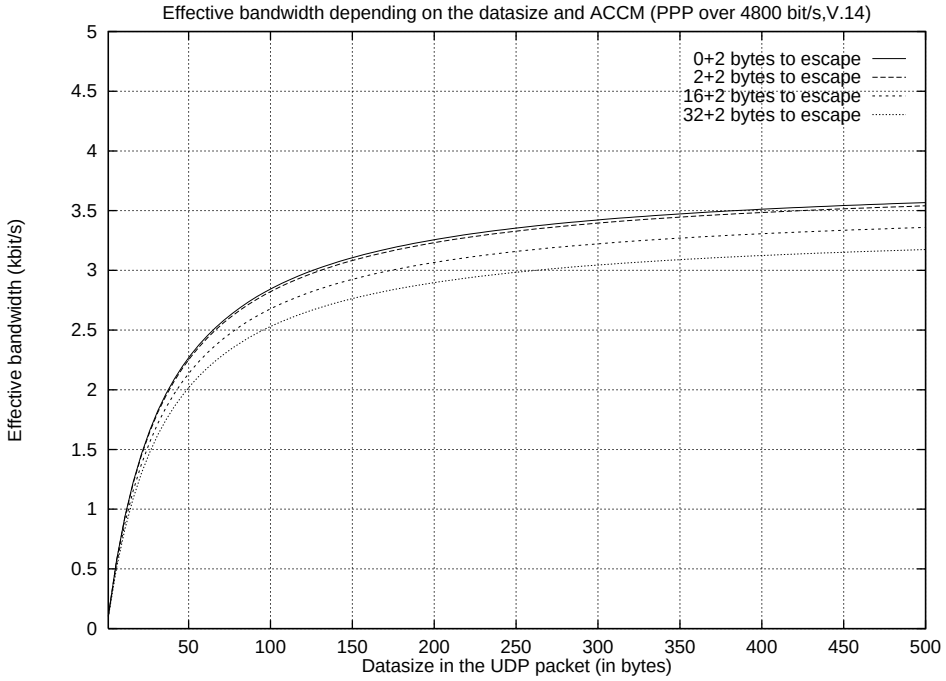


Abbildung 5.6: Theoretisch meßbare Bandbreite bei verschieden großen Datenpaketen und ACCM-Einstellungen

Zuvor haben wir bereits die Anzahl der zu übertragenden Bytes durch die Gleichung 5.1 beschrieben. Setzen wir dort unsere neue Funktion für N_{esc} ein, so ergibt sich folgendes:

$$\begin{aligned} N(n,e) &= O_{\text{HDLC-FLAGS}} + N_{\text{esc}}(O_{\text{UDP8}} + O_{\text{IP8}} + O_{\text{PPP8}} + O_{\text{HDLC8}} + n, e) \\ &= 2 + (1 + e/256) * (8 + 20 + 2 + 4 + n) \\ &= 2 + (1 + e/256) * (34 + n) \end{aligned}$$

Setzen wir diese in die Gleichung zur Berechnung der effektiven Bandbreite 5.3 ein, so erhalten wir folgende Gleichung für die effektive Bandbreite in Abhängigkeit von der Anzahl der gesetzten Bits e in der ACCM:

$$\begin{aligned} B_{\text{eff}}(n,e) &= B - (B * ((N_{\text{bit}}(n) - 8 * n)/N_{\text{bit}}(n))) \\ &= B - (B * ((10 * (2 + (1 + e/256) * (34 + n)) - 8 * n)/(10 * (2 + (1 + e/256) * (34 + n))))) \\ &= B - (B * ((20 + (10 + e/25,6) * (34 + n) - 8 * n)/(20 + (10 + e/25,6) * (34 + n)))) \end{aligned}$$

Die graphische Darstellung ist in Abbildung 5.6 zu sehen

Es ist deutlich zu erkennen, daß die ACCM einen großen Einfluß auf die gemessene Bandbreite hat. Sie schwankt um ca. 0.5 kbit/s je nach Einstellung der ACCM.

5.6 Messungen bei Benutzung von V.42

Heutzutage arbeiten nahezu alle Modems mit Fehlerkorrektur, wobei meistens V.42 [Int93g] verwendet wird. Dies hat nun wiederum einen erheblichen Einfluß auf die zur Verfügung stehenden Bandbreite. Die auf dem ersten Blick widersprüchlich erscheinende Aussage ist, daß sich der Durchsatz erhöht, trotz zusätzlichem Overhead durch V.42.

Der Grund für die Zunahme der Bandbreite ist darin begründet, daß bei V.42 keine Bytes, sondern HDLC Frames (es sind hierbei **nicht** die PPP-HDLC-Frames gemeint!) übertragen werden (genauer LAP-M). Somit entfallen die sonst nötigen Start- und Stopbits pro Byte, die immerhin einen Overhead von 25% ausmachen (10 Bits im Gegensatz zu 8). Anstelle eines Bytestroms, der übertragen wird, haben wir es nun mit einem bit-seriellen Datenstrom zu tun. Um Frame-Grenzen etc. erkennen zu können, müssen besondere Kodierungsverfahren verwendet werden.

Um die reinen Daten von Framing-Informationen unterscheiden zu können, wird bit-stuffing benutzt. Hierbei wird nach einer Folge von fünf aufeinanderfolgender 1-Bits im Datenstrom ein 0-Bit eingestreut. Der Empfänger entfernt auf seiner Seite dies Bit wieder, und erhält somit den reinen Datenstrom. Tritt dabei eine Verletzung des Verfahrens auf (es wurden mehr als fünf hintereinanderfolgende 1-Bits empfangen), so handelt es sich hierbei um Framing-Informationen.

Wie bei HDLC üblich kommen nun wiederum Octets zum Datenstrom dazu, wie Flag, Address, Control etc. Dennoch erhöht sich der Datendurchsatz durch den Wegfall von Start- und Stopbit und dem Einstreuen von Bits durch bit-stuffing erheblich.

Betrachten wir diesen Fall zunächst aus theoretischer Sicht. Der Nutzdatenstrom besteht aus einer Folge von Bytes, zu je acht Bits. Um unsere effektive Bandbreite zu bestimmen, müssen wir den Datenstrom (inklusive Paket-Overhead) als einen reinen Bitstrom betrachten. Die Wahrscheinlichkeit, daß in einem Datenstrom Folgen von fünf 1-Bits austreten ist gegeben durch die folgende arithmetische Reihe:

$$
\begin{aligned}
p &= 1 * .5^7(p('0-11111-0')) \\
&= +1 * .5^8(p('0-11111-10')) \\
&= +1 * .5^9(p('0-11111-110')) \\
&= +1 * .5^10(p('0-11111-1110')) \\
&= +1 * .5^11(p('0-11111-11110')) \\
&= +2 * .5^12(p('0-11111-11111-0')) \\
&= +2 * .5^13 \\
&\ldots \\
&= +3 * .5^17(p('0-11111-11111-11111-0')) \\
&\ldots
\end{aligned}
$$

Mit Hilfe von *Rainer Baier* (er antwortete auf diese Frage in der Fokus Mailing List) konnte diese Reihe in die Summenformel

$$p = 31/64 * \sum_{i=1}^{n}(i/(32^i))$$

transformiert werden. Mit Hilfe des folgenden kleinen `bc` Programms

```
scale=99
x=0
for (i=1; i<100; i++) {
        x += i / (32 ^ i)
}
31 / 64 * x
```

wurde der angenährte Wert ermittelt:

$$p = .016129\ldots$$

Der Overhead beträgt somit nur 1,6129%, deutlich weniger als bei Verwendung von Start- und Stopbits (mit 25%). $n_1 = (1 + p) = 1,016129$ stellt zugleich den Faktor dar, um den sich der Datenstrom (mit Ausnahme der FLAGS) vergrößert.

Zu den zu übertragenden Nutzdaten kommt nun noch der relativ statische Overhead durch das HDLC ähnliche Framing beim LAP-M (vgl. auch 5.1). Dies sind

- 1 Byte für starting-flag
- 1 Byte für address
- 2 Bytes für control (I-frame)
- 2 Bytes FCS
- 1 Byte für closing-flag

Es kommen also noch 7 Bytes Overhead dazu, wobei in den Flags kein bit-stuffing ausgeführt wird. Die statistisch durchschnittliche Größe der zu übertragenden Daten $N(n)$ bei Paketen mit n Bytes PPP-Daten ergibt sich zu

$$N(n) \quad = \quad (n + 5) * 1,016129 + 2$$

Setzen wir dies in unsere Gleichung 5.1 ein, so ergibt sich folgende Gleichung:

$$\begin{aligned} N(n,e) &= 2 + n_1 * (5 + O_{\text{HDLC-FLAGS}} + N_{\text{esc}}(O_{\text{UDP8}} + O_{\text{IP8}} + O_{\text{PPP8}} + O_{\text{HDLC8}} + n, e)) \\ &= 2 + n_1 * (5 + 2 + (1 + e/256) * (8 + 20 + 2 + 4 + n)) \\ &= 2 + n_1 * (7 + (1 + e/256) * (34 + n)) \end{aligned}$$

Diese wiederum verwendet in 5.5, unter der Berücksichtigung, daß nun kein Start- Stopbit-Verfahren mehr verwendet wird, ergibt folgende Gleichung für die Berechnung der zu erwartenden effektiven Bandbreite:

$$\begin{aligned} B_{\text{eff}}(n,e) &= B - (B * ((N_{\text{bit}}(n) - 8 * n)/N_{\text{bit}}(n))) \\ &= B - (B * ((8 * (2 + n_1 * (7 + (1 + e/256) * (34 + n))) - 8 * n)/(8 * (2 + n_1 * (7 + (1 + e/256) * (\end{aligned}$$

Wiederum führten wir einige Messungen mit verscheiden großen UDP Paketen durch. Die Ergebnisse sind in der Abbildung 5.7 graphisch dargestellt. Ebenfalls wurde die zu erwartende effektive Bandbreite mit dargestellt. Es ist sehr deutlich zu erkennen, daß die theoretisch zu erwartende effektive Bandbreite von der gemessenen mit zunehmender Paketgröße, stärker abweicht, sie ermittelt einen Wert, der stets größer ist. Der vermutliche Grund hierfür könnte die Beschränkung der maximalen verwendeten Frame-Größe des LAB-M sein (der Parameter `N401` nach V.42). Aus diesem Grunde haben wir noch zwei weitere Funktionen für die zu erwartenden effektiven Bandbreite dargestellt, die eine berücksichtigt eine Wert von 128, die andere einen von 256 Octets für den Parameter `N401`.

Anhand der Graphiken könnte man darauf schließen, daß die verwendete Frame-Größe zwischen den beiden Modem ca. 128 Octets betragen dürfte. Leider gibt es für uns als Benutzer eines Modem keine Möglichkeit, die zwischen den beiden Modems ausgehandelten Werte für diesen Parameter zu erfragen. Auch die `ITU` [Int93g] Norm gibt uns hierfür keinerlei Hinweise, daß einzige was erwähnt wird ist, daß jede Implementierung eine Frame-Größe von 64 Bytes unterstützen sollte. Die oben gegebene Erklärung für die Abweichungen bleiben also nur eine reine, wenn auch unserer Meinung nach sinnvolle Vermutung unsererseits, die wir jedoch nicht genau belegen können!

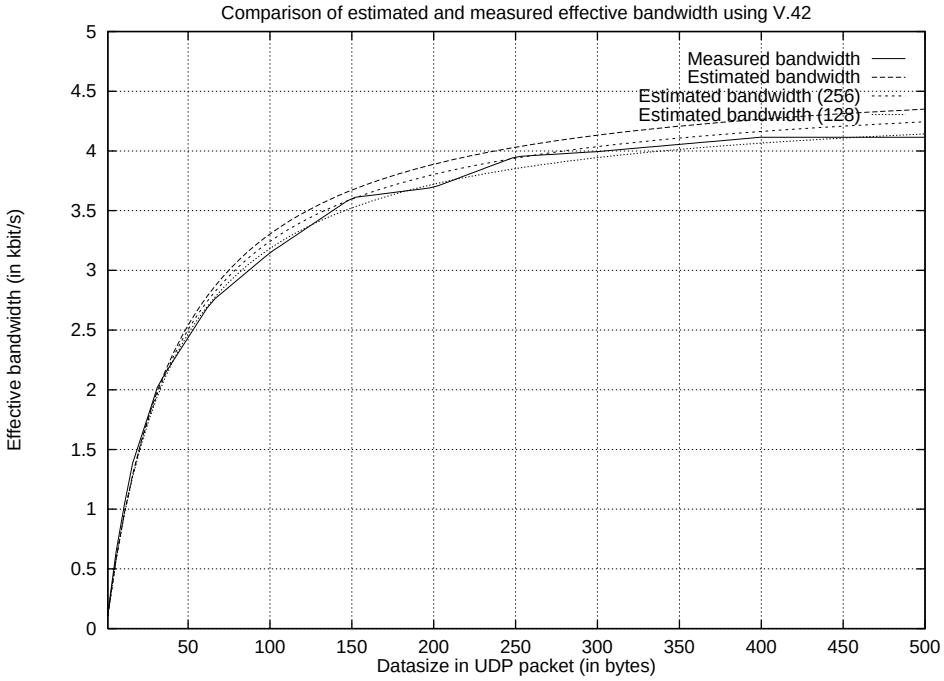


Abbildung 5.7: Messungen mit verschiedenen Paketgrößen bei V.42

5.7 Kompression von PPP-Headern

PPP setzt die Möglichkeit zur Verfügung, einige Protokoll-Felder zu komprimieren. So sind die Werte für das PPP Protokoll-Feld so gewählt, daß auch eine 1-Byte Übertragung zur Erkennung ausreicht. Dies ist beispielsweise bei der Übertragung von `IP` über PPP-Verbindungen der Fall. Wie in RFC 1332 [McG92] beschrieben, beträgt der Wert für das Protokoll-Feld `0x0021`. Entsprechend den Definitionen in [Sim94a] können Werte, bei denen der erste Bytewert ein `00` ist, durch weglassen dieses Bytes komprimiert werden. Aufgrund der Verwendung eines bestimmten Verfahrens für die Bestimmung der gültigen Werte für das erste und zweite Byte, kann bei jedem gültigen Protokoll-Wert festgestellt werden, ob es sich um ein ein-Byte oder ein zwei-Byte Wert handelt (das genaue Verfahren ist für uns an dieser Stelle nicht weiter von Bedeutung).

PPP-Verbindungen werden, wie der Name bereits andeutet, als Protokoll zwischen zwischen exakt zwei Endpunkten benutzt. Aus diesem Grunde ist es eigentlich auch unnötig, das HDLC-Address-Feld (vgl. 5.1), daß immer eine Broadcast Adresse (`0xFF`) enthält, zu übertragen. Aus diesem Grunde kann mittels LCP Nachrichten vereinbart werden, daß dieses Feld entfallen kann.

Wie dem auch sei, je nach gewählten Optionen können somit maximal zwei Bytes im Datenstrom eingespart werden, so daß die zu erwartende effektive Bandbreite nur um zwei Bytes nach links verschoben sein dürfte (sie müßte nun bei einer Nutzdatengröße von $n + 2$ so groß sein, wie zuvor bei einer Größe von n. Aufgrund der Einfachheit der Kompression verzichten wir an dieser Stelle aus entsprechende Messungen und Berechnungen der zu erwartenden effektiven Bandbreite.

5.8 Messungen mit V.42bis

Viel interessanter ist es, die Auswirkungen der von den Modems auszuführenden Kompressionen des Bitdatenstromes zu betrachten. Alle gängigen Modems arbeiten heutzutage mit der Datenkompression

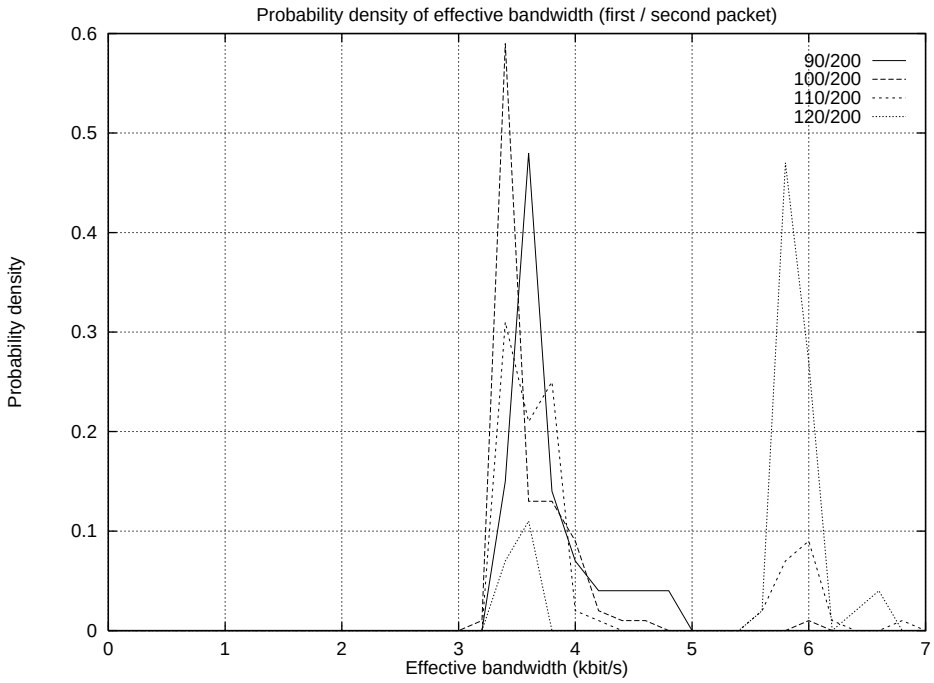


Abbildung 5.8: Häufigkeitsverteilung bei verschieden großen Start Paketen (200 Byte Meßpaket)

nach V.42bis [Int90]. Das früher häufig verwendete Verfahren nach MNP5[2] betrachten wir hier nicht, da hiervon kein großer Zuwachs an Bandbreite bei der Übertragen von Multimediadaten zu erwarten ist, im Gegenteil, sie verschlechtert sich angeblich sogar[3].

Die große Frage ist, in wie weit sich diese Datenkompression auf die Bandbreite auswirkt. Unser Ziel war es, anhand von Messungen während einer bestehenden Modemverbindung die Bandbreite bestimmen zu können, und mit Hilfe von abgeleiteten Formeln die für uns nutzbare effektive Bandbreite bestimmen zu können.

Wiederum führten wir einige Messungen über eine aufgebaute PPP-Verbindung durch. Bei der Betrachtung der Ergebnisse viel auf, daß sehr starke Schwankungen bei Messungen mit gleichen Paketgrößen auftraten. Bei einer Nutzdatengröße von 100 Bytes geb es sogar regelmäßige Peaks von über 25 kbit/s, obwohl der zu übertragene Datenstrom zufällig war!

Wir führten daraufhin einige weitere Versuche durch und stellten fest, daß bei der Verwendung von `V.42bis` sogar die Größe des ersten Paketes (für das Packet-Pair-Verfahren) einen sehr starken Einfluß auf die gemessene effektive Bandbreite hat. Aus diesem Grunde untersuchten wir das Verhalten bei Datenpaketen mit 200 Bytes und verschieden große Werten für das erste Paket.

Da es aufgrund der großen Schwankungen keinen Sinn hat, einen arithmetischen Mittelwert für jede Meßreihe zu verwenden, verglichen wir die Häufigkeitsdichte der gemessenen Bandbreite. Eine graphische Gegenüberstellung für verschiedenen Messungen ist in der Abbildung 5.8 zu sehen. Es fällt sofort auf, daß zwei extreme Werte für die gemessene Bandbreite auftreten. Erstaunlich ist hierbei, daß die Bandbreite sogar die der physischen Modemverbindung um fast das Doppelte übersteigt. Eventuell könnte das auf das Kompressionsverfahren hindeuten, jedoch scheint dies recht unwahrscheinlich zu sein, da die versendeten Daten bereits komprimiert waren (wir verwendeten Daten aus einer gziped Daten, genauer die gziped'te PostScript Daten der V.42bis Recommendation). Aus diesem Grunde betrachteten wir das Verhalten bei einer anderen Paketgröße, wie in Abbildung 5.9 gezeigt. Bei diesen Werten scheint das Verhalten noch weit aus extremer auszufallen. Es zeigen sich deutlich mehrere Maxima, bis zu über

[2] MNP5 ist eine Kompressionsverfahren der Firma Microlink

[3] Zumindest ist dieser Hinweis in vielen Handbüchern zu Modem zu finden.

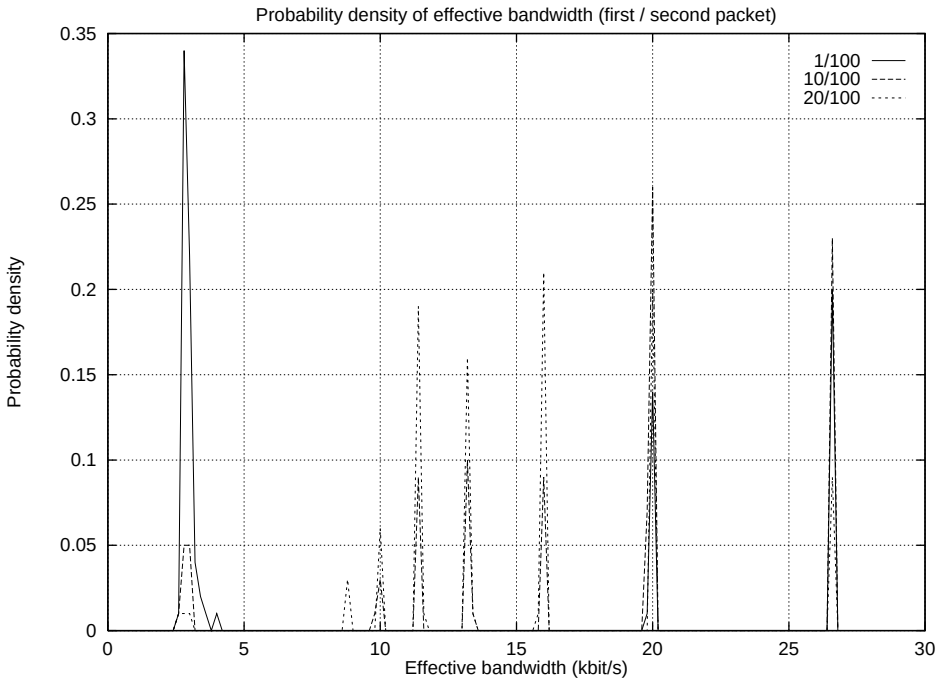


Abbildung 5.9: Häufigkeitsverteilung bei verschieden großen Start-Paketen (100 Byte Meßpaket)

25 kbit/s. Der Grund scheint wiederum die maximale Größe der LAP-M HDLC Frames zu sein, die versendet werden können.

5.9 Entwicklung eines Verfahrens zur Bestimmung der Bandbreite

In den letzten Abschnitten haben wir diverse Messungen der effektiven Bandbreite durchgeführt und versuchten, die gemessenen Werte auch belegen zu können. Solange wir mit keinerlei Form von Kompression oder wie im letzten Fall, mit Framing-Eigenschaften zu tun hatten, stellten die unsere angegebenen Gleichungen eine sehr gute Annäherung an die tatsächlich zur Verfügung stehende Bandbreite dar.

Speziell bei der Verwendung von V.42bis haben wir jedoch gesehen, daß die Messungen direkt keine Hinweise auf die Bandbreite zulassen (vgl. 5.9). Das Bilden eine Mittelwertes führte hierbei zu keinem brauchbaren Ergebnis. Unser Ziel war es jedoch, ein Verfahren zu finden womit wir anhand von Messungen auf die tatsächliche effektive Bandbreite schließen können.

Im folgenden beschreiben wir eine Methode, die dennoch eine sehr gute Schätzung der effektiven Bandbreite zuläßt.

5.9.1 Normalisieren der gemessenen Werte

Unser Hauptproblem ist es, die vielen Peaks, die bei der Verwendung von V.42bis auftreten, einschätzen zu können. Das Bilden von Mittelwerten führte hierbei nicht sehr viel weiter und ist sehr stark Abhängig vom Datenstrom.

Aus diesem Grunde beschreiten wir nun einen etwas anderen Weg. Zunächst müssen wir die Meßergebnisse bei verschieden großen Datenpaketen normieren. Hierzu müssen wir nur den Overhead, der durch die Protokoll-Header hinzukommt bei den Messungen mit berücksichtigen.

Der Grund, weshalb das sinnvoll ist, ist folgender: Unser Ziel ist es, die Bandbreite einer Verbindung bestimmen zu können, und diese Information wiederum für unsere Format-Wandlung zu benutzen. Die

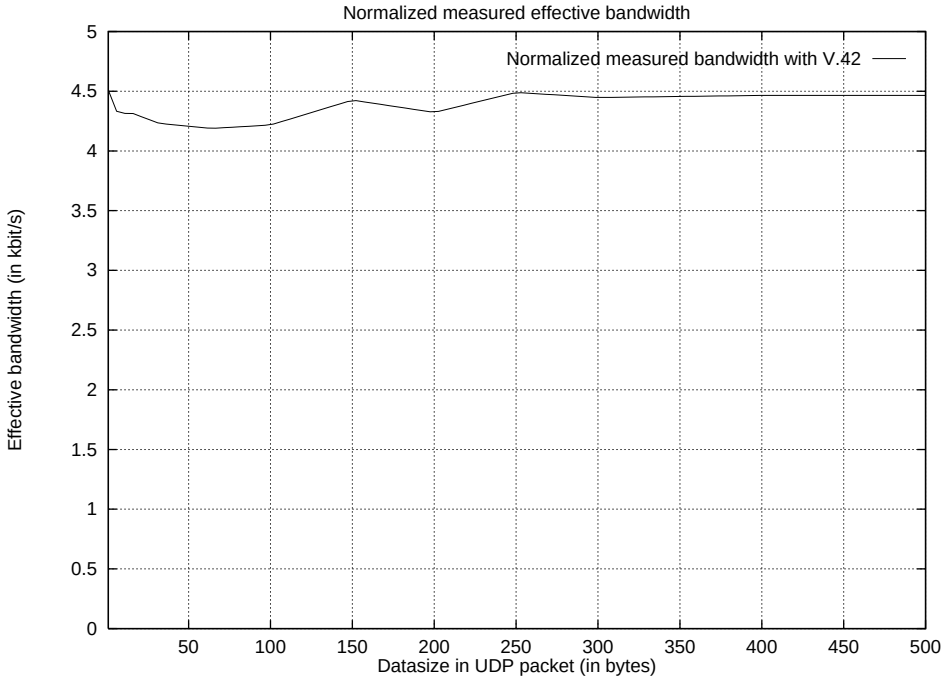


Abbildung 5.10: Normierte gemessene effektive Bandbreite mit V.42

Messungen sollten auf jedem Fall in der PPP-Implementierung realisiert werden. Das heißt auch, daß wenn wir ein Paket empfangen wir alle Header und Trailer mit empfangen. Wir messen also wirklich das, was „durch die Leitung paßt“.

Wir definieren also die angenommene effektive Bandbreite E_{eff} unter Berücksichtigung des Protokoll-Overheads O, bei einer Nutzdatengröße von n wie folgt:

$$E_{\text{eff}}(n) = B_{\text{eff}}(n) * (O - n)/n$$

Angewandt auf unsere Messungen (mit Ausnahme der mit V.42bis) müßte diese Gleichung eine Gerade ergeben, die in etwas waagerecht verläuft. Die Abbildung 5.10 zeigt das Ergebnis für die Messungen mit V.42 aus Abschnitt 5.6. Es ist deutlich die geradenähnliche Form zu erkennen. Die Schwankungen im unteren Bereich sind auf statistische Abweichungen zurückzuführen.

Wir können somit diese Gleichung nehmen, um unsere Messungen zu Normieren, und auf die tatsächliche effektive Bandbreite schließen.

5.9.2 Bestimmung der effektiven Bandbreite

Nachdem wir die tatsächliche effektive Bandbreite bestimmen können, betrachten wir die kumulierten Häufigkeiten (auch empirische Verteilungskurve genannt) der gemessenen Bandbreite. Der Theorie nach zu folge müßte die Bandbreite, bei der das 0,5-Quartil liegt, der real zur Verfügung stehenden Bandbreite entsprechen.

Wir verwenden hierzu Quantisierungsstufen von $0,1$kbit/s und stellten die Kurven für die Messungen der letzten Abschnitte bei V.42 und V.42bis zusammen in der Abbildung 5.11 dar. Bei der Betrachtung der Kurven fällt folgendes auf: Bis zu dem Punkt, an dem die reale effektive Bandbreite liegt, laufen die beiden Kurven nahezu synchron, auch wenn sie ein wenig versetzt sind. Nach dem 0,5 Quartil weicht die Kurve für V.42bis extrem von der für V.42 ab.

Die Kurve für V.42 entspricht auch exakt den Erwartungen für eine empirische Verteilungsfunktion.

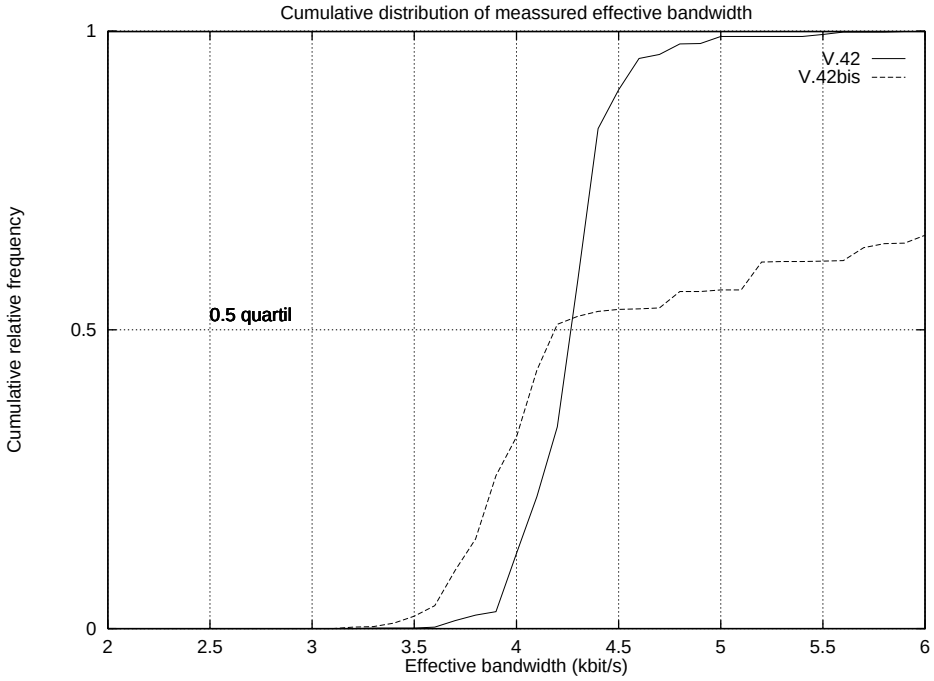


Abbildung 5.11: Kumulierte Häufigkeit der normierten gemessenen effektive Bandbreite

Daß die zweite Kurve im oberen Teil derart abfällt war ebenfalls zu erwarten, da sich hier die vielen verschiedenen Peaks in den oberen Bereichen bemerkbar machen.

Der wichtigste Punkt ist jedoch, daß das 0,5 Quartil bei beiden sehr eng beieinander liegt. Mit Hilfe dieser Erkenntnis haben wir nun ein Verfahren zur Verfügung, mit dem wir die reale effektive Bandbreite durch Messungen ziemlich gut approximieren zu können.

Wir können somit folgende Aussage treffen: Die zu reale effektive Bandbreite kann durch Messungen bestimmt werden, und ist der Wert am 0,5-Quartil der empirischen Verteilungsfunktion der Bandbreite.

5.10 Zusammenfassung der Ergebnisse

Wir haben in dieser Arbeit eine Verfahren entwickelt, mit die Bandbreite einer bestehenden Verbindung gemessen werden kann. Das Verfahren erlaubt auch die Messung, wenn Datenkompression und Fehlerkorrekturmechanismen verwendet werden.

Die Messungen wurden bisher jedoch nur in einer optimalen Umgebung durchgeführt. Die Frage, in wie weit dieses Verfahren auch in einer realen Umgebung zu verwenden ist, werden wir später in unserer Testimplementierung (Kapitel 6) noch klären.

Kapitel 6

Beschreibung unserer Testimplementierung

In den letzten Kapiteln beschäftigten wir uns mit der Frage, wie wir Multimediadaten, insbesondere RTP-Daten, über eine PPP-Verbindung übertragen können. Wir beschrieben verschiedene Probleme und zeigten zum Teil Lösungen auf.

Damit unsere Überlegungen nicht isoliert im Raum stehen, haben wir eine Testumgebung geschaffen, in der wir unsere Überlegungen probieren konnten.

6.1 Welche Testumgebung können wir verwenden?

Eigentlich war es unser Ziel, eine existierende PPP-Implementierung zu nehmen und zu erweitern, so daß wir mit dieser Multimediadaten übertragen können. Wir heben jedoch aus zwei Gründen davon Abstand genommen:

- Ein Teil Änderungen hatten in dem Kernel Streams Modul vorgenommen werden müssen. Es wäre in der ersten Testphase mit vielen Abstürzen zu rechnen, insbesondere da wir etwas derartiges bisher noch nicht gemacht heben.
- Das Testen wäre weitaus schwerer. Wir bräuchten dann root Zugriff auf einen Rechner. In der Umgebung, in der wir unsere Arbeit realisierten, war dies nicht ohne weiteres möglich.

Wir überlegten uns somit, was die günstigste Simulationsumgebung sein könnte. Es standen dabei mehrere Möglichkeiten zur Diskussion, von denen zwei in den nächsten Abschnitten genauer beschrieben sind.

6.1.1 Verwendung eines Tunnels

Als erstes kam uns der Gedanke eine Art Tunnel über eine PPP-Verbindung zu realisieren. Dieser sollte mit Hilfe zweier Prozesse, die miteinander über eine UDP-Verbindung verbunden sind, realisiert werden. Diese Variante hätte die folgenden Vorteile:

- Sie kann bei allen Implementierungen verwendet werden, sogar bei nicht PPP-Verbindungen.
- Es sind keine speziellen Privilegien erforderlich, jeder Benutzer kann diese Implementierung benutzen.

Es ergeben sich jedoch einige gravierende Nachteile:

- Wir haben keine Kontrolle über zur Verfügung stehende Bandbreite. Wir können sie dann nicht exakt messen. Der Grund hierfür liegt in der Eigenschaft der Netzwerkimplementierungen, anstehende Netzwerkpakete zu puffern. Erst wenn der Puffer voll wäre, bei welchem Wert das auch immer sein möge, würden wir merken daß die von uns verwendete Bandbreite viel zu hoch ist.
- Es bestünde keine vernünftige Möglichkeit, die bereits erwähnte IP/UDP/RTP-Header-Compression zu verwenden. Wir müßten immer mit vollen Headern arbeiten, womit Messung in der Realität nicht möglich wären. Man könnte diese natürlich dennoch machen und ganz normal als UDP-Paket verschicken. Jedoch ist der Overhead insgesamt dann so hoch, daß wir nur noch eine sehr starke Komprimierung benutzen könnten.

Alles im allen erschien es uns recht fraglich, ob man mit dieser Umgebung vernünftige Ergebnisse erzielen kann.

6.2 Eine eigene kleine PPP-Implementierung

Aus diesem Grunde entschieden wir uns, eine eigene Simulationsumgebung zu implementieren, bei der im Grunde alle Komponenten einer PPP-Implementierung mit enthalten sind. Dies erforderte zwar einen höheren Aufwand, jedoch konnten wir unsere Testumgebungen somit optimal gestalten. Es auch möglich, Messungen an allen Stellen der Implementierung vorzunehmen, was im Falle einer echten Implementierung im PPP-Kern nicht ohne weiteres möglich gewesen wäre.

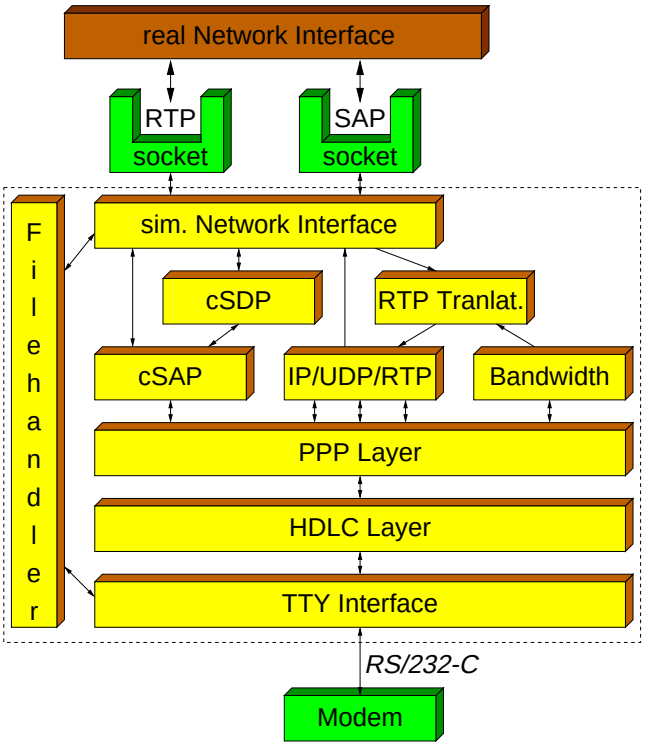


Abbildung 6.1: Überblick über unsere Testimplementierung

Die Abbildung 6.1 gibt einen Überblick über die von uns implementieren Komponenten und deren Interaktionen. In den folgenden Abschnitten ist jedes der implementieren Module separat beschrieben. Der Source-Code der Testimplementierung ist im Anhang C enthalten.

6.2.1 TTY Interface

Das unterste Modul realisiert den Zugriff auf das Modem, das wir für unsere physikalische Verbindung benutzten (C.1.1). In erster Linie initialisiert es die „TTY Line Characteristics“, so daß es möglich ist binäre Daten zu übertragen. Da wir in unserer Testumgebung immer eine Modemverbindung zur TU-Berlin herstellten, war auch eine „Not-Aus-Taste“ nötig, um im Fehlerfall die Implementierung auf der Seite der TU ohne Abbruch der Verbindung terminieren konnten. Wir wählen einfach die Tastenkombination `CTRL/B`. Sobald diese empfangen wurde, wurde das gesammte Programm beendet.

Neben dieser Rolle stellte es auch das eigentliche Hauptmodul auf der entfernten Seite dar. Für die Seite des „Anrufers“ war noch eine weitere Zusatzkomponente nötig, bei der es möglich ist, die Eingaben der Tastatur direkt zum Modem zu schicken. Nur so war es möglich, mit Hilfe des Modems zu wählen und das Programm auf der Seite in der TU zu starten. Auch hier galt, daß durch Eingabe von `CTRL/B` die Verbindung unterbrochen werden konnte. Es wurde jedoch nicht die physikalische Modemverbindung getrennt, sondern nur der in der Abbildung sichtbare Protokollstack sauber terminiert. Danach befand man sich wieder im „Terminal-Modus“.

6.2.2 HDLC-Layer

Um auch genausoviele Bytes für die Übertragung der Daten zu benötigen, wie bei einer echten PPP-Implementierung, haben wir auch das Versenden der Daten nach RFC 1662 [Sim94b] implementiert. Hierbei werden die PPP-Pakete in HDLC [Int96] Frames eingekapselt, und durch Flags getrennt verschickt. Wir implementierten auch die Option, das Adress- und Kontrollfeld zu komprimieren. In unserer Testumgebung wurde diese vom Hauptprogramm stets eingeschaltet, weil das auch der Realität entspricht[1]. Auch die Verwendung der `ACCM`, der „Asynch Control Character Map“ wurde berücksichtigt. Durch den Server werden bei uns

- `XON`
- `XOFF`
- `CTRL/B`

escaped. `XON` und `XOFF` wird bei Modemverbindungen oft eingeschaltet, da die unterliegenden Verbindungen diese oft von sich aus einstreuen. Durch das escapen dieser Werte ist man somit immer auf der sicheren Seite. Das `CTRL/B` wird nun bei uns escaped, da wir es (wie oben schon erwähnt) zum Abbruch der Verbindung benutzen[2].

In unserer Implementierung wird das FCS-Feld mit einen 16 Bit CRC berechnet und übertragen. Für unsere Testimplementierung sollte das ausreichen. Verwendet wurde hierzu die in RFC 1662 angegebene CRC Tabelle aus Appendix C.2, sowie das dort beschriebene Verfahren zur Berechnung.

6.2.3 PPP Layer

Aufsetzend auf dem HDLC-Layer ist der PPP-Layer angelegt. Dieser ist als Empfänger für Pakete beim HDLC-Layer registriert, so daß alle korrekt auf HDLC-Ebene empfangenen Pakete zum PPP weitergereicht werden.

[1] Bei einer realen Implementierung ist diese Komprimierung zu Beginn einer Verbindung immer ausgeschaltet. Mit Hilfe von LPC [Sim94a] Nachrichten kann jedoch ausgehandelt werden, diese Kompression zu verwenden. Aus Effizienzgründen wird diese Form der Kompression bei PPP-Verbindungen in der Regel immer eingeschaltet.

[2] Der Wert wurde nicht ganz zufällig gewählt. 0 wurde nicht verwendet, da dieser des öfteren in realen Daten vorkommt. Würde dieser escaped erden müssen, hätte das ein erhöhtes zu übertragendes Datenvolumen zur Folge. `CRTL/A` wurde von unserem Terminal Programm bereits benutzt, so daß sie die Verwendung von `CTRL/B` anbot.

Das PPP-Modul entspricht eigentlich einer ganz normalen PPP-Implementierung jedoch mit der Einschränkung, daß kein LCP (Link Control Protocol)[3] vorhanden ist. Da sich unser Interesse nicht auf die Realisierung einer weiteren PPP-Implementierung richtet, gingen wir von festen, bereits ausgehandelten Voraussetzungen aus:

- Benutzung der HDLC Address-Control-Compression
- Benutzung der PPP Protocol-Compression
- Die `ACCM` ist auf beiden Seiten auf einen festen Wert eingestellt.
- Keine Aushandlung bzgl. der zu verwendenden Network-Protocols.

Wie jedes PPP-Modul wird anhand des Protokoll-Feldes das verwendete Protokoll erkannt. Andere Module, die ein spezielles Protokoll behandeln können, müssen sich für dieses beim PPP-Modul mit einer Callback-Routine registrieren.

6.2.4 IP-UDP-RPT Header-Compression-Layer

Diese Modul realisiert die eigentliche IP/UDP/RTP Header-Compression, wie im Internet-Draft [CJ96] von Stephen Casner und Van Jacobson beschrieben[4]. Der Draft ist derzeit jedoch noch nicht in allen Teilen vollständig implementiert. Fertig sind derzeit:

- Die Behandlung der Kontexte.
- Das Senden und Empfangen von `FULL_HEADER`, `COMPRESSED_UDP` COMPRESSED_RTP, und `CONTEXT_STATE` Paketen.

Es fehlen:

- Behandlung von RTCP Paketen.
- Senden und Empfangen von `COMPRESSED_NON_UDP` Paketen.

Für die drei verschiedenen Pakettypen wurde auch drei verschiedene Werte für das PPP Protokoll-Feld genommen. Diese Werte sind von uns frei erfunden und wurden NICHT beim IANA registriert. Sie dienten nur für unsere Testzwecke.

6.2.5 RTP Mixer / Translator

Das eigentliche Ziel war es, einen RTP-Mixer zu benutzen. Falls mehrere Sender (für eine Multicast-Adresse) zur gleichen Zeit aktiv sind, sollte dieser die eingehenden RTP-Ströme mixen und dann übertragen. Da es jedoch ein wenig aufwendig ist, einen RTP-Mixer zu implementieren und der Kern dieser Arbeit nicht die Implementierung eines solchen ist, entschlossen wir und für den einfacheren Weg. Wir implementierten nur einen RTP-Translator. Dieser ist bei weitem einfacher zu realisieren. Da wir bei unserer Testumgebung auch nicht mit mehreren Quellen arbeiteten (das war für unsere Tests auch nicht nötig), war dieser durchaus ausreichend.

[3] Dieses wird benutzt, um verschieden Optionen, die die PPP-Verbindung betreffen, ausgehandelt werden.

[4] Während des Implementieren gab es einige Fragen an Stephen, die er mir alle sehr bereitwillig beantwortete. Durch dieses Dialoges klärten sich viele Fragen meinerseits. Wir stießen auch auf einige unsauber formulierte Passagen in dem Draft, die in der nächsten Version klarer formuliert werden.

Ziel- / Quellformat	PCM μ-law	PCM A-law	GSM	LPC
GSM	GSM	GSM	GSM	LPC
LPC	LPC	LPC	LPC	LPC

Tabelle 6.1: Ausgangsformat in Abhängigkeit von Quell- und Zielformat

Es existiert ein entscheidender Unterschied zwischen einem RTP-Mixer und einem Translator. Der Translator empfängt einen RTP-Strom und sendet ihn wieder neu aus. Jede Daten-Quelle bleibt für den Empfänger weiterhin erkennbar. Der Translator hat auch die Option, eine Formatwandlung durchzuführen. Hauptsächliches Einsatzgebiet sind Proxies für Firewalls, Multicast zu Unicast Konverter und Wandler für exotische oder nicht überall unterstützte Datenformate.

Der RTP-Mixer hingeben mischt mehre eingehende Datenquellen, und erscheint als eigener Sender. Er hat also den ganzen Aufwand, die Daten zu ordnen, Lücken zu erkennen, die Daten zu mischen, dabei eventuell das Format zu ändern etc. Er wird hauptsächlich dort eingesetzt, wo es um die Verringerung der Bandbreite geht, wenn es oft vorkommt, daß viele Quellen gleichzeitig aktiv sind.

Neben der Funktion als Proxy für die Multicast Daten zu fungieren, führt unser Translator die notwendige Formatwandlung der RTP Daten durch. In der Tabelle 6.2.5 ist abgegeben, welche Ausgangsformate bei welchem Ziel- und Quellformat verwendet wird. Andere Formate als in der Tabelle aufgeführt werden derzeit nicht unterstützt.

6.2.6 cSAP

Neben dem IP/UDP/RTP Header Compression Modul, ist auch das cSAP (Compressed Session Announcement Protocol) Module beim PPP mit einem Transportprotokoll registriert. Es realisiert die komprimierte Übertragung von Session Announcements wie in Abschnitt 4.4 und A beschrieben. Auch hier verwenden wir eine eigene, nicht beim IANA registrierte Protokollnummer.

6.2.7 cSDP

Dieses Modul ist für die Komprimieren einer einzelnen Session Description, also einem SDP [Han96b] Paket verantwortlich. In der derzeitigen Implementierung ist diese Modul zwar vorgesehen, wurde jedoch noch nicht richtig getestet.

6.2.8 Bandbreiten-Messung

Um die Bandbreite während einer Verbindung messen zu können, implementieren wir noch ein weiteren Modul (Bandwidth). Dieses benutzt das in Kapitel 5 vorgestellte Verfahren zur Messung der Bandbreite. Derzeit werden die Ergebnisse noch NICHT für die Entscheidung mit verwendet, welches Datenformat für die ausgehenden RTP-Pakete zu benutzen ist.

Die Beschreibung der Meßergebnisse ist in Abschnitt 6.3 enthalten.

6.2.9 Simuliertes Netzwerk-Interface

Da unsere Implementierung ohne root Privilegien läuft, ist es auch nicht möglich direkt das Netzwerk-Interface zu benutzten. Wir simulierten aus diesem Grunde dieses Interface unter der Benutzung von

Standard-Sockets. Für jede RTP-Multicast-Adresse und für den Zugriff auf SAP erstellten wir ein normalen UDP Socket. Die eingehenden Pakete wurden dann um einen künstlich erzeugten IP und UDP-Header ergänzt, und an die untere Schicht weitergegeben. Somit war sichergestellt, daß die anderen Module (cSAP, RTP-Mixer etc.) auch so arbeiten, wie sie es in einer realen Implementierung würden. Alle Protokollköpfe waren somit ebenfalls vorhanden.

Wenn das simulierte Netzwerkinterface ein Paket von einer unteren Schicht empfing, wurden die IP und UDP-Header wieder entfernt. Mit Hilfe der Zieladresse des IP-Headers und der Zielportnummer des UDP-Headers, wurden die Daten an exakt diese Adresse gesendet. Da diese Felder von dem Sendeteil der Implementierung auf der anderen Seite des Modems generiert wurden, bestimmte diese auch das Ziel. Insofern kommt dies Implementierung einem realen Netzwerkinterface recht nahe.

6.2.10 Filehandler

Wie in der Beschreibung aufgefallen sein dürfte, heben wir es hierbei mit mehreren Datenquellen zu tun:

- dem Socket für den RTP Strom,
- dem Socket für die SAP Daten,
- dem Filedescriptor für das Modem,
- dem Filedescriptor für das Terminal (nur auf der Seite des Anrufers).

Es ist somit nötig, mit `select` oder `poll`[5] zu arbeiten. Damit sich dies in der Implementierung der einzelnen Module nicht weiter berücksichtigt werden muß, implementierten wir ein weiteres Modul (Filehandler). Dieses besitzt eine Main-Loop, in der ein `select` ausgeführt wird. Routinen, die auf Daten eines Filedescriptors warten, müssen sich und den Filedescriptor beim Filehandler mit einer Callback-Routine registrieren. Wenn Daten für einen solchen Filedescriptor bereitstehen, ruft der Filehandler die entsprechende Callback-Routine auf.

Wie schon zu Beginn dieses Kapitels erwähnt, kann das Programm durch Eingabe eines bestimmten Zeichenwertes (`CRTL/B`) beendet werden. Realisiert wird dies durch eine globale Variable, mit Hilfe die Main-Loop des Filehandlers beendet werden kann.

6.3 Messungen an der Implementierung

Nachdem die oben beschriebene Testumgebung implementiert war, galt es diese zu Testen. Als erstes Testszenario versuchten wir die zu der Zeit zufälliger Weise aktive Übertragung der NASA STS-82 Space-Shuttle-Mission zu verfolgen. Wir verwendeten hierzu die in Abbildung 6.3 dargestellte Umgebung.

Die RTP-Daten wurden im `PCMU2` Format (40ms Frames mit `PCM` μ-law) gesendet. Dies entspricht 320 Bytes je Frame. Für die Komprimierung der Daten verwendeten wir `GSM`. Anhand von Tracinginformationen, die auf unserer Seite ausgegeben wurden waren deutliche Paketverluste zu erkennen, auch kam es vor, daß die Pakete in der falschen Reihenfolge ankamen. Unsere Implementierung verhielt sich jedoch stabil, so daß es möglich war der Übertragung zu folgen. Nach diesem ersten Test gingen wir davon aus, daß die Implementierung hinreichend arbeitet, um Messungen durchführen zu können.

Die erste Messung beschäftigte sich mit Fragestellung, wie hoch die tatsächliche benötigte ausgehende Bandbreite für unseren komprimierten RTP-Datenstrom ist. Wir sendeten hierzu von einem Windows

[5] `select` und `poll` sind zwei C-Funktionen unter UNIX, die es ermöglichen auf Daten und Ereignissen von mehren Filedescriptoren gleichzeitig zu warten.

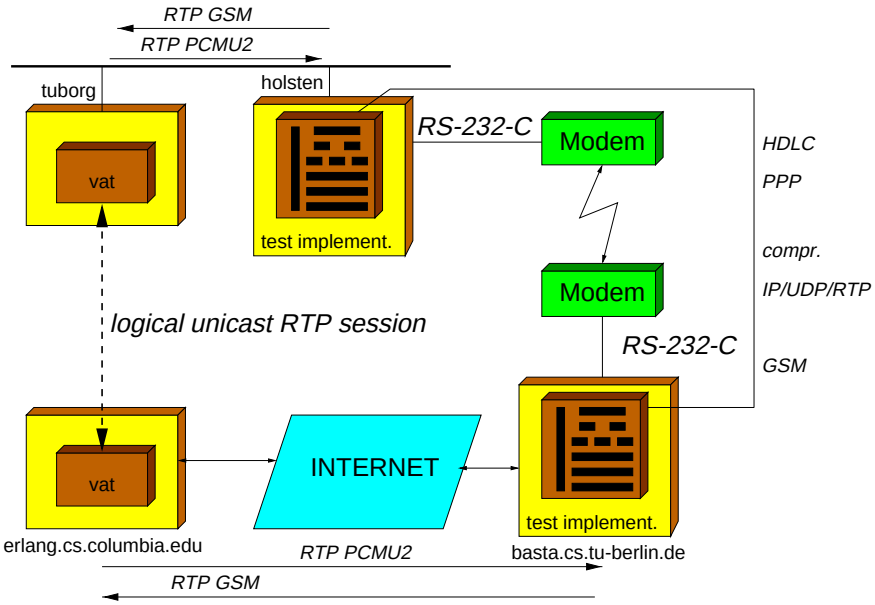


Abbildung 6.2: Verwendete Testumgebung

95 PC mit Hilfe von `vat` RTP-Daten (PCMU2, 40ms Frames) in das Netz. In unserer Implementierung protokollierten wir dann die Anzahl der pro Sekunde über das Netz empfangenen Bytes, und addierten den Overhead für einen einfachen IP und UDP Header (28 Bytes)[6]. Ebenfalls wurden die Bytes gezählt, die durch die Modemverbindung jede Sekunde gesendet wurden. In Abbildung 6.3 sind die beiden Messungen gegenübergestellt. Der eingehende Datenstrom beinhaltete 40 ms RTP Frames, mit `PCM` μ-law codierten Audiosamples. Für den komprimierten Datenstrom verwendeten wie die `GSM` Kodierung. Für die Messungen wurde ein Text vorgelesen und übertragen.

Es sind deutlich die kurzen Unterbrechungen zu erkennen, die aus den Sprechpausen resultieren. Ebenfalls gut zu erkennen ist, daß für die eingehenden Daten etwas mehr als 70 kbit/s benötigt werden. Dieser gemessene Wert deckt sich auch sehr gut mit der theoretischen Betrachtung aus Abschnitt 3.2.1 (wir ermittelten dort 72 kbit/s). Die benötigte Bandbreite zum Senden der komprimierten RTP Daten schwingt im gleichen Takt, wie die für den eingehenden Datenstrom. Im Mittel liegt sie bei ca. 15 kbit/s. Zum Vergleich leiten wir uns den zu erwartenden Wert her:

$$\begin{aligned} B &= (GSM\text{bytes} + IP/UDP/RTP - Header + PPP + HDLC)\text{bytes}/40\text{ms} \\ &= (66 + 2 + 2 + 4)\text{bytes}/40\text{ms} \\ &= 14,8 \text{ kbit/s} \end{aligned} \tag{6.1}$$

Der gemessene Wert liegt somit sehr nahe an dem zu erwartenden theoretischen Wert.

Die gleiche Messung führten wir nocheinmal durch, jedoch wurde nun die ausgehenden Daten nicht mit `GSM`, sondern mit `LPC` komprimiert. Die graphische Auswertung zeigt die Abbildung 6.3. Die nun benötigte Bandbreite beträgt nun nur noch ca. 7 kbit/s. Setzt man anstelle der bei `GSM` benötigten 66 Bytes für 40 ms Audiodaten die für `LPC` nötigen 24 Bytes ein, so ergibt sich ein theoretischer Wert von $6,4$ kbit/s. Dieser Wert liegt ebenfalls dicht bei dem von uns gemessenen.

[6] Zur Erinnerung, in unserer Implementierung haben wir nur Zugriff auf die Nutzdaten, jedoch nicht auf den IP und UDP-Header.

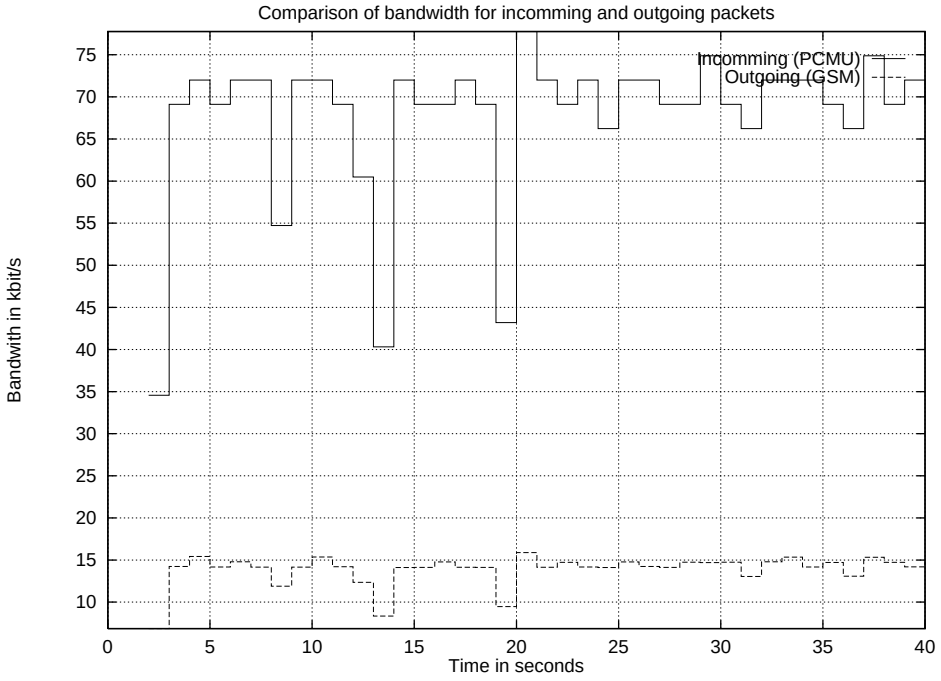


Abbildung 6.3: Messung der benötigten Bandbreite des empfangenen und gesendeten (GSM kodierten) Datenstroms

6.4 Vorhersage der verfügbaren Bandbreite

Nachdem unsere Implementierung so weit funktionierte, waren wir interessiert zu sehen, ob es möglich ist die Bandbreite während einer Verbindung zu messen. Wir verwendeten hierzu das bereits in Kapitel 5.1 beschriebene Packet-Pair Verfahren. Um Messungen durchführen zu können, benötigten wir somit zwei Pakete, von denen wir wissen, daß sie direkt aufeinander folgen. Da wir jedoch nur die RTP-Daten übertrugen, und diese in einem größeren Abstand gesendet wurden, reichten diese nicht. Versuchsmessungen bestätigten dies Annahme auch. Aus diesem Grunde führten wir ein weiteres PPP-Protokoll-Paket ein. Dieses Paket wurde vor jedem eigentlichem PPP-Paket gesendet, so daß wir sicher waren, daß es die beiden Pakete auch wirklich in Folge gesendet wurden. Dieses Triggerpaket bestand nur aus der PPP-Protokollnummer ohne Daten.

Für die Bestimmung machten wir immer 100 Messungen. Gemessen wurde die Zeit die verging, vom Eintreffen des Triggerpaketes bis zum Eintreffen des nächsten PPP-Paketes. Die Messungen sind wie in Abschnitt 5.9.2 als kumulierte Häufigkeitsverteilung dargestellt (siehe Abbildung 6.4). Die Messungen wurden während eines realen Gespräches mit Dr. Henning Schulzrinne aufgenommen, und entsprechen somit einem real vorkommenden Datenaufkommen. Zur Erinnerung, der definierten, daß die Bandbreite, bei der das 0,5-Quartil liegt näherungsweise als die real zur Verfügung stehende Bandbreite angesehen werden kann.

Im Gegensatz zu unseren Testmessungen aus Kapitel 5 verlaufen die Kurven ein wenig linearer. Das war auch zu erwarten, da wie es bei den Testmessungen mit optimalen Verhältnissen zu tun hatten. Dennoch ist deutlich zu sehen, daß alle Meßreihen, mit zwei Ausnahmen, sehr dicht beieinander liegen. Das heißt, daß solche Messungen durchaus ein reproduzierbares Ergebnis liefern können. Die statistische Streuung der Meßergebnisse ist insofern als verhältnismäßig klein zu bezeichnen. Mit Ausnahme der beiden „Ausreißer“ liegt die gemessene Bandbreite somit zwischen 17 und 20 kbit/s.

Um die Güte dieser Messungen bestimmen zu können, müssen wir zunächst jedoch die theoretisch zu erwartende Bandbreite mathematisch herleiten. Gemessen wurde die Bandbreite, die für PPP-Nutzdaten zur Verfügung steht. Es entfallen also die HDLC-Framing-Informationen (Flag und FCS), sowie das

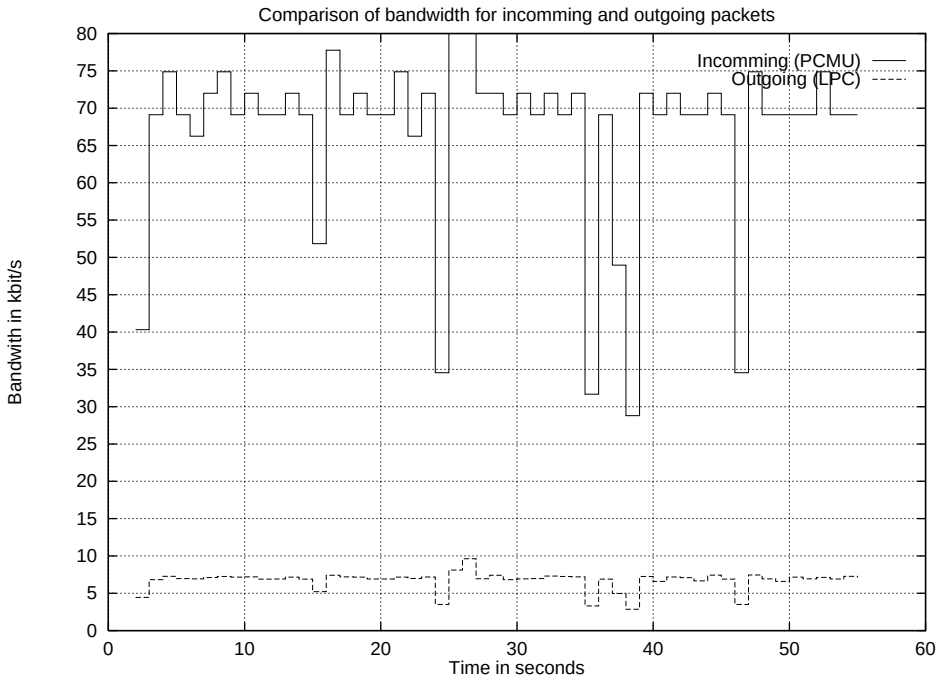


Abbildung 6.4: Messung der benötigten Bandbreite des empfangenen und gesendeten (LPC kodierten) Datenstroms

PPP-Protokoll-Feld. Übertragen wurden in der Regel 40 ms GSM Frames (66 Bytes), die zusammen mit dem komprimierten IP/UDP/RTP-Headern 68 Bytes ausmachten. Für die Framinginfomrationen fielen 2 Bytes für die HDLC Flags, 2 Bytes für das FCS Feld, und ein Byte für das komprimierte PPP-Protokoll-Feld (zusammen 5 Bytes) an.

Laut Aussage des Modems[7] betrug die Datenrate in der von uns gemessenen Richtung 24 kbit/s. Wie bereits in Abschnitt 5.6 errechnet wurde, kommen zu den zu übertragenden Daten ca. 1,6% Overhead durch bitstuffing hinzu. Die für Modemdaten zur Verfügung stehende Bandbreite B_{modem} beträgt somit:

$$\begin{aligned} B_{\text{modem}} &= 24000 \text{ bit/s} - 1,6\% \\ &= 23616 \text{ bit/s} \end{aligned}$$

Bei der von uns verwendeten ACCM waren 3 Werte zum Escapen maskiert. Es ergibt sich somit ein Overhead O für E_{ACCM} gesetzte Bits in der ACCM von:

$$\begin{aligned} O &= (E_{\text{ACCM}} + E_{\text{FLAG}} + E_{\text{ESC}})/256 \\ &= (3 + 1 + 1)/256 = .01953 \\ O(n) &= .01953 * n \end{aligned}$$

Übertragen wurden immer Frames mit $68 + 5$ Bytes, so daß sich die theoretische Bandbreite B_{theor} ergibt zu:

$$\begin{aligned} B_{\text{theor}} &= 68/(68 + 5 + O(68 + 5)) * 23616 \text{ bit/s} \\ &= 21,6 \text{ kbit/s} \end{aligned}$$

Die Bandbreite, die unser vorgestelltes Meßverfahren ermittelte, lag mit ca. $18,5$ kbit/s um ca. 15% niedriger, als die theoretisch zu erwartende.

[7] Bei dem von uns verwendeten US-Robotics Modem können die Parameter der letzten Verbindung mit dem Befehl ATI6 abgefragt werden.

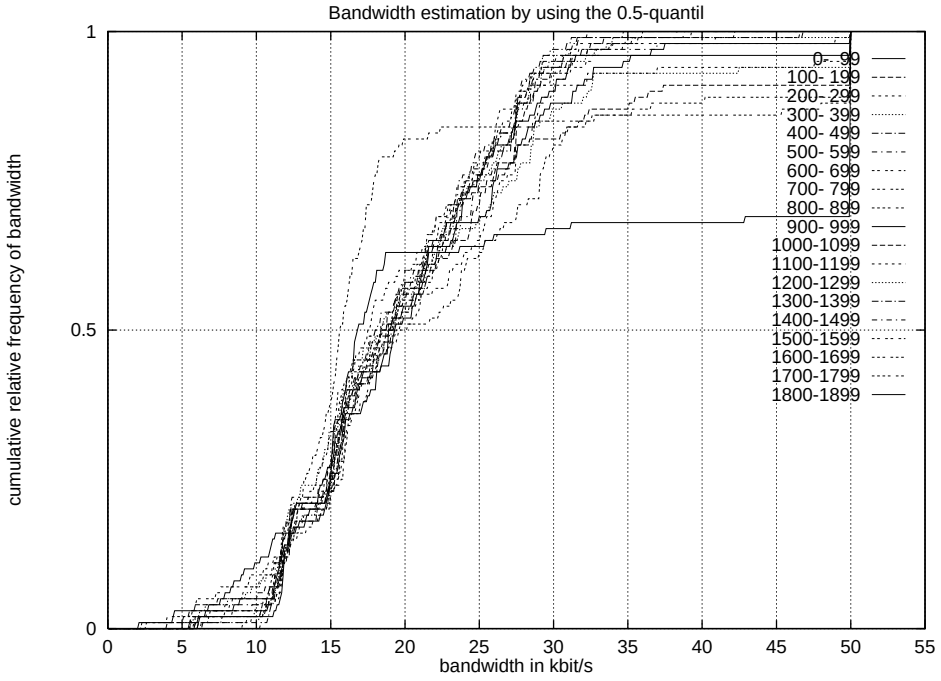


Abbildung 6.5: Bandbreitenmessung mittels 0,5-Quartil

6.5 Zusammenfassung

Wir haben während der letzten Monate eine Testumgebung entwickelt, mit der wir unsere zuvor theoretisch Überlegungen der letzten Kapitel überprüfen wollten. Aus verschiedenen, oben beschriebenen Gründen waren wir nicht in der Lage, diese direkt in einer PPP-Implementierung einzubetten. Wir versuchten daher eine Referenzimplementierung zu schaffen, die die gleichen Aufgaben zu bewältigen hat, wie eine reale PPP-Implementierung.

Durch die Integrierung von Timern an verschiedenen Stellen waren wir in der Lage, die verschiedenen Messungen durchzuführen. So konnten wir die resultierende benötigte Bandbreite für den ausgehenden Datenstrom in Relation zum eingehenden betrachten. Wie zu erwarten zeigte sich eine nahezu proportionale Verringerung der benötigten Bandbreite. Durch die Konvertierung der eingehenden IP/UDP/RTP-Daten, konnte die benötigte Bandbreite bei PCM μ-law auf ca. 21% verringert werden, wenn `GSM`, sowie IP/UDP/RTP-Header-Komprimierung eingesetzt wurden. Bei der Verwendung von `LPC`, verringerte sie sich sogar auf ca. 10%.

Mit Hilfe dieses Verfahrens ist es somit möglich, ausgesendete RTP-Datenströme aus dem INTERNET über eine Verbindung mit niedriger Bandbreite transparent zu übertragen. Eine Modemverbindung erwies sich dabei als durchaus brauchbar.

Weiterhin konnten wir durch Messungen zeigen, daß unser entwickeltes Verfahren zur Bestimmung der zur Verfügung stehenden Bandbreite sehr gute Ergebnisse liefert. Durch geeignetes Koppeln dieser beiden Komponenten müßte es somit auch möglich sein, das RTP-Datenformat entsprechend der gemessenen Bandbreite dynamisch anzupassen.

Kapitel 7

Beschreibung der IsdnLib

Vor einigen Monaten, als wir nach einem geeigneten Thema für diese Diplomarbeit suchten, kam uns die Internet-Telefonie in den Sinn. Wir überlegten in wie es möglich ist, gewöhnliche Telefongespräche über eine ISDN-Verbindung von einem angeschlossenen Rechner aus führen zu können.

7.1 ISDN und SUN

Zu diesem Zweck bot sich die Beschaffung einer ISDN-Karte für eine SUN an, auch besorgt wurde. Mitbestellt wurde auch die ISDN-Software *SunISDN 1.0.2*, mit der es möglich sein sollte, diese Karte zu benutzen. Kurz nach der Installation stellte sich jedoch ernüchternd schnell heraus, daß es sich hierbei nur um ein PPP Netzwerk für ISDN handelte. Es war mit dieser Software nicht möglich die ISDN Karte für andere Zwecke zu benutzen. Insbesondere stellte diese keine Programmierschnittstelle zur Verfügung.

Wir versuchten zwar durch dissassemblieren zweier beiliegender Testprogramme (zum Verschicken und zum Empfangen einer Audiodatei) einen Weg zu finden dennoch mit der Karte arbeiten zu können. So fanden wir heraus, daß diese Programme mit der Software über eine *named-pipe* kommunizierten, und waren auch in der Lage einen Anruf zu initiieren. Jedoch fehlten Informationen über die Art und Weise von Signalisierungen.

Wir fragten natürlich auch bei SUN direkt nach und erhielten die Antwort, daß diese Software ausschließlich für den Einsatz von PPP konzipiert worden war. Die Testprogramme seinen nur aus historischen Gründen mitgeliefert worden. Von der direkten Benutzung der *named-pipe* riet man uns auch ab, da diese Schnittstelle sich von Version zu Version ändern könnte.

Dennoch verwiesen uns die Leute von SUN auf ein Produkt, mit dem es möglich sein sollte die ISDN Karte für unsere Zwecke nutzen zu können. Dieses Produkt (*SunXTL 1.2*) wurde dann auch bestellt und installiert.

7.1.1 SunXTL 1.2

SUN Microsystems bot zu dieser Karte eine interessante Programmumgebung an, genannt *SunXTL 1.2* [Mic94c, Mic94a, Mic94b, Mic94d]. Mit dieser sollte es sehr leicht möglich sein, Telefonie-Applikationen zu entwickeln, die nicht nur auf das ISDN beschränkt sind, sondern auch Netzwerke wie ATM, Ethernet und Modem-Verbindungen. Die Abbildung 7.1.1 (entnommen aus [Mic94a]) gibt einen kleinen Überblick über die gesamte Architektur.

Wie in der Abbildung zu erkennen ist, stellt diese Architektur eine verteilte Telefonie Umgebung zur

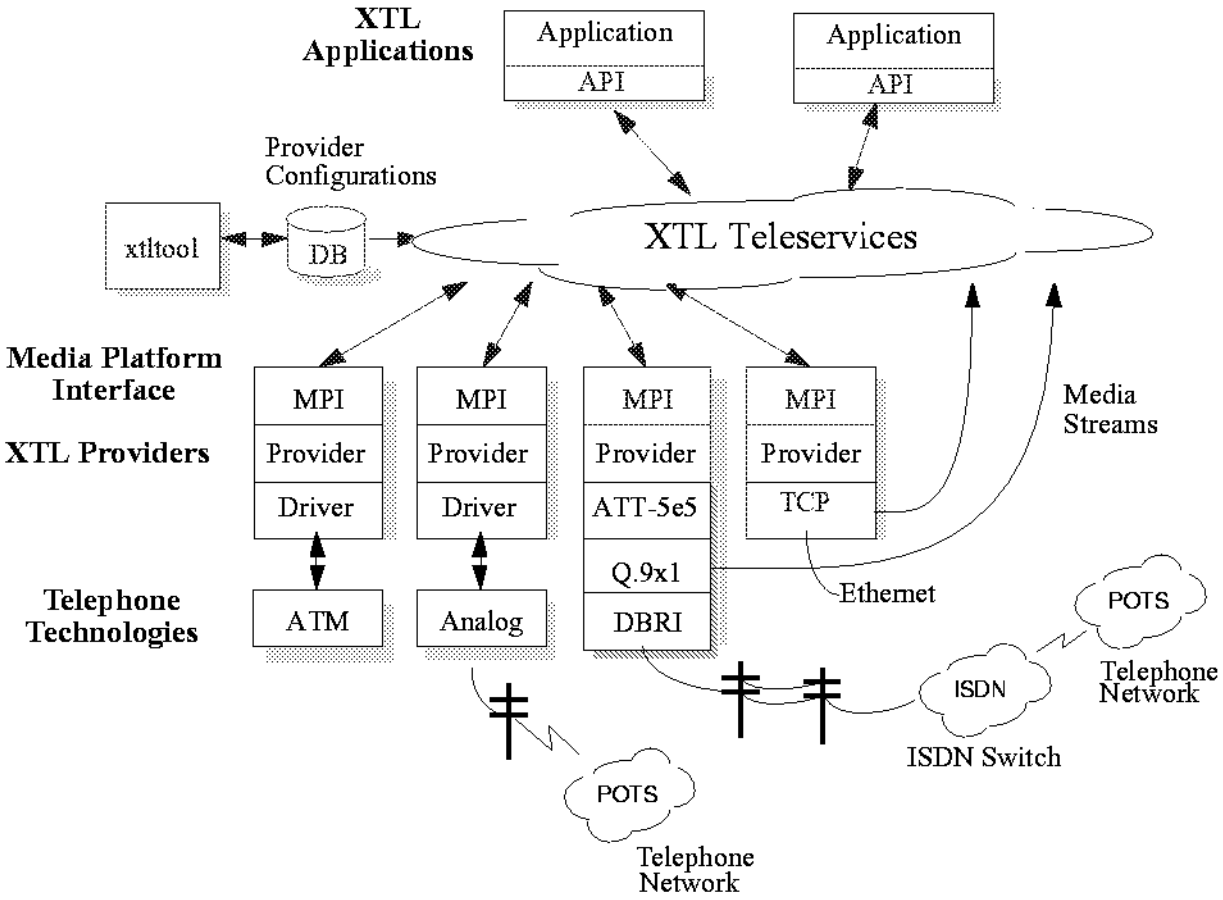


Figure 1-1 Teleservices Ar chitectur e

Abbildung 7.1: Überblick über die SunXTL Architektur

Verfügung. So weit so gut, als wir jedoch nach dem MPI[1] für die ISDN Karte suchten, stellte sich heraus daß dieses nicht zum Lieferumfang gehört. Auf Nachfragen hieß es dann, daß wir einen derartigen MPI von einem Fremdhersteller beziehen könnten. Die Software stellte auch kein sonstige Möglichkeit zur Verfügung, die ISDN Karte zu verwenden.

Nach dieser wiederum ernüchternden Erkenntnis fragten wir in der GMD nach, ob dort schon irgendwo eine Implementierung existiert. Dort verhielt man sich eher zurückhaltend der XTL Architektur gegenüber und riet uns, noch einige Zeit zu warten. Man war dort selber dabei eine ISDN Karte für eine SUN zu entwickeln, die dann auch besser liefe und unterstützt werden sollte.

7.1.2 Nachfragen im INTERNET

Wir fragten nach diesen Erkenntnissen im INTERNET nach, ob jemand etwas genaueres über einen MPI für XTL wisse. Wir erhielten jedoch nur negative Antworten; es wurde sogar davon gesprochen, daß SUN dieses Paket bald wieder vom Markt nehmen würde. Der mangelhafte Support von SUN bezüglich dieser Software wurde allerdings von mehreren Leuten bestätigt. Auch wurde uns nochmals bestätigt, daß es im XTL Paket keinerlei Software gibt, die die Benutzung einer ISDN Karte erlaubt.

[1]Das MPI (Media Provider Interface) verbindet einen Typ eines physikalischen Gerätes (wie einer ISDN Karte) mit den XTL Teleservices. Es wandelt somit die internen Funktionen der XTL in Aktionen auf dem Gerät um (z.B. Initiieren eines Telefonanrufs), und erzeugt für eingehende Ereignisse (z.B. ein eingehender Anruf) in die internen Nachrichten. Ohne diesen MPI ist es somit nicht möglich, die unterliegende Hardware zu benutzen.

7.1.3 SunShine

Nach diesen recht frustrierenden Erkenntnissen, stießen wir jedoch auf ein Projekt namens SunShine, an dem man an der Helsinki Universität in Finnland arbeitete. Es hieß, daß dort etwas erstellt wurde, mit dem die Benutzung der ISDN Karte möglich sein sollte.

Nach der Kontaktaufnahme erhielten wir auch gleich Zugang zu den Quellen, und konnten diese auch benutzen. Es existierten zwei Testprogramme, mit denen es möglich war eine Datei über das ISDN zu versenden.

Ein wenig enttäuschend waren jedoch,

- daß es keine Library gab, die zur Applikation dazugebunden werden konnte,
- daß die Implementierung sehr instabil war (die SUN stürzte des öfteren ab)
- und daß der Sourcecode ziemlich chaotisch aussah.

Hinzu kam, daß das Semester dort gerade zu Ende war, und mit einer Weiterentwicklung erst in einigen Monaten zu rechnen sei.

Das SunShine Projekt war die einzige Referenz, die wir in Hinblick auf Unterstützung zur Programmierung der ISDN-Karte gefunden haben. Aus diesem Grunde entschlossen wir uns zu einer Zusammenarbeit. Ein Teil der Entwicklung wurde dort weiter von Bengt Olof Shalin betrieben, der andere Teil von uns.

7.2 Ein kurzer Überblick über ISDN

Um verstehen zu können, welchen Teil des SunShine Projektes wir übernommen haben, müssen wir zuvor einen kurzen Überblick über ISDN geben. ISDN wurde von der International Telecommunication Union (ITU) standardisiert. In der Recommendation I.120 [Int93i] ist eine Übersicht über das Rahmenwerk der ISDN Recommendations (siehe Abbildung 7.2. Zum Verständnis der weiteren Ausführung ist nur ein Überblick über die einzelnen Schichten im ISDN nötig.

Die physikalische Schicht (physical layer) ist beim ISDN durch die ITU Empfehlung I.430 [Int93a] für einen Basisanschluß[2] spezifiziert. Die ISDN Karte ist die Instanz, die eine Schnittstelle zu dieser physikalischen Schicht herstellt. Da diese bereits vorhanden ist, beschäftigen wir uns hier nicht weiter damit.

Die zweite Schicht (Link Layer) ermöglicht den Datenaustausch mittels des Physical Layers. Sie erfüllt dabei Aufgaben wie die Adressierung von Absender und Empfänger und ermöglicht einen fehlerfreien Datenaustausch durch Fehlererkennungs- und Fehlerkorrekturmaßnahmen. Sie ermöglicht auch das Mulitplexen mehrerer Zugangspunkte. Diese Schicht ist durch Q.920 [Int93f] (gleich I.440 [Int93k]) und Q.921 [Int93j] (entspricht I.441 [Int93l]) spezifiziert.

Die dritte Schicht (Network Layer) ermöglicht die Kommunikation zwischen zwei oder mehreren Teilnehmern. Beim ISDN entspricht dies beispielsweise einer Möglichkeit, daß zwei Teilnehmer über ISDN miteinander telefonieren können. Diese Schicht ist spezifiziert durch Q.930 [Int93c] (I.450 [Int93m]) und Q.931 [Int93d] (I.451 [Int93n]).

Die anderen Schichten sind für unsere Arbeit nicht weiter von Bedeutung und werden daher nicht weiter betrachtet.

[2] Es werden drei Arten von Anschlüssen unterschieden, der Basisanschluß, der Primärmultiplexeranschluß und das Breitband-ISDN. Diese unterscheiden sich im wesentlichen nur in der Übertragungsrate und den physikalischen Eigenschaften. Für die weitere Betrachtung beschränken wir uns hier auf den Basisanschluß.

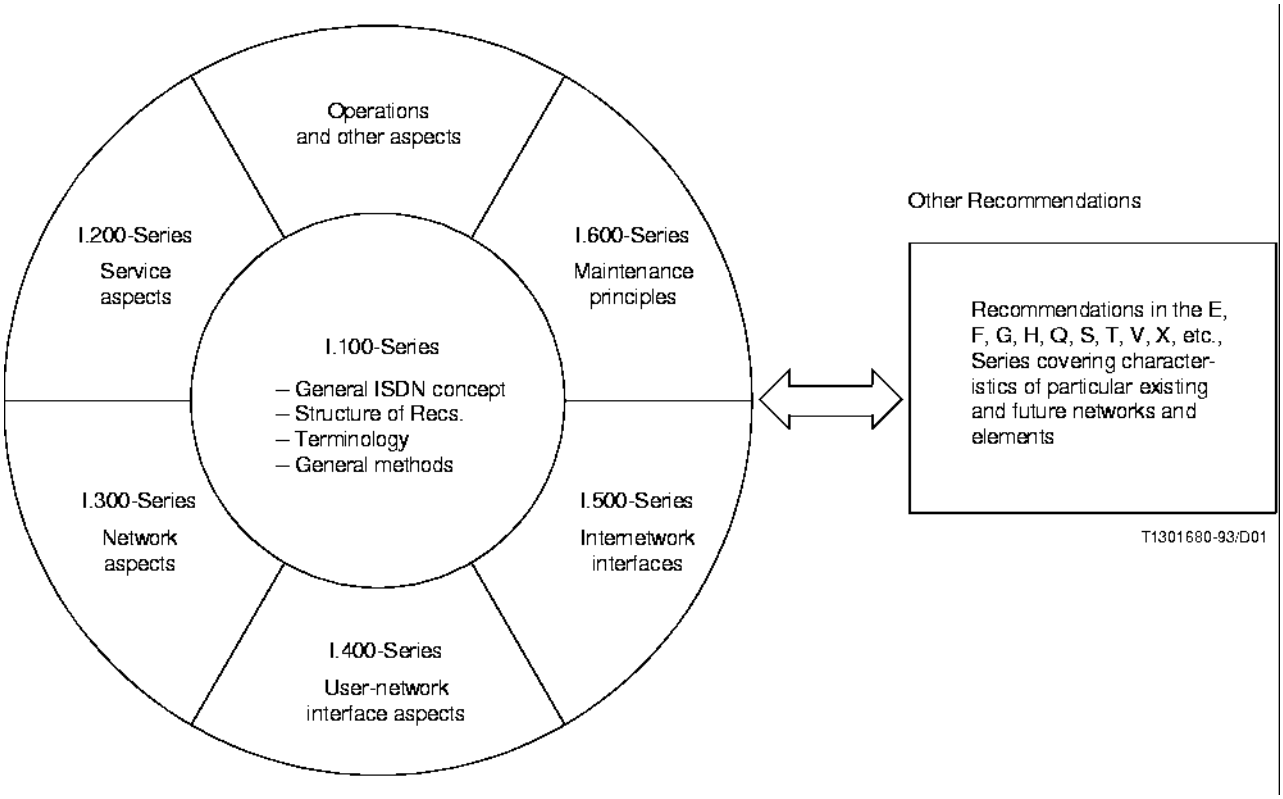


Abbildung 7.2: Übersicht über die ITU-T Normen bzgl. ISDN

7.3 Aufgabenteilung

Wie bereits erwähnt benötigten wir eine Software, die den Zugriff auf die ISDN-Karte ermöglicht. Genauer formuliert benötigen wir eine Software, die genau die Schicht 2 und 3 behandeln kann, und eine den Benutzer eine Programmierschnittstelle zur Verfügung stellt. Ohne die Behandlung (ohne eine Implementierung dieser beiden Schichten entsprechend der oben genannten ITU-T Empfehlungen) dieser beiden Schichten ist es auch nicht möglich, die ISDN-Karte in einer realen Umgebung (d.h. angeschlossen an des öffentliche ISDN-Netz der Telekom) zu benutzen.

In dem SunShine Projekt wurden diese beiden Schichten implementiert, so das es mit dieser Software möglich war, auf das „normale“ ISDN zuzugreifen und es zu nutzen. Wir zuvor schon erwähnt gab es jedoch eine Menge Fehler in der Implementierung. Weiterhin war diese auch nicht für einen echten Einsatz geplant, sondern diente nur als Versuchsobjekt. Hinzu kam noch, daß die Implementierung kein Programmierinterface zur Verfügung stellte.

Es wurde daher beschlossen, einen Teil neu zu entwickeln. Glücklicherweise waren die Implementierungen der beiden Schichten von einander völlig getrennt. Die Implementierung der Schicht 2, genauer des Q.921 LAP-D Protokolls, wurde als eine eigene Streams-Extention für den SUN Kernel implementiert. Die Implementierung des Q.931 Protokolls hingegen wurde zur Applikation dazugelinkt.

Nach einigen Überlegungen entschlossen wir uns für eine Implementierung des Q.931 Moduls. Zusätzlich sollte eine Programmierschnittstelle zur Verfügung gestellt werden. Beng Olof Shalin entschied sich währenddessen, das Q.921 Modul völlig zu überarbeiten.

7.4 Beschreibung der Implementierung

Im folgenden Abschnitt wollen wir den von uns neu implementierten Teil beschreiben. Die Abbildung 7 gibt einen Überblick über die beteiligten Module.

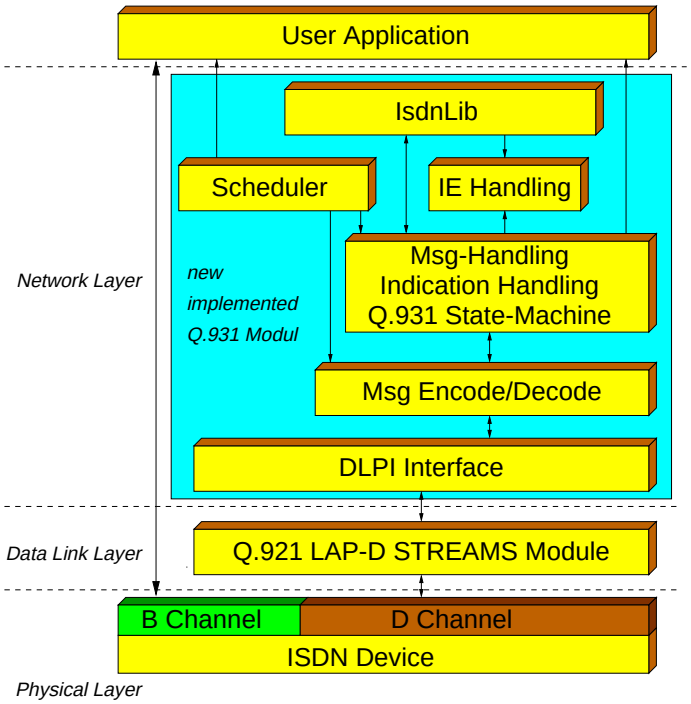


Abbildung 7.3: Aufbau unserer Implementierung

7.4.1 DLPI Interface

Das DLPI (Data Link Provider Interface) Interface Modul ermöglicht einen relativ einfachen Zugriff auf des Q.921 Moduls. Da dieses als STREAMS Extension im Kernel der SUN integriert ist, steht nach Außen hin nur eine DLPI Schnittstelle zur Verfügung. Der Nachrichtenaustausch ist somit ein wenig umständlich, da es verschiedene Typen von Nachrichten gibt, die je nach Type einen anderen Aufbau besitzen. Da dieses Interface weitaus mehr Möglichkeiten zur Verfügung stellt, als wir für den Zugriff auf das ISDN benötigen, wurden einige Routinen geschrieben, die den Nachrichtenaustausch erheblich vereinfachen.

7.4.2 Msg Encode / Decode

Dieses Modul ist beim Scheduler als Callback-Routine für eingehende Nachrichten vom Q.921 Modul. Es nutzt das DLPI Interface Modul, um die Nachrichten über die DLPI Verbindung zu empfangen. Nachdem die Nachricht empfangen wurde, wird je nach Nachrichtentyp weiter verfahren. Die im allgemeinen eingehenden Daten Nachrichten werden dann im ersten Schritt dekodiert.

Jede nach Q.931 eingehende Nachricht besitzt den gleichen Aufbau (siehe Abbildung 7.4.2). Nachdem die Nachricht anhand des *Protocol Discriminator* Feldes als Q.931 Nachricht identifiziert wurde, wird die *Call Reference Value* (*CRF*) decodiert, sowie der *Message Type* (*MT*). Die dann optional folgende Liste mit weiteren *Information Elements* (*IE*) wird dann Stück für Stück dekodiert und in einer internen Struktur abgelegt. So wie in Q.931 beschrieben werden unbekannte IEs ignoriert, sie erzeugen keinen Fehler.

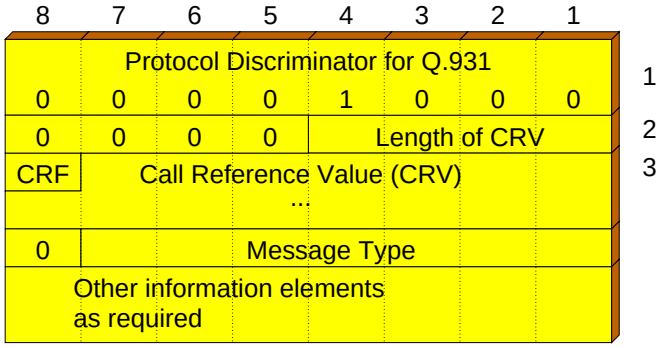


Abbildung 7.4: Genereller Aufbau einer Q.931 Nachricht

Nach dem der generelle Nachrichtenaufbau überprüft wurde, wird anhand der CRV, die eine logische Verbindung zwischen dem ISDN-NT und der ISDN-Karte identifiziert, der zugehörige Kontext gesucht. Existiert keiner, so wird zunächst ein neuer angelegt, unabhängig davon, ob um welchen Nachrichtentype es sich handelt. Die dekodierte Nachricht wird dann an je nach *MT* an eine entsprechende Funktion weitergereicht, die diese Nachricht dann behandeln kann.

Dieses Modul ist auch für das Verschicken einer vollständig aufbereiteten Nachricht an das DLPI Modul zuständig.

7.4.3 Msg Handling, Indication Handling, Q931 State-Machine

Dieses Modul ist der Kern der gesamten Q.931 Implementierung. Zunächst beinhaltet es alle Funktionen, die einen speziellen *MT* behandeln. Dazu gehören

- das Behandeln der eingetroffenen Nachricht (vom Msg Encode/Decode Modul)
- die Behandlung der Q.931 Finite State Machine für eingehende Nachrichten,
- die Generierung von Indications für die IsdnLib,
- die Behandlung der Q.931 Finite State Machine für ausgehende Nachrichten,
- die Generierung der ausgehenden Q.931 Datenpakete.

Alle Funktionen, die für die Behandlung eines speziellen *Message Type* nötig sind, wurden in einem eigenen Source-Files zusammengefaßt. Derzeit werden die folgenden Q.931 *Message Types* behandelt (vergleiche auch Q.931 [Int93d], Abschnitt 3.1):

- `ALERTING`
- `CALL PROCEEDING`
- `CONNECT`
- `CONNECT ACKNOWLEDGE`
- `DISCONNECT`
- `INFORMATION`
- `NOTIFY`

- PROGRESS
- RELEASE
- RELEASE COMPLETE
- SETUP
- SETUP ACKNOWLEDGE

Es fehlen somit nach Q.931 die folgenden Typen:

- RESUME
- RESUME ACKNOWLEDGE
- RESUME REJECT
- SUSPEND
- SUSPEND ACKNOWLEDGE
- SUSPEND REJECT
- STATUS
- STATUS ENQUIRY

Für die Benutzung der Implementierung reichte dies in der Vergangenheit aus, die fehlenden Nachrichten wurden auch nie empfangen. Um eine Q.931 konforme Implementierung zu haben, müßten diese dennoch behandelt werden.

Für jede eingehende und ausgehende Nachricht wird die Q.931 Finite State Machine des entsprechenden Kontextes abgearbeitet. Zusätzliche werden die (alle) ebenfalls in Q.931 definierten Timer entsprechend den SDL [Int93b] Diagrammen behandelt. Realisiert wird dies mit Hilfe des Scheduler Moduls.

Wenn eingehende (zum Teil auch ausgehende) Nachrichten eine Indikation zur Applikation erfordern, wird in den einzelnen Funktionen noch eine *IsdnLib Indication* generiert und die zuvor registrierte Callback Funktion der Applikation aufgerufen.

7.4.4 IE Handling

Neben der Behandlung der *MT*, müßten die *Information Elements* kodiert und dekodiert werden. Bei der Betrachtung des Aufbaus einzelner *IE*s wird sehr schnell klar, daß dies eine etwas kompliziertere Aufgabe darstellt. Dies ist darin begründet, daß einige *IE*s sehr viele optionale Felder besitzen, so daß sich der Aufbau sehr stark ändert.

Aus diesem Grunde und aus dem Grunde der Übersichtlichkeit, haben diese Kodierung und Dekodierung der *IE*s in eine eigenen Modul ausgelagert. Für jedes *IE* existieren zwei Funktionen, die jeweils in einem Source-File implementiert wurden. Die eine Funktion Realisiert die Kodierung, die andere die Dekodierung. Zu Debugging Zwecken haben die meisten *IE*s noch eine weitere Funktion zur Verfügung, eine mit der der Inhalt formatiert ausgegeben werden kann.

In der jetzigen Implementierung werden folgende *IE*s unterstützt (in alphabetischer Reihenfolge):

- Bearer capability

- `Called party number`
- `Called party subaddress` †
- `Calling party number`
- `Calling party subaddress` †
- `Cause`
- `Channel identification`
- `High layer compatibility` †
- `Low layer compatibility` †
- `Keypad facillity`
- `Notify indication` †
- `Progress Indication`

Die markierten *IE*s sind noch nicht vollständig implementiert.

7.4.5 Scheduler

Der Scheduler ist für die Behandlung der asynchronen Ereignisse zuständig. Er besitzt eine Main-Loop, in der er auf ein Ereignis wartet. Er besitzt eine kleine Programmierschnittstelle, über die eine Routine für die Behandlung eines bestimmten Ereignisses registriert werden kann. Dies kann

- ein Ereignis bzgl eine Filedescriptors sein (lesbar, schreibbar, Ausnahmebehandlung)
- oder ein abgelaufener Timer (in Millisekunden Auslösung) sein.

Das Modul mit der Q.931 Finite State Machine benutzt diesen beispielsweise für die Realisierung der Timer.

7.4.6 IsdnLib

Die IsdnLib stellt die eigentliche Programmierschnittstelle nach außen hin zur Verfügung. Sie stellt eine Reihe von Funktionen zur Verfügung, mit deren Hilfe eine Applikation das ISDN relativ einfach benutzen kann.

Die IsdnLib basiert maßgeblich auf den anderen zuvor beschriebenen Modulen, so auch das Scheduler Modul. Bei der Entwicklung der IsdnLib wurde jedoch auch daran gedacht, diese später einmal in tcl/tk[3] einbinden zu können. Zu diesem Zweck muß es möglich sein, die Schedulling Routinen des tcl benutzen zu können. Die Implementierung wurde daher so realisiert, daß nur einige „Hüllfunktionen“ geschrieben werden müssen, die den Namen der Funktionen des Scheduler Moduls besitzen, und die entsprechenden Funktionen der tcl Implementierung aufrufen. Dies wurde bisher jedoch noch nie getestet!

[3] tcl/tk ist eine Scriptsprache, die von John K. Ousterhout entwickelt wurde.

7.5 Warum eine neue Library?

Nachdem in dem letzten Abschnitt der Aufbau unserer Implementierung beschrieben wurde, wollen wir noch einmal kurz die Gründe beschreiben, die zu der Entscheidung führten die Library in der Art zu realisieren, wie sie jetzt existiert.

Zu Beginn des Projektes überlegten wir, was das einfachste Interface für eine ISDN Unterstützung sein könnte. Wir überlegten uns hierzu die folgenden Varianten:

1. Realisierung als eigenständigen Prozess mit einem socket() ähnlichem Interface.
2. Implementierung als CAPI 2.0 Schnittstelle.
3. Definition einer eigenen Library, die zur Applikation gelinkt wird.

7.5.1 Socket-Interface

Die erste Version sah auf dem ersten Blick sehr einfach aus und hatte den Vorteil, daß das ISDN eventuell auch über das Netz zu benutzten wäre. Bei weiteren Überlegungen stellte sich jedoch schnell heraus, daß sich eine Reihe von Problemen ergeben, die den Realisierung und vor allem die Benutzung ziemlich erschwert hätten. Hinzu kommt noch die Tatsache, daß hierdurch eine erhöhtes Schedulling auf dem Rechner zu erwarten wäre, auf dem dieser Server läuft.

Besonders schwierig war es jedoch, die asynchrone Signalisierung realisieren zu können. Im ISDN können zu jeder Zeit Nachrichten für existierende oder noch aufzubauende Verbindungen eintreffen. Um diese den einzelnen Verbindungen zuordnen zu können, müßte etwas ähnliches verwendet werden, wie die *Call Reference Value* der Q.931 Nachrichten. Wir würden dann ein neues Protokoll benötigen, in dem wiederum verschieden Nachrichtenformate definiert werden müßten.

Er stellte sich somit heraus, daß wir wiederum eine neues Protokoll definieren müßten, das ähnlich Komplex wäre wie das Q.931 Protokoll selber, womit wir dann im Grunde überhaupt nichts gewonnen hätten.

7.5.2 CAPI 2.0

Es erschien uns daher eher sinnvoll auf eine Schnittstelle zurückzugreifen, die bereits definiert war und als quasi Standard gilt, CAPI 2.0. Wir besorgten uns darauf hin die CAPI-Spezifikation und analysierten inwieweit es möglich wäre, diese Schnittstelle zu implementieren.

Bei der Betrachtung der Spezifikation fiel jedoch schnell auf, daß dieser ebenfalls recht komplex zu sein scheint. Zumal das Problem auftrat, daß diese Implementierung dann wiederum eine STREAMS Kernel Extension realisiert werden müßte. Da wir einen derartigen Aufwand scheuten, und da die Benutzung von CAPI auch recht kompliziert erschien, verwarfen wir auch diese Idee.

7.5.3 Implementierung als eigene Library

Somit blieb noch die letzte Variante übrig, die Implementierung als eine eigenständige Library, die zu der Applikation dazugelinkt werden muß. Diese Variante erschien uns als sehr vernünftig, da es möglich war direkt anzufangen und einige Tests durchzuführen. Es mußten nicht erst umständlich Protokolle definiert, oder Kernel Extensions geschrieben werden. Und vor allem erschien uns diese Variante immer flexibel erweiterbar zu sein. Wird ein neuer Dienst mit angeboten, so muß nur eine neue Funktion implementiert

werden. Sollten weitere Datenfelder zu einer Nachricht hinzukommen, würde sich hierdurch nur eine Struktur ändern, womit sich die Änderung für Applikation dann als transparent darstellt.

Ein kleinen Nachteil gab es bei dieser Variante jedoch auch. Da die Library als Teil des Prozesses läuft, würde ein Blockieren des Programms auch die ISDN-Schnittstelle beeinflussen. Eingehende Q.931-Nachrichten würden dann zeitlich nicht mehr Q.931 Konform behandelt werden können.

Wir entschieden uns dennoch für diese letzte Variante, und nannten die entstandene Library IsdnLib.

7.6 Beschreibung der entwickelten IsdnLib

Der Kern der Library stellt die interne „Event-Loop“ dar. Sie ist mit der im tcl zu vergleichen. Aus diesem Grunde ist die IsdnLib in der ähnlichen Art und Weise zu benutzen, wie eine Applikation, die tcl zur Benutzung mit dazulinkt.

Die einzelnen Funktionen der IsdnLib sind im Anhang D in der Form von Manualpages beschrieben. Die Implementierung der IsdnLib ist in Anhang E.

Kapitel 8

Zusammenfassung

Diese Diplomarbeit beschäftigte sich mit vier verschiedenen Problemkreisen, die bei der Übertragung von Multimediadaten über Verbindungen niedriger Bandbreite auftreten:

1. Übertragung von IP-Multicast-Daten
2. Verringerung der benötigte Bandbreite von RTP-Strömen
3. Effiziente Übertragung von Session-Announcements
4. Messung der zur Verfügung stehenden Bandbreite

Jeder Problemkreis wurde dabei genau untersucht und beschrieben. Für die Problemkreise 2 – 4 erarbeiteten wir Lösungsvorschläge, die wir schließlich auch in einer Testimplementierung (vgl. Kapitel 6) verifizieren konnten.

Wir sind der Meinung, daß der von gewählte Weg einen guten Ansatz darstellt, die Probleme, die bei Übertragung von IP-Multimediadaten über Verbindungen niedriger Bandbreite auftreten, zu analysieren und eine Lösung vorzustellen. Während der Erstellung dieser Arbeit wurde auch stets die aktuellen Ereignisse im INTERNET mit berücksichtigt. So wurde von uns die, nach unseren Informationen nach, erste Implementierung der IP/UDP/RTP-Header-Komprimierung implementiert und im realen Einsatz erprobt.

Zu diesem eigentlichen Hauptteil der Arbeit wurde noch eine Library entwickelt, die den Zugriff auf das ISDN über eine ISDN-Karte in einer SUN ermöglicht. Die Benutzung dieser Library als Basis für die Diplomarbeit von Frank Oertel zeigt, daß diese durchaus real eingesetzt werden kann.

8.1 Kritikpunkte

Dennoch gibt es einige Stellen, die Anlaß zur Kritik geben.

Zunächst einmal wurde auf die Reservierung von Bandbreite für die Übertragung der Multimediadaten zu wenig eingegangen. Eine genauere Betrachtung der entsprechenden Protokolle wie BACP oder RSVP eingegangen.

Es wäre sinnvoller gewesen, wenn unsere Testimplementierung direkt im PPP integriert wäre. Die Implementierung würde dann in einem weiten Bereich zum direkten Einsatz kommen. In der jetzigen Situation müssen die verschiedenen Schichten im Nachhinein in einer PPP-Implementierung integriert werden.

In Hinblick auf die implementierte IsdnLib bleibt zu erwähnen, daß es besser gewesen wäre, wenn sich diese an einem Standard wie CAPI orientieren würde. Es wäre dann einfacher, bestehende Applikationen

auf diese Plattform zu portieren und dort nutzen zu können. Wie jedoch schon beschrieben, erschien uns der Aufwand zu hoch zu sein, und die Entwicklungszeit der jetzigen Library hat gezeigt, daß diese Annahme wohl begründet ist. Es verging eine sehr lange Zeit, bis diese nun endlich benutzt werden konnte.

Anhang A

A Compressed Session Announcement Protocol

A.1 Abstract

Today session announements are definied by using the session description protocol (SDP, currently in ID state) and are transported by using the session announcement protocol (SAP) version 0. The following document specifies an efficient compression mechanism to transport such announcement messages of slow-speed links like PPP over modem links.

A.2 Introduction and Motivations

Today multicast or MBONE sessions on the INTERNET are announced by using the session description protocol (SDP) and session announcement protocol (SAP), version 0. The SDP defines a compact format for providing a description of the session. The SAP is the protocol used, to transmit such SDP announcement.

The SAP transmits the announcements in IP UDP datagamms. They are periodically retransmitted, aproximately between every 5 and 10 minutes, depending on the scope of the announcement (refer to [Han96a]. When trying to give computers access to such announcements using slow links, like PPP links over a modem line, the bandwidth occupied by the announcements will be very high. Each announcement transmitted will contain the SDP announcement, which is usually between 400 and 1000 bytes, plus the overhead for the IP/UDP/SAP protocols (at least 36 bytes). When sending such announcements uncompressed, it can happen that the link is unusable for other traffic for more than a second. This is too long when transfering multimedia data over the same link.

The major idea in this document is to transmit changes to the global session description database only. While doing so, we can reduce the needed bandwidth for transmitting such announcements. Because such announcements will be retransmitted after some time, the amount of saved bandwidth can be expected to be very high.

Today, only a few number of session announcements are online at a given time (less than 100). But one can expect that the number will increase significantly in the near future. Bacause of this, and because of the need to brind the mbone into the home, this compression technique was designed.

A.3 Description of the basic protocol

The basic point is that we can assume to have a point-to-point connection over the slow-speed link. Because of this we can use a client-server relationship, in contrast to the global shared database. Each side can be a client, a server or both at the same time. The client and server of one side are totally independent of each other. The server and the client of the other side keep a shared set of contexts. Each context represents exactly one session announcement.

In order to keep the occupied bandwidth low, we will only send changes for a given context. On startup, each client and server has an empty set of contexts. Each context is identified by a context identifier (context-ID). This context-ID is maintained by the server, and the client has to allocate a new context if the server asks the client to do so.

A.3.1 Context Assignment

After a connection between the server and client has been established, the server starts to assign context-IDs for each session announcement. So, whenever the server receives a session announcement of some type it serves, it will assign a context for this announcement. After doing so, he HAS TO inform the client of the assignment. This is done by transmitting an `ASSIGN` packet to the client. This packet includes the original announcement, perhaps compressed by some way, and the context-ID.

When the client receives such an `ASSIGN` packet, it allocates a new context for the given context-ID, and stores all the information about the announement in the context. After the client has stored the data in the context, it sends out the received announement. This SHOULD BE done with the destination address and portnumber supported by the server. If the context-ID was in use by another announcement, the client assumes that this announement has been invalidated by the server and frees the context.

A.3.2 Retransmitting announcements

After some time, the announcements will arrive again at the server. The server searches it's contexts to find the context-ID for this announcement. After doing so, he sends a `RETRANSMIT` packet to the client. This packet contains only the context-ID, because the client allready has all the necessarry informations about the announement.

When the client receives the `RETRANSMIT` packet, he looks into it's set of contexts to find the one with the same context-ID. After he has found it, he transmitts the announement on it's side of the network.

A.3.3 Automatic invalidation

The server maintains a timeout timer for each context. This timeout timer will be initialized every time he receives the announement which belongs to this context. When the time has been elapsed, the context will be freed by the server. No indication will be sent to the client. So it can happen that the same context-ID will be reused after some time for another announcement. Because this one is new, the server sends an `ASSIGN` packet to the client. The receiving client overrides the current context settings, so that the old entry will be removed automatically.

The client has also the option to maintain a timer for each context. The timer will be restarted every time the client receives a `RETRANSMIT` packet. When the time has been esapsed, the client assumes that the announement has been deleted and frees the context and context-ID. There are two resulting possibilites:

1. The assumption was right and the server has also deleted the context. In this case there will be no

more RETRANSMIT packets for this announement. Only new assignments (ASSIGN packet) will arrive which assigns a new announcement to the context.

2. It was wrong and the client receives a RETRANSMIT packet for the prior freed context. In this case, the client notices that he doesn't have a context for the given context-ID and HAVE TO send back a DELETED packet. After the server receives this packet, he HAS TO send back an ASSIGN packet to resynchronize with the client.

A.3.4 Error condition: handling of possibly packet loss

Let's assume that the client has a valid context with the context-ID C. Assume that the server decided to invalidate this context at some time. Because of the protocol handling descibed above, the client has no change to know this fact. Well, after some time the server assigns another announcement to the context C, but because for some reasons the ASSIGN package was lost. When the server receives the retransmitted annoouncement he assumes that the client also have an up-to-date context for this announcement. Because of this, he only sends a RETRANSMIT packet to the client.

At this point we get a serious problem, because the client receives the RETRANSMIT packet and transmits the old and wrong announcement which was stored in the given context. So the announcement received at the server will never be sended by the client; it get's lost.

To minimize this possible error, we also assign a generation-ID for each context-ID. Whenever the server assigns a new announcement to a context-ID, it HAS TO change this generation-ID, best by incrementing it by one. This generation-ID will be transmitted in every packet belonging to this context.

When the client receives a RETRANSMIT packet, it compares the generation-ID of the context with the one in the packet. If it is the same then the packet realy belongs to this announcement. Otherwise a packet has been lost and the context is out-of-date with the one in the server. So the client frees the context and context-ID, and sends a DELETED packet back to the server. Doing so informs the server that the client has no such context any more and the server HAS TO send a new ASSIGN packet to the client. Doing so results in a new assignment of the context and context-ID with the announcement. After the client has received this ASSIGN packet, both are now resynchronized.

Another possible problem can occur, when the very first ASSIGN packet for a context was lost. In this case, the client receives a RETRANSMIT packet after some time, but is not able to find the context-ID in it's list. The client HAS TO do exactly the same as described above, he has to send back a DELETED message to force the server to retansmit the assignment.

A.3.5 Broken link handling

When using point-to-point links, it can happen that the underlying link gets broken for a short time. To reduce the bandwidth needed to rebuild the cache in the client by sending lots of ASSIGN packets for every announcement, we have added a mechanism which alows both sides to keep their current cache. This is done by using the DELETED packet.

Whenever a new link will be established between two peers, each client normaly HAS TO send a global DELETED packet to the peer client and has to free each context. Each server receiving this global DELETED packet HAS TO free all its contexts. This ensures a consistent state on both sides.

However, if the link was interrupted for a short time only, each client has the oportunity not to send the global DELETED packet. This SHOULD BE done only when the client is sure that he has talked with this server directly before. The server then keep it's current set of contexts and therefore only needs to send RETRANSMIT packets, when the announement is known. If the server is not sure that he has talked with

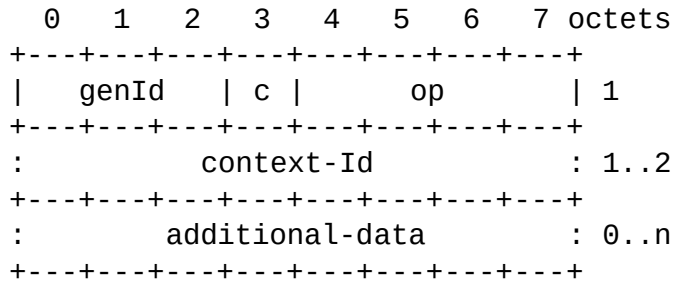
this client directly before, he has to oportunity to ignore this global DELETE packet. Doing to results in a server which has currently no assigned contexts.

The procedure described above enables a further kind of saving bandwidth. Even if the server has been crashed and the client accidently assumes that the server still knows it's context, no further overhead arrises. The server has an empty set of contexts and send ASSIGN packets for each announement. This results in overwriting the old context entries in the client without any problem. Because of the timeout timer the client HAS TO maintain, very old entries in it's context will also freed after some time automatically.

The global context-Id is a context-Id with all bits set to 1. So when using a ....

A.4 Packet Format

The following diagramm shows the general format of the cSAP packet:



The genId ist the generation-ID of the context, to which this packet belongs. This gives the client some kind of error detection in the case of packet loss due to an error. When the server assigns a new announement to a previously used context, it HAS TO change the generation-ID of the context by incrementing by 1. When this ASSIGN packet gets lost, and the client only reveives the RETRANSMIT packet, it detects an error, because the generation-ID is different. Having three bits, resulting in 8 different generation-IDs, the possibility of accidently receiving the save generation-ID is very small.

The operation code defines the type of the packet. Currently only three different packet types are defined, so 2 bits are enough to code them. But to let the room for further extensions, the operation code field was extended to 4 bits.

Today there the number of available announements is very small, less than 100 (received in Germany), so 1 octet, resulting in 256 possible contexts should be enough. But one can expect that the number will increase in the next time. Because of this, we extended the contest-ID up to 2 octets, resulting in 65536 different active announcements for a given time.

To minimize the overhead to send the context-ID, the context-ID can de transmitted with 1 or 2 octets. To let the client determine, who long the context-ID is, the C bit is used. If it is 0, then 1 octet is used for the context-ID, otherwide 2 octets.

In section A.3.4 we introduced the use global DELETE packet. Ro reduce the number of packet types, we decided to reserve a special bit combination of the context-ID. So, when all bits in the 1 or 2 octet long context-ID are set to 1, we say that this is the global context-ID. This context-ID HAS NEVER to be used for a normal context-ID. It HAS ONLY to be used for the globel DELETE packet.

Depending of the operation code in the packet, additional data can follow the context ID.

Whenever the client or server receives an packet with an unknown op-code, it SHOULD silently discard this packet. There is no kind of an error indication available.

The next sections describes the detailed format of each packet type defined.

A.4.1 ASSIGN Packet

Description

The `ASSIGN` packet is used to assign a session announcement to a given context-ID. The desciption of the announcement consists of the following fields:

- The IPv4 source and destination addresses of the announcement.
- The UDP source and destination port numbers. Together with the addresses, they give the client a hint, how to handle this packet and what to do with it. The client also needs this to retransmit the announcement on it's side of the network.
- An identifier for the session announcement protocol used for this announcement.
- An identifier for the session description protocol used. This can also be used to indicate the used compression.
- The announcement itself.

Knowing these parameters, the client is able to handle the announcement and can transmit it in it's side of the network. If he does not understand the session announcement protocol or the session description protocol, it SHOULD never transmit a packet for this announcement. No error indication HAS TO be send back to the server.

If the client knows both formats, it HAVE TO copy these fields into the context specified by the server. If the context was allready in use by another (or even the same) announement, it HAS TO BE silently overwritten (the server kows the context of the client, so he knows what he does). After the announement has been stored in the context, the client SHOULD tranmit the announcement using the right protocol on it's side of the network.

The server HAS TO change the generation-ID field for this announcement on every time he makes a new assignment for this context. The client needs this context-ID to perform some kind of packet loss checks. It is recommended that the server increments this generation-ID by one.

This packet can only be sent from the server to the client.

Packet format

```
  0   1   2   3   4   5   6   7 octets
+---+---+---+---+---+---+---+---+
|   genId   | c | 0 | 0 | 0 | 0 | 1
+---+---+---+---+---+---+---+---+
:           context-Id          : 1..2
+---+---+---+---+---+---+---+---+
:     IPv4 source address       : 4
+---+---+---+---+---+---+---+---+
:    IPv4 destination address   : 4
+---+---+---+---+---+---+---+---+
:       UDP source port         : 2
+---+---+---+---+---+---+---+---+
:     UDP destination port      : 2
+---+---+---+---+---+---+---+---+
|      sap      |      sdp      | 1
+---+---+---+---+---+---+---+---+
:       announcement data       : 0..n
+---+---+---+---+---+---+---+---+
```

GenId

Specifies the generation ID for this context. The client HAVE to store this generation ID in it's context. The server HAVE to maintain the last generation ID for this context-ID and HAVE to use a different value than the one stored. The server SHOULD increment the generation ID by 1 on each time it makes a new assignment for this context-iD.

C-bit and Context-id

The C-bit specifies, if the following context-Id is 1 or 2 octets long (see A.4).

IPv4 Source Address

Specifies the IPv4 address of the host, from which the server received this announcement.

IPv4 Destination Address

Specifies the IPv4 address of the destination address. Usually, this is a IPv4 class D address (IP multicast). The client SHOULD send the announcement to this specified address.

UDP source port

Specifies the port of the originating host.

UDP destination port

Specifies the port for the destination address. In the case of an IPv4 multicast address, this will (normally) also specify the type of the announcement, but is not restricted to. The client HAVE to send the announement to this specified port.

Sap

Specifies the protocol, which is used to transmit this announcement. The header and tralier for the given protocol is still included in the Announcement-data. The function of this field is to allow some kind of compression to the specific announcement protocol, before it will be sent to the client. Before the compressed announcement protocol can be send, it HAVE to be uncompressed, so the client must be noticed of the type of compression used. Whenever an announcement is transmitted in it's original form, it SHOULD be marked as transparent.

Currently, the following session announcement protocols are defined:

```
0000
transparent (i.e. don't care about the contents, simply use it)
```

When the receiving client does not understand the protocol used, it DON'T HAVE to transmit any outgoing packets for this context, until it has been assigned to another announcement.

Sdp

Specifies the session description protocol used. Like the Sap, the major use of this field is to indicate the kind of compression used.

Currently, the following values have been defined:

```
0000 transparent (i.e. use it as is)
0001 gzip'd
0010 cSDP (currently work-in-progress)
```

The same rules as for the Sap field applies here.

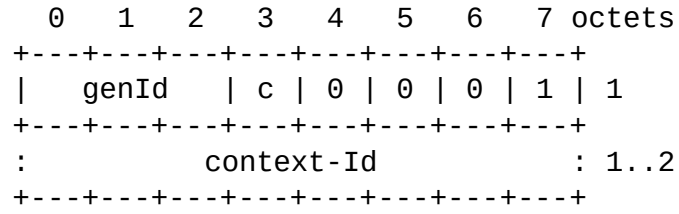
A.4.2 RETRANSMIT Packet

Description

Whenever the server receives an announcement which has allready a assigned context on the client, it HAVE TO send the `RETRANSMIT` packet. When the client receives this packet, it HAS TO use the data stored in the context and SHOULD retransmit the stored announcement to the given address. Prior to sending, it HAVE TO check if the context is up to date. This is done by comparing the received generation-ID with the one stored in the context. If they are not the same, the client DO NOT HAVE to send out the announcement. Instead, it HAVE TO send a `DELETED` packet back to the server. The same has to be done if the client don't have a context with the given context-ID.

This packet can only be sent from the server to the client.

Packet format



GenId

Specifies the generation ID for this context. If the generation ID in the context specified is different to the one received, the client HAVE TO invalidate the context and HAVE TO send a `DELETED` packet back to the server. It NEVER HAVE TO send any more announcements for this context, until it receives an `ASSIGN` packet.

C-bit and Context-id

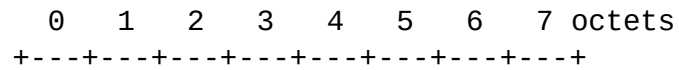
The C-bit specifies the length of the following context-Id (see A.4).

A.4.3 DELETED Packet

Description

... This packet can only be sent by a client to a server.

Packet format



```
|   genId   | c | 0 | 0 | 1 | 0 | 1
+---+---+---+---+---+---+---+---+
|          Context-Id           | 1..2
+---+---+---+---+---+---+---+---+
```

Gen-Id

Specifies the generation-ID, which has caused the client to invalidate the context. Generally, this is the same as the one received from the server in the previous `RETRANSMIT` packet.

Whenever the server receives a `DELETED` packet, it HAS TO reply with an `ASSIGN` packet to resynchronize the context with the client. If the context does no longer exists on the server, he HAS TO silently discard this packet.

C-bit and Context-id

The C-bit specifies the length of the following context-Id (see A.4).

The client can use the special encoded context-ID, known as the global context-ID to force the server to clear all its contexts. The server DON'T HAVE to send `ASSIGN` packets directly to the receipt of this packet.

A.5 Security Considerations

Sesurity considerations are not covered in any way by this draft.

Anhang B

Compressing SDP Packets

Session announcements are transmitted as SDP packets. SDP is a simple protocol, defined as ASCII text lines describing the announcement, separated by a CR LF. These lines all have the same format:

```
x=value
```

, where `x` is any letter which defines the attribute to be defined (case sensitive) and `value` is the value given to the attribute. Several attributes are defined by SDP so far. The following is a SDP packet arbitrarily catched from the INTERNET:

```
v=0
o=james 3054298051 3054298223 IN IP4 129.89.142.50
s=FreeBSD Lounge
i=Channel to discuss FreeBSD related issues.  Please keep video bandwidth be
e=Jim Lowe <james@cs.uwm.edu>
p=Jim Lowe (414) 229-6634
c=IN IP4 224.2.100.100/127
t=0 0
a=tool:sdr v2.2a23
a=type:test
m=audio 16400 RTP/AVP 0
c=IN IP4 224.2.100.100/127
a=ptime:40
m=video 49200 RTP/AVP 31
c=IN IP4 224.2.100.102/127
m=whiteboard 32800 udp wb
c=IN IP4 224.2.100.101/127
a=orient:portrait
```

This encoding technique is quite simple to implement, because a parser for recognizing a SDP packet is very simple. But the format is very inefficient and waste a lot of bandwidth. This is a problem on slow-speed links like PPP links on normal telephone lines. Because of this the SDP packet should be compressed. The SAP protocol which is normally used on the INTERNET to carry SDP packets, defines a way to simply compress the SDP packet by using the gzip algorithm. The following document describes a better technique, which utilizes a better compression ratio then simply using the gzip algorithm. The resulting protocol will be called compressed SDP (CSDP).

B.1 CSDP

The SDP packet will be compressed sequential on a per attribute basis. Because the order of the attributes is specified in the SDP draft, it is not necessary to identify the attributes explicit by an ID. They can simply be identified by the order. If an optional attribute is not present in an SDP packet, it will be coded as not present (see below).

There are two groups of attributes which can be repeated several times, the *Time Description* and the *Media Description*. The first attribute of each group is mandatory, if the group is present. So these groups will also be repeated in the compressed SDP packet. The last (or even missing) group will be indicated by the first attribute. If it is not present in the compressed packet (violation of the SDP specification), then this group will no longer repeated.

B.2 The Presence Bit

Because there are some attributes which are optional, and others which can be repeated, every attribute will be preceded by an `<presence-bit>`. If this bit is equal to 1, then the attribute is present, if the bit is 0, it is not. If it is present (i.e. the bit was 1) then the attribute will also be repeated. For example, to specify an attribute once, the following sequence is produced:

```
+-+-+-+-+-+ ... +-+-+-+
|1| the attribute   |0|
+-+-+-+-+ ... +-+-+-+-+
```

If the attribute occurs two times, then the following sequence will be generated:

```
+-+-+-+-+-+ ... +-+-+-+-+-+-+-+ ... +-+-+-+
|1| the attribute   |1| the attribute   |0|
+-+-+-+-+ ... +-+-+-+-+-+-+-+ ... +-+-+-+-+
```

According to the SDP specification, there are some fields which are mandatory and have to be specified exactly once. So the usage of the `<presence-bit>` is not needed for such attributes. But to simplify the compression / decompression, and to make the algorithm more clear, every attribute will be encapsulated by the `<presence-bit>`. This makes it also possible to extend the SDP spec, so that such attribute can be repeated, of skipped. The overhead of two bits (1 and 0) can be accepted.

There is one exception regarding the usage of the `<presence-bit>`. The very first attribute, the version of the SDP packet, will NOT be encapsulated. This is to let us define new versions of SDP or this compression protocol, which uses a fully different encapsulation. Therefore, we need the version information in plain.

B.3 Text Compression

Several attributes include plain ASCII text designated to be read by humans. Such fields cannot be compressed like an IP address. So they these fields will be compressed by the gzip algorithm.

Tests have shown that, when the compressed header information for each attribute (e.g. the `<presence-bit>` for the i= attribute) were are directly followed by the text value (the textual description), the gzip algorithm works very poor. Because of this, the compressed SDP packet will be splited in four sections.

```
+----------------------+
| version              |
+----------------------+
| header-length        |
+----------------------+
| header               |
+----------------------+
| compressed text data |
+----------------------+
```

The **version** section simply defines the SDP and CSDP version. The **header-length** section defines the number of bytes occupied by the **version**, the **header-length** and the **header**. The **header** section contains all the compressed SDP data, excluding the compressed text data. The last section contains the sequential appended text, which is compressed by gzip (the whole collected text-buffer will be compressed, not each attribute separately!).

The last section begins on a byte boundary. Because the compression is bit oriented, the last octet of the **header** section may be padded to the byte boundary. This can be done by appending 0 to 7 `<final-bits>`.

Theoretically, the length field can be ommited, because the end of the `header` can be detected automatically. The last *Media Description* group defines the end of the SDP packet, so the last byte is the one with the `0 presence-bit` for the group. But during trying to implement the CSDP decompressing algorithm, we found out that it is a difficult to handle the text sections. Without the length indication, the parser has to parse the packet twice, resulting in a poor performance. So we decided to include the length indication.

B.4 CSDP Specification

The following sections specify the format of the compressed version of every attribute defined by SDP. The order given is the same as in the SDP draft.

For every length filed defined below, a value with all bits set to `1` is reserved to indicate a length extension. This allows us to efficient compress attributes with a usual length, and makes it still possible to encode larger attributes. The format and usage of the extension mechanism has not been defined up to now.

Each attribute but the version one, is preceded by a `<presence-bit>` as defined above. It is indicated by a **P**-bit.

B.4.1 Version (v=)

This field is mandatory for all SDP packets, and it is the only one, which is NOT preceded by the presence-bit!

- **Uncompressed format**
  `v=<version>`
- **Compressed format**

```
 0 1
+-+-+
```

```
|ver|
+-+-+
```

`Ver` is the binary encoded version field. Currently, the value `00` is defined for the actual version `0`, and the description given here applies only to this version!

B.4.2 Header-Length

This is not a SDP attribute, but is necessary for decoding the compressed SDP packet. This field indicates the length of the header, which preceeds the compressed text section.

- **Uncompressed format**
  not available
- **Compressed format**

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|<header-length>|
+-+-+-+-+-+-+-+-+
```

`<headler-length>` is the binary encoded length of the header, given int octets. The value includes the overall header, including the `<version>` and `<header-length>` field, so it an offset, to the point where the compressed text section begins.

B.4.3 Origin (o=)

The origin field specifies the origin of the announcement.

- **Uncompressed format**
  `o=<username> <session-id> <version> <net-type> <addr-type> <addr>`
- **Compressed format**

```
                                    3 3                   3 3
 0   0 1 2 3   0 1 2 3 4          0 1   0 1 2 3          0 1
+-+ +-+-+-+-+ +-+-+-+-+ ... +-+-+-+-+ +-+-+-+-+ ... +-+-+-+
|P| | u-len | | 32 bit <session-id> | | 32 bit <version>  |
+-+ +-+-+-+-+ +-+-+-+ ... +-+-+-+-+-+ +-+-+-+ ... +-+-+-+-+
 0 1 2 3 4 5
+-+-+-+-+-+-+ ... +-+-+-+-+-+-+-+-+
| <net-type> <addr-type> <addr> |
+-+-+-+-+-+-+ ... +-+-+-+-+-+-+-+-+
```

`u-len` specifies the length of the username in octets and is 6 bits long. Because the the `<username>` is raw ASCII, is will be appended to the text section to be compressed. Because in practice, the `<session-id>` and `<version>` fields seems to be derived from an 32 bit integer, it will be represented in binary version. The address-triple will be encoded as defined in B.4.16.

B.4.4 Session Name (s=)

This field gives the session a reasonable name.

- **Uncompressed format**
  `s=<session-name>`
- **Compressed format**

```
 0   0 1 2 3 4 5
+-+ +-+-+-+-+-+-+
|P| | 6 bit len |
+-+ +-+-+-+-+-+-+
```

The 6 bit `len` field specified the length of the session name in octets. The `<session-name>` will be appended to the text section to be compressed.

B.4.5 Session Description (i=) or Media Title

This optional attribute describes the session.

- **Uncompressed format**
  `i=<session-description>`
- **Compressed format**

```
 0  0 1 2 3 4 5 6 7 8 9
+-+ +-+-+-+-+-+-+-+-+-+-+
|P| | 10 bit length     |
+-+ +-+-+-+-+-+-+-+-+-+-+
```

The 10 bit `length` field specifies the length of the description in octets. It is possible to specify a length of up to 1023 octets, which is enough because of the length restriction of an SDP packet to 1024 octets (well, the packet „should“ not be larger than 1024). The `<session-description>` will be appended to the text section to be compressed.

B.4.6 URI (u=)

This optional attribute specifies an URI for further informations about the session.

- **Uncompressed format**
  `u=<uri>`
- **Compressed format**

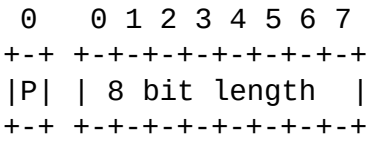
```
 0   0 1 2 3 4 5 6 7
+-+ +-+-+-+-+-+-+-+-+
|P| | 8 bit length  |
+-+ +-+-+-+-+-+-+-+-+
```

The 8 bit `length` field specifies the length of the `<uri>` in octets. The `<uri>` field will be appended to the text section to be compressed.

B.4.7 E-Mail (e=)

This optional attribute defines an e-mail address for a contact person regarding the session. This field can be repeated.

- **Uncompressed format**
  `e=<e-mail>`
- **Compressed format**

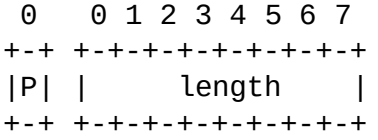


The 8 bit `length` field specifies the length of the `<e-mail>` address in octets. The `<e-mail>` address will be appended to the text section to be compressed.

B.4.8 Phone Number (p=)

This attribute defines a phone number for a contact person regarding the session. This field can be repeated and is optional.

- **Uncompressed format**
  `p=<phone-number>`
- **Compressed format**

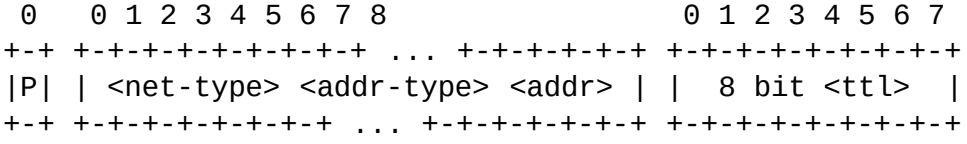


The 8 bit `length` field specifies the length of the `<phone-number>` in octets. The `<phone-number>` will be appended to the text section to be compressed.

B.4.9 Connection Information (c=)

This attribute defines where the session is being transmitted (i.e. on which network address). It is optional and can be repeated. This filed can also be used in the *Media Description* group.

- **Uncompressed format**
  `c=<net-type> <addr-type> <addr>[/<ttl>]`
- **Compressed format**
  Format for an IN IP4 address:

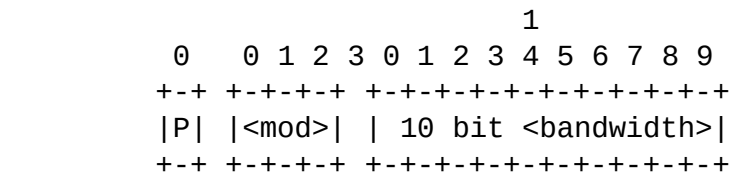


The address-triple (<net-type>, <addr-type> and <addr>) will be encoded as defined in B.4.16. The 8 bit <ttl> field is the binary encoded <ttl>.

B.4.10 Bandwidth (b=)

This optional attribute specifies the bandwidth consumed by the session.

- **Uncompressed format**
  b=<modifier>:<bandwidth-value>
- **Compressed format**



The 3 bit <mod> is encoded as following:

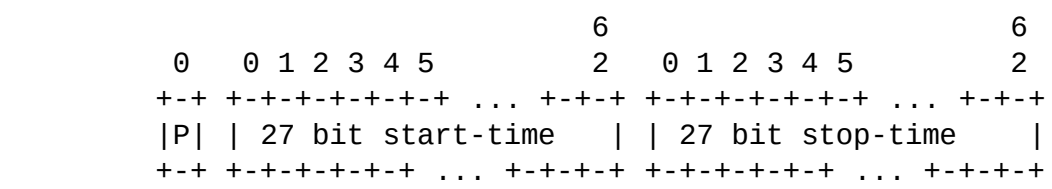
```
000 CT
001 AS
111 (reserved for extension)
```

The <bandwidth> will be encoded as 10 bit binary value.

B.4.11 Times (t=)

This attributes defines the time, when the session will be active. It also starts the *Time Description* group. If it is present, then the group definition (t=, r= and z=) will come next. Otherwise these other attributes will be skipped.

- **Uncompressed format**
  t=<start-time> <stop-time>
- **Compressed format**



The 27 bit start-time is the binary encoded value of the <start-time>, given in minutes. The 27 bit stop-time is the binary encoded value of the <stop-time>, given in minutes.

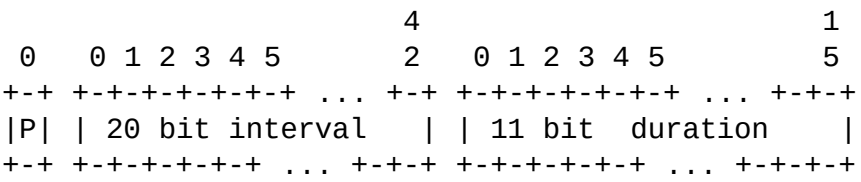
Because these attributes are represented as NTP values, there should be no limitation by using the 27 bit binary encoding, given in minutes.

Well it can happen that the values have to be rounded. In fact, this does not affect the real meaning. Anyway, we define that the <start-time> has to be rounded down, and the <stop-time> has to be rounded up.

B.4.12 Repeat Interval (r=)

This optional attribute specifies, when the session will be repeated. Note that this field can only occur within a *Time Description* group.

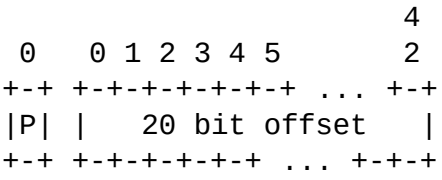
- **Uncompressed format**
  `r=<repeat-interval> <active-duration> <list-of-offsets>`
- **Compressed format**



The 20 bit `interval` is the binary encoded value of `<repeat-interval>`, given in minutes. This largest value is larger than 1 year, so 20 bit should be sufficient.

The 11 bit `duration` is the binary encoded value of `<active-duration>`, given in minutes. The largest value is about 34 hours (longer than one day), which should be sufficient.

The variable length `list-of-offsets` is encoded like an attribute, i.e. it is preceded by the presence-bit. This field is repeated until a presence-bit of `0` is received.



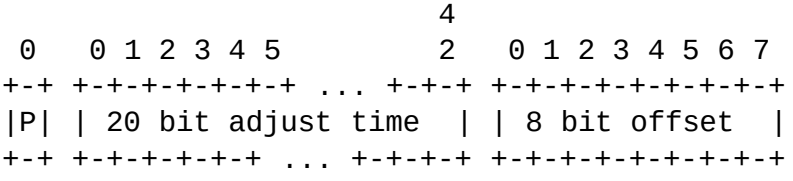
The 20 bit `offset` is the binary encoded value of one `<list-of-offsets>` element, given in minutes. The largest value is more than a year, which should be sufficient.

It can happen that the values were truncated, which doesn't have any real meaning. Anyway, the `<repeat-interval>` should be rounded down, the `<active-duration>` up and the `<offset>` down.

B.4.13 Timezone Adjust (z=)

This optional attribute specifies, some kind of timezone adjustments. Note that this field can only occur within a *Time Description* group.

- **Uncompressed format**
  `z={<adjust-time> <offset>} ...`
- **Compressed format**
  Because this attribute consists of an variable length list of the two fields, it will be repeated until the presence-bit is `0`.



The 20 bit `adjust time` is the binary encoded value of the `<adjust-time>` field, given in minutes. The 8 bit `offset` is the binary encoded value of the `<offset>` field, given in minutes! So we can have an adjustment of up to 4 hours; should be far enough!

It can happen that the values were truncated, which doesn't have any real meaning. Anyway, the `<adjust-time>` should be rounded down and the `<offset>` up.

B.4.14 Encryption Key (k=)

- **Uncompressed format**

- **Compressed format**

```
(currently not defined)
```

B.4.15 Session Attribute (a=)

This attributes defines additional attributes.

- **Uncompressed format**
  `a=<flag>`
  `a=<attrib>:<value>`

- **Compressed format**
  Because the values for this attribute can vary very much and will extend in the future, it makes no sense to define any special mappings. Instead, it will be encoded like text.

```
 0   0 1 2 3 4 5
+-+ +-+-+-+-+-+-+
|P| |   length  |
+-+ +-+-+-+-+-+-+
```

The 6 bit `length` field specifies the length of the remaining attribute definition, either the `<flag>` or the `<attrib>:<value>`. These fields will be appended to the text section to be compressed.

B.4.16 Media (m=)

- **Uncompressed format**
  `m=<media> <port> <transport> <fmt-list>`

- **Example**
  m=video 60450 RTP/AVP 31

- **Compressed format**

```
                                    1
 0   0 1 2   0 1                    5    0 1 2
+-+ +-+-+-+ +-+-+-+-+-+-+ ... +-+-+ +-+-+-+
|P| |media| | 16-bit port          | |trans|
+-+ +-+-+-+ +-+-+-+-+-+ ... +-+-+-+ +-+-+-+
```

The <media> field will be encoded as an 3 bit binary value. The following values have been defined:

```
000 audio
001 video
010 whiteboard
011 html
100 text
111 (reserved for extensions)
```

The 16 bit port field is the binary encoded value of <port>. NR IS MISSING!!! The next field is the transfer format, and will be encoded as 3 bit value. The following values have been defined:

```
000 RTP/AVP
001 VAT
010 UDP
111 (reserved for extensions)
```

Because the <fmt-list> is a variable length list, it will be encoded as present-bit, value. If the present-bit is 0, there are no more fmt entries. Otherwise, a media specific 5 bit encoded fmt description will follow. For the media type audio, the following values are defined:

```
00001 0 (pcm)
...
00010 pcm
00011 gsm
00100 dvi4
0
0001 pcm
0010 gsm
11111 (reserved for extension)
```

For the media type video, the following values are defined:

```
00000 h261, 31
00001 ???, 96
11111 (reserved for extension)
```

For the media type whiteboard, the following values are defined:

```
00000 none
00001 wb
11111 (reserved for extension)
```

For the media type html, the following values are defined:

```
00000 Mosaic
00001 Netscape
11111 (reserved for extension)
```

For the media type text, the following values are defined:

```
00000 nt
11111 (reserved for extension)
```

B.4.17 Combined Compression of <net-type> <addr-type> <addr>

Network-addresses are specified as a triple in SDP. In contrast of coding each part separately, we will combine these three field into one logical unit.

The compressed version consists of a fix 4 bit header which defines the combined type of the address format. The following formats has been defined up to now:

- IN IP4 <addr>

```
                        1               3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3            5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+
|0 0 0 0|  32 bit binary IP4 address  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+-+
```

- IN IP6 <addr>

```
                        1               5
 0 1 2 3 4 5 6 7 8 9 0 1 2 3            1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+
|0 0 0 1|  48 bit binary IP6 address  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+-+
```

- IN DNS <addr> (currently not defined in SDP)

```
                        1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+-+-+-+-+-+-+
|0 0 1 0| length    | length octets of DNS-address  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+-+-+-+-+-+-+
```

  Length (6 bits) specifies the length of the DNS-address in octets (i.e. 8 bit units). Addresses longer than 64 octets cannot be represented by this format.

- The following header is reserved for further extensions.

```
 0 1 2 3
+-+-+-+-+
|1 1 1 1|
+-+-+-+-+
```

B.5 Comparison of SDP and CSDP

The following is a comparison of the SDP, the gziped SDP and the CSDP sizes. The SDP packets has been grabbed from the INTERNET during several hours, so they represent a set of usual transmitted SDP packets. The packets are ordered on the original size:

```
Nr SDP/bytes gzip'd ratio/\% CSDP ratio/\%

19 178 157 11.7 86 51.6
04 274 210 23.3 158 42.3
20 336 245 27.0 194 42.2
16 356 260 26.9 195 45.2
```

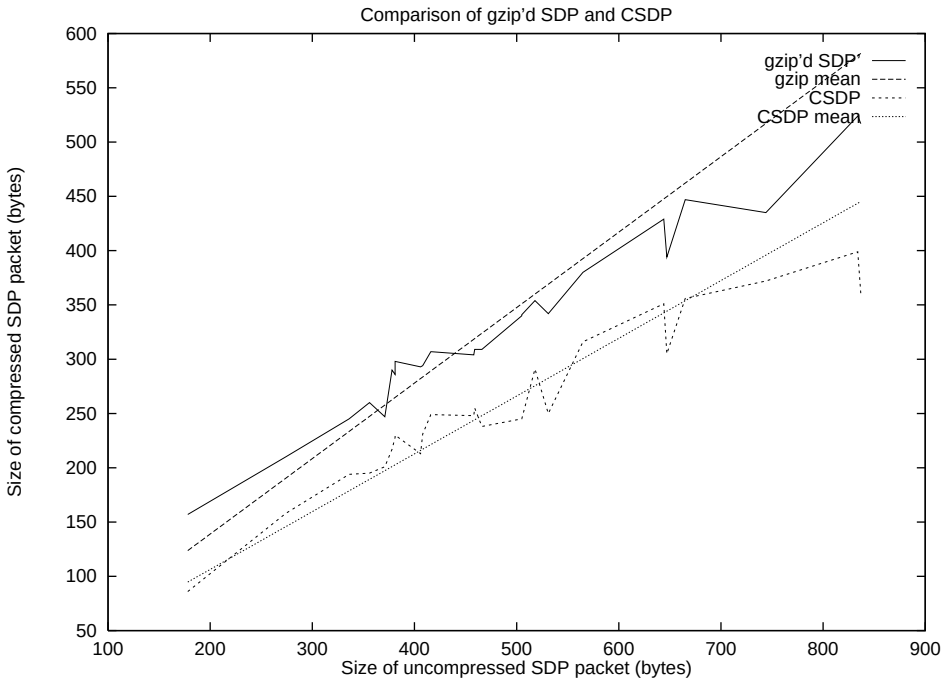
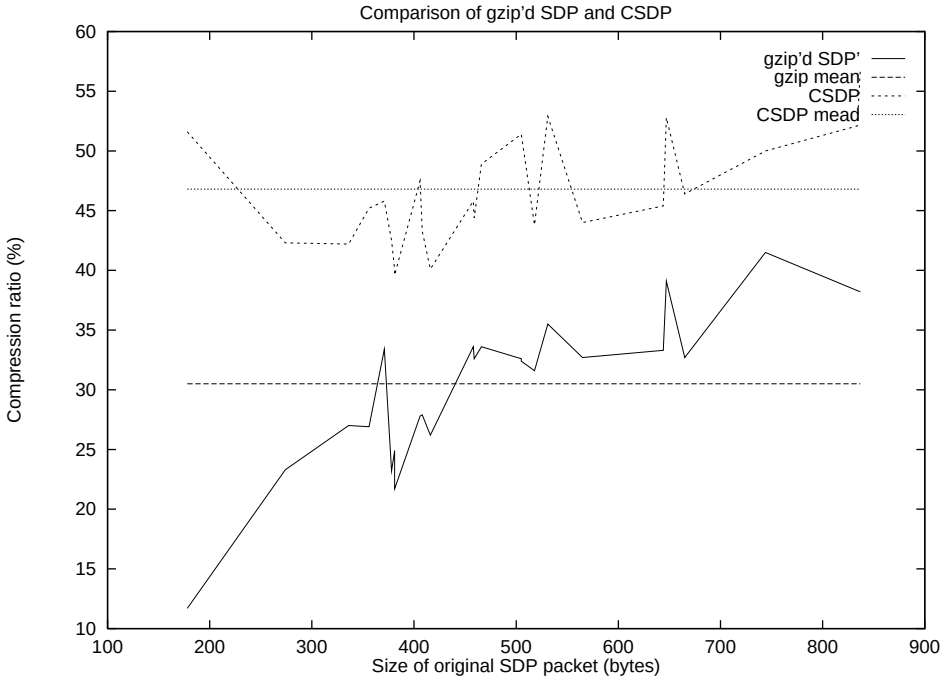


Abbildung B.1: Comparing gzip and CSDP

```
09 371 247 33.4 201 45.8
18 378 290 23.2 217 42.5
01 381 286 24.9 228 40.1
14 381 298 21.7 230 39.6
05 406 293 27.8 213 47.5
03 408 294 27.9 231 43.3
24 416 307 26.2 249 40.1
07 458 304 33.6 248 45.8
06 459 309 32.6 255 44.4
21 466 309 33.6 238 48.9
23 505 340 32.6 245 51.4
15 505 341 32.4 245 51.4
17 505 341 32.4 245 51.4
11 518 354 31.6 291 43.8
22 531 342 35.5 250 52.9
10 565 380 32.7 316 44.0
13 644 429 33.3 351 45.4
08 647 394 39.1 305 52.8
12 665 447 32.7 356 46.4
02 744 435 41.5 372 50.0
25 834 524 38.3 399 52.1
00 837 517 38.2 360 56.9
```

The following is a graph, which compares the size of the compressed SDP packet, once compressed by gzip and once with the method described here: You can see very clear that the CSDP compression is much more effective than simply applying gzip to the SDP packet, as expexected by the author. BTW, it is interesting that the two plots have nearly the same shape, but they the CSDP plot is lowered by an offset compared to gzip.



Another interesting comparison is the compression ratio using the two different compression tachniques: Once again you can see that the shape is nearly the same. But the major result is, that the CSDP method has a compression ratio of approximately 40% to 55 %, compared to 15% to 40% when using gzip. The arithmetic mean of the gziped SDP packet is 30.5%. It is 46.8% when using the CSDP compression.

B.6 Summary

As shown in the last sections, the here explained method for compressing `SDP` packets is much more efficient than simply using `gzip`. The average gain compared to gzip is about 15% (46.8% - 30.5%), which seems to be a notable value.

But there are also some kind of pitfalls. First, the compression algorithm shown here strictly relies on the order of the `SDP` lines. While implementing the algorithm explained above, I realized that there are still a lot of `SDP` packets on the net, which do not strictly care about the order, even `sdr`. So prior to compression, a reordering of the fields was necessary. Doing so results in correctly ordered `SDP` packets, so that there was no packet, which couldn't be compressed.

Of course, extending the `SDP` (e.g. adding new tags) will cause a problem. But it should be easy to adapt such changes to CSDP as well.

C.1.1 server.c

Anhang C

Source-Code der Testimplementierung

C.1 Das TTY-Interface, Simuliertes Netzwerkinterface

```
/*
 *      file:           server.c
 *      written by:     Christian Zahl
 *      description:    Server for PPP link
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <sys/socket.h>

#include <fcntl.h>
#include <sys/stat.h>
#include <termio.h>

/*----- user includefiles --------------------------------------------------*/
#include "misc.h"
#include "hdlc.h"
#include "ppp.h"
#include "ipUdpRtpCompr.h"
#include "filehandler.h"
#include "rtp.h"

/*----- defines ------------------------------------------------------------*/
#define SERVER_EXIT             2
#define DEF_TTL                 127

/*----- type definitions ---------------------------------------------------*/
typedef struct {
        u32bit                  srcAddr;
        u32bit                  dstAddr;
        u16bit                  srcPort;
        u16bit                  dstPort;
        u16bit                  id;
        u8bit                   ttl;
        int                     s;
} t_Cxt;

/*----- local functions ----------------------------------------------------*/
void RtpOutCb (u8bit *frame, int len, int type);

/*----- global functions ---------------------------------------------------*/
        u32bit                  srcAddr;
        u16bit                  srcPort;
        u32bit                  dstAddr;
        u16bit                  dstPort;

/*----- local variables ----------------------------------------------------*/
static  int                     linkInFd;
static  int                     linkOutFd;
static  int                     udpOutSocket;
static  int                     dstPt;                  /* dst RTP payload-type */
static  t_Cxt                   cxtList[10];
```

```
#ifdef _AIX
 #define WITH_LOG               0
 #define WITH_RCV_LOG           0
 #define DEBUG(xxx)             xxx
#else
 #define DEBUG(xxx)
#endif

#if WITH_LOG
 static FILE                    *log = NULL;
#endif
#if WITH_RCV_LOG
 static time_t                  t0;
 static FILE                    *rcvLog = NULL;
 static time_t                  lastRcv;
 static int                     nRcv;
#endif

/*----- global variables -----------------------------------------------------*/

/*--------------------------------------------------------------------------*/
void TTYIn (int fd, int mode, void *date)
{
        char                    buf[100];
        int                     n;
        int                     i;

        n = read (fd, buf, sizeof (buf));
        for (i=0; i<n; i++)
                if (buf[i] == SERVER_EXIT)
                        FileHandlerStop ();
                else {
                        if (buf[i] == '\n')
                                write (linkOutFd, "\r", 1);
fprintf (stderr, "[1]");
                        write (linkOutFd, buf+i, 1);
                }
fflush (stderr);
} /* TTYIn */
/*--------------------------------------------------------------------------*/
void LinkInCb (int fd, int mode, void *data)
/*
 * Callback will be called when there are data available from the link
 */
{
        u8bit                   buf[512];
        int                     n;
        int                     i;

        n = read (fd, buf, sizeof (buf));
        if (n == -1 && errno == EINTR)
                return;
        if (n == 0) {                                   /* EOF condition */
                FileHandlerStop ();
                return;
        }
        for (i=0; i<n; i++)
                if (buf[i] == SERVER_EXIT) {
                        FileHandlerStop ();
                        return;
                }
#ifdef TEST
        write (1, buf, n);
#else
        HDLCReceive (buf, n);
#endif
} /* LinkInCb */
/*--------------------------------------------------------------------------*/
void LinkOutCb (u8bit *buf, int len)
{
#ifdef LOOPBACK_TEST
        HDLCReceive (buf, len);
#else
        write (linkOutFd, buf, len);
#endif
} /* LinkOutCb */
/*--------------------------------------------------------------------------*/
void LinkInit  ()
{
        FileHandlerCreate (linkInFd, FILEHANDLER_READ, LinkInCb, NULL);
} /* LinkInit */
/*--------------------------------------------------------------------------*/
void LinkFini ()
{
        FileHandlerStop ();
} /* LinkFini */
/*--------------------------------------------------------------------------*/
void IPInit ()
{
        PPPRegister (PPP_PROT_FULL_HEADERS, RtpOutCb, TYPE_FULL_HEADERS);
        PPPRegister (PPP_PROT_COMPRESSED_UDP, RtpOutCb, TYPE_COMPRESSED_UDP);
        PPPRegister (PPP_PROT_COMPRESSED_RTP, RtpOutCb, TYPE_COMPRESSED_RTP);
        PPPRegister (PPP_PROT_CONTEXT_STATE, RtpOutCb, TYPE_CONTEXT_STATE);
#if WITH_LOG
        {
                char                    fn[100];
                int                     now;
                now = time (NULL);
```

```
                sprintf (fn, "log.%d.rtp", now);
                log = fopen (fn, "wb");
        }
#endif
} /* IPInit */
/*---------------------------------------------------------------------------*/
void IPFini ()
{
#if WITH_LOG
        fclose (log);
#endif
} /* IPFini */
/*---------------------------------------------------------------------------*/
static int SendRtpPacket (t_Cxt *cxt, u8bit *rtpBuf, int len)
/*
 * Generate a new IP/UDP/RTP packet and send it over the PPP link.
 */
{
        u8bit                   buf[1500];
        u8bit                   out[1500];
        struct ip               *ip = (struct ip*) buf;
        struct udphdr           *udp = (struct udphdr*) (buf +20);
        u8bit                   *rtp = buf +20 +8;
        int                     type;
        int                     outLen;

/*** generate a new IP header ***/
        ip->ip_v = 4;                                           /* IP v4 */
        ip->ip_hl = 20 /4;                                      /* fix 20 byte hdr */
        ip->ip_tos = 0;                                         /* no TOS */
        ip->ip_len = 20 +8 +len;                                /* unimportant, because of compression */
        ip->ip_id = cxt->id++;
        ip->ip_off = 0;                                         /* no fragment, so no offset */
        ip->ip_ttl = cxt->ttl;                                  /* copy TTL */
        ip->ip_p = 17;                                          /* set to UDP */
        ip->ip_sum = 0;
        ip->ip_src.s_addr = cxt->srcAddr;
        ip->ip_dst.s_addr = cxt->dstAddr;                       /* dst mcast addr */
/*** generate the UDP header ***/
        udp->uh_sport = cxt->srcPort;
        udp->uh_dport = cxt->dstPort;
        udp->uh_ulen = len +8;
        udp->uh_sum = 0;                                        /* no checksum */
/*** generate the RTP header + data ***/
        memcpy (rtp, rtpBuf, len);
/*** compress the generated packet ***/
        len += 20 +8;
        outLen = IpUdpRtpCompress (buf, len, out, &type);
        if (outLen <= 0)
                return 0;
/*** send it over the link ***/
        if (type == TYPE_FULL_HEADERS)
                PPPSend (out, outLen, PPP_PROT_FULL_HEADERS);
        else if (type == TYPE_COMPRESSED_UDP)
                PPPSend (out, outLen, PPP_PROT_COMPRESSED_UDP);
        else if (type == TYPE_COMPRESSED_RTP)
                PPPSend (out, outLen, PPP_PROT_COMPRESSED_RTP);
        else {
                fprintf (stderr, "?");
                return 0;
        }
        return outLen;
} /* SendRtpPacket */
/*---------------------------------------------------------------------------*/
void RtpInCb (int fd, int mode, t_Cxt *cxt)
/*
 * This callback is registered for every local RTP UDP socket. It receives the
 * RTP packet, and sends it to the RTP translator. If the translator has
 * something to transmit, we send the RTP packet over the link.
 */
{
        u8bit                   in[1500];
        u8bit                   out[1500];
        int                     len;
        struct sockaddr_in      sin;
        int                     sinLen = sizeof (sin);
        int                     outLen;

        len = recvfrom (fd, in, sizeof (in), 0, (struct sockaddr*)&sin, &sinLen);
        if (len <= 0)
                return;
#if WITH_RCV_LOG
        {
                time_t                  t;
                time (&t);
                if (t != lastRcv) {
                        if (lastRcv)
                                for (; lastRcv<t; lastRcv++) {
                                        fprintf (rcvLog, "%d %9.6f\n", lastRcv-t0, 8.0*nRcv/1000.0);
                                        nRcv = 0;
                                }
                        lastRcv = t;
                        nRcv = 0;
                }
                nRcv += len + 20 + 8;
        }
#endif
#ifdef _AIX
        if (sin.sin_addr.s_addr == 0xC06F6F03)
```

```
                return;
#endif
        outLen = RtpTranslator (NULL, in, len, out);
        if (outLen <= 0)
                return;
        outLen = SendRtpPacket (cxt, out, outLen);
#if WITH_LOG
        len += 20 +8;           /* + IP + UDP header */
        fprintf (log, "%d\t%d\t%d\n", len, outLen, outLen *100/len);
#endif
} /* RtpInCb */
/*--------------------------------------------------------------------------*/
void RtpOutCb (u8bit *frame, int len, int type)
/*
 * CB, which receives the compressed IP/UDP/RTP frame from the PPP link and
 * sends it into the local network.
 */
{
        t_Cxt                   *cxt;
        u8bit                   buf[1500];
        struct sockaddr_in      sin;
        struct ip               *ip = (struct ip*)buf;
        struct udphdr           *udp;
        struct rtp              *rtp;
        int                     rtpLen;
static  u8bit lastRtpBuf[1500];
static  struct rtp* lastRtp = (struct rtp*)lastRtpBuf;
static int lastSeq = 0;
        int s;

        len = IpUdpRtpDecompress (frame, len, type, buf);
/*** frame sucsessfully decompressed, so send it on the way ***/
        if (len > 0) {
                udp = (struct udphdr*)IP_DATA(ip);
                rtp = (struct rtp*)UDP_DATA(udp);
                rtpLen = len - IP_HDR_BYTES (ip) - UDP_HDR_BYTES (udp);
/*** search the context ***/
                cxt = &cxtList[0];
/*** generate the output packet ***/
                sin.sin_family = AF_INET;
                sin.sin_port = udp->uh_dport;
                sin.sin_addr.s_addr = ip->ip_dst.s_addr;
#ifdef _AIX
                if (rtp->m)
                        fprintf (stderr, "M");
                else if (rtp->seq-lastRtp->seq == 1) {
                        ;
                } else {
                        fprintf (stderr, "%d:", rtp->seq-lastRtp->seq);
                        s = (rtp->timestamp -lastRtp->timestamp) / (rtp->seq -lastRtp->seq);
                        if (rtp->seq-lastSeq < 10) {
                                while (++lastRtp->seq < rtp->seq) {
                                        lastRtp->timestamp += s;
                                        sendto (cxt->s, (void*)lastRtp, rtpLen, 0, (struct sockaddr*)&sin, sizeof (sin));
                                }
                        }
                }
                memcpy (lastRtp, rtp, rtpLen);
                lastSeq = rtp->seq;
#endif
                sendto (cxt->s, (void*)rtp, rtpLen, 0, (struct sockaddr*)&sin, sizeof (sin));
        } else if (len < -2) {
                DEBUG (fprintf (stderr, "c");)
                PPPSend (buf, -len, PPP_PROT_CONTEXT_STATE);
        }
} /* RtpOutCb */
/*--------------------------------------------------------------------------*/
RTPInit (u32bit srcAddr, u16bit srcPort, u32bit dstAddr, u16bit dstPort)
/*
 * Initializes the RTP proxy server. This is done in a constant manner here
 * and should be done by some other kind of signalling.
 */
{
        int                     on = 1;
        int                     s;
        struct sockaddr_in      sin;
        struct ip_mreq          mr;
        int                     ttl = DEF_TTL;
        int                     off = 0;

        s = socket (AF_INET, SOCK_DGRAM, 0);
        if (s < 0) {
#ifdef _AIX
                perror ("RTPInit: socket");
#endif
                return;
        }
        setsockopt (s, SOL_SOCKET, SO_REUSEADDR, (void*)&on, sizeof (on));
        memset (&sin, '\0', sizeof (sin));
        sin.sin_family = AF_INET;
        sin.sin_port = htons (srcPort);
        sin.sin_addr.s_addr = htonl (srcAddr);
        if (bind (s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
                sin.sin_addr.s_addr = INADDR_ANY;
                bind (s, (struct sockaddr*)&sin, sizeof(sin));
        }
        if (IN_CLASSD (srcAddr)) {
                mr.imr_multiaddr.s_addr = htonl (srcAddr);
                mr.imr_interface.s_addr = INADDR_ANY;
```

```
                setsockopt (s, IPPROTO_IP, IP_ADD_MEMBERSHIP, (void*)&mr, sizeof (mr));
                setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, (char *)&ttl, sizeof(ttl));
                setsockopt(s, IPPROTO_IP, IP_MULTICAST_LOOP, (char *)&off, sizeof (off));
        }
/*** update the context ***/
        cxtList[0].s = s;
        cxtList[0].srcAddr = srcAddr;
        cxtList[0].srcPort = srcPort;
        cxtList[0].dstAddr = dstAddr;
        cxtList[0].dstPort = dstPort;
/*** install the filehandler ***/
        FileHandlerCreate (s, FILEHANDLER_READ, RtpInCb, &cxtList[0]);
        udpOutSocket = s;
        RtpTranslatorInit ();
#if WITH_RCV_LOG
        {
                char                    fn[100];
                int                     now;
                now = time (NULL);
                t0 = now;
                sprintf (fn, "log.%d.rtp.rcv", now);
                rcvLog = fopen (fn, "wb");
        }
#endif
} /* RTPInit */
/*-------------------------------------------------------------------------*/
RTPFini ()
{
        close (cxtList[0].s);
        RtpTranslatorFini ();
#if WITH_RCV_LOG
        fclose (rcvLog);
#endif
} /* RTPFini */
/*-------------------------------------------------------------------------*/
ServerMainLoop (int _linkInFd, int _linkOutFd)
{
        u32bit                  accm;

        FileHandlerInit ();
/*** initialize the physical layer ***/
        linkInFd = _linkInFd;
        linkOutFd = _linkOutFd;
        LinkInit ();
/*** initialize the HDLC layer ***/
        accm = 0;
        accm |= (0x01 << SERVER_EXIT);                          /* special exit key */
        accm |= (0x01 << 0x11);                                 /* XON */
        accm |= (0x01 << 0x13);                                 /* XOFF */
        HDLCInit (LinkOutCb);
        HDLCSetSndAccm (accm);
        HDLCSetRcvAccm (accm);
        HDLCSetAddrCtrlCompr (1);
/*** initialize PPP layer ***/
        PPPInit ();
/*** initialize IP layer ***/
        IPInit ();
        RTPInit (srcAddr, srcPort, dstAddr, dstPort);
/*** enter the server's mainloop ***/
#ifdef _AIX
        FileHandlerCreate (0, FILEHANDLER_READ, TTYIn, NULL);
#endif
        FileHandler ();
/*** teminate the whole stack ***/
        RTPFini ();
        IPFini ();
        PPPFini ();
        HDLCFini ();
        LinkFini ();
        FileHandlerFini ();
} /* ServerMainLoop */
/*-------------------------------------------------------------------------*/
#ifndef _AIX
main (int argc, char *argv[])
{
        struct termio           savedTermio;
        struct termio           termio;
        int                     i;

        if (argc >= 2) {
                for (i=0; argv[1][i]; i++)
                        if (argv[1][i] == '/') {
                                argv[1][i] = '\0';
                                i++;
                                break;
                        }
                srcAddr = inet_addr (argv[1]);
                srcPort = atoi (argv[1]+i);
        } else {
                srcAddr = inet_addr ("224.2.0.1");
                srcPort = 23456;
        }
        if (argc >= 3) {
                for (i=0; argv[2][i]; i++)
                        if (argv[2][i] == '/') {
                                argv[2][i] = '\0';
                                i++;
                                break;
                        }
```

```
                dstAddr = inet_addr (argv[2]);
                dstPort = atoi (argv[2]+i);
        } else {
                dstAddr = inet_addr ("224.2.0.1");
                dstPort = 23456;
        }
/*** save the current tty settings ***/
        if (ioctl (0, TCGETA, &savedTermio) == -1) {
                perror ("ioctl");
                exit (1);
        }
/*** setup the tty interface ***/
        ioctl (0, TCGETA, &termio);
        termio.c_iflag = 0;
        termio.c_oflag = 0;
        termio.c_cflag &= CBAUD;
        termio.c_cflag |= CS8 |CREAD |HUPCL;
        termio.c_lflag = 0;
        termio.c_cc [VMIN] = 1;
        termio.c_cc [VTIME] = 0;
        ioctl (0, TCSETA, &termio);
/*** start the PPP interface ***/
        ServerMainLoop (0, 1);
/*** reset the TTY settings ***/
        if (ioctl (0, TCSETA, &savedTermio) == -1) {
                perror ("ioctl");
        }
} /* main */
#endif
```

C.1.2 modem.c

Dieses Modul ist nur für Seite des Anrufer nötig. Es bahendelt auch das Controlling TTY /dev/tty, so daß die Tastatureingaben zum Modem gesendet werden.

```
/*
 *      file:           modem.c
 *      creation date:  11-Feb-1994
 *      last update:    <obsolete>
 *      written by:     Christian Zahl
 *      description:
 *
 *update history:
 * date      who        why
 */
static char rcsid[] = "$Id: modem.c,v 1.18 1997/02/05 15:47:41 czahl Exp $\n";

/*** standard includefiles ***/
#include <stdio.h>
#include <fcntl.h>
#include <sgtty.h>
#include <sys/ioctl.h>
#include <sys/devinfo.h>
/* #include <sys/tty.h> */
#include <signal.h>
#include <termio.h>
#include <sys/select.h>
#include <sys/types.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/errno.h>
#include <sys/stat.h>
#include <time.h>
#include <netdb.h>

#include <sys/wait.h>

/*** user includefiles ***/
#include "misc.h"

/*** defines ***/
#define BUF_LEN                 1024
#define PORT                    8000

#define CHEXIT                  1
#define MAX(a,b)                ((a>b) ? a : b)

#define TERM_LOCAL              0
#define TERM_MODEM              1
#define TIMEOUT                 60
#define DEBUG(xxx)              xxx; fflush (stdout);
#define LOG_FN                  "LOG.MODEM"

/*** type definitions ***/
typedef struct {
        int                     f;              /* has to be the first! */
        struct termio           Termio;
        struct sgttyb           Sgttyb;
        struct tchars           Tchars;
```

```
        struct ltchars          LTchars;
        int                     Localmode;
} t_TermState;

/*** local functions ***/
static  void                    Alarm (int SigNo, int Pid);
static  void                    Terminate (int SigNo, int Pid);
static  void                    SigAlarm ();

/*** global functions ***/

/*** local variables ***/
        int                     baudRate;
        int                     dataBits;
        int                     parity;
        int                     stopBits;
        t_TermState             Term[2] = {{-1}, {-1}};
        int                     modemFlowControl = 99999;
        char                    *modemTtyName;
        char                    *modemTty;
        int                     modemFd;
        int                     channelNo = 0;
        int                     dataSend = 4;
        struct sigaction        sa;
        int                     stop = 0;

/*** global variables ***/
extern  u32bit                  srcAddr;
extern  u16bit                  srcPort;
extern  u32bit                  dstAddr;
extern  u16bit                  dstPort;

/*-----------------------------------------------------------------*/
Usage ()
{
        printf ("usage: modem [-b baud] [-d databits] [-p parity] [-f{x|r}] <tty>\n");
        printf ("\t<tty>\te.g. /dev/tty0\n");
        printf ("\t<baud>\t50 75 110 150 3[00] 6[00] 12[00] 24[00] 48[00] 96[00] 192[00] 384[00]\n");
        exit (1);
}
/*-----------------------------------------------------------------*/
ParseArgs (int argc, char *argv[])
{
        int                     c;

        baudRate = 38400;
        dataBits = 8;
        parity = 'n';
        stopBits = 1;
        while ((c = getopt (argc, argv, "b:d:f:p:s:")) != -1) {
                switch (c) {
                case 'b':
                        baudRate = atoi (optarg);
                        break;
                case 'd':
                        dataBits = atoi (optarg);
                        break;
                case 'f':
                        modemFlowControl = *optarg;
                        break;
                case 'p':
                        parity = *optarg;
                        break;
                case 's':
                        stopBits = atoi (optarg);
                        break;
                default:
                case '?':
                        Usage ();
                }
        }
        switch (baudRate) {
                case 50: baudRate = B50; break;
                case 75: baudRate = B75; break;
                case 110: baudRate = B110; break;
                case 150: baudRate = B150; break;
                case 3:
                case 300: baudRate = B300; break;
                case 6:
                case 600: baudRate = B600; break;
                case 12:
                case 1200: baudRate = B1200; break;
                case 24:
                case 2400: baudRate = B2400; break;
                case 48:
                case 4800: baudRate = B4800; break;
                default: printf ("\aInvalid baudRate, using 9600!\n");
                case 96:
                case 9600: baudRate = B9600; break;
                case 192:
                case 19200: baudRate = B19200; break;
                case 384:
                case 38400: baudRate = B38400; break;
        }
        switch (dataBits) {
                case 5: dataBits = CS5; break;
                case 6: dataBits = CS6; break;
                case 7: dataBits = CS7; break;
                default:
```

```
                        printf ("\aInvalid databits, using 8!\n");
                case 8: dataBits = CS8; break;
        }
        switch (modemFlowControl) {
                case 'x':
                        modemFlowControl = IXON;
                        break;
                case 'r':
                        modemFlowControl = 0;
                        break;
                default:
                        modemFlowControl = 0;
                        break;
        }
        switch (parity) {
                default:
                        printf ("\aInvaid parity, using no parity!\n");
                case 'n': parity = 0; break;
                case 'e': parity = PARENB; break;
                case 'o': parity = PARENB | PARODD; break;
                case 'm': parity = PARENB | PAREXT; break;
                case 's': parity = PARENB | PAREXT; break;
        }
        switch (stopBits) {
                default:
                        printf ("\aInvaid stopbits, using 1!\n");
                case 1: stopBits = 0; break;
                case 2: stopBits = CSTOPB; break;
        }
        if (optind == argc) {
                printf ("\asorry, no tty given, aborting!\n");
                exit (1);
        } else {
                modemTty = argv[optind++];
        }
        if (optind != argc) {
                printf ("\ainvalid argument!\n");
                Usage ();
                exit (1);
        }
} /* ParseArgs */
/*---------------------------------------------------------------------*/
CmdMode ()
{
        char                    cmd[100];

        printf ("COMMAND MODE (type ? for help)\n");
        while (1) {
                printf ("==> ");
                fflush (stdout);
                if (!gets (cmd))
                        continue;
                switch (cmd[0]) {
                        case 'f':
                                if (cmd[1] == 'i')
                                        tcflush (Term[TERM_MODEM].f, TCIFLUSH);
                                else if (cmd[1] == 'o')
                                        tcflush (Term[TERM_MODEM].f, TCOFLUSH);
                                else if (cmd[1] == 'b')
                                        tcflush (Term[TERM_MODEM].f, TCIOFLUSH);
                                else
                                        printf ("\aillegal option for f(lush) [i|o|b]\n");
                                break;
                        case 'o':
                                printf ("USE CTRL/A FOR COMMAND MODE\n");
                                Server ();
                                break;
                        case 'r':
                                RtpProxy (cmd+1);
                                break;
                        case 'q':
                                Terminate (0, 0);
                                break;
                        case '?':
                        case 'h':
                                printf ("AVAILABLE COMMANDS:\n");
                                printf ("\tflush [i|o|b]  flush IN, OUT or BOTH\n");
                                printf ("\tr)tp proxy     serve as RTP proxy\n");
                                printf ("\to)nline        go online\n");
                                printf ("\tq)uit          say good-bye\n");
                                printf ("\t? or h)elp     prints this help information\n");
                                break;
                        case '\0':
                                break;
                        default:
                                printf ("\aINVALID COMMAND (h for help)\n");
                                break;
                } /* switch */
        } /* while */
} /* CmdMode */
/*---------------------------------------------------------------------*/
Server ()
{
#define BUFSIZE                 (32*1024)
        struct termio           Termio;
        int                     i;
        char                    Buf[BUFSIZE];
        fd_set                  ReadSet;
        int                     Fds;
```

```
        int                     j;

        signal (SIGTERM, Terminate);
        signal (SIGINT, Terminate);
        signal (SIGQUIT, Terminate);
        signal (SIGHUP, Terminate);
        sa.sa_handler = SigAlarm;
        sa.sa_flags = SA_RESTART;
        sigaction (SIGALRM, &sa, NULL);
        alarm (TIMEOUT);

/*** Einstellungen für local Term ***/
        ioctl (0, TCGETA, &Termio);
/*      Termio.c_iflag = IXON|IXOFF; */
        Termio.c_iflag = 0;
        Termio.c_oflag = 0;
        Termio.c_lflag = 0;
        Termio.c_cc [VMIN] = BUFSIZE;
        Termio.c_cc [VTIME] = 3;
        ioctl (0, TCSETA, &Termio);
/*** MAINLOOP ***/
        Fds = 0;
        Fds = MAX (Fds, 0);
        Fds = MAX (Fds, Term[TERM_MODEM].f);
        Fds++;
        while (1) {
                FD_ZERO (&ReadSet);
                FD_SET (0, &ReadSet);
                FD_SET (Term[TERM_MODEM].f, &ReadSet);
                if ((i = select (Fds, &ReadSet, NULL, NULL, NULL)) == -1) {
                        if (errno == EINTR)
                                continue;
                        perror ("select");
                }
/*** Daten vom Modem gekommen? ***/
                if (FD_ISSET (Term[TERM_MODEM].f, &ReadSet)) {
                        if ((i = read (Term[TERM_MODEM].f, Buf, BUFSIZE)) >= 1) {
                                write (1, Buf, i);
                        } /* if */
                } /* if */
/*** Tasten vom tty behandeln ***/
                if (FD_ISSET (0, &ReadSet)) {
                        i = read (0, Buf, BUFSIZE);             /* von KBD lesen */
                        dataSend = 4;
                        if (i == 1) {
                                if (Buf[0] == CHEXIT)           /* in den cmd Modem */
                                      break;
                                else
                                write (Term[TERM_MODEM].f, Buf, i);             /* hinschicken */
                        } else
                                write (Term[TERM_MODEM].f, Buf, i);
                } /* if */
        } /* while */
        alarm (0);
        SetTtyStates (&Term[TERM_LOCAL]);
} /* Server */
/*--------------------------------------------------------------------*/
static void Terminate (int SigNo, int Pid)
/*
 * Signal Händler für den bösen Fall und für den Fall, das jemand
 * ^A gedrückt hat.
 */
{
        int             i;

/*** flush in and output ***/
        tcflush (Term[TERM_MODEM].f, TCIOFLUSH);
/*** restore tty states ***/
/*      SetTtyStates (&Term[TERM_MODEM]); */
        SetTtyStates (&Term[TERM_LOCAL]);
        close (Term[TERM_MODEM].f);
        DEBUG (printf ("\rSignal %d catched...", SigNo);)
/*** remove the lock file ***/
        DEBUG (printf ("\nRemoving TTY lock...");)
        ttyunlock (modemTtyName);
        DEBUG (printf ("done!\n");)
        DEBUG (printf ("Bye!\n");)
        exit (0);
} /* Terminate */
/*--------------------------------------------------------------------*/
SaveTtyStates (t_TermState *Term)
{
        if (Term->f == -1)
          return 0;
        if (ioctl (Term->f, TCGETA, &Term->Termio) == -1)
          perror ("tty TCGETA\n");
        if (ioctl (Term->f, TIOCGLTC, &Term->LTchars) == -1)
          perror ("tty TIOCGLTC");
        if (ioctl (Term->f, TIOCGETP, &Term->Sgttyb) == -1)
          perror ("tty TIOCGETP");
        if (ioctl (Term->f, TIOCGETC, &Term->Tchars) == -1)
          perror ("tty TIOCGETC");
        if (ioctl (Term->f, TIOCLGET, &Term->Localmode) == -1)
          perror ("tty TIOCLGET");
} /* SaveTtyStates */
/*--------------------------------------------------------------------*/
SetTtyStates (t_TermState *Term)
{
        if (Term->f == -1)
```

```
          return 0;
        if (ioctl (Term->f, TIOCLSET, &Term->Localmode) == -1)
          perror ("TIOCLSET");
        if (ioctl (Term->f, TIOCSETC, &Term->Tchars) == -1)
          perror ("TIOCSETC");
        if (ioctl (Term->f, TIOCSETP, &Term->Sgttyb) == -1)
          perror ("TIOCSETP");
        if (ioctl (Term->f, TIOCSLTC, &Term->LTchars) == -1)
          perror ("TIOCSLTC");
        if (ioctl (Term->f, TCSETA, &Term->Termio) == -1)
          perror ("TCSETA");
} /* SetTtyStates */
/*---------------------------------------------------------------------*/
InitModem (int f)
{
        struct termio   Term;
        int             i;

        DEBUG (printf ("Getting modem paramters...");)
        ioctl (f, TCGETA, &Term);
        DEBUG (printf ("done!\n");)
        if (modemFlowControl == IXON)
                Term.c_iflag = IXON|IXOFF;
        else
                Term.c_iflag = 0;
        Term.c_oflag = 0;
        Term.c_cflag = baudRate |CS8 |CREAD |CLOCAL;
        Term.c_cflag = baudRate |dataBits |parity |stopBits |CREAD |CLOCAL;
        Term.c_lflag = 0;
        Term.c_cc [VMIN] = 1;
        Term.c_cc [VTIME] = 0;
        DEBUG (printf ("Setting modem paramter...");)
        ioctl (f, TCSETAF, &Term);
        DEBUG (printf ("done!\n");)
        return 0;
} /* InitModem */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
RtpProxy (char *buf)
{
        int                     i;

        while (*buf && *buf == ' ')
                buf++;
        if (srcAddr == 0) {
                srcAddr = inet_addr ("224.2.0.1");
                srcPort = 23456;
                dstAddr = inet_addr ("224.2.0.1");
                dstPort = 23456;
        }
        for (i=0; buf[i]; i++)
                if (buf[i] == '/') {
                        buf[i] = '\0';
                        srcAddr = inet_addr (buf);
                        i++;
                        buf += i;
                        srcPort = atoi (buf);
                        break;
                }
        while (*buf && *buf != ' ')
                buf++;
        while (*buf && *buf == ' ')
                buf++;
        for (i=0; buf[i]; i++)
                if (buf[i] == '/') {
                        buf[i] = '\0';
                        dstAddr = inet_addr (buf);
                        i++;
                        buf += i;
                        dstPort = atoi (buf);
                        break;
                }
        printf ("ENTERING RTP / PPP PROXY MODE NOW, LEAVE WITH ^B...\n");
        printf ("%08X/%d -> %08X/%d\n", srcAddr, srcPort, dstAddr, dstPort);
        ServerMainLoop (modemFd, modemFd);
} /* RtpProxy */
/*---------------------------------------------------------------------*/
static void SigAlarm ()
{
        if (dataSend == 0) {
                write (Term[TERM_MODEM].f, "\x11", 1);
                dataSend = 4;
        }
        dataSend--;
        alarm (TIMEOUT);
} /* SigAlarm */
/*---------------------------------------------------------------------*/
main (int argc, char *argv[])
{
        int                     i;
        int                     a;
        int                     fd;

        printf ("[compiled on %s at %s]\n", __DATE__, __TIME__);
        ParseArgs (argc, argv);
/*** strip the modem tty name from the device path ***/
        modemTtyName = modemTty;
        for (i=strlen (modemTtyName)-1; i>0; i--)
```

```
                if (modemTtyName[i] == '/') {
                        modemTtyName += i+1;
                        break;
                }
        Term[TERM_LOCAL].f = 0;
/*** check for any locks to this tty ***/
        if (ttylock (modemTtyName) == -1) {
                fprintf (stderr, "Sorry, but the TTY %s is locked by another process!\n", modemTty);
                exit (1);
        }
/*** istall various signal handler ***/
        signal (SIGTERM, Terminate);
        signal (SIGINT, Terminate);
        signal (SIGQUIT, Terminate);
        signal (SIGHUP, Terminate);
/*** open the modem tty without waiting ***/
        if ((Term[TERM_MODEM].f = open (modemTty, O_RDWR |O_NDELAY)) == -1) {
                perror ("fopen");
                ttyunlock (modemTtyName);
                exit (1);
        }
        modemFd = Term[TERM_MODEM].f;
        DEBUG (printf ("done!\n");)
/*** save the tty states ***/
        SaveTtyStates (&Term[TERM_LOCAL]);
        SaveTtyStates (&Term[TERM_MODEM]);
        InitModem (Term[TERM_MODEM].f);
        CmdMode ();
} /* main */
/*---------------------------------------------------------------------*/
```

C.2 HDLC Layer

C.2.1 hdlc.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           HDLC.c
 *      written by:     Christian Zahl
 *      description:    Impelements an HDLC framing protocol (RFC-1662).
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>
#include <sys/time.h>

/*----- user includefiles ---------------------------------------------------*/
#include "misc.h"
#include "hdlc.h"

/*----- defines -------------------------------------------------------------*/
#define FRAME_LEN               1500
#define HDLC_FLAG               0x7E
#define HDLC_ADDR               0xFF
#define HDLC_CTRL               0x03
#define HDLC_ESC                0x7D
#define HDLC_PUT_BYTE(out,nnn,byte) { \
                                u8bit b = byte; \
                                if (sndAccm[b]) { \
                                        out[nnn++] = HDLC_ESC; \
                                        out[nnn++] = b ^ 0x20; \
                                } else \
                                        out[nnn++] = b;\
                                }
#define PPPINITFCS16            0xffff                          /* Initial FCS value */
#define PPPGOODFCS16            0xf0b8                          /* Good final FCS value */
#define FCS16(fff,xxx)          fff = (fff >> 8) ^ fcstab[(fff ^ (xxx)) & 0xff]
#ifdef _AIX
 #define DEBUG(xxx)             xxx
#else
 #define DEBUG(xxx)
#endif

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/
static  u8bit                   rcvAccm[256];
static  u8bit                   sndAccm[256];
static  u8bit                   frame[FRAME_LEN];
static  int                     frameLen;
static  int                     esc;
static  int                     addrCtrlCompr;
static  void                    (*cb) ();                       /* cb for incomming HDLC frames */
static  void                    (*linkOutCb) ();                /* cb for sending HDLC frames */
```

```
static  int                     errFCS = 0;                     /* # of FCS errors */
static  int                     errTooShort = 0;                /* # of frames too short */
static  int                     errTooLong = 0;                 /* # of frames too long */
static  int                     errFrame = 0;                   /* # of general framing errors */
static  int                     goodFrames = 0;                 /* # of received good frames */
static  u16bit                  fcstab[256] = {
        0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
        0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
        0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
        0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
        0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
        0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
        0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
        0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
        0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
        0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
        0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
        0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
        0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
        0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
        0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
        0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
        0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
        0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
        0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
        0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
        0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
        0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
        0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
        0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
        0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
        0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
        0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
        0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
        0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
        0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
        0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
        0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
        };
#ifdef _AIX
 #define WITH_LOG               0
 #define WITH_XMIT_LOG          0
 #define WITH_RCV_LOG           0
#endif
#if WITH_LOG
 static FILE                    *log = NULL;
 static struct timeval          now;
 static struct timeval          before;
#endif
#if WITH_XMIT_LOG
 static FILE                    *xmitLog = NULL;
 static time_t                  lastXmit;               /* time of last xmit */
 static int                     nXmit;                  /* octets xmited in last second */
#endif
#if WITH_RCV_LOG
 static FILE                    *rcvLog = NULL;
 static time_t                  lastRcv;                /* time of last rcv */
 static int                     nRcv;                   /* octets received in last second */
#endif
#if WITH_LOG || WITH_RCV_LOG || WITH_XMIT_LOG
 static time_t                  t0;
#endif

/*----- global variables ----------------------------------------------------*/

/*--------------------------------------------------------------------------*/
void HDLCInit (void (*_linkOutCb) ())
/*
 * Initializes the HDLC layer.
 */
{
        int                     i;
#if WITH_LOG || WITH_XMIT_LOG || WITH_RCV_LOG
        char                    fn[100];
        int                     now;
        now = time (NULL);
        t0 = now;
#endif

/*** init the ACCM map ***/
        for (i=0; i<32; i++)
                sndAccm[i] = rcvAccm[i] = 1;
        for (; i<256; i++)
                sndAccm[i] = rcvAccm[i] = 0;
        sndAccm[HDLC_FLAG] = 1;
        sndAccm[HDLC_ESC] = 1;
/*** init various stuff ***/
        frameLen = 0;
        esc = 0;
        addrCtrlCompr = 0;
        cb = NULL;
        linkOutCb = _linkOutCb;
        errFCS = 0;
        errTooShort = 0;
        errTooLong = 0;
        errFrame = 0;
        goodFrames = 0;
#if WITH_LOG
        sprintf (fn, "log.%d.hdlc", now);
```

```
        log = fopen (fn, "wb");
#endif
#if WITH_XMIT_LOG
        sprintf (fn, "log.%d.hdlc.xmit", now);
        xmitLog = fopen (fn, "wb");
#endif
#if WITH_RCV_LOG
        sprintf (fn, "log.%d.hdlc.rcv", now);
        rcvLog = fopen (fn, "wb");
#endif
} /* HDLCInit */
/*--------------------------------------------------------------------------*/
void HDLCFini ()
{
        printf ("HDLC Statistics:\r\n");
        printf ("\tFCS errors:    %d\r\n", errFCS);
        printf ("\ttoo short:     %d\r\n", errTooShort);
        printf ("\ttoo long:      %d\r\n", errTooLong);
        printf ("\tframing error: %d\r\n", errFrame);
        printf ("\tgood frames:   %d\r\n", goodFrames);
        cb = NULL;
        linkOutCb = NULL;
#if WITH_LOG
        if (log)
                fclose (log);
        log = NULL;
#endif
#if WITH_XMIT_LOG
        if (xmitLog)
                fclose (xmitLog);
        xmitLog = NULL;
#endif
#if WITH_RCV_LOG
        if (rcvLog)
                fclose (rcvLog);
        rcvLog = NULL;
#endif
} /* HDLCFini */
/*--------------------------------------------------------------------------*/
void HDLCSetRcvAccm (u32bit map)
/*
 * Sets the internal receiving ACCM map according to the ACCM map given.
 */
{
        int                     i;

        for (i=0; i<32; i++)
                if ((map >> i) & 0x01)
                        rcvAccm[i] = 1;
                else
                        rcvAccm[i] = 0;
} /* HDLCSetRcvAccm */
/*--------------------------------------------------------------------------*/
void HDLCSetSndAccm (u32bit map)
/*
 * Sets the internal sending ACCM map according to the ACCM map given.
 */
{
        int                     i;

        for (i=0; i<32; i++)
                if ((map >> i) & 0x01)
                        sndAccm[i] = 1;
                else
                        sndAccm[i] = 0;
} /* HDLCSetSndAccm */
/*--------------------------------------------------------------------------*/
void HDLCSetAddrCtrlCompr (int compr)
{
        addrCtrlCompr = compr;
}
/*--------------------------------------------------------------------------*/
void HDLCSend (u8bit *buf, int len)
/*
 * Generates an HDLC frame containing the data from 'in' of 'len' bytes.
 * It then calls the link output callback.
 * RC   >0      length of frame in bytes
 *      <=0     error
 */
{
        u8bit                   frame[FRAME_LEN];
        int                     n = 0;
        u16bit                  fcs;

        if (!linkOutCb)
                return;
        frame[n++] = HDLC_FLAG;
        fcs = PPPINITFCS16;
        fcs = FCS16 (fcs, HDLC_ADDR);
        fcs = FCS16 (fcs, HDLC_CTRL);
        if (!addrCtrlCompr) {
                HDLC_PUT_BYTE (frame, n, HDLC_ADDR);
                HDLC_PUT_BYTE (frame, n, HDLC_CTRL);
        }
        while (len-- > 0)
                if (n < FRAME_LEN -1 -2 -1) {
                        fcs = FCS16 (fcs, *buf);
                        HDLC_PUT_BYTE (frame, n, *buf++);
                }
```

```
        fcs ^= 0xFFFF;
        HDLC_PUT_BYTE (frame, n, (fcs >> 0) & 0xff);
        HDLC_PUT_BYTE (frame, n, (fcs >> 8) & 0xff);
        frame[n++] = HDLC_FLAG;
#if WITH_XMIT_LOG
        {
                time_t                  t;
                time (&t);
                if (t != lastXmit) {
                        if (lastXmit)
                                for (; lastXmit<t; lastXmit++) {
                                        fprintf (xmitLog, "%d %9.6f\n", lastXmit-t0, 8.0*nXmit/1000.0);
                                        nXmit = 0;
                                }
                        lastXmit = t;
                        nXmit = 0;
                }
                nXmit += n;
        }
#endif
        linkOutCb (frame, n);
} /* HDLCSend */
/*-------------------------------------------------------------------------*/
void HDLCReceive (u8bit *buf, int len)
/*
 * Handles the received byte.
 */
{
        int                     n;
        u8bit                   byte;
        u16bit                  fcs;
        int                     i;

        while (len-- > 0) {
                byte = *buf++;
/*** check for start or end of frame ***/
                if (byte == HDLC_FLAG) {                        /* end of frame? */
#if WITH_LOG
                        if (frameLen >= 1+1+2) {
                                int                     t;

                                gettimeofday (&now, NULL);
                                t = (now.tv_sec -before.tv_sec)*1000*1000+(now.tv_usec -before.tv_usec);
                                fprintf (log, "%2.6f\t# %4d in %3d.%03d ms\n", 8.0*frameLen/t*1000, frameLen, t /(1000), t%(1000));
                                fflush (log);
                        }
                        before = now;
#endif
#if WITH_RCV_LOG
                        time_t                  t;
                        time (&t);
                        if (t != lastRcv) {
                                if (lastRcv)
                                        for (; lastRcv<t; lastRcv++) {
                                                fprintf (rcvLog, "%d %9.6f\n", lastRcv-t0, 8.0*nRcv/1000.0);
                                                nRcv = 0;
                                        }
                                lastRcv = t;
                                nRcv = 0;
                        }
                        nRcv += frameLen +1;
#endif
                        if (frameLen == 0) {                            /* FLAG after FLAG? */
                                n = 0;
                        } else if (frameLen <= 1+1+2) {                 /* too short? ADDR+CRTL+FCS16 */
                                n = 0;
                                errTooShort++;
                        } else if (frame[0] != HDLC_ADDR) {             /* ADDR missing? */
                                n = 0;
                                errFrame++;
                        } else if (frame[1] != HDLC_CTRL) {             /* CTRL missing? */
                                n = 0;
                                errFrame++;
                        } else if (frameLen >= FRAME_LEN) {             /* too long? */
                                n = 0;
                                errTooLong++;
                        } else {
/*** check the CFS ***/
                                fcs = PPPINITFCS16;
                                for (i=0; i<frameLen; i++)
                                        fcs = FCS16 (fcs, frame[i]);
                                if (fcs != PPPGOODFCS16) {
                                        DEBUG (fprintf (stderr, "FCS[%04X]\a\n", fcs);)
                                        n = 0;
                                        errFCS++;
                                } else {
                                        goodFrames++;
                                        n = frameLen -1 -1 -2;          /* len of payload */
                                        if (cb)
                                                cb (frame+2, n);
                                }
                        }
                        frameLen = 0;
                        esc = 0;
/*** check for flaged ctrl-seq ***/
                } else if (rcvAccm[byte]) {
                        ;
/*** check for ESC ***/
                } else if (byte == HDLC_ESC) {
```

```
                        esc = 1;
/*** check for pending escape ***/
                } else if (esc) {
                        byte ^= 0x20;
                        if (frameLen == 0 && byte != HDLC_ADDR) {
                                frame[frameLen++] = HDLC_ADDR;
                                frame[frameLen++] = HDLC_CTRL;
                        }
                        if (frameLen < FRAME_LEN)
                                frame[frameLen++] = byte;
                        esc = 0;
/*** store the byte for the frame ***/
                } else {
                        if (frameLen == 0 && byte != HDLC_ADDR) {
                                frame[frameLen++] = HDLC_ADDR;
                                frame[frameLen++] = HDLC_CTRL;
                        }
                        if (frameLen < FRAME_LEN)
                                frame[frameLen++] = byte;
                }
        } /* while */
} /* HDLCReceive */
/*---------------------------------------------------------------------------*/
void HDLCRegister (void (*_cb) ())
/*
 * Registers the given callback _cb for incomming HDLC frames.
 */
{
        cb = _cb;
} /* HDLCRegister */
/*---------------------------------------------------------------------------*/
void HDLCUnregister (void (*_cb) ())
/*
 * Removes the registered callback for HDLC frames.
 */
{
        cb = NULL;
} /* HDLCUnregister */
/*---------------------------------------------------------------------------*/
```

C.3 PPP Layer, Bandwidth Layer

C.3.1 ppp.h

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           ppp.h
 *      written by:     Christian Zahl
 *      description:    Defines various types for PPP
 *
 * $Id$
 *
 * $Log$
 */

#ifndef INCL_PPP_H
#define INCL_PPP_H

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/

/*----- defines -------------------------------------------------------------*/
#define PPP_PROT_FULL_HEADERS   0x0073
#define PPP_PROT_COMPRESSED_UDP 0x0075
#define PPP_PROT_COMPRESSED_RTP 0x0077
#define PPP_PROT_CONTEXT_STATE  0x0079
#define PPP_PROT_START_TIMER    0x8073
#define PPP_PROT_CSAP           0x8075

/*----- type definitions ----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_PPP_H */
```

C.3.2 ppp.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           ppp.c
 *      written by:     Christian Zahl
 *      description:    Implements an part of the PPP (RFC-1661)
 *
 * $Id$
 *
 * $Log$
```

```
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>
#include <errno.h>
#include <sys/time.h>

/*----- user includefiles --------------------------------------------------*/
#include "misc.h"
#include "hdlc.h"
#include "ppp.h"

/*----- defines ------------------------------------------------------------*/
#define CB_LIST                 100
#define PPP_MTU                 1490

/*----- type definitions ---------------------------------------------------*/
typedef struct {
        u16bit                  proto;
        void                    (*cb)();
        void                    *data;
} t_Cb;

/*----- local functions ----------------------------------------------------*/

/*----- global functions ---------------------------------------------------*/
        void                    PPPReceive (u8bit *frame, int len);

/*----- local variables ----------------------------------------------------*/
static  t_Cb                    cbList[CB_LIST];
 static struct timeval          bandwTimer;
#ifdef _AIX
 #define WITH_LOG               0
 #define WITH_BLOG              1
#endif
#if WITH_LOG
        FILE                    *log;
        struct timeval          now;
        struct timeval          before;
        int                     octets;
#endif
#if WITH_BLOG
 static FILE                    *bLog;
#endif

/*----- global variables ---------------------------------------------------*/

/*--------------------------------------------------------------------------*/
void PPPInit ()
{
        int                     i;

#if WITH_LOG
        int t;
        char fn[100];
        time (&t);
        sprintf (fn, "log.%d.ppp", t);
        log = fopen (fn, "wb");
#endif
#if WITH_BLOG
        {
                int t;
                char fn[100];
                time (&t);
                sprintf (fn, "log.%d.ppp.bandw", t);
                bLog = fopen (fn, "wb");
        }
#endif
        for (i=0; i<CB_LIST; i++) {
                cbList[i].proto = 0;
                cbList[i].cb = NULL;
                cbList[i].data = NULL;
        }
        HDLCRegister (PPPReceive);
} /* PPPInit */
/*--------------------------------------------------------------------------*/
void PPPFini ()
{
#if WITH_LOG
        fclose (log);
#endif
#if WITH_BLOG
        fclose (bLog);
#endif
        HDLCUnregister (PPPReceive);
} /* PPPFini */
/*--------------------------------------------------------------------------*/
static void PPPBandwidthTimerStart ()
/*
 * Signals that we have to start the time for the bandwidth estimation.
 */
{
        gettimeofday (&bandwTimer, NULL);
} /* PPPBandwidthTimerStart */
/*--------------------------------------------------------------------------*/
static void PPPBandwidthTimerStop (int len)
/*
 * Signals that we have to stop the time for the bandwidth estimation. Len
```

```
 * specifies the number of bytes received in the meanwhile.
 */
{
        struct timeval          now;
        int                     delta;

        gettimeofday (&now, NULL);
        delta = (now.tv_sec -bandwTimer.tv_sec) *1000 *1000;
        delta += now.tv_usec -bandwTimer.tv_usec;
#if WITH_BLOG
        fprintf (bLog, "%9.6f # %d bytes in %7.3f ms\n",
                8.0 *len /delta *1000, len, 1.0*delta /1000);
#endif
} /* PPPBandwidthTimerStop ***/
/*---------------------------------------------------------------------------*/
void PPPRegister (u16bit proto, void (*cb)(), void *data)
{
        int                     i;
        int                     j = -1;

        for (i=0; i<CB_LIST; i++)
                if (cbList[i].proto == proto) {
                        j = i;
                        break;
                } else if (j == -1 && cbList[i].proto == 0)
                        j = i;
        if (j < CB_LIST && j != -1) {
                cbList[j].proto = proto;
                cbList[j].cb = cb;
                cbList[j].data = data;
        }
} /* PPPRegister */
/*---------------------------------------------------------------------------*/
void PPPUnregister (u16bit proto, void (*cb)())
{
        int                     i;

        for (i=0; i<CB_LIST; i++)
                if (cbList[i].proto == proto)
                        break;
        if (i < CB_LIST) {
                cbList[i].proto = 0;
                cbList[i].cb = NULL;
                cbList[i].data = NULL;
        }
} /* PPPUnregister */
/*---------------------------------------------------------------------------*/
void PPPReceive (u8bit *frame, int len)
/*
 * Handles the received PPP frame. Calls the approriate protocol callback,
 * if the protocol has been registered. Otherwise the packet will be
 * silently discarded.
 */
{
        u16bit                  proto;
        int                     i;

#if WITH_LOG
        int                     t;
        gettimeofday (&now, NULL);
        t = (now.tv_sec -before.tv_sec)*1000*1000+(now.tv_usec -before.tv_usec);
        fprintf (log, "%2.6f\t", 8.0*len/t*1000);
        fprintf (log, "# %4d in %3d.%03d ms", len, t /(1000), t%(1000));
        fprintf (log, "\n");
        fflush (log);
        before = now;
#endif
        if (len < 2)
                return;
/*** check for PPP protocol field compression ***/
        if (frame[0] & 0x01) {
                proto = *frame++;
                len -= 1;
        } else {
                proto = *frame++ << 8;
                proto |= *frame++;
                len -= 2;
        }
/*** check for the special START_TIMER ***/
        if (proto == PPP_PROT_START_TIMER) {
                PPPBandwidthTimerStart ();
                return;
        }
        PPPBandwidthTimerStop (len);
/*** search the cb for the protocol ***/
        for (i=0; i<CB_LIST; i++)
                if (cbList[i].proto == proto)
                        break;
        if (i >= CB_LIST) {
#ifdef _AIX
                fprintf (stderr, "p[%04x,%d]", proto, len);
                fflush (stderr);
#endif
                return;
        }
        (cbList[i].cb) (frame, len, cbList[i].data);
} /* PPPReceive */
/*---------------------------------------------------------------------------*/
void PPPSend (u8bit *data, int len, u16bit proto)
```

```
{
        u8bit                   frame[PPP_MTU];
        int                     n = 0;

/*** send the START_TIMER frame first ***/
        n = 0;
        frame[n++] = (PPP_PROT_START_TIMER >> 8) & 0xFF;
        frame[n++] = (PPP_PROT_START_TIMER >> 0) & 0xFF;
        HDLCSend (frame, n);
        n = 0;
/*** build the real frame ***/
        if (len > PPP_MTU -2)
                return;
        if (proto <= 0xff) {
                frame[n++] = proto;
        } else {
                frame[n++] = (proto >> 8) & 0xFF;
                frame[n++] = (proto >> 0) & 0xFF;
        }
        while (len-- > 0)
                frame[n++] = *data++;
        HDLCSend (frame, n);
} /* PPPSend */
/*---------------------------------------------------------------------------*/
```

C.4 IP/UDP/RTP Header Compression

C.4.1 ipUdpRtpCompr.h

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           ipudprtpcomrpe.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id$
 *
 * $Log$
 */

#ifndef INCL_IP_UDP_RTP_COMPRESS_H
#define INCL_IP_UDP_RTP_COMPRESS_H

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/

/*----- defines -------------------------------------------------------------*/
#define TYPE_FULL_HEADERS       1
#define TYPE_COMPRESSED_UDP     3
#define TYPE_COMPRESSED_RTP     4
#define TYPE_CONTEXT_STATE      5

/*----- type definitions ----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_IP_UDP_RTP_COMPRESS_H */
```

C.4.2 ipUdpRtpCompr.c

```
/*
 *      file:           ipUdpRtpCompr.c
 *      written by:     Christian Zahl
 *      description:    Compressing of IP/UDP/RTP packets
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>
#include <sys/types.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>

/*----- user includefiles ---------------------------------------------------*/
#include "misc.h"
#include "rtp.h"
#include "ipUdpRtpCompr.h"

/*----- defines -------------------------------------------------------------*/
#define MAX_CXT                 256
```

```
#define TRACE                   printf ("##########%s:%d\n", __FILE__, __LINE__);
#ifdef _AIX
 #define DEBUG(xxx)             xxx
#else
 #define DEBUG(xxx)
#endif

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        u32bit                  srcAddr;                        /* IP src */
        u32bit                  dstAddr;                        /* IP dst */
        u16bit                  srcPort;                        /* UDP src */
        u16bit                  dstPort;                        /* UDP dst */
        u32bit                  ssrc;                           /* RTP ssrc */
} t_Ident;
typedef struct {
        int                     valid;
        t_Ident                 ident;
        int                     cxtId;
        u32bit                  timestamp;                      /* when this cxt was touched last */
        struct ip               *ip;
        u8bit                   ipHdr[60];
        int                     ipHdrLen;
        struct udphdr           *udp;
        u8bit                   udpHdr[8];
        int                     udpHdrLen;
        struct rtp              *rtp;
        u8bit                   rtpHdr[12+15*4+32];
        int                     rtpHdrLen;
        int                     fullHeaderReq;                  /* pending req to send FULL_HEADER */
        int                     seqNo;                          /* next sequence number */
        int                     deltaI;
        int                     deltaT;
        int                     X;
        int                     XLen;
} t_Cxt;

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/
static  u32bit                  timestamp = 0;
static  t_Cxt                   compressCxt[MAX_CXT];
static  t_Cxt                   decompressCxt[MAX_CXT];

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void SaveHeaders (t_Cxt *cxt, const u8bit *p)
/*
 * Saves the IP, UDP and RTP headers in the context.
 */
{
/*** save IP header ***/
        cxt->ipHdrLen = IP_HDR_BYTES ((struct ip*)p);
        memcpy (cxt->ipHdr, p, cxt->ipHdrLen);
        p += cxt->ipHdrLen;
/*** save UDP header ***/
        cxt->udpHdrLen = UDP_HDR_BYTES ((struct udphdr*)p);
        memcpy (cxt->udpHdr, p, cxt->udpHdrLen);
        p += cxt->udpHdrLen;
/*** save RTP header ***/
        cxt->rtpHdrLen = RTP_HDR_BYTES ((struct rtp*)p);
        memcpy (cxt->rtpHdr, p, cxt->rtpHdrLen);
} /* SaveHeaders */
/*---------------------------------------------------------------------------*/
static void GetIdent (const u8bit *in, t_Ident *ident)
/*
 * Generates an identifier for the connection out of the IP/UDP/RTP packet
 */
{
        struct ip               *ip = (struct ip*)in;
        struct udphdr           *udp = (struct udphdr*)IP_DATA (ip);
        struct rtp              *rtp = (struct rtp*)UDP_DATA (udp);

        ident->dstAddr = ip->ip_dst.s_addr;
        ident->srcAddr = ip->ip_src.s_addr;
        ident->dstPort = udp->uh_dport;
        ident->srcPort = udp->uh_sport;
        ident->ssrc = rtp->ssrc;
} /* GetIdent */
/*---------------------------------------------------------------------------*/
static CompIdent (const t_Ident *ident1, const t_Ident *ident2)
{
        return memcmp (ident1, ident2, sizeof (t_Ident));
} /* CompIdent */
/*---------------------------------------------------------------------------*/
static void InitCxt (t_Cxt *cxt)
/*
 * Initializes the context.
 */
{
        memset (&cxt->ident, '\0', sizeof (cxt->ident));
        cxt->fullHeaderReq = 1;
        cxt->seqNo = 0;
        cxt->deltaI = 1;
        cxt->deltaT = 0;
        cxt->X = 0;
```

```
        cxt->ip = (struct ip*)cxt->ipHdr;
        cxt->udp = (struct udphdr*)cxt->udpHdr;
        cxt->rtp = (struct rtp*)cxt->rtpHdr;
        cxt->valid = 0;
} /* InitCxt */
/*-------------------------------------------------------------------------*/
static int NewSeqNo (t_Cxt *cxt)
/*
 * Generates a new sequence no for the given context.
 */
{
        int                     seqNo;

        seqNo = cxt->seqNo++;
        if (cxt->seqNo >= 0x0f)
                cxt->seqNo = 0;
        return seqNo;
} /* NewSeqNo */
/*-------------------------------------------------------------------------*/
static t_Cxt *GetCompressCxt (const t_Ident *ident)
/*
 * Gets the context for this connection identifier. If it is a new one,
 * get a new context for the array by removing the oldest one.
 * This is for the compression context only.
 */
{
        int                     oldest = 0;
        int                     i;

        for (i=0; i<MAX_CXT; i++)
                if (CompIdent (&compressCxt[i].ident, ident) == 0)
                        return &compressCxt[i];
                else if (compressCxt[i].timestamp < compressCxt[oldest].timestamp)
                        oldest = i;
        InitCxt (&compressCxt[oldest]);
        compressCxt[oldest].ident = *ident;
        compressCxt[oldest].cxtId = oldest;
        return &compressCxt[oldest];
} /* GetCompressCxt */
/*-------------------------------------------------------------------------*/
static int CompressValue (u8bit *buf, u32bit val)
/*
 * Compresses the delta-values for the compressed IP/UDP/RTP header.
 * Returns the # of occupied bytes.
 */
{
        if (val <= 0x7f) {
                *buf = val;
                return 1;
        } else if (val <= 0x3fff) {
                *buf++ = ((val >> 8) & 0x3f) | 0x80;
                *buf = val & 0xff;
                return 2;
        } else if (val <= 0x3fffff) {
                *buf++ = ((val >> 16) & 0x3f) | 0xC0;
                *buf++ = ((val >> 8) & 0xff);
                *buf = val & 0xff;
                return 3;
        } else
                return -1;
} /* CompressValue */
/*-------------------------------------------------------------------------*/
static int DecompressValue (u8bit *buf, u32bit *val)
/*
 * Decompresses the value from an compressed IP/UDP/RTP header.
 * Returns the # of occupied bytes.
 */
{
        if ((buf[0] & 0x80) == 0x00) {                          /* 1 byte */
                *val = buf[0] & 0x7f;
                return 1;
        }
        if ((buf[0] & 0xC0) == 0x80) {                          /* 2 bytes */
                *val = ((buf[0] & 0x3f) <<8) | buf[1];
                return 2;
        }
        if ((buf[0] & 0xC0) == 0xC0) {                          /* 3 bytes */
                *val = ((buf[0] & 0x3f) <<16) | (buf[1]<<8) | buf[2];
                return 3;
        }
} /* CompressValue */
/*-------------------------------------------------------------------------*/
static int GenerateFullHeader (t_Cxt *cxt, const u8bit *in, int inLen, u8bit *out, int *type)
/*
 * Generates a FULL_HEADER packet in out, sets the type and returns the # of
 * bytes of the new packet.
 */
{
        struct ip               *ip = (struct ip*)out;

/*** set the type of this new package ***/
        *type = TYPE_FULL_HEADERS;
/*** place the input package in the new paket ***/
        memcpy (out, in, inLen);
/*** place the sequence-no ***/
        IP_DATA (ip) [5] = NewSeqNo (cxt);                      /* low byte of UDP length */
/*** place the context-id ***/
        ip->ip_len = (ip->ip_len & 0xff00) | cxt->cxtId;        /* cxtId into low byte of IP-len */
/*** reset delta values to default ***/
```

```
        cxt->deltaI = 1;
        cxt->deltaT = 0;
        cxt->valid = 1;
/*** return the length of the new packet ***/
        return inLen;
} /* GenerateFullHeader */
/*-------------------------------------------------------------------------*/
static int GenerateCompressedUdp (t_Cxt *cxt, const u8bit *in, int inLen, u8bit *out, int *type)
/*
 * Generates a COMPRESSED_UDP packet in out, sets the type, and returns the #
 * of bytes of the new packet.
 */
{
        struct ip               *ip = (struct ip*)in;
        struct udphdr           *udp = (struct udphdr*) IP_DATA (ip);
        struct rtp              *rtp = (struct rtp*) UDP_DATA (udp);
        int                     deltaI;
        int                     n;
        int                     outLen = 0;

/*** set the type of this new package ***/
        *type = TYPE_COMPRESSED_UDP;
/*** place context-id and sequence-no ***/
        out[outLen++] = cxt->cxtId;
        out[outLen++] = NewSeqNo (cxt);
/*** place UDP checksum, if necessarry ***/
        if (udp->uh_sum) {
                if (!(cxt->udp->uh_sum))
                        return GenerateFullHeader (cxt, in, inLen, out, type);
                out[outLen++] = (udp->uh_sum >> 8) & 0xff;
                out[outLen++] = (udp->uh_sum >> 0) & 0xff;
        } else if (cxt->udp->uh_sum)
                return GenerateFullHeader (cxt, in, inLen, out, type);
/*** place IP identifier delta, if ncessarry ***/
        if ((deltaI = ip->ip_id - cxt->ip->ip_id) != cxt->deltaI) {
                out[1] |= 0x10;
                outLen += CompressValue (out+outLen, deltaI);
                cxt->deltaI = deltaI;
        }
/*** place RTP header and payload ***/
        n = inLen -IP_HDR_BYTES (ip) -UDP_HDR_BYTES (udp);
        memcpy (out+outLen, (u8bit*)rtp, n);
/*** reset the timestamp delta in the cxt ***/
        cxt->deltaT = 0;
/*** return size of generated package ***/
        return outLen +n;
} /* GenerateCompressedUdp */
/*-------------------------------------------------------------------------*/
static int GenerateCompressedRtp (t_Cxt *cxt, const u8bit *in, int inLen, u8bit *out, int *type)
/*
 * Generates a COMPRESSED_RTP packet in out, sets the type and returns the
 * # of bytes of the new packet.
 */
{
        int                     outLen = 0;
        struct ip               *ip = (struct ip*)in;
        struct udphdr           *udp = (struct udphdr*)IP_DATA (ip);
        struct rtp              *rtp = (struct rtp*)UDP_DATA (udp);
        int                     M = 0;
        int                     S = 0;
        int                     T = 0;
        int                     I = 0;
        int                     deltaS;
        int                     deltaT;
        int                     deltaI;
        int                     MSTI = 0;
        int                     MSTIOff;
        int                     i, j, n;

/*** set the type of this new package ***/
        *type = TYPE_COMPRESSED_RTP;
/*** place context-id and sequence-no ***/
        out[outLen++] = cxt->cxtId;
        MSTIOff = outLen;
        out[outLen++] = NewSeqNo (cxt);
/*** place UDP checksum ***/
        if (udp->uh_sum) {
                if (!cxt->udp->uh_sum)
                        return GenerateFullHeader (cxt, in, inLen, out, type);
                out[outLen++] = (udp->uh_sum >> 8) & 0xff;
                out[outLen++] = (udp->uh_sum >> 0) & 0xff;
        } else if (cxt->udp->uh_sum)
                return GenerateFullHeader (cxt, in, inLen, out, type);
/*** get some indications and deltas ***/
        M = rtp->m;
        if ((deltaS = rtp->seq -cxt->rtp->seq) < 0)
                deltaS += 0x10000;
        if (deltaS != 1)
                S = 1;
        if ((deltaT = rtp->timestamp -cxt->rtp->timestamp) != cxt->deltaT)
                T = 1;
        if ((deltaI = ip->ip_id - cxt->ip->ip_id) != cxt->deltaI)
                I = 1;
/*** check for changes in CSRC ***/
        n = 0;
        for (i=0; i<rtp->cc; i++)
                for (j=0; j<cxt->rtp->cc; j++)
                        if (RTP_CSRC(rtp)[i].csrc == RTP_CSRC(cxt->rtp)[j].csrc) {
                                n++;
```

```
                                break;
                        }
        /*** check for the need of special MSTI setting ***/
                if ((rtp->cc != cxt->rtp->cc)
                || (rtp->cc != n)
                || (M & S & T & I)) {
                        out[1] |= 0xf0;                                 /* set MSTI */
                        MSTIOff = outLen;
                        out[outLen++] = rtp->cc;                        /* CC */
                        MSTI = 1;
                }
        /*** place M bit ***/
                if (M)
                        out[MSTIOff] |= 0x80;
        /*** place I bit and IP identification delta ***/
                if (I) {
                        out[MSTIOff] |= 0x10;
                        outLen += CompressValue (out+outLen, deltaI);
                        cxt->deltaI = deltaI;
                }
        /*** place S bit and RTP sequence-no delta ***/
                if (S) {
                        out[MSTIOff] |= 0x40;
                        outLen += CompressValue (out+outLen, deltaS);
                }
        /*** place T bit and RTP timestamp delta ***/
                if (T) {
                        out[MSTIOff] |= 0x20;
                        outLen += CompressValue (out+outLen, deltaT);
                        cxt->deltaT = deltaT;
                }
        /*** place RTP CC count and RTP CC list ***/
                if (MSTI) {
                        u32bit                  csrc;

                        for (i=0; i<rtp->cc; i++) {
                                csrc = RTP_CSRC (rtp)[i].csrc;
                                out[outLen++] = (csrc >> 24) & 0xff;
                                out[outLen++] = (csrc >> 16) & 0xff;
                                out[outLen++] = (csrc >> 8) & 0xff;
                                out[outLen++] = (csrc >> 0) & 0xff;
                        }
                }
        /*** place RTP header extension ***/
                if (rtp->x) {
                        u8bit                   *p;

                        p = RTP_EXT_DATA (rtp);
                        for (i=0; i<RTP_EXT_BYTES (rtp); i++)
                                out[outLen++] = *p++;
                }
        /*** place RTP payload ***/
                n = inLen;
                n -= IP_HDR_BYTES (ip);
                n -= UDP_HDR_BYTES (udp);
                n -= RTP_HDR_BYTES (rtp);
                memcpy (out+outLen, RTP_DATA (rtp), n);
        /*** return size of generated package ***/
                return outLen +n;
        } /* GenerateCompressedRtp */
        /*-------------------------------------------------------------------------*/
        IpUdpRtpCompress (const u8bit *in, int inLen, u8bit *out, int *type)
        /*
         * Performs the IP/UDP/RTP compression
         * Returns the # of byte of the resulting package to be send and the type
         */
        {
                t_Ident                 ident;
                struct ip               *ip = (struct ip*)in;
                struct udphdr           *udp = (struct udphdr*)IP_DATA (ip);
                struct rtp              *rtp = (struct rtp*)UDP_DATA (udp);
                int                     outLen;
                t_Cxt                   *cxt;

                GetIdent ((u8bit*)ip, &ident);
                cxt = GetCompressCxt (&ident);
                cxt->timestamp = timestamp++;
        /*** FULL_HEADER if reqested or uncompressable IP hdr field changed ***/
                if ((cxt->fullHeaderReq)
                || (ip->ip_hl   != cxt->ip->ip_hl)
                || (ip->ip_tos  != cxt->ip->ip_tos)
                || (ip->ip_off  != cxt->ip->ip_off)
                || (ip->ip_ttl  != cxt->ip->ip_ttl)
                || (ip->ip_p    != cxt->ip->ip_p))
                        outLen = GenerateFullHeader (cxt, in, inLen, out, type);
        /*** COMPRESSED_UDP if uncompressable RTP field changes ***/
                else if ((rtp->x != cxt->rtp->x)
                || (rtp->pt != cxt->rtp->pt)
                || (rtp->seq - cxt->rtp->seq > 0x3fffff)
                || (rtp->timestamp - cxt->rtp->timestamp > 0x3fffff))
                        outLen = GenerateCompressedUdp (cxt, in, inLen, out, type);
        /*** COMPRESSED_RTP ***/
                else
                        outLen = GenerateCompressedRtp (cxt, in, inLen, out, type);
        /*** save the current headers in the context ***/
                SaveHeaders (cxt, in);
                cxt->fullHeaderReq = 0;
                return outLen;
        } /* IpUdpRtpCompress */
```

```
/*--------------------------------------------------------------------------*/
static int GenerateContextState (t_Cxt *cxt, u8bit *out)
/*
 * Generates a CONTEXT_STATE packet for the gicen context. Return the
 * length of the new packet in bytes.
 */
{
        int                     n = 0;

        out[n++] = 0x01;
        out[n++] = 0x01;                /* only one context state */
        out[n++] = cxt->cxtId;
        out[n++] = 0x80 | cxt->seqNo;
        out[n++] = 0x00;                /* currently no support for gen-id */
        return n;
} /* GenerateContextState */
/*--------------------------------------------------------------------------*/
static int DecompressFullHeaders (u8bit *_in, int inLen, u8bit *out)
/*
 * Decompresses a FULL_HEADER compressed frame and updates the context for
 * this session.
 * RC   > 0     len of uncompressed frame
 */
{
        u8bit                   tmp[1500];
        u8bit                   *in = tmp;
        int                     cxtId;
        t_Cxt                   *cxt;
        int                     outLen = inLen;
        struct ip               *ip = (struct ip*)in;
        struct udphdr           *udp;
        struct rtp              *rtp;

/*** copy to make sure that the IP header is word alligned (needed for SPARC) ***/
        memcpy (tmp, _in, inLen);
/*** it's IP, so handle and save the IP header ***/
        cxtId = ip->ip_len & 0xff;                              /* get the encoded context id */
        cxt = &decompressCxt[cxtId];
        InitCxt (cxt);
        cxt->cxtId = cxtId;
        cxt->ipHdrLen = IP_HDR_BYTES (ip);
        memcpy (cxt->ipHdr, in, cxt->ipHdrLen);
        cxt->ip->ip_len = inLen;                                /* set the real IP len */
        memcpy (out, cxt->ipHdr, cxt->ipHdrLen);
        out += cxt->ipHdrLen;
        in += cxt->ipHdrLen;
        inLen -= cxt->ipHdrLen;
/*** if it's UDP, handle UDP header ***/
        if (ip->ip_p == IPPROTO_UDP) {
                udp = (struct udphdr*)in;
                cxt->udpHdrLen = UDP_HDR_BYTES (udp);
                memcpy (cxt->udpHdr, udp, cxt->udpHdrLen);
                cxt->seqNo = udp->uh_ulen & 0xff;                       /* encoded sequence no */
                NewSeqNo (cxt);                                         /* generate the next sequence no to expect */
                cxt->udp->uh_ulen = inLen;                              /* set the original length */
                memcpy (out, cxt->udpHdr, cxt->udpHdrLen);
                out += cxt->udpHdrLen;
                in += cxt->udpHdrLen;
                inLen -= cxt->udpHdrLen;
/*** if it seems to be RTP version 2, handle it ***/
                rtp = (struct rtp*)in;
                if (rtp->v == 2) {
                        cxt->rtpHdrLen = RTP_HDR_BYTES (rtp);
                        memcpy (cxt->rtpHdr, in, cxt->rtpHdrLen);
                        memcpy (out, cxt->rtpHdr, cxt->rtpHdrLen);
                        out += cxt->rtpHdrLen;
                        in += cxt->rtpHdrLen;
                        inLen -= cxt->rtpHdrLen;
                } else {
                        cxt->rtpHdrLen = 0;
                        memset (cxt->rtpHdr, '\0', sizeof (cxt->rtpHdr));
                }
        } else {
                cxt->udpHdrLen = 0;
                memset (cxt->udpHdr, '\0', sizeof (cxt->udpHdr));
        }
/*** update some context fields ***/
        cxt->fullHeaderReq = 0;                         /* no pending FULL_HEADER */
        cxt->deltaT = 0;
        cxt->deltaI = 1;
        cxt->X = 0;
        cxt->valid = 1;
/*** place the remaining data ***/
        memcpy (out, in, inLen);
        return outLen;
} /* DecompressFullHeaders */
/*--------------------------------------------------------------------------*/
static short IpChksum (u16bit *buf, int words)
/*
 * Generates the IP header checksum, according to Comer.
 */
{
        u32bit                  sum;

        for (sum=0; words>0; words--)
                sum += *buf++;
        sum = (sum >> 16) + (sum & 0xffff);
        sum += (sum >> 16);
        return  sum;
```

```
} /* IpChnksum */
/*-------------------------------------------------------------------------*/
static int DecompressCompressedUdp (u8bit *in, int inLen, u8bit *out)
{
        int                     outLen = 0;
        int                     n = 0;
        int                     s;
        int                     cxtId;
        t_Cxt                   *cxt;
        u16bit                  chks;
        struct rtp              *rtp;

        cxtId = in[n++];
        cxt = &decompressCxt[cxtId];
        if (!cxt->valid)
                return -2;
        s = NewSeqNo (cxt) - in[n++] & 0x0F;
/*** check for pending FULL_HEADER request (i.e. context invalid) ***/
        if (cxt->fullHeaderReq) {
                DEBUG (fprintf (stderr, "s");)
                return -GenerateContextState (cxt, out);
                return 0;
        }
/*** check the sequence no ***/
        if (s) {
                cxt->fullHeaderReq = 1;
                DEBUG (fprintf (stderr, "S");)
                return -GenerateContextState (cxt, out);
                return -1;
        }
/*** get the UDP checksum, if present ***/
        if (cxt->udp->uh_sum) {
                chks = (in[n++] << 8);
                chks |= in[n++];
                cxt->udp->uh_sum = chks;
        }
/*** check if I bit is set ***/
        if (in[1] & 0x10)
                n += DecompressValue (in+n, &cxt->deltaI);
/*** update the saved headers ***/
        cxt->ip->ip_id += cxt->deltaI;
        cxt->ip->ip_len = inLen -n +cxt->ipHdrLen +cxt->udpHdrLen +cxt->rtpHdrLen;
        cxt->udp->uh_ulen = cxt->ip->ip_len -cxt->ipHdrLen;
/*** place the original IP header ***/
        memcpy (out +outLen, cxt->ipHdr, cxt->ipHdrLen);
        outLen += cxt->ipHdrLen;
/*** place the original UDP header ***/
        memcpy (out +outLen, cxt->udpHdr, cxt->udpHdrLen);
        outLen += cxt->udpHdrLen;
/*** save the RTP header ***/
        in += n;
        inLen -= n;
        rtp = (struct rtp*) in;
        if (rtp->v == 2 && RTP_HDR_BYTES (rtp) <= sizeof (cxt->rtpHdr)) {
                cxt->rtpHdrLen = RTP_HDR_BYTES (rtp);
                memcpy (cxt->rtpHdr, in, cxt->rtpHdrLen);
        } else
                ;
/*** place the RTP header and data ***/
        memcpy (out +outLen, in, inLen);
        outLen += inLen;
        return outLen;
} /* DecompressCompressedUdp */
/*-------------------------------------------------------------------------*/
static int DecompressCompressedRtp (u8bit *in, int inLen, u8bit *out)
{
        int                     n = 0;
        int                     outLen = 0;
        int                     cxtId;
        t_Cxt                   *cxt;
        int                     CC;
        u8bit                   MSTI;
        int                     i;
        u32bit                  deltaS;
        u32bit                  s;
        u32bit                  chks;
        u32bit                  csrc;

        cxtId = in[n++];
        cxt = &decompressCxt[cxtId];
        if (!cxt->valid)
                return -2;
        MSTI = in[n] & 0xf0;
        s = NewSeqNo (cxt) - (in[n++] & 0x0F);                  /* delta expected and received sequence-no */
/*** check for pending FULL_HEADER request (i.e. context invalid) ***/
        if (cxt->fullHeaderReq) {
                DEBUG (fprintf (stderr, "s");)
                return -GenerateContextState (cxt, out);
                return 0;
        }
/*** check the sequence no ***/
        if (s) {
                cxt->fullHeaderReq = 1;
                DEBUG (fprintf (stderr, "S");)
                return -GenerateContextState (cxt, out);
                return -1;
        }
/*** get the UDP checksum, if present ***/
        if (cxt->udp->uh_sum) {
```

```
                chks = (in[n++] << 8);
                chks |= in[n++];
                cxt->udp->uh_sum = chks;
        }
/*** ***/
        if (MSTI == 0xf0) {                                     /* MSTI = 1111 */
                MSTI = in[n];
                CC = in[n++] & 0x0f;                            /* CC */
        } else
                CC = 0;
/*** delta IPv4 identifier ***/
        if (MSTI & 0x10)                                        /* I */
                n += DecompressValue (in+n, &cxt->deltaI);
/*** delta RTP sequence ***/
        if (MSTI & 0x40)                                        /* S */
                n += DecompressValue (in+n, &deltaS);
        else
                deltaS = 1;
/*** delta RTP timestamp ***/
        if (MSTI & 0x20)                                        /* T */
                n += DecompressValue (in+n, &cxt->deltaT);
/*** get the RTP CSRC list ***/
        for (i=0; i<CC; i++) {
                csrc = (in[n+0]<<24) | (in[n+1]<<16) | (in[n+2]<<8) | (in[n+3]);
                RTP_CSRC(cxt->rtp)[i].csrc = csrc;
                n += 4;
        }
/*** get the RTP header extension ***/
        if (cxt->X) {
                /* currently not supported */
                n += cxt->XLen;
        }
/*** update the header fields ***/
        cxt->ip->ip_id += cxt->deltaI;
        cxt->ip->ip_len = inLen -n +IP_HDR_BYTES (cxt->ip) +UDP_HDR_BYTES (cxt->udp) +RTP_HDR_BYTES (cxt->rtp);
        cxt->udp->uh_ulen = cxt->ip->ip_len -IP_HDR_BYTES (cxt->ip);
        cxt->rtp->seq += deltaS;
        cxt->rtp->timestamp += +cxt->deltaT;
        cxt->rtp->m = (MSTI & 0x80) >>7;
        if (CC)
                cxt->rtp->cc = CC;
        cxt->rtpHdrLen = RTP_HDR_BYTES (cxt->rtp);
        cxt->ip->ip_sum = 0;
        cxt->ip->ip_sum = IpChksum ((u16bit*)cxt->ip, cxt->ipHdrLen /2);
/*** place the original IP header ***/
        memcpy (out +outLen, cxt->ipHdr, cxt->ipHdrLen);
        outLen += cxt->ipHdrLen;
/*** place the original UDP header ***/
        memcpy (out +outLen, cxt->udpHdr, cxt->udpHdrLen);
        outLen += cxt->udpHdrLen;
/*** place the original RTP header ***/
        memcpy (out +outLen, cxt->rtpHdr, cxt->rtpHdrLen);
        outLen += cxt->rtpHdrLen;
/*** place the recevied RTP data ***/
        memcpy (out +outLen, in+n, inLen -n);
        outLen += inLen -n;
        return outLen;
} /* DecompressCompressedRtp */
/*-------------------------------------------------------------------------*/
static int DecompressContextState (u8bit *in, int inLen, u8bit *out)
/*
 * Handles the received CONTEXT_STATE packet
 */
{
        int                     n = 0;
        int                     cxtId;
        t_Cxt                   *cxt;

        if (in[n++] != 0x01)
                return 0;
        if (in[n++] != 0x01)            /* we can handle only one context */
                return 0;
        cxtId = in[n++];
        cxt = &compressCxt[cxtId];
        if (!cxt->valid) {
                DEBUG (fprintf (stderr, "(CS%d not valid)", cxtId);)
                return 0;
        }
        if (in[n++] & 0x80) {           /* FULL_HEADER request ? */
                cxt->fullHeaderReq = 1;
        }
        return 0;
} /* DecompressContextState */
/*-------------------------------------------------------------------------*/
IpUdpRtpDecompress (u8bit *in, int inLen, int packetType, u8bit *out)
/*
 * Uncompresses the packet in with inLen bytes into out. The packetType
 * defines, which kind of packet we have.
 * RC   > 0     # byte of out
 *      0       # error
 */
{
        int                     n;

        if (packetType == TYPE_FULL_HEADERS) {
                DEBUG (fprintf (stderr, "F");)
                n = DecompressFullHeaders (in, inLen, out);
        } else if (packetType == TYPE_COMPRESSED_UDP) {
                DEBUG (fprintf (stderr, "U");)
```

```
                n = DecompressCompressedUdp (in, inLen, out);
        } else if (packetType == TYPE_COMPRESSED_RTP) {
/*              DEBUG (fprintf (stderr, "R");) */
                n = DecompressCompressedRtp (in, inLen, out);
        } else if (packetType == TYPE_CONTEXT_STATE) {
                DEBUG (fprintf (stderr, "C");)
                n = DecompressContextState (in, inLen, out);
        }
        return n;
} /* IpUdpRtpUncompress */
/*--------------------------------------------------------------------------*/

#include "dump.c"
```

C.5 RTP Translator

C.5.1 rtpTranslator.c

```
/*
 *      file:           rtpTranslator.c
 *      written by:     Christian Zahl
 *      description:    Implements an RTP translator with payload compression
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>
#include <sys/time.h>

/*----- user includefiles --------------------------------------------------*/
#include "misc.h"
#include "gsm.h"
#include "rtp.h"
#include "tabxlaw.h"
#include "lpc.h"

/*----- defines ------------------------------------------------------------*/
#define CXT_LIST                10                      /* # of contexts available */

/*----- type definitions ---------------------------------------------------*/
typedef struct {
        gsm                     gsmHandle;
        int                     lpcInit;
        int                     dstPt;                  /* dest. payload type */
        u32bit                  ssrc;
        u32bit                  csrc[15];
        u16bit                  seq;
        u32bit                  timestamp;
        u8bit                   data[8000];
        int                     dataLen;                /* current len in buf */
        int                     pt;                     /* current pt in buf */
        int                     m;                      /* marker bit */
        int                     cc;                     /* # CSRC */
} t_Cxt;

/*----- local functions ----------------------------------------------------*/

/*----- global functions ---------------------------------------------------*/

/*----- local variables ----------------------------------------------------*/
static  t_Cxt                   cxtList[CXT_LIST];

/*----- global variables ---------------------------------------------------*/

/*--------------------------------------------------------------------------*/
void RtpTranslatorInit ()
/*
 * Initializes the RTP translator.
 */
{
        int                     i;

        for (i=0; i<CXT_LIST; i++) {
                cxtList[i].gsmHandle = NULL;
                cxtList[i].dstPt = RTP_PT_LPC;
                cxtList[i].dstPt = RTP_PT_GSM;
        }
} /* RtpTranslatorInit */
/*--------------------------------------------------------------------------*/
void RtpTranslatorFini ()
{
} /* RtpTranslatorFini */
/*--------------------------------------------------------------------------*/
static int ConvertPcmuGsm (t_Cxt *cxt, u8bit *in, int inLen, u8bit *out)
/*
 * Converts len bytes in buf from PCMU to GSM.
 * Return # of bytes
 */
{
```

```
        short                   frame[160];
        int                     n = 0;
        int                     i;

/*** create the GSM handle, if not done so ***/
        if (!cxt->gsmHandle)
                cxt->gsmHandle = gsm_create ();
/*** convert each frame of 160 bytes (20ms) to GSM ***/
        while (inLen >= 160) {
                for (i=0; i<160; i++)
                        frame[i] = ULAW_TO_LIN (*in++);
                gsm_encode (cxt->gsmHandle, frame, out+n);
                n += 33;
                inLen -= 160;
        }
        return n;
} /* ConvertPcmuGsm */
/*---------------------------------------------------------------------------*/
static int ConvertPcmaGsm (t_Cxt *cxt, u8bit *in, int inLen, u8bit *out)
/*
 * Converts len bytes in buf from PCMU to GSM.
 * Return # of bytes
 */
{
        short                   frame[160];
        int                     n = 0;
        int                     i;

        if (!cxt->gsmHandle)
                cxt->gsmHandle = gsm_create ();
        while (inLen >= 160) {
                for (i=0; i<160; i++)
                        frame[i] = ALAW_TO_LIN (*in++);
                gsm_encode (cxt->gsmHandle, frame, out+n);
                n += 33;
                inLen -= 160;
        }
        return n;
} /* ConvertPcmaGsm */
/*---------------------------------------------------------------------------*/
static int ConvertPcmuLpc (t_Cxt *cxt, u8bit *in, int inLen, u8bit *out)
/*
 * Converts len bytes in buf from PCMU to LPC.
 * Return # of bytes
 */
{
        int                     n = 0;

/*** init LPC context if necessarry ***/
        if (!cxt->lpcInit) {
                lpc_init (160);
                cxt->lpcInit = 1;
        }
/*** convert each frame of 160 bytes (20ms) to LPC ***/
        while (inLen >= 160) {
                lpc_analyze (in, out+n);
                n += 14;
                inLen -= 160;
                in += 160;
        }
        return n;
} /* ConvertPcmuLpc */
/*---------------------------------------------------------------------------*/
static int RtpCompressPt (t_Cxt *cxt, u8bit *out)
/*
 * Compresses the payload of the RTP data for the given cxt.
 */
{
/*** convert to GSM ***/
        if (cxt->dstPt == RTP_PT_GSM) {
                switch (cxt->pt) {
                        case RTP_PT_PCMU:
                                cxt->pt = RTP_PT_GSM;
                                return ConvertPcmuGsm (cxt, cxt->data, cxt->dataLen, out);
                        case RTP_PT_PCMA:
                                cxt->pt = RTP_PT_GSM;
                                return ConvertPcmaGsm (cxt, cxt->data, cxt->dataLen, out);
                        case RTP_PT_GSM:
                                cxt->pt = RTP_PT_GSM;
                                memcpy (out, cxt->data, cxt->dataLen);
                                return cxt->dataLen;
                        case RTP_PT_LPC:
                                cxt->pt = RTP_PT_LPC;
                                memcpy (out, cxt->data, cxt->dataLen);
                                return cxt->dataLen;
                        default:
                                return 0;
                }
        } else if (cxt->dstPt == RTP_PT_LPC) {
                switch (cxt->pt) {
                        case RTP_PT_PCMU:
                                cxt->pt = RTP_PT_LPC;
                                return ConvertPcmuLpc (cxt, cxt->data, cxt->dataLen, out);
                        case RTP_PT_LPC:
                                cxt->pt = RTP_PT_LPC;
                                memcpy (out, cxt->data, cxt->dataLen);
                                return cxt->dataLen;
                        default:
                                return 0;
```

```
                }
/*** unsoported conversion... ***/
        } else {
                return 0;
        }
} /* RtpCompressPt */
/*---------------------------------------------------------------------------*/
int RtpTranslator (struct sockaddr_in *dst, struct rtp *in, int inLen, struct rtp *out)
/*
 * Generiert ein eigenes neues RTP paket, aufsetzend auf den empfangenen
 * und cxt. Komprimiert die Payload des RPT Paketes.
 * IN
 *      cxt     Context der session
 *      rtp     dst buffer fr rtp paket
 *
 * RC   = -1    skip
 *      >= 0    len
 */
{
        t_Cxt                   *cxt;
        u8bit                   *buf;
        int                     len;

/*** search the context of the session ***/
        cxt = &cxtList[0];              /* simulated for now... */
/*** translate the packet ***/
        cxt->pt = in->pt;
        cxt->m = in->m;
        cxt->timestamp = in->timestamp;
        cxt->seq = in->seq;
        cxt->dataLen = inLen -RTP_HDR_BYTES (in);
        memcpy (cxt->data, RTP_DATA (in), cxt->dataLen);
/*** generate a new output packet, if data available ***/
        out->v = 2;
        out->p = 0;
        out->x = 0;
        out->cc = 0;
        out->m = cxt->m;
        out->pt = cxt->dstPt;
        out->seq = cxt->seq;
        out->timestamp = cxt->timestamp;
        out->ssrc = cxt->ssrc;
/*** RTP data ***/
        buf = RTP_DATA (out);
        len = RtpCompressPt (cxt, buf);
        out->pt = cxt->pt;
        return len +RTP_HDR_BYTES (out);
} /* RtpTranslator */
/*---------------------------------------------------------------------------*/
```

C.6 cSAP

C.6.1 csap.c

```
/*
 *      file:           csap.c
 *      written by:     Christian Zahl
 *      description:    Implements the cSAP (compressed session announcement
 *                      protocol) to be used over PPP
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- user includefiles ---------------------------------------------------*/
#include "misc.h"

/*----- defines -------------------------------------------------------------*/
#define OP_ASSIGN               0x00
#define OP_RETRANSMIT           0x01
#define OP_DELETED              0x02
#define OP_RESET                0x03
#define GLOBAL_CONTEXT_ID       0xffffffff
#define STATE_FREE              0
#define STATE_INUSE             1
#define DEF_TIMEOUT             3600
#define CONTEXTS                256
#define SAP_TRANSPARENT         0
#define SDP_TRANSPARENT         0

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        u32bit                  srcAddr;
        u32bit                  dstAddr;
        u16bit                  srcPort;
        u16bit                  dstPort;
        u8bit                   sapSdp;
} t_Assign;
```

```
typedef struct _t_Cxt {
        int                     state;
        int                     genId;
        u32bit                  srcAddr;
        u32bit                  dstAddr;
        u16bit                  srcPort;
        u16bit                  dstPort;
        int                     sap;
        int                     sdp;
        int                     timeout;
        int                     announcementLen;
        u8bit                   *announcement;
} t_Cxt;

/*----- local functions -------------------------------------------------*/

/*----- global functions ------------------------------------------------*/

/*----- local variables -------------------------------------------------*/
static  t_Cxt                   serverCache[CONTEXTS];
static  t_Cxt                   clientCache[CONTEXTS];

/*----- global variables ------------------------------------------------*/

/*-----------------------------------------------------------------------*/
static void FlushCxt (t_Cxt *cxt)
{
        if (cxt->state == STATE_INUSE)
                free (cxt->announcement);
        cxt->announcement = NULL;
        cxt->state = STATE_FREE;
} /* FlushCxt */
/*-----------------------------------------------------------------------*/
static void FlushClientCache ()
{
        int                     i;

        for (i=0; i<CONTEXTS; i++)
                FlushCxt (&clientCache[i]);
} /* FlushClientCache */
/*-----------------------------------------------------------------------*/
static void FlushServerCache ()
{
        int                     i;

        for (i=0; i<CONTEXTS; i++)
                FlushCxt (&serverCache[i]);
} /* FluchServerCache */
/*-----------------------------------------------------------------------*/
static void SendAnnouncement (t_Cxt *cxt)
/*
 * Sends the announcement in the given context, if we are able to handle
 * it.
 */
{
        if (cxt->sap == SAP_TRANSPARENT && cxt->sdp == SDP_TRANSPARENT) {
;
/*              ... send out the announcement */
        } else {
                ;
        }
} /* SendAnnouncement */
/*-----------------------------------------------------------------------*/
static void SendAssignMsg (t_Cxt *cxt, u32bit cxtLen)
{
} /* SendAssignMsg */
/*-----------------------------------------------------------------------*/
static void SendRetransmitMsg (t_Cxt *cxt, u32bit cxtLen)
{
} /* SendRetransmitMsg */
/*-----------------------------------------------------------------------*/
static void SendDeleteMsg (t_Cxt *cxt, u32bit cxtLen)
{
        int                     i;
        u8bit                   msg[2048];
        int                     n = 0;

        msg[n] = cxt->genId << 5;
.......
} /* SendDeleteMsg */
/*-----------------------------------------------------------------------*/
static void HandleAssignMsg (u8bit *msg, int msgLen, int genId, u32bit cxtId)
/*
 * Assigns the announcement and all the other related data in the clients
 * given context. Also transmit the announcement, if we are able to handle
 * it.
 */
{
        u8bit                   buf[2048];
        t_Assign                *assign = (t_Assign*) buf;
        t_Cxt                   *cxt;
        int                     i;
        int                     now;

/*** check the context-id ***/
        if (cxtId == GLOBAL_CONTEXT_ID)
                return;
/*** save the announcement data in the cache ***/
```

```
        time (&now);
        memcpy (buf, msg, msgLen);      /* don't get a problem on a SUN */
        cxt = &clientCache[cxtId];
        FlushCxt (cxt);
        cxt->state = STATE_INUSE;
        cxt->genId = genId;
        cxt->srcAddr = assign->srcAddr;
        cxt->dstAddr = assign->dstAddr;
        cxt->srcPort = assign->srcPort;
        cxt->dstPort = assign->dstPort;
        cxt->sap = (assign->sapSdp >> 4) &0x0f;
        cxt->sdp = (assign->sapSdp >> 0) &0x0f;
        cxt->timeout = now + DEF_TIMEOUT;
        msgLen -= sizeof (t_Assign);
        msg += sizeof (t_Assign);
        cxt->announcementLen = msgLen;
        cxt->announcement = malloc (msgLen);
        memcpy (cxt->announcement, msg, msgLen);
/*** transmit the announcement, if possible ***/
        SendAnnouncement (cxt);
/*** check the cache for any timouts ***/
        for (i=0; i<CONTEXTS; i++)
                if (clientCache[i].timeout < now);
                        FlushCxt (&clientCache[i]);
} /* HandleAssignMsg */
/*---------------------------------------------------------------------------*/
static void HandleRetransmitMsg (u8bit *msg, int msgLen, int genId, u32bit cxtId)
{
        t_Cxt                   *cxt;

/*** check the context-id ***/
        if (cxtId == GLOBAL_CONTEXT_ID)
                return;
/*** search the context ***/
        cxt = &clientCache[cxtId];
/*** send DELETE if context unknown or genId out of band ***/
        if (!cxt || genId != cxt->genId) {
                if (cxt)
                        FlushCxt (cxt);
                SendDeleteMsg (cxt, cxtId);
                return;
        }
/*** transmit the announcement, if possible ***/
        SendAnnouncement (cxt);
} /* HandleRetransmitMsg */
/*---------------------------------------------------------------------------*/
static void HandleDeletedMsg (u8bit *msg, int msgLen, int genId, u32bit cxtId)
/*
 * Deletes the context of cxtId from the server cache. When the cxtId is the
 * global context-ID, we delete the whole server cache.
 */
{
        int                     i;
        t_Cxt                   *cxt;

/*** check the context-id ***/
        if (cxtId == GLOBAL_CONTEXT_ID) {
                FlushServerCache ();
                return;
        }
/*** search the context ***/
        cxt = &serverCache[cxtId];
        if (cxt->state == STATE_FREE)
                return;
/*** reassign the announcement on the client side ***/
        SendAssignMsg (cxt, cxtId);
} /* HandleDeletedMsg */
/*---------------------------------------------------------------------------*/
void CSAPIRecv (u8bit *msg, int msgLen)
{
        int                     op;
        int                     genId;
        u32bit                  cxtId;

        op = *msg & 0x0F;
        genId = (*msg >> 5) & 0x07;
/*** get the context-ID ***/
        if (*msg++ & 0x10) {
                if (msgLen < 3)
                        return;
                cxtId = *msg++ << 8;
                cxtId |= *msg++;
                if (cxtId == 0xffff)
                        cxtId = GLOBAL_CONTEXT_ID;
                msgLen -= 3;
        } else {
                cxtId = *msg++;
                if (cxtId == 0xff)
                        cxtId = GLOBAL_CONTEXT_ID;
                msgLen  -= 2;
        }
/*** ignore context-ID values too high ***/
        if (cxtId != GLOBAL_CONTEXT_ID && cxtId >= CONTEXTS)
                return;
/*** handle the packet */
        switch (op) {
                case OP_ASSIGN:
                        HandleAssignMsg (msg, msgLen, genId, cxtId);
                        break;
```

```
                case OP_RETRANSMIT:
                        HandleRetransmitMsg (msg, msgLen, genId, cxtId);
                        break;
                case OP_DELETED:
                        HandleDeletedMsg (msg, msgLen, genId, cxtId);
                        break;
                default:
                        break;
        }
} /* CSAPIRecv */
/*---------------------------------------------------------------------------*/
void CSAPInit ()
{
        FlushServerCache ();
        FlushClientCache ();
} /* CSAPInit */
/*---------------------------------------------------------------------------*/
void CSAPIFini ()
{
} /* CSAPIFini */
/*---------------------------------------------------------------------------*/
```

C.7 cSAP

C.7.1 csap.c

```
/*
 *      file:           csap.c
 *      written by:     Christian Zahl
 *      description:    Implements the cSAP (compressed session announcement
 *                      protocol) to be used over PPP
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- user includefiles ---------------------------------------------------*/
#include "misc.h"

/*----- defines -------------------------------------------------------------*/
#define OP_ASSIGN               0x00
#define OP_RETRANSMIT           0x01
#define OP_DELETED              0x02
#define OP_RESET                0x03
#define GLOBAL_CONTEXT_ID       0xffffffff
#define STATE_FREE              0
#define STATE_INUSE             1
#define DEF_TIMEOUT             3600
#define CONTEXTS                256
#define SAP_TRANSPARENT         0
#define SDP_TRANSPARENT         0

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        u32bit                  srcAddr;
        u32bit                  dstAddr;
        u16bit                  srcPort;
        u16bit                  dstPort;
        u8bit                   sapSdp;
} t_Assign;

typedef struct _t_Cxt {
        int                     state;
        int                     genId;
        u32bit                  srcAddr;
        u32bit                  dstAddr;
        u16bit                  srcPort;
        u16bit                  dstPort;
        int                     sap;
        int                     sdp;
        int                     timeout;
        int                     announcementLen;
        u8bit                   *announcement;
} t_Cxt;

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/
static  t_Cxt                   serverCache[CONTEXTS];
static  t_Cxt                   clientCache[CONTEXTS];

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void FlushCxt (t_Cxt *cxt)
{
```

```
        if (cxt->state == STATE_INUSE)
                free (cxt->announcement);
        cxt->announcement = NULL;
        cxt->state = STATE_FREE;
} /* FlushCxt */
/*---------------------------------------------------------------------------*/
static void FlushClientCache ()
{
        int                     i;

        for (i=0; i<CONTEXTS; i++)
                FlushCxt (&clientCache[i]);
} /* FlushClientCache */
/*---------------------------------------------------------------------------*/
static void FlushServerCache ()
{
        int                     i;

        for (i=0; i<CONTEXTS; i++)
                FlushCxt (&serverCache[i]);
} /* FluchServerCache */
/*---------------------------------------------------------------------------*/
static void SendAnnouncement (t_Cxt *cxt)
/*
 * Sends the announcement in the given context, if we are able to handle
 * it.
 */
{
        if (cxt->sap == SAP_TRANSPARENT && cxt->sdp == SDP_TRANSPARENT) {
;
/*              ... send out the announcement */
        } else {
                ;
        }
} /* SendAnnouncement */
/*---------------------------------------------------------------------------*/
static void SendAssignMsg (t_Cxt *cxt, u32bit cxtLen)
{
} /* SendAssignMsg */
/*---------------------------------------------------------------------------*/
static void SendRetransmitMsg (t_Cxt *cxt, u32bit cxtLen)
{
} /* SendRetransmitMsg */
/*---------------------------------------------------------------------------*/
static void SendDeleteMsg (t_Cxt *cxt, u32bit cxtLen)
{
        int                     i;
        u8bit                   msg[2048];
        int                     n = 0;

        msg[n] = cxt->genId << 5;
.......
} /* SendDeleteMsg */
/*---------------------------------------------------------------------------*/
static void HandleAssignMsg (u8bit *msg, int msgLen, int genId, u32bit cxtId)
/*
 * Assigns the announcement and all the other related data in the clients
 * given context. Also transmit the announcement, if we are able to handle
 * it.
 */
{
        u8bit                   buf[2048];
        t_Assign                *assign = (t_Assign*) buf;
        t_Cxt                   *cxt;
        int                     i;
        int                     now;

/*** check the context-id ***/
        if (cxtId == GLOBAL_CONTEXT_ID)
                return;
/*** save the announcement data in the cache ***/
        time (&now);
        memcpy (buf, msg, msgLen);      /* don't get a problem on a SUN */
        cxt = &clientCache[cxtId];
        FlushCxt (cxt);
        cxt->state = STATE_INUSE;
        cxt->genId = genId;
        cxt->srcAddr = assign->srcAddr;
        cxt->dstAddr = assign->dstAddr;
        cxt->srcPort = assign->srcPort;
        cxt->dstPort = assign->dstPort;
        cxt->sap = (assign->sapSdp >> 4) &0x0f;
        cxt->sdp = (assign->sapSdp >> 0) &0x0f;
        cxt->timeout = now + DEF_TIMEOUT;
        msgLen -= sizeof (t_Assign);
        msg += sizeof (t_Assign);
        cxt->announcementLen = msgLen;
        cxt->announcement = malloc (msgLen);
        memcpy (cxt->announcement, msg, msgLen);
/*** transmit the announcement, if possible ***/
        SendAnnouncement (cxt);
/*** check the cache for any timouts ***/
        for (i=0; i<CONTEXTS; i++)
                if (clientCache[i].timeout < now);
                        FlushCxt (&clientCache[i]);
} /* HandleAssignMsg */
/*---------------------------------------------------------------------------*/
static void HandleRetransmitMsg (u8bit *msg, int msgLen, int genId, u32bit cxtId)
{
```

```
        t_Cxt                   *cxt;

/*** check the context-id ***/
        if (cxtId == GLOBAL_CONTEXT_ID)
                return;
/*** search the context ***/
        cxt = &clientCache[cxtId];
/*** send DELETE if context unknown or genId out of band ***/
        if (!cxt || genId != cxt->genId) {
                if (cxt)
                        FlushCxt (cxt);
                SendDeleteMsg (cxt, cxtId);
                return;
        }
/*** transmit the announcement, if possible ***/
        SendAnnouncement (cxt);
} /* HandleRetransmitMsg */
/*---------------------------------------------------------------------------*/
static void HandleDeletedMsg (u8bit *msg, int msgLen, int genId, u32bit cxtId)
/*
 * Deletes the context of cxtId from the server cache. When the cxtId is the
 * global context-ID, we delete the whole server cache.
 */
{
        int                     i;
        t_Cxt                   *cxt;

/*** check the context-id ***/
        if (cxtId == GLOBAL_CONTEXT_ID) {
                FlushServerCache ();
                return;
        }
/*** search the context ***/
        cxt = &serverCache[cxtId];
        if (cxt->state == STATE_FREE)
                return;
/*** reassign the announcement on the client side ***/
        SendAssignMsg (cxt, cxtId);
} /* HandleDeletedMsg */
/*---------------------------------------------------------------------------*/
void CSAPIRecv (u8bit *msg, int msgLen)
{
        int                     op;
        int                     genId;
        u32bit                  cxtId;

        op = *msg & 0x0F;
        genId = (*msg >> 5) & 0x07;
/*** get the context-ID ***/
        if (*msg++ & 0x10) {
                if (msgLen < 3)
                        return;
                cxtId = *msg++ << 8;
                cxtId |= *msg++;
                if (cxtId == 0xffff)
                        cxtId = GLOBAL_CONTEXT_ID;
                msgLen -= 3;
        } else {
                cxtId = *msg++;
                if (cxtId == 0xff)
                        cxtId = GLOBAL_CONTEXT_ID;
                msgLen  -= 2;
        }
/*** ignore context-ID values too high ***/
        if (cxtId != GLOBAL_CONTEXT_ID && cxtId >= CONTEXTS)
                return;
/*** handle the packet */
        switch (op) {
                case OP_ASSIGN:
                        HandleAssignMsg (msg, msgLen, genId, cxtId);
                        break;
                case OP_RETRANSMIT:
                        HandleRetransmitMsg (msg, msgLen, genId, cxtId);
                        break;
                case OP_DELETED:
                        HandleDeletedMsg (msg, msgLen, genId, cxtId);
                        break;
                default:
                        break;
        }
} /* CSAPIRecv */
/*---------------------------------------------------------------------------*/
void CSAPInit ()
{
        FlushServerCache ();
        FlushClientCache ();
} /* CSAPInit */
/*---------------------------------------------------------------------------*/
void CSAPIFini ()
{
} /* CSAPIFini */
/*---------------------------------------------------------------------------*/
```

C.8 Filehandler

C.8.1 filehandler.h

```
/*
 *      file:           filehandler.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id$
 *
 * $Log$
 */

#ifndef INCL_FILEHANDLER_H
#define INCL_FILEHANDLER_H

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/
#define FILEHANDLER_READ                1
#define FILEHANDLER_WRITE               2
#define FILEHANDLER_EXCEPTION           4

/*----- type definitions --------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

#endif /* INCL_FILEHANDLER */
```

C.8.2 filehandler.c

```
/*
 *      file:           filehandler.c
 *      written by:     Christian Zahl
 *      description:    Impelements a filehander concept
 *
 * $Id$
 *
 * $Log$
 */
static  char                    rcsid[] = "$Id$\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <errno.h>
#include <sys/select.h>

/*----- user includefiles -------------------------------------------------*/
#include "filehandler.h"

/*----- defines -----------------------------------------------------------*/
#define CB_LIST                 32

/*----- type definitions --------------------------------------------------*/
typedef struct {
        void                    (*cb)();
        void                    *data;
        int                     mode;
} t_Cb;

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/
static  int                     stop = 0;
static  t_Cb                    cbList[CB_LIST];
static  fd_set                  readSet;

/*----- global variables --------------------------------------------------*/

/*-------------------------------------------------------------------------*/
void FileHandlerFdSet ()
{
        int                     i;

        FD_ZERO (&readSet);
        for (i=0; i<CB_LIST; i++)
                if (cbList[i].cb)
                        if (cbList[i].mode & FILEHANDLER_READ)
                                FD_SET (i, &readSet);
} /* FileHandlerFdSet */
/*-------------------------------------------------------------------------*/
void FileHandlerInit ()
{
        int                     i;

        for (i=0; i<CB_LIST; i++) {
```

```
                cbList[i].cb = NULL;
                cbList[i].data = NULL;
                cbList[i].mode = 0;
        }
        FileHandlerFdSet ();
        stop = 0;
} /* FileHandlerInit */
/*--------------------------------------------------------------------------*/
void FileHandlerStop ()
{
        stop = 1;
} /* FileHandlerStop */
/*--------------------------------------------------------------------------*/
void FileHandlerFini ()
{
        FileHandlerStop ();
} /* FileHandlerFini */
/*--------------------------------------------------------------------------*/
FileHandlerCreate (int fd, int mode, void (*cb)(), void *data)
{
        if (fd < 0)
                return;
        if (fd >= CB_LIST)
                return;
        if (mode == FILEHANDLER_READ) {
                cbList[fd].cb = cb;
                cbList[fd].data = data;
                cbList[fd].mode = mode;
                FileHandlerFdSet ();
        }
} /* FileHandlerCreate */
/*--------------------------------------------------------------------------*/
FileHandlerDelete (int fd)
{
        if (fd < 0)
                return;
        if (fd >= CB_LIST)
                return;
        cbList[fd].cb = NULL;
        FileHandlerFdSet ();
} /* FileHandlerDelete */
/*--------------------------------------------------------------------------*/
void FileHandler ()
{
        fd_set                  tmpReadSet;
        int                     i;

        stop = 0;
        while (!stop) {
                memcpy (&tmpReadSet, &readSet, sizeof (tmpReadSet));
                i = select (CB_LIST, &tmpReadSet, NULL, NULL, NULL);
                if (i == -1 && errno == EINTR)
                        continue;
                for (i=0; i<CB_LIST; i++)
                        if (FD_ISSET (i, &tmpReadSet))
                                (cbList[i].cb) (i, 0, cbList[i].data);
        } /* while */
} /* FileHandler */
/*--------------------------------------------------------------------------*/
```

C.9 Sonstige Files

C.9.1 misc.h

```
/*
 *      file:           misc.h
 *      written by:     Christian Zahl
 *      description:    Defines for accessing IP, UDP and RTP headers
 *
 * $Id$
 *
 * $Log$
 */

#ifndef INCL_MISC
#define INCL_MISC

/*----- standard includefiles ----------------------------------------------*/

/*----- local includefiles -------------------------------------------------*/

/*----- defines ------------------------------------------------------------*/
#define IP_HDR_BYTES(xxx)       ((xxx)->ip_hl*4)
#define IP_DATA(xxx)            (((unsigned char*)(xxx))+(xxx)->ip_hl*4)

#define UDP_HDR_BYTES(xxx)      (8)
#define UDP_DATA(xxx)           (((unsigned char*)xxx)+UDP_HDR_BYTES(xxx))
        /*** handling of RTP header fields ***/
#define RTP_CSRC_BYTES(xxx)     ((xxx)->cc*4)
#define RTP_CSRC(xxx)           ((struct rtpCsrc*)(((unsigned char*)xxx)+12))
#define RTP_EXT(xxx)            ((struct rtpExt*)(((unsigned char*)xxx)+12+RTP_CSRC_BYTES(xxx)))
#define RTP_EXT_DATA(xxx)       ((((unsigned char*)xxx)+12+RTP_CSRC_BYTES(xxx)))
#define RTP_EXT_BYTES(xxx)      ((xxx)->x?RTP_EXT(xxx)->xlen:0)
```

```
#define RTP_HDR_BYTES(xxx)      (12+(xxx)->cc*4+RTP_EXT_BYTES(xxx))
#define RTP_DATA(xxx)           (((unsigned char*)xxx)+RTP_HDR_BYTES(xxx))

#define RTP_CSRC_SET(rtp,csrc,ii) \
                                {rtp[12+ii*4]=(csrc)>>24;rtp[13+ii*4]=(csrc)>>16;\
                                rtp[14+ii*4]=(csrc)>>8;rtp[15+ii*4]=(csrc)>>0;}


/*----- type definitions ----------------------------------------------------*/
typedef unsigned char           u8bit;
typedef unsigned short          u16bit;
typedef unsigned int            u32bit;

/*----- global functions ----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/


#endif /* INCL_MISC */
```

C.9.2 rtp.h

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           rtp.h
 *      written by:     Christian Zahl
 *      description:    Definition of RTP header
 *
 * $Id$
 *
 * $Log$
 */

#ifndef INCL_RTP_H
#define INCL_RTP_H

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/

/*----- defines -------------------------------------------------------------*/
#define RTP_PT_PCMU             0
#define RTP_PT_GSM              3
#define RTP_PT_LPC              7
#define RTP_PT_PCMA             8

/*----- type definitions ----------------------------------------------------*/
struct rtp {
        unsigned int            v:2;
        unsigned int            p:1;
        unsigned int            x:1;
        unsigned int            cc:4;
        unsigned int            m:1;
        unsigned int            pt:7;
        unsigned int            seq:16;
        unsigned int            timestamp;
        unsigned int            ssrc;
};
struct rtpCsrc {
        unsigned int            csrc;
};

struct rtpExt {
        unsigned short          type;
        unsigned short          xlen;
};

/*----- global functions ----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_RTP_H */
```

C.9.3 tabxlaw.h

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           tabxlaw.h
 *      written by:     Christian Zahl
 *      description:    Defines four tables and some macros for converting
 *                      alaw and ulaw to linear and vice versa.
 *
 * $Id$
 *
 * $Log$
 */

#ifndef INCL_TABXLAW
#define INCL_TABXLAW

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/
```

```
/*----- defines -------------------------------------------------------------*/
#define ULAW_TO_LIN(x)          (tabulaw2lin[(x)])
#define LIN_TO_ULAW(x)          (tablin2ulaw[(int)(x)+32767])
#define ALAW_TO_LIN(x)          (tabalaw2lin[(x)])
#define LIN_TO_ALAW(x)          (tablin2alaw[(int)(x)+32767])

/*----- type definitions ----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/
extern  const short             tabulaw2lin[256];
extern  const unsigned char     tablin2ulaw[65536];
extern  const short             tabalaw2lin[256];
extern  const unsigned char     tablin2alaw[65536];

#endif /* INCL_TABXLAW */
```

C.9.4 tabxlaw.c

Diese Datei enthällt vier Tabellen, die die Umwandlung zwischen linearem PCM, PCM μ-law und PCM A-law ermöglichen. Wir haben diese NICHT ausgedruckt, da diese ca. 16000 Zeilen lang ist!

C.9.5 Makefile

```
CFLAGS=-O2 -g -I/usr/local/include

.SUFFIXES:
.DEFAULT:


OBJ_DIR=$(ARCH)

OBJ=    $(OBJ_DIR)/server.o \
        $(OBJ_DIR)/hdlc.o \
        $(OBJ_DIR)/ppp.o \
        $(OBJ_DIR)/filehandler.o \
        $(OBJ_DIR)/ipUdpRtpCompr.o \
        $(OBJ_DIR)/rtpTranslator.o \
        $(OBJ_DIR)/tabxlaw.o \
        $(OBJ_DIR)/lpc.o \
        $(NULL)

server: $(OBJ_DIR)/server

sun5/server:    $(OBJ)
        $(CC) -g $(OBJ) -o$(OBJ_DIR)/server -L. -L/usr/local/lib -lgsm -lnsl -lsocket -lm

aix4/server:    $(OBJ) $(OBJ_DIR)/modem.o
        $(CC) -g $(OBJ) $(OBJ_DIR)/modem.o -o$(OBJ_DIR)/server -L/usr/local/lib -lgsm -lm

test: $(OBJ_DIR)/test

aix4/test:      $(OBJ_DIR)/test.o $(OBJ_DIR)/ipUdpRtpCompr.o
        $(CC) -g $(OBJ_DIR)/test.o $(OBJ_DIR)/ipUdpRtpCompr.o -o $(OBJ_DIR)/test

sun5/test:      $(OBJ_DIR)/test.o $(OBJ_DIR)/ipUdpRtpCompr.o
        $(CC) -g $(OBJ_DIR)/test.o $(OBJ_DIR)/ipUdpRtpCompr.o -o $(OBJ_DIR)/test

###############################################################################
$(OBJ_DIR)/server.o:            server.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/hdlc.o:                      hdlc.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/ppp.o:                       ppp.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/filehandler.o:               filehandler.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/ipUdpRtpCompr.o:     ipUdpRtpCompr.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/rtpMixer.o:          rtpMixer.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/rtpTranslator.o:             rtpTranslator.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/tabxlaw.o:           tabxlaw.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/modem.o: modem.c
        $(CC) -c $(CFLAGS) $? -o $@

$(OBJ_DIR)/test.o: test.c
        $(CC) -c $(CFLAGS) $? -o $@
```

```
$(OBJ_DIR)/lpc.o: lpc.c
        $(CC) -c $(CFLAGS) $? -o $@
```

Anhang D

Manualpages für die IsdnLib

In den folgenden Abschnitten beschreiben wir die Funktionen der IsdnLib. Dabei werden die Funktionen einzeln in der Art von Manualpages beschrieben.

D.1 IsdnInitialize

ZWECK

Initialisiert die IsdnLib

SYNTAX

```
#include <IsdnLib.h>

int IsdnInitialize (void (*callback)());
```

BESCHREIBUNG

`IsdnInitialize` wird benutzt um die IsdnLib zu initialisieren und den Q.931 Stack mit dem DLPI Interface zu verbinden. Sie muß vor allen anderen Funktionen der Library aufgerufen werden, oder diese liefern einen Fehler.

`callback` ist eine Callback Funktion, die die Applikation angeben muß. Sobald eine Q.931 Nachricht eingeht, die für die Applikation von Interesse sein könnte, wird diese Funktion aufgerufen.

Nachdem die Library initialisiert wurde, können andere IsdnLib Funktionen benutzt werden, um das ISDN zu benutzen.

Bevor die Applikation terminiert, sollte diese `IsdnFini()` aufrufen, um alle belegten Ressourcen wieder freizugeben.

DEBUGGING

Es existiert ein Debugging Support, falls dieser bei der Generierung der Library nicht ausgeschaltet wurde. Durch Setzen der Environment Variablen `ISDN_DEBUG` kann angegeben werden, welche Informationen ausgegeben werden sollen. Der angegebene Wert spezifiziert eine Bitmaske, die durch Angabe des Prefixes `0x` auch als Hexadezimalwert angegeben werden. Folgende Werte sind definiert:

```
0x0001 Ausgabe von Informationen des Schedullers
0x0002 Ausgabe der IsdnLib CallId
0x0008 Ausgabe von Timerinformationen
```

```
0x0010 Anzeige der ein- und ausgehenden Q.931 Nachrichten
0x0020 Anzeige der ein- und ausgehenden Q.921 Nachrichten
0x0040 Anzeige interner Q.931 Requests und Indikationen
0x1000 Trace des Beenden einer Funktion
0x2000 Trace jedes Funktionsaufrufs
0x4000 Ausgabe der Warnungen
0x8000 Ausgabe der Fehler
```

RÜCKGABEWERT

`IsdnInitialize` liefert 0 wenn die Initialisierung erfolgreich war, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

FEHLER

`EISDN_SYSCALL`
Der Aufruf eines Systemcalls lieferte einen unerwarteten Fehler.

BEISPIEL

```
#include <IsdnLib.h>

void            (*IsdnCallback) (t_IsdnInd *ind);

main ()
{
        /* initalize the application */
        if (IsdnInitialize (IsdnCallback)) < 0) {
                /* handle error */
                exit (1);
        }
        IsdnMainLoop ();
                ...
        IsdnFini ();
        exit (0);
}
```

D.2 IsdnFini

ZWECK

Freigabe aller Ressourcen, die von der IsdnLib benutzt wurden

SYNTAX

```
#include <IsdnLib.h>

void IsdnFini ();
```

BESCHREIBUNG

`IsdnFini` beendet die IsdnLib und gibt alle belegten Ressourcen wieder frei.

D.3 IsdnMainLoop

ZWECK

Startet die Mainloop der IsdnLib

SYNTAX

```
#include <IsdnLib.h>

int IsdnMainLoop ();
```

BESCHREIBUNG

`IsdnMainLoop` wird verwendet, um die Mainloop der IsdnLib zu starten. Bevor diese nicht gestartet wurde ist es nicht möglich, Q.931 Nachrichten zu empfangen oder ordnungsgemäß zu versenden.

Da jederzeit, nachdem das ISDN Device aktiviert wurde, Q.931 Nachrichten eintreffen können, sollte `IsdnMainLoop` relativ schnell nach `IsdnInitialize` aufgerufen werden. Anderenfalls kann es passieren, das noch alte, nicht behandelte Nachrichten in der Queue stehen, die dann zu Verwirrungen führen könnten.

Der Aufruf von `IsdnMainLoop` kehrt erst zurück, wenn dies ausdrücklich durch Setzen der globalen Variable `schedulerStop` auf einen Wert ungleich 0.

RÜCKGABEWERT

`IsdnInitialize` liefert 0 wenn die Mainloop erfolgreich gestartet werden konnte, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
#include <IsdnLib.h>

void            (*IsdnCallback) (t_IsdnInd *ind);

main ()
{
        ...
        IsdnInitialize (IsdnCallback);
        IsdnMainLoop ();
        IsdnFini ();
        exit (0);
}
```

D.4 IsdnError

ZWECK

Formatierte Ausgabe des Wertes der Variable `isdnErrno`

SYNTAX

```
#include <IsdnLib.h>

void IsdnError (const char *str, ...);
```

BESCHREIBUNG

`IsdnError` gibt den Wert der globalen Fehlervariablen `isdnErrno` formatiert auf `stderr` aus. Als Parameter erwartet die Funktion eine variable Anzahl von Argumenten, analog zu `printf`.

BEISPIEL

```
...
IsdnError ("Error occured at %s, in line %d", ctime (&now), __LINE__);
...
```

D.5 IsdnCreateFileHandler

ZWECK

Registriert einen Filehandler

SYNTAX

```
#include <IsdnLib.h>

void cb (int mode, void *data);
int IsdnCreateFileHandler (int fd, int mode, void (*cb) (), void *data);
```

BESCHREIBUNG

Mit `IsdnCreateFileHandler` registriert die Applikation einen Filehandler (`cb`) für den angegebenen Filedescriptor `fd`. Dieser wird aufgerufen, sobald einer der in `mode` angegebenen Ereignisse eintrifft. Folgende Ereignisse können spezifiziert werden:

```
ISDN_READABLE
ISDN_WRITABLE
ISDN_EXCEPTION
```

Ein installierter Filehandler kann mit `IsdnDeleteFileHandler` jederzeit wieder gelöscht werden.

RÜCKGABEWERT

`IsdnInitialize` liefert 0 wenn der Filehandler erfolgreich installiert werden konnte, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
#include <IsdnLib.h>

Echo (void *data, int mode)
{
        ...
}
main ()
{
        ...
        IsdnCreateFileHandler (0, ISDN_READABLE, Echo, NULL);
        ...
}
```

D.6 IsdnDeleteFileHandler

ZWECK

Löscht einen zuvor installierten Filehandler

SYNTAX

```
#include <IsdnLib.h>

void IsdnDeleteFileHandler (int fd);
```

BESCHREIBUNG

`IsdnDeleteFileHandler` löscht den zuvor für den Filedescriptor `fd` installierten Filehandler.

BEISPIEL

```
#include <IsdnLib.h>

Echo (void *data, int mode)
{
        ...
}
main ()
{
        ...
        IsdnCreateFileHandler (0, ISDN_READABLE, Echo, NULL);
        ...
        IsdnDeleteFileHandler (0);
}
```

D.7 IsdnCreateTimerHandler

ZWECK

Startet einen Timer und installiert einen Timerhandler

SYNTAX

```
#include <IsdnLib.h>

void cb (int mode, void *data);
int IsdnCreateTimerHandler (int ms, void (*cb) (), void *data);
```

BESCHREIBUNG

`IsdnCreateTimerHandler` startet einen Timer mit `ms` Millisekunden. Sollte der Timer ablaufen, so wird der registrierte Callback `cb` aufgerufen.

Ein
gestarteter Timer und der installierte Timerhandler können mit `IsdnDeleteTimerHandler` jederzeit wieder gelöscht werden.

RÜCKGABEWERT

Wurde der Timer erfolgreich installiert, liefert `IsdnCreateTimerHandler` eine TimerId für den Timer, dessen Wert größer 0 ist, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
#include <IsdnLib.h>

Beep (void *data, int mode)
{
        ...
}
main ()
{
        ...
/* create timer for one second ***/
        IsdnCreateTimerHandler (1000, Echo, NULL);
        ...
}
```

D.8 IsdnDeleteTimerHandler

ZWECK

Stoppt einen Timer und den installierten Timerhandler

SYNTAX

```
#include <IsdnLib.h>

void IsdnDeleteTimerHandler (int timerId);
```

BESCHREIBUNG

`IsdnDeleteTimerHandler` stoppt einen zuvor gestarteten Timer `timerId` und löscht den installierten TimerHandler.

BEISPIEL

```
#include <IsdnLib.h>

void Cb ();
main ()
{
        ...
/* create timer for one second ***/
        timerId = IsdnCreateTimerHandler (1000, Cb, NULL);
        ...
        IsdnDeleteTimerHandler (timerId);
        ...
}
```

D.9 IsdnAlertingReq

ZWECK

Signalisierung der Bereitschaft, den eingehenden Anruf entgegenzunehmen.

SYNTAX

```
#include <IsdnLib.h>

int IsdnAlertingReq (const t_IsdnAlertingReq *ireq)
```

BESCHREIBUNG

Die IsdnLib signalisiert einen eingehenden Anruf durch eine `IsdnCallInd`. Durch Aufruf von `IsdnAlertingReq` signalisiert die Applikation ihre Bereitschaft, den Anruf eventuell entgegenzunehmen. Handelt es sich um einen Telefonanruf, wird beim Anrufer hierdurch das Klingelzeichen generiert.

Der Parameter `ireq` definiert weitere Parameter für den Request:

```
typedef struct {
        int                     callId;                 /* req. */
} t_IsdnAlertingReq;
```

callId

Referenziert die logische Verbindung.

Wenn die Applikation den Anruf nicht behandeln möchte, sollte sie den Anruf mit einem `IsdnReleaseReq` beenden. Wird auf einen eingehenden Anruf nicht innerhalb von 8 Sekunden reagiert, generiert die IsdnLib selbst eine `IsdnReleaseInd`.

RÜCKGABEWERT

`IsdnAlertingReq` liefert 0 wenn der Request erfolgreich bearbeitet wurde, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

FEHLER

EISDN_INV_CALLID

Die in `ireq` angegebene `callId` konnte keiner logischen Verbindung zugeordnet werden.

EISDN_Q931

Es wurde eine unerwartete Q.931 Nachricht empfangen, die nicht der Protokollspezifikation entspricht.

BEISPIEL

```
HandleIncommingCall (int callId)
{
        t_IsdnAlertingReq       req;

        ...                             /* check the incomming call */
        req.callId = callId;
        if (IsdnAlertingReq (&req))
                IsdnError ("Error in IncommingCall");
        else
                ...                     /* ok */
}
```

D.10 IsdnCallReq

ZWECK

Initiiert einen ausgehenden Anruf

SYNTAX

```
#include <IsdnLib.h>

int IsdnCallReq (t_IsdnCallReq *req)
```

BESCHREIBUNG

`IsdnCallReq` wird benutzt um einen ausgehenden Anruf zu initiieren. Die IsdnLib erzeugt eine neue Q.931 CallReference Value und versucht den Anruf zu tätigen.

Der Parameter `ireq` definiert weitere Parameter für den Request:

```
typedef struct {
        int                     callId;                 /* n/a */
        t_CalledPartyNo         calledNo;               /* opt. no to be called */
        t_CalledPartySubAddr    calledSubAddr;          /* opt. subAddr to be called */
        t_CallingPartyNo        callingNo;              /* opt. calling no */
        t_CallingPartySubAddr   callingSubAddr;         /* opt. calling subAddr */
        t_IsdnService           service;                /* opt. complete service */
        t_BearerCap             bearerCap[2];           /* opt. bearer cap */
        t_HighLayerComp         highLayerComp[4];       /* opt. HLC */
        t_LowLayerComp          lowLayerComp[4];        /* opt. LLC */
} t_IsdnCallReq;
```

Bedeutung der Felder:

callId
: Hat beim Aufruf keine Bedeutung. Wenn der Request erfolgreich bearbeitet wurde, enthält dieses Feld die `callId` für diese logische Verbindung, die von der IsdnLib vergeben wurde.

calledNo
: Spezifiziert die anzurufende Nummer. Wurde die Nummer nicht angegeben, so kann die Applikation im *Overlap Sending* Modus die Nummer auch mit Hilfe von `IsdnDialReq` Nachrichten angegeben.

calledSubAddr
: Spezifiziert die Subadresse der angerufenen Nummer.

callingNo
: Spezifiziert die Nummer des Anrufers, falls angegeben. In vielen Fällen kommt es vor, daß das zwischenliegende Netz diese überprüft und gegebenenfalls ändert.

callingSubAddr
: Spezifiziert die Subadresse des Anrufers, falls angegeben.

service
: Definiert den Dienst, mit der der Anruf initiiert werden soll, falls angegeben. Mit diesem Feld ist es möglich, den Dienst auf einfache Art zu definieren:

  ISDN_S_VOICE_ALAW
  : Telefonie mit 3.1 kHz, PCM A-law kodiert

  ISDN_S_VOICE_MULAW
  : Telefonie mit 3.1 kHz, PCM μ-law kodiert

  Wurde dieses Feld nicht angegeben, so muß der Dienst mit Hilfe der folgenden Felder spezifiziert werden. Anderenfalls haben diese Felder keine Bedeutung.

bearerCap
: Definiert die zu verwenden *Bearer Capability*, analog zu Q.931.

highLayerComp
: Definiert die zu verwenden *High Layer Compatibility*, analog zu Q.931.

lowLayerComp
: Definiert die zu verwenden *Low Layer Compatibility*, analog zu Q.931.

Die erfolgreiche Benutzung dieses Requests bedeutet nicht, daß hierdurch eine Verbindung aufgebaut wurde, der Aufbau wurde nur initiiert. Die Verbindung ist erst dann vollständig aufgebaut, wenn die IsdnLib eine `IsdnConnectInd` generiert.

Der initiierte Anruf kann jederzeit durch Verwendung von `IsdnReleaseReq` abgebrochen werden.

RÜCKGABEWERT

`IsdnCallReq` liefert 0 wenn der Request erfolgreich bearbeitet wurde, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
InitiateCall ()
{
        t_IsdnCallReq           req;

        memset (&req, '\0', sizeof (req));
        strcpy (req.calledNo.number, "+493025499203");
        strcpy (req.callingNo.number, "+4930999888777");
        req.service = ISDN_S_VOICE_ALAW;
        if (IsdnCallReq (&req))
                IsdnError ("Error in InitiateCall");
        else
                return req.callId;              /* return the callId */
}
```

D.11 IsdnConnectReq

ZWECK

Annahme eines eingehenden Anrufes.

SYNTAX

```
#include <IsdnLib.h>

int IsdnConnectReq (const t_IsdnConnectReq *ireq)
```

BESCHREIBUNG

`IsdnConnectReq` nimmt einen eingegangenen Anruf entgegen. Bevor der Anruf entgegengenommen werden kann, muß die Applikation die Bereitschaft zur Annahme mit `IsdnAlertingReq` signalisiert haben..

Der Parameter `ireq` definiert weitere Parameter für den Request:

```
typedef struct {                                /* connect to incomming call */
        int                     callId;         /* req. */
        t_IsdnService           service;        /* opt. short cut service */
        t_BearerCap             bearerCap;      /* opt. selected bearer cap */
        t_HighLayerComp         highLayerComp;  /* opt. selected HLC */
        t_LowLayerComp          lowLayerComp;   /* opt. selected LLC */
} t_IsdnConnectReq;
```

Bedeutung der Felder:

callId

Identifiziert die logische Verbindung.

service
: Definiert den angenommenen Dienst, falls angegeben. Im ISDN ist es möglich, daß der Anrufer mehrere mögliche Dienste angibt. Der Angerufene muß aus dieser Liste einen auswählen, den er am besten behandeln kann. Ist dieser Parameter nicht angegeben, so muß der ausgewählte Dienst über die folgenden Felder spezifiziert werden. Ansonsten haben diese keine Bedeutung.

bearerCap
: Definiert die zu verwenden *Bearer Capability*, analog zu Q.931.

highLayerComp
: Definiert die zu verwenden *High Layer Compatibility*, analog zu Q.931.

lowLayerComp
: Definiert die zu verwenden *Low Layer Compatibility*, analog zu Q.931.

Auch nach erfolgreicher Benutzung dieses Requests ist die Verbindung noch nicht hergestellt! Erst nachdem die IsdnLib eine `IsdnConnectInd` signalisiert, wurde die Verbindung aufgebaut.

RÜCKGABEWERT

`IsdnCallReq` liefert 0 wenn der Request erfolgreich bearbeitet wurde, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
ConnectToCaller (int callId, int service)
{
        t_IsdnConnectReq        req;

        memset (&req, '\0', sizeof (req));
        req.callId = callId;
        req.service = service;
        if (IsdnConnectReq (&req))
                IsdnError ("Error in ConnectToCaller");
        else
                ...                     /* ok */
}
```

D.12 IsdnDialReq

ZWECK

Angabe der zu wählenden Nummer bei *Overlap Sending*.

SYNTAX

```
#include <IsdnLib.h>

int IsdnDialReq (const t_IsdnDialReq *req)
```

BESCHREIBUNG

`IsdnDialReq` überträgt Wählinformationen im *Overlap Sending* Modus. Es ist somit möglich, wie bei einem normalen Telefon zu wählen (Abnahme des Höhrers, Warten auf das Freizeichen, Wählen).

Der Parameter `ireq` definiert weitere Parameter für den Request:

```
typedef struct {
        int                     callId;                 /* req. */
        t_CalledPartyNo         calledPartyNo;          /* req. */
} t_IsdnDialReq;
```

Bedeutung der Felder:

callId
: Identifiziert die logische Verbindung.

calledPartyNo
: Gibt die zu übermittelnde Wählinformation an.

RÜCKGABEWERT
: `IsdnDialReq` liefert 0 wenn der Request erfolgreich bearbeitet wurde, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
Dial (int callId, char *number)
{
        t_IsdnDialReq           req;

        memset (&req, '\0', sizeof (req));
        req.callId = callId;
        strcpy (req.calledPartyNo.number, number);
        if (IsdnCallReq (&req))
                IsdnError ("Error in Dial");
        else
                ...                     /* OK */
}
```

D.13 IsdnKeypadReq

ZWECK
: Übermittlung von Zusatzinformationen nach Q.932 [Int93e]

SYNTAX

```
#include <IsdnLib.h>

int IsdnKeypadReq (const t_IsdnKeypadReq *req)
```

BESCHREIBUNG
: `IsdnKeypadReq` übermittelt Zusatzinformationen, mit deren Hilfe weitere Features im ISDN realisiert werden können (z.B. Markeln). Nicht alle Netzwerkbetreiber unterstützen dieses Feature.

Der Parameter `ireq` definiert weitere Parameter für den Request:

```
typedef struct {
        int                     callId;                 /* req. */
        char                    keypad[20+1];           /* req. 1..20 chars, \0 terminate
} t_IsdnKeypadReq;
```

Bedeutung der Felder:

callId
: Identifiziert die logische Verbindung.

keypad
: Gibt zu übermittelnde Informationen an. Die Bedeutung dieser liegt außerhalb der Q.931 Protokollspezifikation. Der Benutzer muß sich daher beim ISDN Dienstanbieter nach den unterstützten Features erkundigen.

RÜCKGABEWERT

`IsdnKeypadReq` liefert 0 wenn der Request erfolgreich bearbeitet wurde, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
SendKeypad (int callId, char *keypad)
{
        t_IsdnKeypadReq         req;

        memset (&req, '\0', sizeof (req));
        req.callId = callId;
        strcpy (req.keypad, keypad);
        if (IsdnKeypadReq (&req))
                IsdnError ("Error in SendKeypad");
        else
                ...                     /* OK */
}
```

D.14 IsdnReleaseReq

ZWECK

Beendet eine logische Verbindung

SYNTAX

```
#include <IsdnLib.h>

int IsdnReleaseReq (const t_IsdnReleaseReq *req)
```

BESCHREIBUNG

`IsdnReleaseReq` initiiert den Abbruch einer bestehenden oder im Aufbau befindlichen Verbindung.

Der Parameter `ireq` definiert weitere Parameter für den Request:

```
typedef struct {
        int                     callId;                 /* req. */
        t_Cause                 cause;                  /* opt. reason */
} t_IsdnReleaseReq;
```

Bedeutung der Felder:

callId
: Identifiziert die logische Verbindung.

cause
: Spezifiziert einen Grund für den Verbindungsabbruch. Die Bedeutung der verschieden Werte sind der ITU Empfehlung Q.850 [Int93p] zu entnehmen.

Es ist zu beachten, daß die Verbindung erst dann vollständig beendet ist, wenn die IsdnLib eine `IsdnReleaseInd` generiert.

RÜCKGABEWERT

`IsdnReleaseReq` liefert 0 wenn der Request erfolgreich bearbeitet wurde, -1 anderenfalls. Der Fehler wird dann in der Variablen `IsdnErrno` gespeichert.

BEISPIEL

```
Disconnect (int callId, int cause)
{
        t_IsdnReleaseReq        req;

        memset (&req, '\0', sizeof (req));
        req.callId = callId;
        req.cause.value = cause;
        req.cause.location = CA_L_USER;
        if (IsdnReleaseReq (&req))
                IsdnError ("Error in Disconnect");
        else
                ...                     /* OK */
}
```

D.15 IsdnAlertInd

ZWECK

Signalisiert, daß der Anrufer prinzipiell bereit ist, den Anruf entgegenzunehmen

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG

Die registrierte Callback Funktion wird mit einer `IsdnAlertInd` Indikation von der IsdnLib aufgerufen, wenn der zuvor Angerufene bereit ist, den Anruf entgegenzunehmen. Sie ersetzt somit das „Klingelzeichen“ beim alten Telefon.

Der Parameter `t_IsdnAlertInd` liefert weitere Informationen über die Indikation:

```
typedef struct {                                /* alerting indication */
        t_IsdnIndType           indType;        /* type of inication */
        int                     callId;         /* unique ID of the call */
} t_IsdnAlertInd;
```

Bedeutung der Felder:

indType
: ISDN_ALERT_IND

callId
: Identifiziert die logische Verbindung, für die das Klingeln signalisiert wird.

BEISPIEL

```
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnAlertInd          *alert = (t_IsdnAlertInd*)ind;

        if (ind->indType == ISDN_ALERT_IND) {
                printf ("Alerting for %d\n", alert->callId);
        }
}
```

D.16 IsdnCallInd

ZWECK

Signalisiert einen eingehenden Anruf

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG

Die registrierte Callback Funktion wird mit einer `IsdnCallInd` Indikation von der IsdnLib aufgerufen, wenn ein neuer Anruf eingegangen ist. Die Applikation muß dann entscheiden, ob sie bereit ist, diesen zu empfangen. Falls ja, sollte sie mit dies mit Hilfer eines `IsdnAlertingReq` signalisieren. Ansonsten sollte der Anruf mit `IsdnReleaseReq` abgelehnt werden.

Der Parameter `t_IsdnCallInd` liefert weitere Informationen über die Indikation:

```
typedef struct {
        t_IsdnIndType           indType;                /* type of inication */
        int                     callId;                 /* unique ID of the call */
        t_CalledPartyNo         calledPartyNo;
        t_CalledPartySubAddr    calledPartySubAddr;
        t_CallingPartyNo        callingPartyNo;
        t_CallingPartySubAddr   callingPartySubAddr;
        t_ChannelId             channelId;              /* identifies the B* channels */
        t_BearerCap             bearerCap[2];
        t_HighLayerComp         highLayerComp[4];
        t_LowLayerComp          lowLayerComp[4];
} t_IsdnCallInd;
```

Bedeutung der Felder:

indType

ISDN_CALL_IND

callId

Identifiziert die logische Verbindung für diesen neuen Anruf.

calledPartyNo

Gibt die angerufene Nummer an.

calledPartySubAddr

Gibt die angerufene Subadresse an.

callingPartyNo

Gibt die Nummer des Anrufer an, falls diese übertragen wurde.

callingPartySubAddr
: Gibt die Subadresse des Anrufer an, falls diese übertragen wurde.

channelId
: Gibt die zu verwendenden ISDN Kanäle an, die für diese Verbindung reserviert wurden.

bearerCap
: Gibt die angebotenen *Bearer Capabilities* des Anrufers an.

highLayerComp
: Gibt die *High Layer Compatibility* an, falls vorhanden.

lowLayerComp
: Gibt die *Low Layer Compatibility* an, falls vorhanden.

BEISPIEL

```
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnCallInd           *call = (t_IsdnCallInd*)ind;

        if (ind->indType == ISDN_CALL_IND) {
                printf ("Incomming CALL for %d\n", call->callId);
                printf ("  called no  = %s\n", call->calledPartyNo.number);
                printf ("  calling no = %s\n", call->callingPartyNo.number);
                ...
        }
}
```

D.17 IsdnChannelInd

ZWECK

Signalisiert, daß ein ISDN Datenkanal für die Verbindung bereitgestellt wurde

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG

Die registrierte Callback Funktion wird mit einer `IsdnChannelInd` Indikation von der IsdnLib aufgerufen, um der Applikation Zugriff auf den zugewiesenen Datenkanal zu geben. Von diesem Zeitpunkt an, können Daten von diesem logischen Kanal empfangen und gesendet werden.

Der Parameter `t_IsdnAlertInd` liefert weitere Informationen über die Indikation:

```
typedef struct {                                        /* indicates the available channe
        t_IsdnIndType           indType;                /* type of inication */
        int                     callId;                 /* unique ID of the call */
        t_ChannelId             channelId;              /* the ID of the selected channel
        int                     fd;                     /* fd for the selected channel */
} t_IsdnChannelInd;
```

Bedeutung der Felder:

indType
: ISDN_CHANNEL_IND

callId
: Identifiziert die logische Verbindung.

channelId
: Gibt die verwendeten ISDN Kanäle an.

fd
: Der Filedescriptor, mit dem auf den logischen Kanal zugegriffen werden kann.

BEISPIEL

```
void DataCb();
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnChannelInd               *channel = (t_IsdnChannelInd*)ind;

        if (ind->indType == ISDN_CHANNEL_IND) {
                printf ("Channel got for %d\n", channel->callId);
                IsdnCreateFileHandler (channel->fd, DataCb, ISDN_READABLE, NULL);
                ...
        }
}
```

D.18 IsdnChargeInd

ZWECK
: Übertragung von Gebühreninformationen

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG
: Die registrierte Callback Funktion wird mit einer `IsdnChargeInd` Indikation von der IsdnLib aufgerufen, wenn Gebühreninformationen aus dem ISDN Netz empfangen wurden (vgl. Q.956 [Int95]).

Der Parameter `t_IsdnChargeInd` liefert weitere Informationen über die Indikation:

```
typedef struct {
        t_IsdnIndType           indType;                /* type of inication */
        int                     callId;                 /* unique ID of the call */
        int                     units;                  /* # of charge units */
} t_IsdnChargeInd;
```

Bedeutung der Felder:

indType
: ISDN_CHARGE_IND

callId
: Identifiziert die logische Verbindung.

units
: Anzahl der Gebühreneinheiten.

BEISPIEL

```
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnChargeInd         *alert = (t_IsdnChargeInd*)ind;

        if (ind->indType == ISDN_CHARGE_IND) {
                printf ("Charging for %d\n", charge->callId);
                printf ("the call costs %d units now!\n", charge->units);
        }
}
```

D.19 IsdnConnectInd

ZWECK

Signalisiert, daß die Verbindung mit der Gegenstelle aufgebaut ist

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG

Die registrierte Callback Funktion wird mit einer `IsdnConnectInd` Indikation von der IsdnLib aufgerufen, wenn die Verbindung mit der Gegenstelle aufgebaut ist. Von diesen Zeitpunkt an können Daten zwischen den beiden Stationen über den Filedescriptor ausgetauscht werden, der zuvor durch die `IsdnChannelInd` mitgeteilt wurde.

Der Parameter `t_IsdnConnectInd` liefert weitere Informationen über die Indikation:

```
typedef struct {                                        /* indicates that the call is cor
        t_IsdnIndType           indType;                /* type of inication */
        int                     callId;                 /* unique ID of the call */
} t_IsdnConnectInd;
```

Bedeutung der Felder:

indType

ISDN_CONNECT_IND

callId

Identifiziert die logische Verbindung.

BEISPIEL

```
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnConnectInd        *alert = (t_IsdnConnectInd*)ind;

        if (ind->indType == ISDN_CONNECT_IND) {
                printf ("Call %d is now connected, ready to speak\n", connect->callId);
        }
}
```

D.20 IsdnDisconnectInd

ZWECK

Signalisiert, daß die Verbindung beendet wurde

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG

Die registrierte Callback Funktion wird mit einer `IsdnDisconnectInd` Indikation von der *IsdnLib* aufgerufen, wenn die Verbindung zwischen zwei Stationen getrennt wurde.

Der Parameter `t_IsdnDisconnectInd` liefert weitere Informationen über die Indikation:

```
typedef struct {
        t_IsdnIndType           indType;        /* type of inication */
        int                     callId;         /* unique ID of the call */
        t_Cause                 cause;
} t_IsdnDisconnectInd;
```

Bedeutung der Felder:

indType
: ISDN_DISCONNECT_IND

callId
: Identifiziert die logische Verbindung.

cause
: Gibt den Grund an, weshalb die Verbindung getrennt wurde. Zur Bedeutung siehe Q.850 [Int93p].

Die Applikation sollte in Folge dieser Indikation den Datenkanal schließen, und die Verbindung durch ein `IsdnReleaseReq` terminieren. Wenn sie das nicht innerhalb von ca. 15 Sekunden macht, wird dies durch die Vermittlungsstelle initiiert.

BEISPIEL

```
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnDisconnectInd     *disc = (t_IsdnDisconnectInd*)ind;

        if (ind->indType == ISDN_DISCONNECT_IND) {
                printf ("%d disconnected\n", disc->callId);
                printf ("Cause:");
                DumpCause (stdout, &disc->cause);
        }
}
```

D.21 IsdnReleaseInd

ZWECK

Signalisiert, daß die logische Verbindung beendet ist.

SYNTAX

```
#include <IsdnLib.h>

IsdnCallback (t_IsdnInd *ind);
```

BESCHREIBUNG

Die registrierte Callback Funktion wird mit einer `IsdnReleaseInd` Indikation von der IsdnLib aufgerufen, um der Applikation mitzuteilen, daß die logische Verbindung beendet ist.

Der Parameter `t_IsdnReleaseInd` liefert weitere Informationen über die Indikation:

```
typedef struct {
        t_IsdnIndType           indType;        /* type of inication */
        int                     callId;         /* unique ID of the call */
        t_Cause                 cause;
} t_IsdnReleaseInd;
```

Bedeutung der Felder:

indType

ISDN_ALERT_IND

callId

Identifiziert die logische Verbindung.

cause

Gibt den Grund an, weshalb die Verbindung getrennt wurde. Zur Bedeutung siehe Q.850 [Int93p].

Spätestens nach dem Erhalt dieser Nachricht hat die Applikation den Filedescriptor für den Datenkanal zu schließen, falls dieser noch geöffnet ist. Nach dieser Indikation darf die Applikation keine weiteren Nachrichten für diese `callId` mehr initiieren, oder es wird ein Fehler gemeldet. Von der IsdnLib sind nach dieser Indikation ebenfalls keine weiteren Nachrichten für diese `callId` zu erwarten.

Es ist zu beachten, daß diese Nachricht zu jeder Zeit für eine gültige `callId` gesendet werden kann.

BEISPIEL

```
IsdnCallback (t_IsdnInd *ind)
{
        t_IsdnReleaseInd                *rel = (t_IsdnReleaseInd*)ind;

        if (ind->indType == ISDN_RELEASE_IND) {
                printf ("%d released now\n", disc->callId);
                printf ("Cause:");
                DumpCause (stdout, &disc->cause);
        }
}
```

Anhang E

Source-Code für die IsdnLib

In den folgenden Abschnitten ist Sourcecode für die IsdnLib. Die Quellen wurden dabei unterteilt nach Gruppen dargestellt.

E.1 Die IsdnLib Routinen

Die folgenen C-Files beinhalten die Implementierungen der Funktionen, die die IsdnLib zur Verfügung stellt.

E.1.1 src/IsdnLib.h

```
/*
 *      file:           header.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: IsdnLib.h,v 1.2 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: IsdnLib.h,v $
 * Revision 1.2  1996/09/30  19:34:31  czahl
 * *** empty log message ***
 *
 * Revision 1.1  1996/06/17  14:52:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_ISDN_LIB
#define INCL_ISDN_LIB

/*----- standard includefiles -------------------------------------------------*/

/*----- local includefiles ----------------------------------------------------*/
#include "Q931.h"

/*----- defines ---------------------------------------------------------------*/
/*** NOTE: these values should match the TCL_* equivalents! ***/
#define ISDN_READABLE           (1<<1)
#define ISDN_WRITABLE           (1<<2)
#define ISDN_EXCEPTION          (1<<3)

#define EISDN_NONE              0                               /* no error */
#define EISDN_Q931              931                             /* Q.931 error */
#define EISDN_INV_CALLID        1                               /* unknown callId */
#define EISDN_INTERNAL          2                               /* internal error */
#define EISDN_NOT_ALLOWED       3                               /* request not allowed */
#define EISDN_INCOMPLETE        4                               /* request incomplete */

/*----- type definitions ------------------------------------------------------*/
typedef enum {
        ISDN_S_VOICE_ALAW = 0x101,
        ISDN_S_VOICE_MULAW = 0x102,
        ISDN_S_DEFAULT = ISDN_S_VOICE_ALAW
} t_IsdnService;

        /*** definition of the various ISDN requests ***/
/*
 * "req." means that this paramter is REQUIRED, "opt." means that it is an
 * optional paramter and can be set, but don't have to be set!
 */
typedef struct {
```

```
        int                     callId;                         /* req. */
} t_IsdnAlertingReq;

typedef struct {
        int                     callId;                         /* req. */
        t_CalledPartyNo         calledNo;                       /* opt. no to be called */
        t_CalledPartySubAddr    calledSubAddr;                  /* opt. subAddr to be called */
        t_CallingPartyNo        callingNo;                      /* opt. calling no */
        t_CallingPartySubAddr   callingSubAddr;                 /* opt. calling subAddr */
        t_IsdnService           service;                        /* opt. complete service */
        t_BearerCap             bearerCap[2];                   /* opt. bearer cap */
        t_HighLayerComp         highLayerComp[4];               /* opt. HLC */
        t_LowLayerComp          lowLayerComp[4];                /* opt. LLC */
} t_IsdnCallReq;

typedef struct {                                                /* connect to incomming call */
        int                     callId;                         /* req. */
        t_IsdnService           service;                        /* opt. short cut service */
        t_BearerCap             bearerCap;                      /* opt. selected bearer cap */
        t_HighLayerComp         highLayerComp;                  /* opt. selected HLC */
        t_LowLayerComp          lowLayerComp;                   /* opt. selected LLC */
} t_IsdnConnectReq;

typedef struct {
        int                     callId;                         /* req. */
        t_CalledPartyNo         calledPartyNo;                  /* req. */
} t_IsdnDialReq;

typedef struct {
        int                     callId;                         /* req. */
        char                    keypad[20+1];                   /* req. 1..20 chars, \0 terminated */
} t_IsdnKeypadReq;

typedef struct {
        int                     callId;                         /* req. */
        t_Cause                 cause;                          /* opt. reason */
} t_IsdnReleaseReq;

        /*** definition of the various ISDN indications ***/
typedef enum {
        ISDN_ALERT_IND = 1,
        ISDN_CALL_IND,
        ISDN_CHANNEL_IND,
        ISDN_CHARGE_IND,
        ISDN_CONNECT_IND,
        ISDN_DISCONNECT_IND,
        ISDN_PROGRESS_IND,
        ISDN_RELEASE_IND
} t_IsdnIndType;

typedef struct {                                                /* alerting indication */
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
} t_IsdnAlertInd;

typedef struct {
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
        t_CalledPartyNo         calledPartyNo;
        t_CalledPartySubAddr    calledPartySubAddr;
        t_CallingPartyNo        callingPartyNo;
        t_CallingPartySubAddr   callingPartySubAddr;
        t_ChannelId             channelId;                      /* identifies the B* channels */
        t_BearerCap             bearerCap[2];
        t_HighLayerComp         highLayerComp[4];
        t_LowLayerComp          lowLayerComp[4];
} t_IsdnCallInd;

typedef struct {                                                /* indicates the available channel(s) */
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
        t_ChannelId             channelId;                      /* the ID of the selected channel(s) */
        int                     fd;                             /* fd for the selected channel */
} t_IsdnChannelInd;

typedef struct {
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
        int                     units;                          /* # of charge units */
} t_IsdnChargeInd;

typedef struct {                                                /* indicates that the call is connected (active) now */
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
} t_IsdnConnectInd;

typedef struct {
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
        t_Cause                 cause;
} t_IsdnDisconnectInd;

typedef struct {
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
        t_ProgressInd           progress;
} t_IsdnProgressInd;

typedef struct {
```

```
        t_IsdnIndType           indType;                        /* type of inication */
        int                     callId;                         /* unique ID of the call */
        t_Cause                 cause;
} t_IsdnReleaseInd;

typedef union {                                                 /* ISDN indications send to callback */
        struct {                                                /* for easy access */
                t_IsdnIndType   indType;                        /* type of inication */
                int             callId;                         /* unique ID of the call */
        } ind;
        t_IsdnAlertInd          alertInd;
        t_IsdnCallInd           callInd;
        t_IsdnChannelInd        channelInd;
        t_IsdnChargeInd         chargeInd;
        t_IsdnConnectInd        connectInd;
        t_IsdnDisconnectInd     disconnectInd;
        t_IsdnProgressInd       progressInd;
        t_IsdnReleaseInd        releaseInd;
} t_IsdnInd;

/*----- global functions -----------------------------------------------------*/
extern  void    IsdnMainLoop (void);
extern  int     IsdnNewCallId (void);
extern  int     IsdnInitialize (void (*callback) ());

extern  int     IsdnAlertingReq (const t_IsdnAlertingReq *req);
extern  int     IsdnCallReq (t_IsdnCallReq *req);
extern  int     IsdnConnectReq (const t_IsdnConnectReq *req);
extern  int     IsdnDialReq (const t_IsdnDialReq *req);
extern  int     IsdnKeypadReq (const t_IsdnKeypadReq *req);
extern  int     IsdnReleaseReq (const t_IsdnReleaseReq *req);

extern  int     IsdnCreateFileHandler (int fd, int mask, void (*cb) (), void *data);
extern  void    IsdnDeleteFileHandler (int fd);
extern  int     IsdnCreateTimerHandler (int milliseconds, void (*cb) (), void *data);
extern  void    IsdnDeleteTimerHandler (int timerId);

extern  void    IsdnError (const char *str, ...);
extern  void    IsdnDumpInd (const t_IsdnInd *ind);

/*----- global variables -----------------------------------------------------*/
extern  void    (*IsdnCallback) ();
extern  int     isdnErrno;

#endif /* INCL_ISDN_LIB */
```

E.1.2 src/IsdnLib.c

```
/*
 *      file:           IsdnLib.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: IsdnLib.c,v 1.2 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: IsdnLib.c,v $
 * Revision 1.2  1996/09/30  19:34:13  czahl
 * *** empty log message ***
 *
 * Revision 1.1  1996/06/17  14:52:28  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: IsdnLib.c,v 1.2 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdarg.h>

/*----- user includefiles ----------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "Scheduler.h"
#include "Q931Main.h"
#include "IsdnLib.h"
#include "Q931Alerting.h"
#include "Q931Connect.h"
#include "Q931Disconnect.h"
#include "Q931Information.h"
#include "Q931Release.h"
#include "Q931Setup.h"
#include "Debug.h"

/*----- defines --------------------------------------------------------------*/

/*----- type definitions -----------------------------------------------------*/

/*----- local functions ------------------------------------------------------*/

/*----- global functions -----------------------------------------------------*/

/*----- local variables ------------------------------------------------------*/
static  int                     init = 0;
static  int                     lastCallId;
```

```
/*----- global variables -------------------------------------------------*/
        void                    (*IsdnCallback) ();
        int                     isdnErrno = 0;

/*--------------------------------------------------------------------------*/
int IsdnNewCallId ()
/*
 * DESC
 *      Requests a new callId. The returned callId is unique!
 */
{
        return ++lastCallId;
} /* IsdnNewCallId */
/*--------------------------------------------------------------------------*/
int IsdnInitialize (void (*isdnCallback) ())
/*
 * DESC
 *      Initializes the IsdnLib and all the other used modules.
 * RET
 *      rc      0 on success, -1 on error.
 */
{
        if (init)
                return 0;
        lastCallId = 4711;
        if (SchedulerInitialize () != 0) {
                isdnErrno = EISDN_INTERNAL;
                return -1;
        }
        if (Q931Initialize () != 0) {
                isdnErrno = EISDN_Q931;
                return -1;
        }
        DebugInitialize ("ISDN_DEBUG");
        IsdnCallback = isdnCallback;
        init = 1;
        return 0;
/*** make gcc happy ***/
        *rcsid = *rcsid;
} /* IsdnInitialize */
/*--------------------------------------------------------------------------*/
void IsdnFini ()
{
        if (!init)
                return;
        init = 0;
        return;
} /* IsdnFini */
/*--------------------------------------------------------------------------*/
int IsdnAlertingReq (const t_IsdnAlertingReq *ireq)
{
        t_Q931AlertingReq       req;
        t_Q931Cxt               *cxt;

        TRACE_ENTER (("IsdnAlertingReq"));
        TRACEX (DFLAG_CALLID, ("callId=%d", ireq->callId))
        if ((cxt = Q931SearchCallId (ireq->callId)) == NULL) {
                isdnErrno = EISDN_INV_CALLID;
                TRACE_EXIT;
                return -1;
        }
        Q931_ZERO (req);
        if (Q931AlertingReq (cxt, &req) < 0) {
                isdnErrno = EISDN_Q931;
                TRACE_EXIT;
                return -1;
        }
        TRACE_EXIT;
        return 0;
} /* IsdnAlertingReq */
/*--------------------------------------------------------------------------*/
int IsdnCallReq (t_IsdnCallReq *req)
/*
 * DESC
 *      Handles the request from the user to initiate a call. This will
 *      be done by 1) selecting a new callId, and 2) by sending a SETUP
 *      message to the ISDN network.
 */
{
        t_Q931SetupReq          setupReq;
        t_Q931Cxt               *cxt;
        t_CallRef               callRef;
        int                     rc;

        TRACE_ENTER (("IsdnCallReq (%p)", req));
/*** get a new context ***/
        Q931AllocCallRef (&callRef);
        cxt = Q931AllocCxt (&callRef);
        cxt->callId = IsdnNewCallId ();
        req->callId = cxt->callId;                              /* feedback to caller */
/*** convert the ISDN req to a Q931 req ***/
        Q931_ZERO (setupReq);
        if (req->service != 0) {
                Q931_FFS ();
        } else {
                memcpy (&setupReq.bearerCap, &req->bearerCap, T_BEARER_CAP_SIZE);
        }
        setupReq.bearerCap.transferCap = BC_TC_DEFAULT;
        setupReq.bearerCap.transferMode = BC_TM_DEFAULT;
```

```
        setupReq.bearerCap.layer1Prot = BC_L1P_DEFAULT;
        setupReq.bearerCap.transferRate = 64;
/*** MISSING: not complete at the moment ... ***/
        memcpy (&setupReq.callingPartyNo, &req->callingNo, T_CALLING_PARTY_NO_SIZE);
        memcpy (&setupReq.calledPartyNo, &req->calledNo, T_CALLED_PARTY_NO_SIZE);
/*** send a SETUP ***/
        if (Q931SetupReq (cxt, &setupReq) < 0) {
                isdnErrno = EISDN_Q931;
                rc = -1;
        } else {
                rc = cxt->callId;
        }
        TRACE_EXIT;
        return rc;
} /* IsdnCallReq */
/*---------------------------------------------------------------------------*/
int IsdnConnectReq (const t_IsdnConnectReq *ireq)
{
        t_Q931ConnectReq        req;
        t_Q931Cxt               *cxt;

        TRACE_ENTER (("IsdnConnectReq"));
        TRACEX (DFLAG_CALLID, ("callId=%d", ireq->callId))
        if ((cxt = Q931SearchCallId (ireq->callId)) == NULL) {
                isdnErrno = EISDN_INV_CALLID;
                TRACE_EXIT;
                return -1;
        }
/*** convert the ISDN req to a Q931 req ***/
        Q931_ZERO (req);
        if (ireq->service != 0) {
                Q931_FFS ();
        } else {
                memcpy (&req.bearerCap, &ireq->bearerCap, T_BEARER_CAP_SIZE);
        }
        memcpy (&req.highLayerComp, &ireq->highLayerComp, T_HIGH_LAYER_COMP_SIZE);
        memcpy (&req.lowLayerComp, &ireq->lowLayerComp, T_LOW_LAYER_COMP_SIZE);
/*** send the request ***/
        if (Q931ConnectReq (cxt, &req) < 0) {
                isdnErrno = EISDN_Q931;
                TRACE_EXIT;
                return -1;
        }
        TRACE_EXIT;
        return 0;
} /* IsdnConnectReq */
/*---------------------------------------------------------------------------*/
int IsdnDialReq (const t_IsdnDialReq *req)
/*
 * DESC
 *      The purpose of this request is to send additional informations for the
 *      calledPartyNo to the providing ISDN network.
 * RET
 *      0 on succes, -1 on error and isdnErrno set to the errorcode.
 */
{
        t_Q931InformationReq    q931Req;
        t_Q931Cxt               *cxt;

        if ((cxt = Q931SearchCallId (req->callId)) == NULL) {
                isdnErrno = EISDN_INV_CALLID;
                return -1;
        }
        if (strlen (req->calledPartyNo.number) <= 0) {
                isdnErrno = -1;
                return -1;
        }
/*** check if sending is allowed ***/
        switch (cxt->state) {
                case 0:
                case 1:
                case 6:
                case 17:
                case 19:
                        isdnErrno = EISDN_NOT_ALLOWED;
                        return -1;
                default:
                        break;
        } /* switch */
/*** setup the Q.931 request ***/
        Q931_ZERO (q931Req);
        memcpy (&q931Req.calledPartyNo, &req->calledPartyNo, T_CALLED_PARTY_NO_SIZE);
        if (Q931InformationReq (cxt, &q931Req) < 0) {
                isdnErrno = EISDN_Q931;
                return -1;
        }
        return 0;
} /* IsdnDialReq */
/*---------------------------------------------------------------------------*/
int IsdnKeypadReq (const t_IsdnKeypadReq *req)
/*
 * DESC
 *      The purpose of this request is to send additional informations to the
 *      providing ISDN network.
 * RET
 *      0 on succes, -1 on error and isdnErrno set to the errorcode.
 */
{
        t_Q931InformationReq    q931Req;
```

```
        t_Q931Cxt               *cxt;

        if ((cxt = Q931SearchCallId (req->callId)) == NULL) {
                isdnErrno = EISDN_INV_CALLID;
                return -1;
        }
        if (!req->keypad[0]) {
                isdnErrno = EISDN_INCOMPLETE;
                return -1;
        }
/*** setup the Q.931 request ***/
        Q931_ZERO (q931Req);
        strcpy (q931Req.keypad.keypad, req->keypad);
        if (Q931InformationReq (cxt, &q931Req) < 0) {
                isdnErrno = EISDN_Q931;
                return -1;
        }
        return 0;
} /* IsdnKeypadReq */
/*--------------------------------------------------------------------------*/
int IsdnReleaseReq (const t_IsdnReleaseReq *req)
{
        t_Q931ReleaseReq        rel;
        t_Q931DisconnectReq     disc;
        t_Q931Cxt               *cxt;
        int                     rc;

        rc = 0;
        if ((cxt = Q931SearchCallId (req->callId)) == NULL) {
                isdnErrno = EISDN_INV_CALLID;
                return -1;
        }
/*** first lets try a RELEASE, then a DISCONNECT :-) ***/
        Q931_ZERO (rel);
        memcpy (&rel.cause, &req->cause, T_CAUSE_SIZE);
        if (!rel.cause.value)
                rel.cause.value = CA_V_DEFAULT;
        if (Q931ReleaseReq (cxt, &rel) < 0) {
                Q931_ZERO (disc);
                memcpy (&disc.cause, &req->cause, T_CAUSE_SIZE);
                if (!disc.cause.value)
                        disc.cause.value = CA_V_DEFAULT;
                if (Q931DisconnectReq (cxt, &disc) < 0) {
                        isdnErrno = EISDN_Q931;
                        rc = -1;
                }
        }
        return rc;
} /* IsdnReleaseReq */
/*--------------------------------------------------------------------------*/
int IsdnCreateFileHandler (int fd, int mode, void (*cb) (), void *data)
{
        return SchedulerCreateFileHandler (fd, mode, cb, data);
} /* IsdnCreateFileHandler */
/*--------------------------------------------------------------------------*/
void IsdnDeleteFileHandler (int fd)
{
        SchedulerDeleteFileHandler (fd);
} /* IsdnCreateFileHandler */
/*--------------------------------------------------------------------------*/
int IsdnCreateTimerHandler (int milliseconds, void (*cb) (), void *data)
{
        return SchedulerCreateTimerHandler (milliseconds, cb, data);
} /* IsdnCreateTimerHandler */
/*--------------------------------------------------------------------------*/
void IsdnDeleteTimerHandler (int timerId)
{
        SchedulerDeleteTimerHandler (timerId);
} /* IsdnCreateTimerHandler */
/*--------------------------------------------------------------------------*/
void IsdnMainLoop ()
{
        SchedulerMainLoop ();
} /* IsdnMainLoop */
/*--------------------------------------------------------------------------*/
static void IsdnDumpCalledPartyNo (const t_CalledPartyNo *no)
{
        if (!no->type)
                return;
        printf ("# CALLED-PARTY-NO: TYPE=%d", no->type & 0xff);
        printf (", NUMBERING-PLAN=%d", no->numberingPlan & 0xff);
        printf (", NO=%s\n", no->number);
} /* IsdnDumpCalledPartyNo */
/*--------------------------------------------------------------------------*/
static void IsdnDumpCallingPartyNo (const t_CallingPartyNo *no)
{
        if (!no->type)
                return;
        printf ("# CALLING-PARTY-NO: TYPE=%d", no->type & 0xff);
        printf (", NUMBERING-PLAN=%d", no->numberingPlan & 0xff);
        printf (", PRESENT=%d", no->presentation & 0xff);
        printf (", SCREENING=%d", no->screening & 0xff);
        printf (", NO=%s\n", no->number);
} /* IsdnDumpCallingPartyNo */
/*--------------------------------------------------------------------------*/
void IsdnDumpInd (const t_IsdnInd *ind)
/*
 * Test function which simply dumps the indication 'ind' is pointing to.
 */
```

```
{
        printf ("#####################################################\n");
        printf ("# %d:\t", ind->ind.callId);
        switch (ind->ind.indType) {
                case ISDN_ALERT_IND:
                        printf ("ALERT-IND\n");
                        break;
                case ISDN_CALL_IND:
                        printf ("CALL-IND\n");
                        IsdnDumpCalledPartyNo (&ind->callInd.calledPartyNo);
                        IsdnDumpCallingPartyNo (&ind->callInd.callingPartyNo);
                        break;
                case ISDN_CHANNEL_IND:
                        printf ("CHANNEL-IND\n");
/*                      Q931DumoChannelId (&ind->channelInd.channelId); */
                        break;
                case ISDN_CHARGE_IND:
                        printf ("CHARGE-IND\n");
                        break;
                case ISDN_CONNECT_IND:
                        printf ("CONNECT-IND\n");
                        break;
                case ISDN_DISCONNECT_IND:
                        printf ("DISC-IND\n# ");
                        Q931DumpCause (stdout, &ind->disconnectInd.cause);
                        break;
                case ISDN_PROGRESS_IND:
                        printf ("PROGRESS-IND\n# ");
                        Q931DumpProgressInd (stdout, &ind->progressInd.progress);
                        break;
                case ISDN_RELEASE_IND:
                        printf ("RELEASE-IND\n# ");
                        Q931DumpCause (stdout, &ind->releaseInd.cause);
                        break;
                default:
                        printf ("<unknown(%d)>\n", ind->ind.indType);
                        break;
        }
        printf ("#####################################################\n");
} /* IsdnDumpInd */
/*---------------------------------------------------------------------------*/
void IsdnError (const char *str, ...)
{
        va_list                 va;

        va_start (va, str);
        vfprintf (stderr, str, va);
        switch (isdnErrno) {
                case EISDN_NONE:
                        fprintf (stderr, "No error\n");
                        break;
                case EISDN_Q931:
                        fprintf (stderr, "error in the Q.931 module\n");
                        break;
                case EISDN_INV_CALLID:
                        fprintf (stderr, "invalid callId\n");
                        break;
                case EISDN_NOT_ALLOWED:
                        fprintf (stderr, "operation currently not allowed\n");
                        break;
                case EISDN_INTERNAL:
                        fprintf (stderr, "internal error\n");
                        break;
                default:
                        fprintf (stderr, "unknown error (%d)\n", isdnErrno);
                        break;
        }
        va_end (va);
} /* IsdnError */
/*---------------------------------------------------------------------------*/
```

E.2 Q.931 Messages

Jedes der folgenden C-Files behandelt einen Q.931 Messagetype.

E.2.1 src/Q931Alerting.h

```
/*
 *      file:           Q931Alerting.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Alerting.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Alerting.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */
```

```
#ifndef INCL_Q931_ALERTING
#define INCL_Q931_ALERTING

/*----- standard includefiles ----------------------------------------------*/

/*----- local includefiles -------------------------------------------------*/
#include "Q931.h"

/*----- defines ------------------------------------------------------------*/

/*----- type definitions ---------------------------------------------------*/
typedef struct {
        int                     dummy;
} t_Q931AlertingReq;

/*----- global functions ---------------------------------------------------*/
extern  void    Q931AlertingInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931AlertingReq (t_Q931Cxt *cxt, const t_Q931AlertingReq *req);

/*----- global variables ---------------------------------------------------*/

#endif /* INCL_Q931_ALERTING */
```

E.2.2 src/Q931Alerting.c

```
/*
 *      file:           Q931Alerting.c
 *      written by:     Christian Zahl
 *      description:    Handles the Q.931 ALERTING related stuff
 *
 * $Id: Q931Alerting.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Alerting.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Alerting.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles --------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Alerting.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines ------------------------------------------------------------*/

/*----- type definitions ---------------------------------------------------*/

/*----- local functions ----------------------------------------------------*/

/*----- global functions ---------------------------------------------------*/

/*----- local variables ----------------------------------------------------*/

/*----- global variables ---------------------------------------------------*/

/*--------------------------------------------------------------------------*/
static void AlertingInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
{
        t_IsdnInd               ind;
        int                     i;

        TRACE_ENTER (("AlertingInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("ALERTING-INDICATION"));
        Q931_ZERO (ind);
        ind.ind.indType = ISDN_ALERT_IND;
        ind.ind.callId = cxt->callId;
        IsdnCallback (&ind);
/*** bearer cap ***/
        /* FFS */
/*** channel selection ***/
        /* FFS */
/*** progress ***/
        for (i=0; msg->progressInd[i].descr; i++) {
                Q931_ZERO (ind);
                ind.progressInd.indType = ISDN_PROGRESS_IND;
                ind.progressInd.callId = cxt->callId;
                ind.progressInd.progress = msg->progressInd[i];
                IsdnCallback (&ind);
        }
/*** display ***/
        /* FFS */
/*** signal ***/
        /* FFS */
/*** HLC ***/
        /* FFS */
/*** to make gcc happy ***/
        *rcsid = *rcsid;
```

```
        TRACE_EXIT;
} /* AlertingInd */
/*------------------------------------------------------------------------*/
void Q931AlertingInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles an incomming ALERTING message
 * NOTE
 *      o no handling of bearer cap selection
 */
{
        TRACE_ENTER (("Q931AlertingInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 ALERTING"));
        switch (cxt->state) {
                case Q931_US_CALL_INITIATED:                    /* 1 */
                        cxt->state = Q931_US_CALL_DELIVERED;    /* 4 */
                        Q931StopTimer (cxt, Q931_TIMER_T303);
                        AlertingInd (cxt, msg);
                        break;
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
                        cxt->state = Q931_US_CALL_DELIVERED;    /* 4 */
                        Q931StopTimer (cxt, Q931_TIMER_T304);
                        AlertingInd (cxt, msg);
                        break;
                case Q931_US_OUT_CALL_PROCEEDING:               /* 3 */
                        cxt->state = Q931_US_CALL_DELIVERED;    /* 4 */
                        Q931StopTimer (cxt, Q931_TIMER_T310);
                        AlertingInd (cxt, msg);
                        break;
                default:
                        /* handle error */
                        Q931_FFS ();
                        break;
        }
        TRACE_EXIT;
} /* Q931AlertingInd */
/*------------------------------------------------------------------------*/
static int Q931SendAlerting (t_Q931Cxt *cxt, const t_Q931AlertingReq *req)
/*
 * DESC
 *      Generates and send the Q.931 ALERTING message
 * NOTE
 *      no support for:
 *              bearer cap
 *              channel id
 *              progress
 *              HLC
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendAlerting ()"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_ALERTING;
        TRACEX (DFLAG_Q931, ("OUT: Q.931 ALERTING"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendAlerting */
/*------------------------------------------------------------------------*/
int Q931AlertingReq (t_Q931Cxt *cxt, const t_Q931AlertingReq *req)
/*
 * DESC
 *      Generate an ALERING to inform the network that we received the incomming
 *      call and that the local user is beeing signaled.
 * RC
 *      0       OK
 *      -1      error
 */
{
        int                     rc;

        TRACE_ENTER (("Q931AlertingReq()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("ALERTING-REQUEST"));
        switch (cxt->state) {
                case Q931_US_CALL_PRESENT:                      /* 6 */
                        cxt->state = Q931_US_CALL_RECEIVED;     /* 7 */
                        Q931StopTimer (cxt, Q931_TIMER_T300);   /* priv., to timeout not acknowledged calls!!! */
                        rc = Q931SendAlerting (cxt, req);
                        break;
                case Q931_US_IN_CALL_PROCEEDING:                /* 9 */
                        cxt->state = Q931_US_CALL_RECEIVED;     /* 7 */
                        rc = Q931SendAlerting (cxt, req);
                        break;
                default:
                        rc = -1;
                        break;
        }
        TRACE_EXIT;
        return rc;
} /* Q931AlertingReq */
/*------------------------------------------------------------------------*/
```

E.2.3 src/Q931CallProceeding.h

```
/*
 *      file:           Q931CallProceeding.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931CallProceeding.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931CallProceeding.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CALL_PROCEEDING
#define INCL_CALL_PROCEEDING

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/
typedef struct {
        t_BearerCap             bearerCap;
        t_ChannelId             channelId;
        t_ProgressInd           progressInd;
        t_HighLayerComp         highLayerComp;
} t_Q931CallProceedingReq;

/*----- global functions --------------------------------------------------*/
extern  void    Q931CallProceedingInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931SendCallProceeding (t_Q931Cxt *cxt, const t_Q931CallProceedingReq *req);
extern  int     Q931CallProceedingReq (t_Q931Cxt *cxt, const t_Q931CallProceedingReq *req);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_CALL_PROCEEDING */
```

E.2.4 src/Q931CallProceeding.c

```
/*
 *      file:           Q931CallProceeding.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931CallProceeding.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931CallProceeding.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931CallProceeding.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931CallProceeding.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void ProceedingInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
{
        t_IsdnInd               ind;

        TRACE_ENTER (("ProceedingInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("CALL-PROCEEDING-INDICATION"));
        Q931_ZERO (ind);
/* FFS */
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* ProceedingInd */
/*---------------------------------------------------------------------------*/
void Q931CallProceedingInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
```

```
{
        TRACE_ENTER (("Q931ProceedingInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 CALL-PROCEEDING"));
        if (cxt->state == Q931_US_CALL_INITIATED) {             /* 1 */
                ProceedingInd (cxt, msg);
                cxt->state = Q931_US_OUT_CALL_PROCEEDING;       /* 3 */
        } else if (cxt->state == Q931_US_OVERLAP_SENDING) {     /* 2 */
                ProceedingInd (cxt, msg);
                cxt->state = Q931_US_OUT_CALL_PROCEEDING;       /* 3 */
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931CallProceeding */
/*------------------------------------------------------------------------*/
int Q931SendCallProceeding (t_Q931Cxt *cxt, const t_Q931CallProceedingReq *req)
/*
 * DESC
 *      Generates and sends a Q.931 CALL PROCEEDING message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendCallProceeding ()"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_CALL_PROCEEDING;
/*** bearer cap ***/
        if (req->bearerCap.transferCap)
                len += Q931PushBearerCap (buf+len, &req->bearerCap);
/*** channel id ***/
        if (req->channelId.interfaceIdPresent)
                len += Q931PushChannelId (buf+len, &req->channelId);
/*** progress ***/
        if (req->progressInd.descr)
                len += Q931PushProgressInd (buf+len, &req->progressInd);
/*** HLC ***/
        if (req->highLayerComp.dummy)
                len += Q931PushHighLayerComp (buf+len, &req->highLayerComp);
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 CALL PROCEEDING"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendCallProceeding */
/*------------------------------------------------------------------------*/
int Q931CallProceedingReq (t_Q931Cxt *cxt, const t_Q931CallProceedingReq *req)
{
        int                     rc;

        TRACE_ENTER (("Q931CallProceedingReq ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("CALL-PROCEEDING-REQUEST"));
        if (cxt->state == Q931_US_CALL_PRESENT) {               /* 6 */
                cxt->state = Q931_US_IN_CALL_PROCEEDING;        /* 9 */
                rc = Q931SendCallProceeding (cxt, req);
        } else if (cxt->state == Q931_US_OVERLAP_RECEIVING) {   /* 25 */
                cxt->state = Q931_US_IN_CALL_PROCEEDING;        /* 9 */
                rc = Q931SendCallProceeding (cxt, req);
        } else {
                rc = -1;
        }
        TRACE_EXIT;
        return rc;
} /* Q931CallProceedingReq */
/*------------------------------------------------------------------------*/
```

E.2.5 src/Q931Connect.h

```
/*
 *      file:           Q931Connect.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Connect.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Connect.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CONNECT
#define INCL_CONNECT

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/
```

```
typedef struct {
        t_BearerCap             bearerCap;
        t_ChannelId             channelId;
        t_ProgressInd           progressInd;
        t_LowLayerComp          lowLayerComp;
        t_HighLayerComp         highLayerComp;
} t_Q931ConnectReq;

/*----- global functions -------------------------------------------------------*/
extern  void    ConnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  void    Q931ConnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931ConnectReq (t_Q931Cxt *cxt, const t_Q931ConnectReq *req);

/*----- global variables -------------------------------------------------------*/

#endif /* INCL_CONNECT */
```

E.2.6 src/Q931Connect.c

```
/*
 *      file:           Q931Connect.c
 *      written by:     Christian Zahl
 *      description:    Handles the Q.931 CONNECT related stuff
 *
 * $Id: Q931Connect.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Connect.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Connect.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles --------------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles ------------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Connect.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines ----------------------------------------------------------------*/

/*----- type definitions -------------------------------------------------------*/

/*----- local functions --------------------------------------------------------*/

/*----- global functions -------------------------------------------------------*/

/*----- local variables --------------------------------------------------------*/

/*----- global variables -------------------------------------------------------*/

/*------------------------------------------------------------------------------*/
void ConnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the inditions of the CONNECT message.
 * NOTE
 *      According to Q.931 this is called "Setup confirm"
 */
{
        t_IsdnInd               ind;

        TRACE_ENTER (("ConnectInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("CONNECT INDICATION"));
        Q931_ZERO (ind);
        ind.ind.indType = ISDN_CONNECT_IND;
        ind.ind.callId = cxt->callId;
        IsdnCallback (&ind);
/*** bearer cap ***/
        /* FFS */
/*** channel id ***/
        /* FFS */
/*** progress ind ***/
        /* FFS */
/*** display ***/
        /* FFS */
/*** data/time ***/
        /* FFS */
/*** signal ***/
        /* FFS */
/*** llc ***/
        /* FFS */
/*** hlc ***/
        /* FFS */
        TRACE_EXIT;
        return;
        *rcsid = *rcsid; /* to make gcc happy */
} /* ConnectInd */
/*------------------------------------------------------------------------------*/
```

```
void Q931ConnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming CONNECT message from the ISDN net.
 */
{
        TRACE_ENTER (("Q931ConnectInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 CONNECT"));
        switch (cxt->state) {
                case Q931_US_CALL_INITIATED:                    /* 1 */
                        cxt->state = Q931_US_ACTIVE;            /* 10 */
                        Q931StopTimer (cxt, Q931_TIMER_T303);
                        ConnectInd (cxt, msg);
                        break;
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
                        cxt->state = Q931_US_ACTIVE;            /* 10 */
                        Q931StopTimer (cxt, Q931_TIMER_T304);
                        ConnectInd (cxt, msg);
                        break;
                case Q931_US_OUT_CALL_PROCEEDING:               /* 3 */
                        cxt->state = Q931_US_ACTIVE;            /* 10 */
                        Q931StopTimer (cxt, Q931_TIMER_T310);
                        ConnectInd (cxt, msg);
                        break;
                case Q931_US_CALL_DELIVERED:                    /* 4 */
                        cxt->state = Q931_US_ACTIVE;            /* 10 */
                        ConnectInd (cxt, msg);
                        break;
                default:
                        /* handle error */
                        Q931_FFS ();
                        break;
        }
        TRACE_EXIT;
} /* Q931ConnectInd */
/*------------------------------------------------------------------------*/
static int Q931SendConnect (t_Q931Cxt *cxt, const t_Q931ConnectReq *req)
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendConnect ()"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_CONNECT;
/*** bearer cap ***/
        if (req->bearerCap.transferCap)
                len += Q931PushBearerCap (buf+len, &req->bearerCap);
/*** channel id ***/
        if (req->channelId.interfaceIdPresent)
                len += Q931PushChannelId (buf+len, &req->channelId);
/*** progress ind ***/
        if (req->progressInd.descr)
                len += Q931PushProgressInd (buf+len, &req->progressInd);
/*** llc ***/
        if (req->lowLayerComp.dummy)
                len += Q931PushLowLayerComp (buf+len, &req->lowLayerComp);
/*** hlc ***/
        if (req->highLayerComp.dummy)
                len += Q931PushHighLayerComp (buf+len, &req->highLayerComp);
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 CONNECT"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendConnect */
/*------------------------------------------------------------------------*/
int Q931ConnectReq (t_Q931Cxt *cxt, const t_Q931ConnectReq *req)
/*
 * DESC:
 *      Request the network to connect the incomming call to the local user.
 * NOTE:
 *      According to Q.931 this is called "SETUP RESPONSE".
 */
{
        int                     rc = 0;

        TRACE_ENTER (("Q931ConnectReq ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("CONNECT-REQUEST"));
        switch (cxt->state) {
                case Q931_US_CALL_PRESENT:                      /* 6 */
                        if (Q931SendConnect (cxt, req) < 0) {
                                rc = -1;
                                break;
                        }
                        Q931StartTimer (cxt, Q931_TIMER_T313);
                        cxt->state = Q931_US_CONNECT_REQUEST;   /* 8 */
                        break;
                case Q931_US_CALL_RECEIVED:                     /* 7 */
                        if (Q931SendConnect (cxt, req) < 0) {
                                rc = -1;
                                break;
                        }
                        Q931StartTimer (cxt, Q931_TIMER_T313);
                        cxt->state = Q931_US_CONNECT_REQUEST;   /* 8 */
                        break;
                case Q931_US_IN_CALL_PROCEEDING:                /* 9 */
                        if (Q931SendConnect (cxt, req) < 0) {
```

```
                                rc = -1;
                                break;
                        }
                        Q931StartTimer (cxt, Q931_TIMER_T313);
                        cxt->state = Q931_US_CONNECT_REQUEST;   /* 8 */
                        break;
                case Q931_US_OVERLAP_RECEIVING:                 /* 25 */
                        if (Q931SendConnect (cxt, req) < 0) {
                                rc = -1;
                                break;
                        }
                        Q931StopTimer (cxt, Q931_TIMER_T302);
                        Q931StartTimer (cxt, Q931_TIMER_T313);
                        cxt->state = Q931_US_CONNECT_REQUEST;   /* 8 */
                        break;
                default:
                        rc = -1;
                        break;
        }
        TRACE_EXIT;
        return rc;
} /* Q931ConnectReq */
/*---------------------------------------------------------------------------*/
```

E.2.7 src/Q931ConnectAck.h

```
/*
 *      file:           Q931ConnectAck.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931ConnectAck.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931ConnectAck.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CONNECT_ACK
#define INCL_CONNECT_ACK

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        int                     dummy;
} t_Q931ConnectAckReq;

/*----- global functions ----------------------------------------------------*/
extern  void    Q931ConnectAckInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931ConnectAckReq (t_Q931Cxt *cxt, const t_Q931ConnectAckReq *req);

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_CONNECT_ACK */
```

E.2.8 src/Q931ConnectAck.c

```
/*
 *      file:           Q931ConnectAck.c
 *      written by:     Christian Zahl
 *      description:    Handles all CONNECT-ACKNOWLEDGE related stuff.
 *
 * $Id: Q931ConnectAck.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931ConnectAck.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931ConnectAck.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <sys/isdnio.h>
#include <unistd.h>
#include <fcntl.h>

/*----- user includefiles ---------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Main.h"
#include "Q931Msg.h"
#include "Q931Timer.h"
#include "Q931Connect.h"
```

```
#include "Q931ConnectAck.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void ConnectAckInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the inditions of the CONNECT ACKNOWLEDGE message.
 * NOTE
 *      According to Q.931 this is called "SETUP COMPLETE INDICATION"
 */
{
        t_IsdnInd                       ind;
        char                    fn[100];
        isdn_format_req_t       isdnFormatReq;
        int                     fd;


        TRACE_ENTER (("ConnectAckInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("CONNECT-ACKNOWLEDGE INDICATION"));
/*** generate the indication ***/
        Q931_ZERO (ind);
        ind.channelInd.indType = ISDN_CHANNEL_IND;
        ind.channelInd.callId = cxt->callId;
        ind.channelInd.channelId = cxt->channelId;
        ind.channelInd.fd = -1;
/*** handle the selected B-Channel ***/
        if (cxt->channelId.interfaceType == CID_IT_BRI) {
                Q931_ZERO (isdnFormatReq);
/*** NOTE: can be something other than ALAW!!! ***/
                ISDN_SET_FORMAT_VOICE_ALAW (&isdnFormatReq.format);
                if (cxt->channelId.infoChannelSel == CID_ICS_B1) {
                        isdnFormatReq.channel = ISDN_CHAN_TE_B1;
                        strcpy (fn, "/dev/isdn/0/te/b1");
                } else if (cxt->channelId.infoChannelSel == CID_ICS_B2) {
                        isdnFormatReq.channel = ISDN_CHAN_TE_B2;
                        strcpy (fn, "/dev/isdn/0/te/b2");
                } else {
                        Q931_FFS ();
                        /** what's that??? ***/
                        return;
                }
                if (ioctl (q931DlpiFd, ISDN_SET_FORMAT, &isdnFormatReq) < 0) {
                        Q931_FFS ();
                        return;
                }
                if (cxt->fd == -1) {
                        if ((fd = open (fn, O_RDWR)) < 0) {
                                Q931_FFS ();
                                return;
                        }
                        cxt->fd = fd;
                }
                ind.channelInd.fd = cxt->fd;
        } else {
                Q931_FFS ();
                return;
        }
        IsdnCallback (&ind);
/*** genereate a CONNECT-IND ***/
        Q931_ZERO (ind);
        ind.ind.indType = ISDN_CONNECT_IND;
        ind.ind.callId = cxt->callId;
        IsdnCallback (&ind);
/*** display ***/
        /* FFS */
/*** signal ***/
        /* FFS */
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* ConnectAckInd */
/*---------------------------------------------------------------------------*/
void Q931ConnectAckInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming CONNECT ACKNOWLEDGE message from the ISDN net.
 */
{
        TRACE_ENTER (("Q931ConnectAckInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 CONNECT ACKNOWLEDGE"));
        switch (cxt->state) {
                case Q931_US_CONNECT_REQUEST:                   /* 8 */
                        Q931StopTimer (cxt, Q931_TIMER_T313);
                        ConnectAckInd (cxt, msg);
                        cxt->state = Q931_US_ACTIVE;            /* 10 */
```

```
                        break;
                default:
                        /* handle error */
                        Q931_FFS ();
                        break;
        }
        TRACE_EXIT;
} /* Q931ConnectAckInd */
/*------------------------------------------------------------------------*/
static int Q931SendConnectAck (t_Q931Cxt *cxt, const t_Q931ConnectAckReq *req)
/*
 * DESC
 *      Generates and send the Q.931 CONNECT-ACKNOWLEDGE message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendConnectAck ()"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_CONNECT_ACK;
        TRACEX (DFLAG_Q931, ("OUT: Q.931 CONNECT-ACKNOWLEDGE"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendConnectAck */
/*------------------------------------------------------------------------*/
int Q931ConnectAckReq (t_Q931Cxt *cxt, const t_Q931ConnectAckReq *req)
/*
 * DESC
 *      Handles the CONNECT ACKNOWLEDGE REQ
 * NOTE:
 *      According to Q.931 there is no explicite name for this primitive.
 *      This message can optionaly send by the user after the receiption
 *      of a CONNECT message.
 */
{
        int                     rc;
        TRACE_ENTER (("Q931ConnectAckReq ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("CONNECT-ACKNOWLEDGE REQUEST"));
        switch (cxt->state) {
                case Q931_US_CALL_INITIATED:                    /* 1 */
                        rc = Q931SendConnectAck (cxt, req);
                        break;
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
                        rc = Q931SendConnectAck (cxt, req);
                        break;
                case Q931_US_OUT_CALL_PROCEEDING:               /* 3 */
                        rc = Q931SendConnectAck (cxt, req);
                        break;
                case Q931_US_CALL_DELIVERED:                    /* 4 */
                        rc = Q931SendConnectAck (cxt, req);
                        break;
                default:
                        rc = -1;
                        break;
        }
        TRACE_EXIT;
        return rc;
} /* Q931ConnectAckReq */
/*------------------------------------------------------------------------*/
```

E.2.9 src/Q931Disconnect.h

```
/*
 *      file:           Q931Disconnect.h
 *      written by:     Christian Zahl
 *      description:    Defines the DISCONNECT related stuff
 *
 * $Id: Q931Disconnect.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Disconnect.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_DISCONNECT
#define INCL_Q931_DISCONNECT

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/
typedef struct {
        t_Cause                 cause;
} t_Q931DisconnectReq;

/*----- global functions --------------------------------------------------*/
```

```
extern  void    Q931DisconnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931DisconnectReq (t_Q931Cxt *cxt, const t_Q931DisconnectReq *req);

/*----- global variables -------------------------------------------------*/

#endif /* INCL_Q931_DISCONNECT */
```

E.2.10 src/Q931Disconnect.c

```
/*
 *      file:           Q931Disconnect.c
 *      written by:     Christian Zahl
 *      description:    Handles all the DISCONNECT related stuff
 *
 * $Id: Q931Disconnect.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Disconnect.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Disconnect.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles --------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles ------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Disconnect.h"
#include "Q931Release.h"
#include "Q931Timer.h"
#include "Debug.h"
#include "IsdnLib.h"

/*----- defines ----------------------------------------------------------*/

/*----- type definitions -------------------------------------------------*/

/*----- local functions --------------------------------------------------*/

/*----- global functions -------------------------------------------------*/

/*----- local variables --------------------------------------------------*/

/*----- global variables -------------------------------------------------*/

/*--------------------------------------------------------------------------*/
static void DisconnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the internal indication of the DISCONNECT IND
 */
{
        t_IsdnInd               ind;
        int                     i;

        TRACE_ENTER (("DisconnectInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931_INTERNAL, ("DISCONNECT INDICATION"));
        Q931_ZERO (ind);
        ind.disconnectInd.indType = ISDN_DISCONNECT_IND;
        ind.disconnectInd.callId = cxt->callId;
/*** cause ***/
        ind.disconnectInd.cause = msg->cause;
        IsdnCallback (&ind);
/*** progress ***/
        for (i=0; msg->progressInd[i].descr; i++) {
                Q931_ZERO (ind);
                ind.progressInd.indType = ISDN_PROGRESS_IND;
                ind.progressInd.callId = cxt->callId;
                ind.progressInd.progress = msg->progressInd[i];
                IsdnCallback (&ind);
        }
/*** display ***/
        /* FFS */
/*** signal ***/
        /* FFS */
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* DisconnectInd */
/*--------------------------------------------------------------------------*/
void Q931DisconnectInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles incomming DISCONNECT-IND messages
 */
{
        t_Q931ReleaseReq        req;

        TRACE_ENTER (("Q931DisconnectInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931, ("IN: Q.931 DISCONNECT INDICATION"));
        Q931_ZERO (req);
        if (cxt->state == Q931_US_CALL_PRESENT) {               /* 6 */
```

```
                DisconnectInd (cxt, msg);
                cxt->state = Q931_US_DISCONNECT_INDICATION;     /* 12 */
        } else if (cxt->state == Q931_US_DISCONNECT_REQUEST ) { /* 11 */
                DisconnectInd (cxt, msg);
                Q931StopTimer (cxt, Q931_TIMER_T305);
        Q931_FFS();
#ifdef FERTIG
        req....
#endif
                Q931ReleaseReq (cxt, &req);
                Q931StartTimer (cxt, Q931_TIMER_T308);
                cxt->state = Q931_US_RELEASE_REQUEST;           /* 19 */
        } else if (cxt->state == Q931_US_DISCONNECT_REQUEST ) { /* 11 */
                Q931StopTimer (cxt, Q931_TIMER_T319);
                DisconnectInd (cxt, msg);
                cxt->state = Q931_US_DISCONNECT_INDICATION;     /* 12 */
        } else if ((cxt->state != Q931_US_NULL)                 /* 0 */
                && (cxt->state != Q931_US_CALL_INITIATED)       /* 1 */
                && (cxt->state != Q931_US_CALL_PRESENT)         /* 6 */
                && (cxt->state != Q931_US_DISCONNECT_REQUEST)   /* 11 */
                && (cxt->state != Q931_US_DISCONNECT_INDICATION) /* 12 */
                && (cxt->state != Q931_US_SUSPEND_REQUEST)      /* 15 */
                && (cxt->state != Q931_US_RESUME_REQUEST)       /* 17 */
                && (cxt->state != Q931_US_RELEASE_REQUEST)) {   /* 19 */
                Q931StopAllTimers (cxt);
                DisconnectInd (cxt, msg);
                cxt->state = Q931_US_DISCONNECT_INDICATION;     /* 12 */
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931DisconnectInd */
/*---------------------------------------------------------------------------*/
static int Q931SendDisconnect (t_Q931Cxt *cxt, const t_Q931DisconnectReq *req)
/*
 * DESC
 *      Generates and send a Q.931 DISCONNECT message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendDisconnect (%p, %p)", cxt, req));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_DISCONNECT;
/*** cause ***/
        len += Q931PushCause (buf+len, &req->cause);
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 DISCONNECT"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendDisconnect */
/*---------------------------------------------------------------------------*/
int Q931DisconnectReq (t_Q931Cxt *cxt, const t_Q931DisconnectReq *req)
/*
 * DESC
 *      Handles the internal DISCONNECT-REQ
 */
{
        int                     rc;

        TRACE_ENTER (("Q931DisconnectReq (%p, %p)", cxt, req));
        TRACEX (DFLAG_Q931_INTERNAL, ("DISCONNECT REQUEST"));
        switch (cxt->state) {
                case Q931_US_CALL_INITIATED:                    /* 1 */
                        if (Q931SendDisconnect (cxt, req) < 0) {
                                rc = -1;
                                break;
                        }
                        Q931StopTimer (cxt, Q931_TIMER_T303);
                        Q931StartTimer (cxt, Q931_TIMER_T305);
                        cxt->state = Q931_US_DISCONNECT_REQUEST; /* 11 */
                        rc = 0;
                        break;
                case Q931_US_NULL:                              /* 0 */
                case Q931_US_CONNECT_REQUEST:                   /* 6 */
                case Q931_US_DISCONNECT_REQUEST:                /* 11 */
                case Q931_US_DISCONNECT_INDICATION:             /* 12 */
                case Q931_US_SUSPEND_REQUEST:                   /* 15 */
                case Q931_US_RESUME_REQUEST:                    /* 17 */
                case Q931_US_RELEASE_REQUEST:                   /* 19 */
                        rc = -1;
                        break;
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
/*** forgotten in Q.931 ??? ***/
                        Q931StopTimer (cxt, Q931_TIMER_T304);
                default:
                        if (Q931SendDisconnect (cxt, req) < 0) {
                                rc = -1;
                                break;
                        }
                        Q931StartTimer (cxt, Q931_TIMER_T305);
                        cxt->state = Q931_US_DISCONNECT_REQUEST; /* 11 */
                        rc = 0;
        } /* switch */
```

```
        TRACE_EXIT;
        return rc;
} /* Q931DisconnectReq */
/*--------------------------------------------------------------------------*/
```

E.2.11 src/Q931Information.h

```
/*
 *      file:           Q931Information.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Information.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Information.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_INFORMATION
#define INCL_Q931_INFORMATION

/*----- standard includefiles ----------------------------------------------*/

/*----- local includefiles -------------------------------------------------*/
#include "Q931.h"

/*----- defines ------------------------------------------------------------*/

/*----- type definitions ---------------------------------------------------*/
typedef struct {
        t_Keypad                keypad;
        t_CalledPartyNo         calledPartyNo;
} t_Q931InformationReq;

/*----- global functions ---------------------------------------------------*/
extern  void    Q931InformationInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931InformationReq (t_Q931Cxt *cxt, const t_Q931InformationReq *req);

/*----- global variables ---------------------------------------------------*/

#endif /* INCL_Q931_INFORMATION */
```

E.2.12 src/Q931Information.c

```
/*
 *      file:           Q931Information.c
 *      written by:     Christian Zahl
 *      description:    Handles the Q.931 INFORMAITON related stuff
 *
 * $Id: Q931Information.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Information.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Information.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles --------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Information.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines ------------------------------------------------------------*/

/*----- type definitions ---------------------------------------------------*/

/*----- local functions ----------------------------------------------------*/

/*----- global functions ---------------------------------------------------*/

/*----- local variables ----------------------------------------------------*/

/*----- global variables ---------------------------------------------------*/

/*--------------------------------------------------------------------------*/
static void InformationInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the internal indication of an INFORMATION-IND
 */
{
        TRACE_ENTER (("InformationInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("INFORMATION INDICATION"));
```

```
/*** sending complete ***/
        /* ffs */
/*** display ***/
        /* ffs */
/*** signal ***/
        /* ffs */
/*** called party no ***/
        /* ffs */
        *rcsid = *rcsid;               /* to make gcc happy */
        TRACE_EXIT;
} /* InformationInd */
/*------------------------------------------------------------------------*/
void Q931InformationInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Receives the INFORMATION-IND messages and handles it.
 */
{
        TRACE_ENTER (("Q931InformationInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 INFORMATION"));
        if (cxt->state == Q931_US_OVERLAP_RECEIVING) {          /* 25 */
                InformationInd (cxt, msg);
                Q931StartTimer (cxt, Q931_TIMER_T302);
        } else if ((cxt->state != Q931_US_NULL)                 /* 0 */
                && (cxt->state != Q931_US_CALL_INITIATED)       /* 1 */
                && (cxt->state != Q931_US_CALL_PRESENT)         /* 6 */
                && (cxt->state != Q931_US_RESUME_REQUEST)       /* 17 */
                && (cxt->state != Q931_US_RELEASE_REQUEST)      /* 19 */
                && (cxt->state != Q931_US_OVERLAP_RECEIVING)) { /* 25 */
                InformationInd (cxt, msg);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931InformationInd */
/*------------------------------------------------------------------------*/
static int Q931SendInformation (t_Q931Cxt *cxt, const t_Q931InformationReq *req)
/*
 * DESC
 *      Generates and sends the Q.931 INFORMATION message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendInformation ()"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_INFORMATION;
/*** sending complete ***/
        /* n/a */
/*** keypad ***/
        len += Q931PushKeypad (buf+len, &req->keypad);
/*** called party no ***/
        if (*req->calledPartyNo.number)
                len += Q931PushCalledPartyNo (buf+len, &req->calledPartyNo);
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 INFORMATION"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendInformation */
/*------------------------------------------------------------------------*/
int Q931InformationReq (t_Q931Cxt *cxt, const t_Q931InformationReq *req)
/*
 * DESC
 *      Handles the internal INFORMATION-REQ
 */
{
        int                     rc = 0;

        TRACE_ENTER (("Q931InformationReq (%p, %p)", cxt, req));
        TRACEX (DFLAG_Q931_INTERNAL, ("INFORMATION REQUEST"));
        switch (cxt->state) {
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
                        Q931StopTimer (cxt, Q931_TIMER_T304);
                        Q931StartTimer (cxt, Q931_TIMER_T304);
                        rc = Q931SendInformation (cxt, req);
                        break;
                case Q931_US_NULL:                              /* 0 */
                case Q931_US_CALL_INITIATED:                    /* 1 */
                case Q931_US_CALL_PRESENT:                      /* 6 */
                case Q931_US_RESUME_REQUEST:                    /* 17 */
                case Q931_US_RELEASE_REQUEST:                   /* 19 */
                        rc = -1;
                        break;
                default:
                        rc = Q931SendInformation (cxt, req);
                        break;
        }
        TRACE_EXIT;
        return rc;
} /* Q931InformationReq */
/*------------------------------------------------------------------------*/
```

E.2.13 src/Q931Notify.h

```
/*
 *      file:           Q931Notify.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Notify.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Notify.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_NOTIFY
#define INCL_Q931_NOTIFY

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/
typedef struct {
        t_NotifyInd             notifyInd;
} t_Q931NotifyReq;

/*----- global functions --------------------------------------------------*/
extern  void    Q931NotifyInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931NotifyReq (t_Q931Cxt *cxt, const t_Q931NotifyReq *req);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_Q931_NOTIFY */
```

E.2.14 src/Q931Notify.c

```
/*
 *      file:           Q931Notify.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Notify.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Notify.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Notify.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Notify.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void NotifyInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the internal NOTIFY-IND
 */
{
        TRACE_ENTER (("NotifyInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("NOTIFY INDICATION"));
/*** bearer cap ***/
        /* ffs */
/*** notify ind ***/
        /* ffs */
/*** display ***/
        /* ffs */
        *rcsid = *rcsid;                /* to make gcc happy */
        TRACE_EXIT;
```

```
} /* NotifyInd */
/*------------------------------------------------------------------------*/
void Q931NotifyInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming NOTIFY message
 */
{
        TRACE_ENTER (("Q931NotifyInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 NOTIFY"));
        switch (cxt->state) {
                case Q931_US_ACTIVE:                            /* 10 */
                        NotifyInd (cxt, msg);
                        break;
                default:
                        /* handle error */
                        Q931_FFS ();
                        break;
        }
        TRACE_EXIT;
} /* Q931NotifyInd */
/*------------------------------------------------------------------------*/
static int Q931SendNotify (t_Q931Cxt *cxt, const t_Q931NotifyReq *req)
/*
 * DESC
 *      Genrates and sned a Q.931 NOTIFY message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendNotify ()"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_NOTIFY;
/*** notification indicator ***/
        len += Q931PushNotifyInd (buf+len, &req->notifyInd);
/*** send the message ***/
        TRACEX (DFLAG_Q931, ("IN: Q.931 NOTIFY"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendNotify */
/*------------------------------------------------------------------------*/
int Q931NotifyReq (t_Q931Cxt *cxt, const t_Q931NotifyReq *req)
/*
 * DESC
 *      Handles the internal NOTIFY-REQ
 */
{
        int                     rc;

        TRACE_ENTER (("Q931NotifyReq ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("NOTIFY REQUEST"));
        switch (cxt->state) {
                case Q931_US_ACTIVE:                            /* 10 */
                        rc = Q931SendNotify (cxt, req);
                        break;
                default:
                        rc = -1;
                        break;
        }
        TRACE_EXIT;
        return rc;
} /* Q931NotifyReq */
/*------------------------------------------------------------------------*/
```

E.2.15 src/Q931Progress.h

```
/*
 *      file:           Q931Progress.h
 *      written by:     Christian Zahl
 *      description:    Handles all Q.931 PROGRESS related stuff
 *
 * $Id: Q931Progress.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Progress.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_PROGRESS
#define INCL_Q931_PROGRESS

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/
typedef struct {
```

```
        t_BearerCap             bearerCap;
        t_Cause                 cause;
        t_ProgressInd           progressInd;
        t_HighLayerComp         highLayerComp;
} t_Q931ProgressReq;

/*----- global functions -------------------------------------------------*/
extern  void    Q931ProgressInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931ProgressReq (t_Q931Cxt *cxt, const t_Q931ProgressReq *req);

/*----- global variables -------------------------------------------------*/

#endif /* INCL_Q931_PROGRESS */
```

E.2.16 src/Q931Progress.c

```
/*
 *      file:           Q931Progress.c
 *      written by:     Christian Zahl
 *      description:    Handles all the Q.931 PROGRESS related stuff
 *
 * $Id: Q931Progress.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Progress.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Progress.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles --------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles ------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Progress.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines ----------------------------------------------------------*/

/*----- type definitions -------------------------------------------------*/

/*----- local functions --------------------------------------------------*/

/*----- global functions -------------------------------------------------*/

/*----- local variables --------------------------------------------------*/

/*----- global variables -------------------------------------------------*/

/*--------------------------------------------------------------------------*/
static void ProgressInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the internal PROGRESS-IND
 */
{
        t_IsdnInd               ind;

        TRACE_ENTER (("ProgressInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("PROGRESS"));
/*** bearer cap ***/
        /* ffs */
/*** cause ***/
        /* ffs */
/*** progress ind ***/
        if (msg->progressInd[0].descr) {
                Q931_ZERO (ind);
                ind.ind.indType = ISDN_PROGRESS_IND;
                ind.ind.callId = cxt->callId;
                memcpy (&ind.progressInd.progress, &msg->progressInd[0], T_PROGRESS_IND_SIZE);
                IsdnCallback (&ind);
        }
/*** display ***/
        /* ffs */
/*** hlc ***/
        /* ffs */
        *rcsid = *rcsid;                /* to make gcc happy */
        TRACE_EXIT;
} /* ProgressInd */
/*--------------------------------------------------------------------------*/
void Q931ProgressInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming Q.931 PROGRESS-IND
 */
{
        TRACE_ENTER (("Q931ProgressInd ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 PROGRESS"));
        switch (cxt->state) {
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
                        if (msg->progressInd[0].descr == PR_D_INTERNETWORKING)
```

```
                        Q931StopTimer (cxt, Q931_TIMER_T304);
                ProgressInd (cxt, msg);
                break;
        case Q931_US_OUT_CALL_PROCEEDING:               /* 3 */
                Q931StopTimer (cxt, Q931_TIMER_T310);
                ProgressInd (cxt, msg);
                break;
        default:
                /* handle error */
                Q931_FFS ();
                break;
    }
    TRACE_EXIT;
} /* Q931ProgressInd */
/*---------------------------------------------------------------------------*/
static int Q931SendProgress (t_Q931Cxt *cxt, const t_Q931ProgressReq *req)
/*
 * DESC
 *      Generates and sends a Q.931 PROGRESS message.
 */
{
    t_Byte                  buf[1000];
    int                     len = 0;
    int                     rc;

    TRACE_ENTER (("Q931SendProgress ()"));
    buf[len++] = Q931_DISCRIMINATOR;
    len += Q931PushCallRef (buf+len, &cxt->callRef);
    buf[len++] = Q931_MT_PROGRESS;
/*** bearer cap ***/
    if (req->bearerCap.transferCap)
        len += Q931PushBearerCap (buf+len, &req->bearerCap);
/*** cause ***/
    if (req->cause.value)
        len += Q931PushCause (buf+len, &req->cause);
/*** progress ind ***/
    if (req->progressInd.descr)
        len += Q931PushProgressInd (buf+len, &req->progressInd);
/*** hlc ***/
 /* NOTE: has to be changed */
    if (req->highLayerComp.dummy)
        len += Q931PushHighLayerComp (buf+len, &req->highLayerComp);
/*** send the message ***/
    TRACEX (DFLAG_Q931, ("OUT: Q.931 PROGRESS"));
    rc = Q931SendMsg (buf, len);
    TRACE_EXIT;
    return rc;
} /* Q931SendProgress */
/*---------------------------------------------------------------------------*/
int Q931ProgressReq (t_Q931Cxt *cxt, const t_Q931ProgressReq *req)
/*
 * DESC
 *      Handles the internal PROGRESS-REQ.
 */
{
    int                     rc;

    TRACE_ENTER (("Q931ProgressReq ()"));
    TRACEX (DFLAG_Q931_INTERNAL, ("PROGRESS REQUEST"));
    switch (cxt->state) {
        case Q931_US_IN_CALL_PROCEEDING:                /* 9 */
                rc = Q931SendProgress (cxt, req);
                break;
        case Q931_US_OVERLAP_RECEIVING:                 /* 25 */
                rc = Q931SendProgress (cxt, req);
                break;
        default:
                rc = -1;
                break;
    }
    TRACE_EXIT;
    return rc;
} /* Q931ProgressReq */
/*---------------------------------------------------------------------------*/
```

E.2.17 src/Q931Release.h

```
/*
 *      file:           Q931Release.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Release.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Release.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_RELEASE
#define INCL_Q931_RELEASE

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
```

```
#include "Types.h"
#include "Q931.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        t_Cause                 cause;
} t_Q931ReleaseReq;

typedef struct {
        t_Cause                 cause;
} t_Q931ReleaseCompleteReq;

/*----- global functions ----------------------------------------------------*/
extern  void    Q931ReleaseInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  void    Q931ReleaseCompleteInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931SendRelease (t_Q931Cxt *cxt, const t_Q931ReleaseReq *req);
extern  int     Q931ReleaseReq (t_Q931Cxt *cxt, const t_Q931ReleaseReq *req);

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_Q931_RELEASE */
```

E.2.18 src/Q931Release.c

```
/*
 *      file:           Q931Release.c
 *      written by:     Christian Zahl
 *      description:    Handles all the Q.931 RELEASE and RELEASE-COMPLETE
 *                      related stuff
 *
 * $Id: Q931Release.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Release.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Release.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <unistd.h>

/*----- user includefiles ---------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Release.h"
#include "Q931Timer.h"
#include "Q931Main.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void ReleaseInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the internal RELEASE-IND
 */
{
        t_IsdnInd               ind;

        TRACE_ENTER (("ReleaseInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931_INTERNAL, ("RELEASE INDICATION"));
        Q931_ZERO (ind);
/*** cause ***/
        ind.releaseInd.indType = ISDN_RELEASE_IND;
        ind.releaseInd.callId = cxt->callId;
        ind.releaseInd.cause = msg->cause;
        IsdnCallback (&ind);
/*** display ***/
        /* ffs */
/*** signal ***/
        /* ffs */
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* ReleaseInd */
/*---------------------------------------------------------------------------*/
static int Q931SendReleaseComplete (t_Q931Cxt *cxt, const t_Q931ReleaseCompleteReq *req)
/*
```

```
 * DESC
 *      Generates and sends a Q.931 RELEASE-COMPLETE message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendReleaseComplete (%p, %p)", cxt, req));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_RELEASE_COMPLETE;
/*** cause ***/
        if (req->cause.value)
                len += Q931PushCause (buf+len, &req->cause);
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 RELEASE-COMPLETE"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendReleaseComplete */
/*------------------------------------------------------------------------*/
void Q931ReleaseInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming RELEASE-IND
 */
{
        t_Q931ReleaseCompleteReq        req;

        TRACE_ENTER (("Q931ReleaseInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931, ("IN: Q.931 RELEASE"));
        Q931_ZERO (req);
        switch (cxt->state) {
                case Q931_US_NULL:                              /* 0 */
                        Q931SendReleaseComplete (cxt, &req);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_CALL_PRESENT:                      /* 6 */
                        Q931SendReleaseComplete (cxt, &req);
                        ReleaseInd (cxt, msg);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_DISCONNECT_REQUEST:                /* 11 */
                        Q931StopTimer (cxt, Q931_TIMER_T305);
                        Q931SendReleaseComplete (cxt, &req);
                        ReleaseInd (cxt, msg);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_DISCONNECT_INDICATION:             /* 12 */
                        Q931SendReleaseComplete (cxt, &req);
                        ReleaseInd (cxt, msg);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_RELEASE_REQUEST:                   /* 19 */
                        Q931StopTimer (cxt, Q931_TIMER_T308);
                        ReleaseInd (cxt, msg);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_CALL_INITIATED:                    /* 1 */
                case Q931_US_SUSPEND_REQUEST:                   /* 15 */
                case Q931_US_RESUME_REQUEST:                    /* 17 */
                        Q931_FFS ();
                        break;
                default:
                        Q931StopAllTimers (cxt);
                        Q931SendReleaseComplete (cxt, &req);
                        ReleaseInd (cxt, msg);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
        } /* switch */
        TRACE_EXIT;
} /* Q931ReleaseInd */
/*------------------------------------------------------------------------*/
void Q931ReleaseCompleteInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming RELEASE-COMPLETE-IND
 */
{
        TRACE_ENTER (("Q931ReleaseCompleteInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931, ("IN: Q.931 RELEASE-COMPLETE"));
```

```
        switch (cxt->state) {
                case Q931_US_NULL:                              /* 0 */
                        cxt->state = Q931_US_NULL;              /* 0 */
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_CALL_INITIATED:                    /* 1 */
                        Q931StopTimer (cxt, Q931_TIMER_T303);
                        ReleaseInd (cxt, msg);                  /* named "reject ind" in Q.931 */
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        cxt->state = Q931_US_NULL;              /* 0 */
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
                case Q931_US_RELEASE_REQUEST:                   /* 19 */
                        Q931StopTimer (cxt, Q931_TIMER_T308);
                        ReleaseInd (cxt, msg);                  /* named "release conf" in Q.931 */
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        cxt->state = Q931_US_NULL;              /* 0 */
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
                default:
                        Q931StopAllTimers (cxt);
                        Q931ReleaseInd (cxt, msg);
                        if (cxt->fd != -1)
                                close (cxt->fd);
                        cxt->state = Q931_US_NULL;              /* 0 */
                        Q931FreeCallRef (&cxt->callRef);
                        Q931FreeCxt (cxt);
                        break;
        } /* switch */
        TRACE_EXIT;
} /* Q931ReleaseCompleteInd */
/*------------------------------------------------------------------------*/
int Q931SendRelease (t_Q931Cxt *cxt, const t_Q931ReleaseReq *req)
/*
 * DESC
 *      Generates and sends a Q.931 RELEASE message
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendRelease (%p, %p)", cxt, req));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_RELEASE;
/*** cause ***/
        if (req->cause.value)
                len += Q931PushCause (buf+len, &req->cause);
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 RELEASE"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendRelease */
/*------------------------------------------------------------------------*/
int Q931ReleaseReq (t_Q931Cxt *cxt, const t_Q931ReleaseReq *req)
/*
 * DESC
 *      Handles the internal RELEASE-REQ
 */
{
        int                     rc;
        t_Q931ReleaseReq        tmp;
        t_Q931ReleaseCompleteReq        relCompl;

        TRACE_ENTER (("Q931ReleaseReq (%p, %p)", cxt, req));
        TRACEX (DFLAG_Q931_INTERNAL, ("RELEASE-REQUEST"));
        switch (cxt->state) {
                case Q931_US_OVERLAP_SENDING:                   /* 2 */
                        Q931StopTimer (cxt, Q931_TIMER_T304);
                        Q931StartTimer (cxt, Q931_TIMER_T308);
#ifdef NEW
/*** set the cause to 6 ***/
                        memcpy (&tmp, req, sizeof (tmp));
                        Q931_ZERO (tmp.cause);
                        tmp.cause.value = CA_V_CAHNNEL_UNACCEPTABLE;
                        rc = Q931SendRelease (cxt, &tmp);
#else
                        rc = Q931SendRelease (cxt, req);
#endif
                        cxt->state = Q931_US_RELEASE_REQUEST;   /* 19 */
                        break;
                case Q931_US_OUT_CALL_PROCEEDING:               /* 3 */
                        Q931StopTimer (cxt, Q931_TIMER_T310);
                        Q931StartTimer (cxt, Q931_TIMER_T308);
/*** set the cause to 6 ***/
                        memcpy (&tmp, req, sizeof (tmp));
                        Q931_ZERO (tmp.cause);
                        tmp.cause.value = CA_V_CAHNNEL_UNACCEPTABLE;
                        rc = Q931SendRelease (cxt, &tmp);
                        cxt->state = Q931_US_RELEASE_REQUEST;   /* 19 */
                        break;
                case Q931_US_CALL_PRESENT:                      /* 6 */
/*
```

```
 * NOTE: called "REJECT REQUEST" in the Q.931 state diagram, s.a. 5.2.5.[23]!
 * NOTE: if no cause is given, we will use "USER BUSY"!
 */
                        Q931StopTimer (cxt, Q931_TIMER_T300);   /* our private timer */
                        Q931_ZERO (relCompl);
                        if (req->cause.value)
                                relCompl.cause = req->cause;
                        else
                                relCompl.cause.value = CA_V_USER_BUSY;
                        rc = Q931SendReleaseComplete (cxt, &relCompl);
                        cxt->state = Q931_US_NULL;              /* 0 */
                        break;
#ifdef OLD
                case Q931_US_DISCONNECT_INDICATION:             /* 11 */
                        /* NOTE: added for timeout of T305 */
                        rc = Q931SendRelease (cxt, req);
                        break;
#endif
                case Q931_US_DISCONNECT_INDICATION:             /* 12 */
                        cxt->state = Q931_US_RELEASE_REQUEST;   /* 19 */
                        Q931StartTimer (cxt, Q931_TIMER_T308);
                        rc = Q931SendRelease (cxt, req);
                        break;
#ifdef OLD
                case Q931_US_RESUME_REQUEST:                    /* 17 */
/* NOTE: added for timeout of T318 */
                        rc = Q931SendRelease (cxt, req);
                        break;
#endif
#ifdef OLD
                case Q931_US_RELEASE_REQUEST:                   /* 19 */
/* NOTE: added for timeout of T308 */
                        rc = Q931SendRelease (cxt, req);
                        cxt->state = Q931_US_RELEASE_REQUEST;   /* 19 */
                        break;
#endif
                default:
                        rc = -1;
                        break;
        } /* switch */
        TRACE_EXIT;
        return rc;
} /* Q931ReleaseReq */
/*------------------------------------------------------------------------*/
```

E.2.19 src/Q931Setup.h

```
/*
 *      file:           Q931Setup.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Setup.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Setup.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_SETUP
#define INCL_Q931_SETUP

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/
typedef struct {
        t_BearerCap             bearerCap;
        t_CallingPartyNo        callingPartyNo;
        t_CallingPartySubAddr   callingPartySubAddr;
        t_CalledPartyNo         calledPartyNo;
        t_CalledPartySubAddr    calledPartySubAddr;
} t_Q931SetupReq;

/*----- global functions --------------------------------------------------*/
extern  void    Q931SetupInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931SendSetup (t_Q931Cxt *cxt, t_Q931SetupReq *req);
extern  int     Q931SetupReq (t_Q931Cxt *cxt, t_Q931SetupReq *req);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_Q931_SETUP */
```

E.2.20 src/Q931Setup.c

```
/*
 *      file:           Q931Setup.c
```

```
 *      written by:     Christian Zahl
 *      description:    Handles all SETUP related stuff
 *
 * $Id: Q931Setup.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Setup.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Setup.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Setup.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*-------------------------------------------------------------------------*/
static void SetupInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC:
 *      We received an incomming SETUP (i.e. a call).
 */
{
        t_IsdnInd               ind;

        TRACE_ENTER (("SetupInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931_INTERNAL, ("SETUP INDICATION"));
        cxt->callId = IsdnNewCallId ();
        cxt->channelId = msg->channelId;
/*** generate the internal indication ***/
        Q931_ZERO (ind);
        ind.callInd.indType = ISDN_CALL_IND;
        ind.callInd.callId = cxt->callId;
        ind.callInd.calledPartyNo = msg->calledPartyNo;
        ind.callInd.callingPartyNo = msg->callingPartyNo;
        ind.callInd.channelId = msg->channelId;
        ind.callInd.bearerCap[0] = msg->bearerCap[0];
        ind.callInd.bearerCap[1] = msg->bearerCap[1];
/*** call the user ***/
        IsdnCallback (&ind);
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* SetupInd */
/*-------------------------------------------------------------------------*/
void Q931SetupInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      The network has send us an SETUP message to signal an incomming
 *      call.
 * NOTE:
 *      When the second SETUP arrives, and noone answered the call within
 *      4s, then the network releases the call, but no indication is
 *      send. But we have to remove the reference (the cxt), so we start
 *      T300.
 */
{
        TRACE_ENTER (("Q931SetupInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931, ("IN: Q.931 SETUP"));
        switch (cxt->state) {
                case Q931_US_NULL:                              /* 0 */
                        cxt->state = Q931_US_CALL_PRESENT;      /* 6 */
                        Q931CopyCallRef (&cxt->callRef, &msg->callRef);
                        Q931StartTimer (cxt, Q931_TIMER_T300);  /* start our local timer */
                        SetupInd (cxt, msg);
                        break;
                default:
                        ;       /* do nothing ! */
        } /* switch */
        TRACE_EXIT;
} /* Q931SetupInd */
/*-------------------------------------------------------------------------*/
int Q931SendSetup (t_Q931Cxt *cxt, t_Q931SetupReq *req)
/*
 * DESC
 *      Generates and sends a Q.931 SETUP message.
 */
{
```

```
        t_Byte                  buf[1000];
        int                     len;
        int                     rc;

        TRACE_ENTER (("Q931SendSetup (%p, %p)", cxt, req));
        if (req == NULL) {
                TRACE_EXIT;
                return -1;
        }
        len = 0;
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_SETUP;
/*** sending complete ***/
        /* ffs */
/*** repeat indicator ***/
        len += Q931PushBearerCap (buf+len, &req->bearerCap);
/*** channel ind ***/
        /* ffs */
/*** progress ind ***/
        /* ffs */
/*** net spec fac ***/
        /* ffs */
/*** keypad ***/
        /* ffs */
/*** calling party no ***/
/* Dump (&req->callingPartyNo, sizeof (req->callingPartyNo)); */
        if (req->callingPartyNo.number[0] != '\0')
                len += Q931PushCallingPartyNo (buf+len, &req->callingPartyNo);
/*** calling party subaddr ***/
        /* ffs */
/*** called party no ***/
        if (req->calledPartyNo.number[0] != '\0')
                len += Q931PushCalledPartyNo (buf+len, &req->calledPartyNo);
/*** called party subaddr ***/
        /* ffs */
/*** transit net sel ***/
        /* ffs */
/*** repeat ind ***/
        /* ffs */
/*** llc ***/
        /* ffs */
/*** hlc ***/
        /* ffs */
/*** send it ***/
        TRACEX (DFLAG_Q931, ("OUT: Q.931 SETUP"));
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendSetup */
/*------------------------------------------------------------------------*/
int Q931SetupReq (t_Q931Cxt *cxt, t_Q931SetupReq *req)
/*
 * DESC
 *      Handles the internal SETUP-REQ.
 */
{
        int                     rc;

        TRACE_ENTER (("Q931SetupReq (&p, %p)", cxt, req));
        TRACEX (DFLAG_Q931_INTERNAL, ("SETUP REQ"));
        switch (cxt->state) {
                case Q931_US_NULL:                              /* 0 */
                        Q931StartTimer (cxt, Q931_TIMER_T303);
                        cxt->state = Q931_US_CALL_INITIATED;    /* 1 */
                        rc = Q931SendSetup (cxt, req);
                        break;
                default:
                        rc = -1;
                        break;
        } /* switch */
        TRACE_EXIT;
        return rc;
} /* Q931SetupReq */
/*------------------------------------------------------------------------*/
```

E.2.21 src/Q931SetupAck.h

```
/*
 *      file:           Q931SetupAck.h
 *      written by:     Christian Zahl
 *      description:    Defines the Q.931 SETUP-ACKNOWLEDGE related stuff
 *
 * $Id: Q931SetupAck.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931SetupAck.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_SETUP_ACK
#define INCL_Q931_SETUP_ACK

/*----- standard includefiles ---------------------------------------------*/
```

```
/*----- local includefiles --------------------------------------------------*/
#include "Types.h"
#include "Q931.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        t_ChannelId             channelId;
        t_ProgressInd           progressInd;
} t_Q931SetupAckReq;

/*----- global functions ----------------------------------------------------*/
extern  void    Q931SetupAckInd (t_Q931Cxt *cxt, const t_Q931Msg *msg);
extern  int     Q931SetupAckReq (t_Q931Cxt *cxt, t_Q931SetupAckReq *req);

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_Q931_SETUP_ACK */
```

E.2.22 src/Q931SetupAck.c

```
/*
 *      file:           Q931SetupAck.c
 *      written by:     Christian Zahl
 *      description:    Handles the Q.931 SETUP-ACKNOWLEDGE related stuff
 *
 * $Id: Q931SetupAck.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931SetupAck.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931SetupAck.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <sys/isdnio.h>
#include <unistd.h>
#include <fcntl.h>

/*----- user includefiles ---------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Main.h"           /* for DLPI fd */
#include "Q931Msg.h"
#include "Q931SetupAck.h"
#include "Q931Timer.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static void SetupAckInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the internal SETUP-ACKNOWLEDGE-IND
 */
{
        t_IsdnInd               ind;
        isdn_format_req_t       isdnFormatReq;
        char                    fn[100];
        int                     fd;

        TRACE_ENTER (("SetupAckInd ()"));
        TRACEX (DFLAG_Q931_INTERNAL, ("SETUP ACKNOWLEDGE"));
/*** channel-id ***/
        Q931_ZERO (ind);
        ind.channelInd.indType = ISDN_CHANNEL_IND;
        ind.channelInd.callId = cxt->callId;
        ind.channelInd.channelId = msg->channelId;
        ind.channelInd.fd = -1;
        if (msg->channelId.interfaceType == CID_IT_BRI) {
                Q931_ZERO (isdnFormatReq);
                ISDN_SET_FORMAT_VOICE_ALAW (&isdnFormatReq.format);
                if (msg->channelId.infoChannelSel == CID_ICS_B1) {
                        isdnFormatReq.channel = ISDN_CHAN_TE_B1;
                        strcpy (fn, "/dev/isdn/0/te/b1");
                } else if (msg->channelId.infoChannelSel == CID_ICS_B2) {
                        isdnFormatReq.channel = ISDN_CHAN_TE_B2;
                        strcpy (fn, "/dev/isdn/0/te/b2");
                } else {
                        /** what's that??? ***/
```

```
                return;
        }
        if (ioctl (q931DlpiFd, ISDN_SET_FORMAT, &isdnFormatReq) < 0) {
                Q931_FFS ();
                return;
        }
        if (cxt->fd == -1) {
                if ((fd = open (fn, O_RDWR)) < 0) {
                        Q931_FFS ();
                        return;
                }
                cxt->fd = fd;
        }
        ind.channelInd.fd = cxt->fd;
} else {
        Q931_FFS ();
        return;
}
IsdnCallback (&ind);
/*** progress ind ***/
if (msg->progressInd[0].location) {
        Q931_ZERO (ind);
        ind.progressInd.indType = ISDN_PROGRESS_IND;
        ind.progressInd.callId = cxt->callId;
        ind.progressInd.progress = msg->progressInd[0];
        IsdnCallback (&ind);
}
*rcsid = *rcsid;               /* to make gcc happy */
TRACE_EXIT;
} /* SetupAckInd */
/*------------------------------------------------------------------------*/
void Q931SetupAckInd (t_Q931Cxt *cxt, const t_Q931Msg *msg)
/*
 * DESC
 *      Handles the incomming Q.931 SETUP-ACKNOWLEDGE-IND
 */
{
        TRACE_ENTER (("Q931SetupAckInd (%p, %p)", cxt, msg));
        TRACEX (DFLAG_Q931, ("IN: Q.931 SETUP-ACKNOWLEDGE"));
        switch (cxt->state) {
                case Q931_US_CALL_INITIATED:                    /* 1 */
                        Q931StopTimer (cxt, Q931_TIMER_T303);
                        SetupAckInd (cxt, msg);                 /* named "more info ind "in Q.931 */
                        Q931StartTimer (cxt, Q931_TIMER_T304);
                        cxt->state = Q931_US_OVERLAP_SENDING;
                        break;
                default:
                        /* handle error */
                        Q931_FFS ();
                        break;
        }
        TRACE_EXIT;
} /* Q931SetupAckInd */
/*------------------------------------------------------------------------*/
static int Q931SendSetupAck (t_Q931Cxt *cxt, const t_Q931SetupAckReq *req)
/*
 * DESC
 *      Generates and sends a Q.931 SETUP-ACKNOWLEDGE message.
 */
{
        t_Byte                  buf[1000];
        int                     len = 0;
        int                     rc;

        TRACE_ENTER (("Q931SendSetupAck ()"));
        TRACEX (DFLAG_Q931, ("IN: Q.931 SETUP-ACKNOWLEDGE"));
        buf[len++] = Q931_DISCRIMINATOR;
        len += Q931PushCallRef (buf+len, &cxt->callRef);
        buf[len++] = Q931_MT_SETUP;
/*** channel id ***/
        if (req->channelId.infoChannelSel)
                len += Q931PushChannelId (buf+len, &req->channelId);
/*** progress ind ***/
        if (req->progressInd.descr)
                len += Q931PushProgressInd (buf+len, &req->progressInd);
/*** send it ***/
        rc = Q931SendMsg (buf, len);
        TRACE_EXIT;
        return rc;
} /* Q931SendSetupAck */
/*------------------------------------------------------------------------*/
int Q931SetupAckReq (t_Q931Cxt *cxt, t_Q931SetupAckReq *req)
/*
 * DESC
 *      Handles the internal SETUP-ACKNOWLEDGEREQ.
 */
{
        int                     rc;

        TRACE_ENTER (("Q931SetupAckReq (%p, %p)", cxt, req));
        TRACEX (DFLAG_Q931_INTERNAL, ("SETUP-ACKNOWLEDGE REQUEST"));
        switch (cxt->state) {
                case Q931_US_CALL_PRESENT:                      /* 6 */
                        cxt->state = Q931_US_OVERLAP_RECEIVING; /* 25 */
                        rc = Q931SendSetupAck (cxt, req);
                        break;
                default:
                        rc = -1;
                        break;
```

```
        }
        TRACE_EXIT;
        return rc;
} /* Q931SetupAckReq */
/*--------------------------------------------------------------------------*/
```

E.3 Behandlung von Information Elements

Jedes Information Element wird durch eines der folgenden C-Files behandelt.

E.3.1 src/BearerCap.h

```
/*
 *      file:           BearerCap.h
 *      written by:     Christian Zahl
 *      description:    Defines the bearer capaibility and its values.
 *
 * $Id: BearerCap.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: BearerCap.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_BEARER_CAP
#define INCL_BEARER_CAP

/*----- standard includefiles ----------------------------------------------*/

/*----- local includefiles -------------------------------------------------*/

/*----- defines ------------------------------------------------------------*/
#define T_BEARER_CAP_SIZE               sizeof (t_BearerCap)

/*----- type definitions ---------------------------------------------------*/
        /*** NOTE: the values MUST be the Q.931 values + 0x100! ***/
typedef enum {
        BC_TC_SPEECH            = 0x100,
        BC_TC_UDI               = 0x108,
        BC_TC_RDI               = 0x109,
        BC_TC_AUDIO             = 0x110,
        BC_TC_UDITA             = 0x111,
        BC_TC_VIDEO             = 0x118,
        BC_TC_DEFAULT           = BC_TC_SPEECH
} t_BearerCapTransferCap;

typedef enum {
        BC_TM_CIRCUIT           = 0x100,
        BC_TM_PACKET            = 0x102,
        BC_TM_DEFAULT           = BC_TM_CIRCUIT
} t_BearerCapTransferMode;

typedef enum {
        BC_L1P_MULAW            = 0x102,
        BC_L1P_ALAW             = 0x103,
        BC_L1P_ADPCM            = 0x104,
        BC_L1P_DEFAULT          = BC_L1P_ALAW
} t_BearerCapLayer1Prot;

typedef enum {
        BC_SA_SYNCH             = 0x100,
        BC_SA_ASYNCH            = 0x101
} t_BearerCapSynchAsynch;

typedef enum {
        BC_NE_NOT_POSSIBLE      = 0x100,
        BC_NE_POSSIBLE          = 0x101
} t_BearerCapNegotiation;

typedef enum {
        BC_UR_600               = 0x101,
        BC_UR_1200              = 0x102,
        /* ... */
        BC_UR_12000             = 0x11f
} t_BearerCapUserRate;

typedef enum {
        BC_NIC_NONE             = 0x100,
        BC_NIC_RX               = 0x101,
        BC_NIC_TX               = 0x102,
        BC_NIC_RX_TX            = 0x103
} t_BearerCapNic;

typedef enum {
        BC_FL_NONE              = 0x100,
        BC_FL_RX                = 0x101,
        BC_FL_TX                = 0x102,
        BC_FL_RX_TX             = 0x103
} t_BearerCapFlow;
```

```
        /* ... */

typedef enum {
        BC_SB_1                 = 0x101,
        BC_SB_15                = 0x102,
        BC_SB_2                 = 0x103
} t_BearerCapStopBits;

typedef enum {
        BC_DB_5                 = 0x101,
        BC_DB_7                 = 0x102,
        BC_DB_8                 = 0x103
} t_BearerCapDataBits;

typedef enum {
        BC_PAR_ODD              = 0x100,
        BC_PAR_EVEN             = 0x102,
        BC_PAR_NONE             = 0x103,
        BC_PAR_SPACE            = 0x104,
        BC_PAR_MARK             = 0x105
} t_BearerCapParity;

typedef enum {
        BC_DUP_HALF             = 0x100,
        BC_DUP_FULL             = 0x101,
        BC_DUP_DEFAULT          = BC_DUP_FULL
} t_BearerCapDuplex;

typedef enum {
        BC_MOD_V21              = 0x111,
        BC_MOD_V22              = 0x112,
        /* ... */
        BC_MOD_V32              = 0x11d
} t_BearerCapModem;

typedef enum {
        BC_L2_Q921              = 0x102,
        BC_L2_X25               = 0x106
} t_BearerCapLayer2Prot;

typedef enum {
        BC_L3_Q921              = 0x102,
        BC_L3_X25               = 0x106
} t_BearerCapLayer3Prot;

typedef struct {
        t_BearerCapTransferCap  transferCap;
        t_BearerCapTransferMode transferMode;
        int                     transferRate;
        t_BearerCapLayer1Prot   layer1Prot;
        t_BearerCapSynchAsynch  synchAsynch;
        t_BearerCapNegotiation  negotiation;
        t_BearerCapUserRate     userRate;
/*      t_BearerCapInterRate    interRate; */
        t_BearerCapNic          nic;
        t_BearerCapFlow         flow;
        /* ... */
        t_BearerCapStopBits     stopBits;
        t_BearerCapDataBits     dataBits;
        t_BearerCapParity       parity;
        t_BearerCapDuplex       duplex;
        t_BearerCapModem        modem;
        t_BearerCapLayer2Prot   layer2Prot;
        t_BearerCapLayer3Prot   layer3Prot;
} t_BearerCap;

/*----- global functions -----------------------------------------------------*/
extern  int     Q931PopBearerCap (const t_Byte *buf, t_BearerCap *bc);
extern  int     Q931PushBearerCap (t_Byte *buf, const t_BearerCap *bc);

/*----- global variables -----------------------------------------------------*/

#endif /* INCL_BEARER_CAP */
```

E.3.2 src/BearerCap.c

```
/*
 *      file:           BearerCap.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: BearerCap.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: BearerCap.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: BearerCap.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ------------------------------------------------*/
#include <string.h>

/*----- user includefiles ----------------------------------------------------*/
#include "Types.h"
```

```
#include "Q931.h"
#include "BearerCap.h"
#include "Misc.h"

/*----- defines -------------------------------------------------------------*/
#define BC_TR_PACKET            0x00
#define BC_TR_64                0x10
#define BC_TR_128               0x11
#define BC_TR_384               0x13
#define BC_TR_1536              0x15
#define BC_TR_1920              0x17
#define BC_TR_MULTIRATE         0x18

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
int Q931PopBearerCap (const t_Byte *buf, t_BearerCap *bc)
/*
 * DESC
 *      Converts the Q.931 bearer capability IE into the internal
 *      C representation.
 * NOTE
 *      Currently the bytes 5a, 5b, 5c and 5d will be skipped.
 * RET
 *      # bytes consumed
 */
{
        int                     len;
        int                     i;
        int                     rate;
        int                     multirate = 0;

        memset (bc, '\0', sizeof (*bc));
        len = buf[1] +2;
        i = 2;
/*** 3 ***/
        if (GET_BITS_76 (buf[i]) != Q931_CODING_Q931)
                return len +2;
        bc->transferCap = GET_BITS_54321 (buf[i]) | 0x100;
        Q931_SKIP_GROUP (buf, i);
/*** 4 ***/
        bc->transferMode = GET_BITS_76 (buf[i]) | 0x100;
        rate = GET_BITS_54321 (buf[i]);
        if (rate == BC_TR_PACKET)
                bc->transferRate = 0;
        else if (rate == BC_TR_64)
                bc->transferRate = 64;
        else if (rate == BC_TR_128)
                bc->transferRate = 128;
        else if (rate == BC_TR_384)
                bc->transferRate = 384;
        else if (rate == BC_TR_1536)
                bc->transferRate = 1536;
        else if (rate == BC_TR_1920)
                bc->transferRate = 1920;
        else if (rate == BC_TR_MULTIRATE)
                multirate = 1;
        else
                bc->transferRate = 0;
        Q931_SKIP_GROUP (buf, i);
/*** 4.1 ***/
        if (multirate) {
                bc->transferRate = 64 * GET_BITS_7654321 (buf[i]);
        }
        Q931_SKIP_GROUP (buf, i);
/*** 5 ***/
        if (i < len) {
                if (GET_BITS_76 (buf[i] == 0x01))
                        bc->layer1Prot = GET_BITS_54321 (buf[i]) | 0x100;
                Q931_SKIP_GROUP (buf, i);
        }
/*** 6 ***/
        if (i < len) {
                if (GET_BITS_76 (buf[i] == 0x02))
                        bc->layer2Prot = GET_BITS_54321 (buf[i]) | 0x100;
                Q931_SKIP_GROUP (buf, i);
        }
/*** 7 ***/
        if (i < len) {
                if (GET_BITS_76 (buf[i] == 0x03))
                        bc->layer3Prot = GET_BITS_54321 (buf[i]) | 0x100;
                Q931_SKIP_GROUP (buf, i);
        }
/*** # bytes consumed ***/
        return len;
        *rcsid = *rcsid;
} /* Q931PopBearerCap */
/*---------------------------------------------------------------------------*/
int Q931PushBearerCap (t_Byte *buf, const t_BearerCap *bc)
/*
 * DESC
```

```
 *      Converts the C structure of the bearer capability into the Q.931
 *      IE format.
 * NOTE
 *      Currently byte 5a, 5b, 5c and 5d are NOT supported.
 */
{
        int                     len = 0;
        t_Byte                  byte;
        int                     stop = 0;
        int                     rate = 0;
        int                     multirate = 0;

        buf[len++] = Q931_IE_BEARER_CAP;
        buf[len++] = 0;
/*** 3 ***/
        byte = 0x80;
        PUT_BITS_76 (byte, Q931_CODING_Q931);
        if (bc->transferCap)
                PUT_BITS_54321 (byte, bc->transferCap);
        else
                PUT_BITS_54321 (byte, BC_TC_DEFAULT);
        buf[len++] = byte;
/*** 4 ***/
        byte = 0x80;
        if (bc->transferMode)
                PUT_BITS_76 (byte, bc->transferMode);
        else
                PUT_BITS_76 (byte, BC_TM_DEFAULT);
        if (bc->transferRate)
                rate = bc->transferRate;
        rate = (rate / 64) * 64;
        if (rate)
                if (rate == 64)
                        PUT_BITS_54321 (byte, BC_TR_64);
                else if (rate == 128)
                        PUT_BITS_54321 (byte, BC_TR_128);
                else if (rate == 384)
                        PUT_BITS_54321 (byte, BC_TR_384);
                else if (rate == 1536)
                        PUT_BITS_54321 (byte, BC_TR_1536);
                else if (rate == 1920)
                        PUT_BITS_54321 (byte, BC_TR_1920);
                else {
                        PUT_BITS_54321 (byte, BC_TR_MULTIRATE);
                        multirate = 1;
                }
        else
                PUT_BITS_54321 (byte, BC_TR_64);
        buf[len++] = byte;
/*** 4.1 ***/
        if (multirate) {
                byte = 0x80;
                PUT_BITS_7654321 (byte, (rate / 64));
                buf[len++] = byte;
        }
/*** 5 opt ***/
        if (bc->layer1Prot) {
                byte = 0x80;
                PUT_BITS_76 (byte, 0x01);
                PUT_BITS_54321 (byte, bc->layer1Prot);
                buf[len++] = byte;
        } else
                stop = 1;
/*** 6 ***/
        if (!stop && bc->layer2Prot) {
                byte = 0x80;
                PUT_BITS_76 (byte, 0x02);
                PUT_BITS_54321 (byte, bc->layer2Prot);
                buf[len++] = byte;
        } else
                stop = 1;
/*** 7 ***/
        if (!stop && bc->layer3Prot) {
                byte = 0x80;
                PUT_BITS_76 (byte, 0x03);
                PUT_BITS_54321 (byte, bc->layer3Prot);
                buf[len++] = byte;
        } else
                stop = 1;
/*** set the real length of the IE ***/
        buf[1] = len -2;
        return len;
} /* Q931PushBearerCap */
/*---------------------------------------------------------------------------*/
```

E.3.3 src/CalledPartyNo.h

```
/*
 *      file:           CalledPartyNo.h
 *      written by:     Christian Zahl
 *      description:    Definition of called party no IE
 *
 * $Id: CalledPartyNo.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: CalledPartyNo.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
```

```
 * Initial revision
 *
 */
#ifndef INCL_CALLED_PARTY_NO
#define INCL_CALLED_PARTY_NO

/*----- standard includefiles ----------------------------------------------*/

/*----- local includefiles -------------------------------------------------*/
#include "Types.h"

/*----- defines ------------------------------------------------------------*/
#define T_CALLED_PARTY_NO_SIZE  sizeof (t_CalledPartyNo)

/*----- type definitions ---------------------------------------------------*/
/*** NOTE: all values MUST be the Q.931 values + 0x100! ***/
typedef enum {
        CDPN_T_UNKNOWN          = 0x100,
        CDPN_T_INTERNATIONAL    = 0x101,
        CDPN_T_NATIONAL         = 0x102,
        CDPN_T_NETWORK          = 0x103,
        CDPN_T_SUBSCRIBER       = 0x104,
        CDPN_T_ABBREV           = 0x106,
        CDPN_T_DEFAULT          = CDPN_T_UNKNOWN
} t_CalledNoType;

typedef enum {
        CDPN_NP_UNKNOWN         = 0x100,
        CDPN_NP_ISDN            = 0x101,
        CDPN_NP_DATA            = 0x103,
        CDPN_NP_TELEX           = 0x104,
        CDPN_NP_NATIONAL        = 0x108,
        CDPN_NP_PRIVATE         = 0x109,
        CDPN_NP_DEFAULT         = CDPN_NP_UNKNOWN
} t_CalledNoNoPlan;

typedef struct {
        t_CalledNoType          type;                           /* opt. type of the number */
        t_CalledNoNoPlan        numberingPlan;                  /* opt. numbering plan of the number */
        char                    number[20+1];                   /* the number itself, \0 terminated */
} t_CalledPartyNo;

/*----- global functions ---------------------------------------------------*/
extern  int     Q931PushCalledPartyNo (t_Byte *buf, const t_CalledPartyNo *no);
extern  int     Q931PopCalledPartyNo (const t_Byte *buf, t_CalledPartyNo *no);

/*----- global variables ---------------------------------------------------*/

#endif /* INCL_CALLED_PARTY_NO */
```

E.3.4 src/CalledPartyNo.c

```
/*
 *      file:           CalledPartyNo.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: CalledPartyNo.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: CalledPartyNo.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: CalledPartyNo.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ----------------------------------------------*/
#include <string.h>

/*----- user includefiles --------------------------------------------------*/
#include "Q931.h"
#include "CalledPartyNo.h"
#include "Misc.h"

/*----- defines ------------------------------------------------------------*/

/*----- type definitions ---------------------------------------------------*/

/*----- local functions ----------------------------------------------------*/

/*----- global functions ---------------------------------------------------*/

/*----- local variables ----------------------------------------------------*/

/*----- global variables ---------------------------------------------------*/

/*--------------------------------------------------------------------------*/
int Q931PopCalledPartyNo (const t_Byte *buf, t_CalledPartyNo *no)
/*
 * DESC
 *      Decodes the called party number IE from a Q.931 message and convert
 *      it into the internal C structure.
 * RET
 *      # of bytes consumed
 */
```

```
{
        int                     len;
        int                     i;
        int                     j;

        Q931_ZERO (*no);
        len = buf[1] +2;
        i = 2;
/*** 3 ***/
        no->type = GET_BITS_765 (buf[i]) | 0x100;
        no->numberingPlan = GET_BITS_4321 (buf[i]) | 0x100;
        Q931_SKIP_GROUP (buf, i);
/*** 4++ ***/
        j = 0;
        while (i < len)
                no->number[j++] = buf[i++];
        no->number[j++] = '\0';
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopCalledPartyNo */
/*------------------------------------------------------------------------*/
int Q931PushCalledPartyNo (t_Byte *buf, const t_CalledPartyNo *no)
/*
 * DESC
 *      Converts the internal C called party no into the Q.931 representation.
 * RET
 *      # bytes occupied
 */
{
        int                     len = 0;
        t_Byte                  byte;
        int                     i;

        buf[len++] = Q931_IE_CALLED_PARTY_NO;
        buf[len++] = 0;
/*** 3 ***/
        byte = 0x80;
        if (no->type)
                PUT_BITS_765 (byte, no->type);
        else
                PUT_BITS_765 (byte, CDPN_T_DEFAULT);
        if (no->numberingPlan)
                PUT_BITS_4321 (byte, no->numberingPlan);
        else
                PUT_BITS_4321 (byte, CDPN_NP_DEFAULT);
        buf[len++] = byte;
/*** 4++ ***/
        for (i=0; no->number[i] != '\0'; i++)
                buf[len++] = no->number[i] & 0x7f;
/*** set the real length ***/
        buf[1] = len -2;
        return len;
} /* Q931PushCalledPartyNo */
/*------------------------------------------------------------------------*/
```

E.3.5 src/CalledPartySubAddr.h

```
/*
 *      file:           CalledPartySubAddr.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: CalledPartySubAddr.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: CalledPartySubAddr.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CALLED_PARTY_SUB_ADDR
#define INCL_CALLED_PARTY_SUB_ADDR

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/
#define T_CALLED_PARTY_SUB_ADDR_SIZE    sizeof (t_CalledPartySubAddr)

/*----- type definitions --------------------------------------------------*/
typedef enum {
        CDPS_T_NSAP             = 0x100,
        CDPS_T_USER             = 0x102
} t_CalledPartySubType;

typedef enum {
        CDPS_P_EVEN             = 0x100,
        CDPS_P_ODD              = 0x101
} t_CalledPartySubParity;

typedef struct {
        t_CalledPartySubType    type;
        t_CalledPartySubParity  evenOdd;
        int                     len;
        int                     subAddr[20];
```

```
} t_CalledPartySubAddr;

/*----- global functions -------------------------------------------------*/

/*----- global variables -------------------------------------------------*/

#endif /* INCL_CALLED_PARTY_SUB_ADDR */
```

E.3.6 src/CallingPartyNo.h

```
/*
 *      file:           CallingPartyNo.h
 *      written by:     Christian Zahl
 *      description:    Calling Party No Information Element
 *
 * $Id: CallingPartyNo.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: CallingPartyNo.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */
#ifndef INCL_CALLING_PARTY_NO
#define INCL_CALLING_PARTY_NO

/*----- standard includefiles --------------------------------------------*/

/*----- local includefiles -----------------------------------------------*/

/*----- defines ----------------------------------------------------------*/
#define T_CALLING_PARTY_NO_SIZE sizeof (t_CallingPartyNo)

/*----- type definitions -------------------------------------------------*/
/*** NOTE: all values MUST be the Q.931 values + 0x100! ***/
typedef enum {
        CGPN_T_UNKNOWN          = 0x100,
        CGPN_T_INTERNATIONAL    = 0x101,
        CGPN_T_NATIONAL         = 0x102,
        CGPN_T_NETWORK          = 0x103,
        CGPN_T_SUBSCRIBER       = 0x104,
        CGPN_T_ABBREV           = 0x106,
        CGPN_T_DEFAULT          = CGPN_T_UNKNOWN
} t_CallingPartyNoType;

typedef enum {
        CGPN_NP_UNKNOWN         = 0x100,
        CGPN_NP_ISDN            = 0x101,
        CGPN_NP_DATA            = 0x103,
        CGPN_NP_TELEX           = 0x104,
        CGPN_NP_NATIONAL        = 0x108,
        CGPN_NP_PRIVATE         = 0x109,
        CGPN_NP_DEFAULT         = CGPN_NP_UNKNOWN
} t_CallingPartyNoNoPlan;

typedef enum {
        CGPN_P_ALLOWED          = 0x100,
        CGPN_P_RESTRICTED       = 0x101,
        CGPN_P_NOT_AVAIL        = 0x102,
        CGPN_P_DEFAULT          = CGPN_P_ALLOWED
} t_CallingPartyNoPresent;

typedef enum {
        CGPN_S_USER_NOT         = 0x100,
        CGPN_S_USER_PASSED      = 0x101,
        CGPN_S_USER_FAILED      = 0x102,
        CGPN_S_NETWORK          = 0x103,
        CGPN_S_DEFAULT          = CGPN_S_USER_NOT
} t_CallingPartyNoScreen;

typedef struct {
        t_CallingPartyNoType    type;
        t_CallingPartyNoNoPlan  numberingPlan;
        t_CallingPartyNoPresent presentation;
        t_CallingPartyNoScreen  screening;
        char                    number[20+1];                   /* calling no, \0 terminated */
} t_CallingPartyNo;

/*----- global functions -------------------------------------------------*/
extern  int                     Q931PushCallingPartyNo (t_Byte *buf, const t_CallingPartyNo *no);
extern  int                     Q931PopCallingPartyNo (const t_Byte *buf, t_CallingPartyNo *no);

/*----- global variables -------------------------------------------------*/

#endif /* INCL_CALLING_PARTY_NO */
```

E.3.7 src/CallingPartyNo.c

```
/*
 *      file:           CallingPartyNo.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
```

```
 * $Id: CallingPartyNo.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: CallingPartyNo.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: CallingPartyNo.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Q931.h"
#include "CallingPartyNo.h"
#include "Types.h"
#include "Misc.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*--------------------------------------------------------------------------*/
int Q931PopCallingPartyNo (const t_Byte *buf, t_CallingPartyNo *no)
/*
 * DESC
 *      Decodes the calling party number IE from a Q.931 message and convert
 *      it into the internal C structure.
 * RET
 *      # of bytes consumed
 */
{
        int                     len;
        int                     i;
        int                     j;

        Q931_ZERO (*no);
        len = buf[1] +2;
        i = 2;
/*** 3 ***/
        no->type = GET_BITS_765 (buf[i]) | 0x100;
        no->numberingPlan = GET_BITS_4321 (buf[i]) | 0x100;
/*** 3a ***/
        Q931_SKIP_BYTE (buf, i);
        if (!GET_BITS_8 (buf[i-1])) {
                no->presentation = GET_BITS_76 (buf[i]) | 0x100;
                no->screening = GET_BITS_21 (buf[i]) | 0x100;
                Q931_SKIP_GROUP (buf, i);
        }
/*** 4++ ***/
        j = 0;
        while (i < len)
                no->number[j++] = buf[i++];
        no->number[j++] = '\0';
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopCallingPartyNo */
/*--------------------------------------------------------------------------*/
int Q931PushCallingPartyNo (t_Byte *buf, const t_CallingPartyNo *no)
/*
 * DESC
 *      Converts the internal C calling party no into the Q.931 representation.
 * RET
 *      # bytes occupied
 */
{
        int                     len = 0;
        t_Byte                  byte;
        int                     i;

        buf[len++] = Q931_IE_CALLING_PARTY_NO;
        buf[len++] = 0;
/*** 3 ***/
        byte = 0x80;
        if (no->type)
                PUT_BITS_765 (byte, no->type);
        else
                PUT_BITS_765 (byte, CGPN_T_DEFAULT);
        if (no->numberingPlan)
                PUT_BITS_4321 (byte, no->numberingPlan);
        else
                PUT_BITS_4321 (byte, CGPN_NP_DEFAULT);
        buf[len++] = byte;
/*** 3a ***/
        if (no->presentation || no->screening) {
                CLR_BITS_8 (buf[len-1]);
                byte = 0x80;
                if (no->presentation)
                        PUT_BITS_76 (byte, no->presentation);
                else
                        PUT_BITS_76 (byte, CGPN_P_DEFAULT);
```

```
                if (no->screening)
                        PUT_BITS_21 (byte, no->screening);
                else
                        PUT_BITS_21 (byte, CGPN_S_DEFAULT);
                buf[len++] = byte;
        }
/*** 4++ ***/
        for (i=0; no->number[i] != '\0'; i++)
                buf[len++] = no->number[i] & 0x7f;
/*** set the real length ***/
        buf[1] = len -2;
        return len;
} /* Q931PushCallingPartyNo */
/*--------------------------------------------------------------------------*/
```

E.3.8 src/CallingPartySubAddr.h

```
/*
 *      file:           CallingPartySubAddr.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: CallingPartySubAddr.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: CallingPartySubAddr.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CALLING_PARTY_SUB_ADDR
#define INCL_CALLING_PARTY_SUB_ADDR

/*----- standard includefiles ----------------------------------------------*/

/*----- local includefiles -------------------------------------------------*/

/*----- defines ------------------------------------------------------------*/
#define T_CALLING_PARTY_SUB_ADDR_SIZE   sizeof (t_CallingPartySubAddr)

/*----- type definitions ---------------------------------------------------*/
typedef enum {
        CGPS_T_NSAP             = 0x100,
        CGPS_T_USER             = 0x102
} t_CallingPartySubType;

typedef enum {
        CGPS_P_EVEN             = 0x100,
        CGPS_P_ODD              = 0x101
} t_CallingPartySubParity;

typedef struct {
        t_CallingPartySubType   type;
        t_CallingPartySubParity evenOdd;
        int                     len;
        int                     subAddr[20];
} t_CallingPartySubAddr;

/*----- global functions ---------------------------------------------------*/

/*----- global variables ---------------------------------------------------*/

#endif /* INCL_CALLING_PARTY_SUB_ADDR */
```

E.3.9 src/Cause.h

```
/*
 *      file:           Cause.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Cause.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Cause.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CAUSE
#define INCL_CAUSE

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>

/*----- local includefiles -------------------------------------------------*/

/*----- defines ------------------------------------------------------------*/
#define T_CAUSE_SIZE            sizeof (t_Cause)

/*----- type definitions ---------------------------------------------------*/
        /*** NOTE: the values MUST be the values from Q.850 + 0x100! ***/
typedef enum {
```

```
        CA_L_USER               = 0x100,        /* user */
        CA_L_PRIV_NET_LOC_USER  = 0x101,        /* private net, local user */
        CA_L_PUB_NET_LOC_USER   = 0x102,        /* public net, local user */
        CA_L_TRANSIT_NETWORK    = 0x103,        /* transit net */
        CA_L_PUB_NET_REM_USER   = 0x104,        /* public net, remote user */
        CA_L_PRIV_NET_REM_USER  = 0x105,        /* provate net, remote user */
        CA_L_INTERNATIONAL_NET  = 0x107,        /* international net */
        CA_L_INTERNETWORKING    = 0x10a         /* net beyond inetnetw. point */
} t_CauseLocation;

typedef enum {
        CA_C_NORMAL             = 0x100,        /* normal */
        CA_C_NORMAL_1           = 0x101,        /* normal */
        CA_C_RESOURCE_UNAVAIL   = 0x102,        /* resourve unavailable */
        CA_C_SERV_OPT_NOT_AVAIL = 0x103,        /* service/option not available */
        CA_C_SERV_OPT_NOT_IMPL  = 0x104,        /* service/option not implemented */
        CA_C_INV_MSG            = 0x105,        /* invalid message */
        CA_C_PROTOCOL_ERROR     = 0x106,        /* protocol error */
        CA_C_INTERNETWORKING    = 0x107         /* internetw. error */
} t_CauseClass;

typedef enum {
        CA_V_NUMBER_UNASSIGNED  = 0x101,        /* the called number is not assigned */
        CA_V_NO_ROUTE_TO_NET    = 0x102,        /* no route to spec. transit network */
        CA_V_NO_ROUTE_TO_DEST   = 0x103,        /* no route to destination */
        CA_V_CAHNNEL_UNACCEPTABLE = 0x106,      /* the B channel is not acceptable */
        /* ... */
        CA_V_NORMAL_CLEARING    = 0x110,        /* normal call clearing */
        CA_V_USER_BUSY          = 0x111,        /* called user is busy */
        CA_V_NO_USER_RESPONDS   = 0x112,        /* no user is responding */
        CA_V_NO_ALERT_NO_ANSWER = 0x113,        /* ? */
        CA_V_CALL_REJECTED      = 0x115,        /* call rejected by user */
        CA_V_NUMBER_CHANGED     = 0x116,        /* number has been changes */
        /* ... */
        CA_V_INVALID_NUMBER_FMT = 0x11c,        /* invalid number format / incomplete */
        /* ... */
        CA_V_NORMAL_UNSPECIFIED = 0x11f,
        /* ... */
        CA_V_TIMER_EXPIRED      = 0x166,        /* recovery on timer expiry */
        /* ... */
        CA_V_INTERNETW_UNSPEC   = 0x127,
        CA_V_DEFAULT            = CA_V_NORMAL_CLEARING
} t_CauseValue;


typedef struct {
        t_CauseLocation         location;
        t_CauseClass            class;
        t_CauseValue            value;          /* see Q.850 */
        int                     diag;           /* = Q.850 + 0x100 */
} t_Cause;

/*----- global functions ----------------------------------------------------*/
extern  void    Q931DumpCause (FILE *f, const t_Cause *cause);
extern  int     Q931PopCause (const t_Byte *buf, t_Cause *cause);
extern  int     Q931PushCause (t_Byte *buf, const t_Cause *cause);

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_CAUSE */
```

E.3.10 src/Cause.c

```
/*
 *      file:           Cause.c
 *      written by:     Christian Zahl
 *      description:    Encodes and decodes the cause IE
 *
 * $Id: Cause.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Cause.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Cause.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <string.h>

/*----- user includefiles ---------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Cause.h"
#include "Misc.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/
```

```
/*----- global variables ----------------------------------------------------*/

/*--------------------------------------------------------------------------*/
void Q931DumpCause (FILE *f, const t_Cause *cause)
{
        if (!cause->value)
                return;
        fprintf (f, "x%02x: CAUSE: ", Q931_IE_CAUSE);
/*** location ***/
        fprintf (f, "LOC=");
        switch (cause->location) {
                case CA_L_USER:
                        fprintf (f, "USER"); break;
                case CA_L_PRIV_NET_LOC_USER:
                        fprintf (f, "PRIV_NET_SERV_LOC_USER"); break;
                case CA_L_PUB_NET_LOC_USER:
                        fprintf (f, "PUB_NET_SERV_LOC_USER"); break;
                case CA_L_TRANSIT_NETWORK:
                        fprintf (f, "TRANSIT_NET"); break;
                case CA_L_PUB_NET_REM_USER:
                        fprintf (f, "PUB_NET_SERV_REM_USER"); break;
                case CA_L_PRIV_NET_REM_USER:
                        fprintf (f, "PRIV_NET_SERV_REM_USER"); break;
                case CA_L_INTERNATIONAL_NET:
                        fprintf (f, "INTERNATIONAL_NET"); break;
                case CA_L_INTERNETWORKING:
                        fprintf (f, "NET_BEYOND_NET_POINT"); break;
                default:
                        fprintf (f, "x%02x", cause->location); break;
        }
/*** class ***/
        fprintf (f, " CLASS=");
        switch (cause->class) {
                case CA_C_NORMAL:
                        fprintf (f, "NORMAL"); break;
                case CA_C_NORMAL_1:
                        fprintf (f, "NORMAL"); break;
                case CA_C_RESOURCE_UNAVAIL:
                        fprintf (f, "RESOURCE_UNAVAIL"); break;
                case CA_C_SERV_OPT_NOT_AVAIL:
                        fprintf (f, "SERVICE_OPTION_NOT_AVAIL"); break;
                case CA_C_SERV_OPT_NOT_IMPL:
                        fprintf (f, "SERVICE_OPTION_NOT_IMPL"); break;
                case CA_C_INV_MSG:
                        fprintf (f, "INVALID_MSG"); break;
                case CA_C_PROTOCOL_ERROR:
                        fprintf (f, "PROTOCOL_ERROR"); break;
                case CA_C_INTERNETWORKING:
                        fprintf (f, "INTERNETWORKING"); break;
                default:
                        fprintf (f, "%d", cause->class &0xff); break;
        }
/*** value ***/
        fprintf (f, " VALUE=");
        switch (cause->value) {
                case CA_V_NUMBER_UNASSIGNED:
                        fprintf (f, "NUMBER_UNASS"); break;
                case CA_V_NO_ROUTE_TO_NET:
                        fprintf (f, "NO_ROUTE_TO_NET"); break;
                case CA_V_NO_ROUTE_TO_DEST:
                        fprintf (f, "NO_ROUTE_TO_DEST"); break;
                case CA_V_CAHNNEL_UNACCEPTABLE:
                        fprintf (f, "CHANNEL_UNACCEPT"); break;
                case CA_V_NORMAL_CLEARING:
                        fprintf (f, "NORMAL_CLEARING"); break;
                case CA_V_USER_BUSY:
                        fprintf (f, "USER_BUSY"); break;
                case CA_V_NO_USER_RESPONDS:
                        fprintf (f, "NO_USER_RESP"); break;
                case CA_V_CALL_REJECTED:
                        fprintf (f, "CALL_REJECTED"); break;
                case CA_V_INVALID_NUMBER_FMT:
                        fprintf (f, "INV_INCOMPL_NO"); break;
                case CA_V_NUMBER_CHANGED:
                        fprintf (f, "NUMBER_CHANGED"); break;
                case CA_V_TIMER_EXPIRED:
                        fprintf (f, "TIMER_EXPIRED"); break;
                default:
                        fprintf (f, "%d", cause->value & 0xff); break;
        }
        printf ("\n");
} /* Q931DumpCause */
/*--------------------------------------------------------------------------*/
int Q931PopCause (const t_Byte *buf, t_Cause *cause)
{
        int                     i;
        int                     len;

        Q931_ZERO (*cause);
        len = buf[1] +2;
        i = 2;
/*** 3 ***/
        if (GET_BITS_76 (buf[i]) != Q931_CODING_Q931)
                return len +2;
        cause->location = GET_BITS_4321 (buf[i]) | 0x100;
        Q931_SKIP_GROUP (buf, i);
/*** 4 ***/
        cause->value = GET_BITS_7654321 (buf[i]) | 0x100;
        cause->class = GET_BITS_765 (buf[i]) | 0x100;
```

```
        Q931_SKIP_GROUP (buf, i);
/*** 5 ***/
        if (i < len) {
                cause->diag = buf[i] | 0x100;
                Q931_SKIP_GROUP (buf, i);
        }
/*** # bytes consumed ***/
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopCause */
/*---------------------------------------------------------------------------*/
int Q931PushCause (t_Byte *buf, const t_Cause *cause)
{
        int                     len = 0;
        t_Byte                  byte;

        buf[len++] = Q931_IE_CAUSE;
        buf[len++] = 0;
/*** 3 ***/
        byte = 0x80;
        PUT_BITS_76 (byte, Q931_CODING_Q931);
        if (cause->location)
                PUT_BITS_4321 (byte, cause->location);
        else
                PUT_BITS_4321 (byte, CA_L_USER);
        buf[len++] = byte;
/*** 4 ***/
        byte = 0x80;
        if (cause->value)
                PUT_BITS_7654321 (byte, cause->value);
        else
                PUT_BITS_7654321 (byte, CA_V_NORMAL_UNSPECIFIED);
        buf[len++] = byte;
/*** 5 ***/
        /* ffs */
/*** set the real length of the IE ***/
        buf[1] = len -2;
        return len;
} /* Q931PushCause */
/*---------------------------------------------------------------------------*/
```

E.3.11 src/ChannelId.h

```
/*
 *      file:           ChannelId.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: ChannelId.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: ChannelId.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_CHANNEL_ID
#define INCL_CHANNEL_ID

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- local includefiles --------------------------------------------------*/
#include "Types.h"

/*----- defines -------------------------------------------------------------*/
#define T_CHANNEL_ID_SIZE       sizeof (t_ChannelId)

/*----- type definitions ----------------------------------------------------*/
typedef enum {
        CID_IP_NOT_PRESENT      = 0x100,
        CID_IP_PRESENT          = 0x101,
        CID_IP_DEFAULT          = CID_IP_NOT_PRESENT
} t_ChannelIdIntPresent;

typedef enum {
        CID_IT_BRI              = 0x100,
        CID_IT_OTHER            = 0x101,
        CID_IT_DEFAULT          = CID_IT_BRI
} t_ChannelIdIntType;

typedef enum {
        CID_PE_PREFERED         = 0x100,
        CID_PE_EXCLUSIVE        = 0x101,
        CID_PE_DEFAULT          = CID_PE_PREFERED
} t_ChannelIdPrefEx;

typedef enum {
        CID_D_NOD               = 0x100,
        CID_D_D                 = 0x101,
        CID_D_DEFAULT           = CID_D_NOD
} t_ChannelIdD;

typedef enum {
        CID_ICS_NONE            = 0x100,
        CID_ICS_B1              = 0x101,
```

```
        CID_ICS_B2              = 0x102,
        CID_ICS_ANY             = 0x103,
        CID_ICS_MAP             = 0x101,
        CID_ICS_DEFAULT         = CID_ICS_ANY
} t_ChannelIdInfoChanSel;

typedef enum {
        CID_ICT_B               = 0x200,
        CID_ICT_H0              = 0x201,
        CID_ICT_H11             = 0x202,
        CID_ICT_H12             = 0x203
} t_ChannelIdInfoChanType;

typedef union {
        struct {
                unsigned                c31:1;
                unsigned                c30:1;
                unsigned                c29:1;
                unsigned                c28:1;
                unsigned                c27:1;
                unsigned                c26:1;
                unsigned                c25:1;
                unsigned                c24:1;
                unsigned                c23:1;
                unsigned                c22:1;
                unsigned                c21:1;
                unsigned                c20:1;
                unsigned                c19:1;
                unsigned                c18:1;
                unsigned                c17:1;
                unsigned                c16:1;
                unsigned                c15:1;
                unsigned                c14:1;
                unsigned                c13:1;
                unsigned                c12:1;
                unsigned                c11:1;
                unsigned                c10:1;
                unsigned                c9:1;
                unsigned                c8:1;
                unsigned                c7:1;
                unsigned                c6:1;
                unsigned                c5:1;
                unsigned                c4:1;
                unsigned                c3:1;
                unsigned                c2:1;
                unsigned                c1:1;
                unsigned                c0:1;
        } BChannels;
        struct {
                unsigned                :3;
                unsigned                c5:1;
                unsigned                c4:1;
                unsigned                c3:1;
                unsigned                c2:1;
                unsigned                c1:1;
        } H0Channels;
        struct {
                unsigned                :7;
                unsigned                c1:1;
        } H11Channels;
        struct {
                unsigned                :7;
                unsigned                c1:1;
        } H12Channels;
} t_ChannelIdChannelMap;

typedef struct {
        t_ChannelIdIntPresent   interfaceIdPresent;
        t_Byte                  interfaceId[10];
        int                     interfaceIdLen;
        t_ChannelIdIntType      interfaceType;
        t_ChannelIdPrefEx       prefExcl;
        t_ChannelIdD            dChannelInd;
        t_ChannelIdInfoChanSel  infoChannelSel;
        t_ChannelIdInfoChanType infoChannelType;
        t_ChannelIdChannelMap   infoChannelMap;
} t_ChannelId;

/*----- global functions -------------------------------------------------------*/
extern  void    Q931DumpChannelId (FILE *f, const t_ChannelId *id);
extern  int     Q931PopChannelId (const t_Byte *buf, t_ChannelId *id);
extern  int     Q931PushChannelId (t_Byte *buf, const t_ChannelId *id);

/*----- global variables -------------------------------------------------------*/

#endif /* INCL_CHANNEL_ID */
```

E.3.12 src/ChannelId.c

```
/*
 *      file:           header.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: ChannelId.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
```

```
 * $Log: ChannelId.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: ChannelId.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "ChannelId.h"
#include "Misc.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*-------------------------------------------------------------------------*/
void Q931DumpChannelId (FILE *f, const t_ChannelId *id)
{
        fprintf (f, "x%02X: CHANNEL-ID: ", Q931_IE_CHANNEL_ID);
        fprintf (f, "\a missing!!!");
/*
        ...
*/
        fprintf (f, "\n");
} /* Q931DumpChannelId */
/*-------------------------------------------------------------------------*/
int Q931PopChannelId (const t_Byte *buf, t_ChannelId *id)
/*
 * DESC
 *      Decodes the Q.931 channel identification IE.
 * NOTE:
 *      Currently only the BRI will be handled correctly.
 */
{
        int                     len;
        int                     i;
        int                     n;

        Q931_ZERO (*id);
        len = buf[1] +2;
        i = 2;
/*** 3 ***/
        id->interfaceIdPresent = GET_BITS_7 (buf[i]) | 0x100;
        id->interfaceType = GET_BITS_6 (buf[i]) | 0x100;
        id->prefExcl = GET_BITS_4 (buf[i]) | 0x100;
        id->dChannelInd = GET_BITS_3 (buf[i]) | 0x100;
        id->infoChannelSel = GET_BITS_21 (buf[i]) | 0x100;
        Q931_SKIP_GROUP (buf, i);
/*** 3.1 ***/
        if (i < len && id->interfaceIdPresent == CID_IP_PRESENT) {
                for (n=0; !(buf[i] & 0x80); n++)
                        id->interfaceId[n] = GET_BITS_7654321 (buf[i++]);
                id->interfaceId[n] = GET_BITS_7654321 (buf[i++]);
                id->interfaceIdLen = n;
        }
/*** 3.2 ***/
        if (i < len && id->interfaceType != CID_IT_BRI) {
                if (GET_BITS_76 (buf[i]) == Q931_CODING_Q931) {
                        /* ffs */
                }
                Q931_SKIP_GROUP (buf, i);
        }
/*** 3.3 ***/
        if (i < len && id->interfaceType != CID_IT_BRI) {
                /* ffs */
        }
/*** # of consumed bytes ***/
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopChannelId */
/*-------------------------------------------------------------------------*/
int Q931PushChannelId (t_Byte *buf, const t_ChannelId *id)
/*
 * DESC
 *      Converts the internal C-struct into the Q.931 format
 */
{
        int                     len = 0;
        t_Byte                  byte;
        int                     i;

        buf[len++] = Q931_IE_CHANNEL_ID;
        buf[len++] = 0;
/*** 3 ***/
        byte = 0x80;
```

```
        if (id->interfaceIdPresent)
                PUT_BITS_7 (byte, id->interfaceIdPresent);
        else
                PUT_BITS_7 (byte, CID_IP_DEFAULT);
        if (id->interfaceType)
                PUT_BITS_6 (byte, id->interfaceType);
        else
                PUT_BITS_6 (byte, CID_IT_DEFAULT);
        if (id->prefExcl)
                PUT_BITS_4 (byte, id->prefExcl);
        else
                PUT_BITS_4 (byte, CID_PE_DEFAULT);
        if (id->dChannelInd)
                PUT_BITS_3 (byte, id->dChannelInd);
        else
                PUT_BITS_3 (byte, CID_D_DEFAULT);
        if (id->infoChannelSel)
                PUT_BITS_21 (byte, id->infoChannelSel);
        else
                PUT_BITS_21 (byte, CID_ICS_DEFAULT);
        buf[len++] = byte;
/*** 3.1 ***/
        if (id->interfaceIdPresent == CID_IP_PRESENT) {
                for (i=0; i<id->interfaceIdLen; i++)
                        buf[len++] = GET_BITS_7654321 (id->interfaceId[i]);
                buf[len-1] |= 0x80;
        }
/*** 3.2 ***/
        if (id->interfaceType == CID_IT_OTHER) {
                /* ffs */
        }
/*** 3.3 ***/
        if (id->interfaceType == CID_IT_OTHER) {
                /* ffs */
        }
/*** set the real length ***/
        buf[1] = len -2;
        return len;
} /* Q931PushChannelId */
/*---------------------------------------------------------------------------*/
```

E.3.13 src/HighLayerComp.h

```
/*
 *      file:           HighLayerComp.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: HighLayerComp.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: HighLayerComp.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_HIGH_LAYER_COMP
#define INCL_HIGH_LAYER_COMP

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"

/*----- defines -----------------------------------------------------------*/
#define T_HIGH_LAYER_COMP_SIZE  sizeof (t_HighLayerComp)

/*----- type definitions --------------------------------------------------*/
typedef struct {
        /* ffs */
        int                     dummy;
} t_HighLayerComp;

/*----- global functions --------------------------------------------------*/
extern  int     Q931PopHighLayerComp (const t_Byte *buf, t_HighLayerComp *hlc);
extern  int     Q931PushHighLayerComp (t_Byte *buf, const t_HighLayerComp *hlc);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_HIGH_LAYER_COMP */
```

E.3.14 src/HighLayerComp.c

```
/*
 *      file:           HighLayerComp.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: HighLayerComp.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: HighLayerComp.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
```

```
 *
 */
static  char                    rcsid[] = "$Id: HighLayerComp.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>

/*----- user includefiles -------------------------------------------------*/
#include "Q931.h"
#include "HighLayerComp.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*-------------------------------------------------------------------------*/
int Q931PopHighLayerComp (const t_Byte *buf, t_HighLayerComp *hlc)
{
        int                     i;
        int                     len;

        len = buf[1];
        i = 2;
/*** 3 ***/
        Q931_FFS ();
/*** return # consumed bytes ***/
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopHighLayerComp */
/*-------------------------------------------------------------------------*/
int Q931PushHighLayerComp (t_Byte *buf, const t_HighLayerComp *hlc)
{
        Q931_FFS ();
        return 0;
} /* Q931PushHighLayerComp */
/*-------------------------------------------------------------------------*/
```

E.3.15 src/Keypad.h

```
/*
 *      file:           Keypad.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Keypad.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Keypad.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_KEYPAD
#define INCL_KEYPAD

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"

/*----- defines -----------------------------------------------------------*/
#define T_KEYPAD_SIZE           sizeof (t_Keypad)

/*----- type definitions --------------------------------------------------*/
typedef struct {
        unsigned char           keypad[32+1];
} t_Keypad;

/*----- global functions --------------------------------------------------*/
extern  int                     Q931PushKeypad (t_Byte *buf, const t_Keypad *keypad);
extern  int                     Q931PopKeypad (const t_Byte *buf, t_Keypad *keypad);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_KEYPAD */
```

E.3.16 src/Keypad.c

```
/*
 *      file:           Keypad.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Keypad.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
```

```
 * $Log: Keypad.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Keypad.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/

/*----- user includefiles -------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Keypad.h"
#include "Misc.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/

/*----- global variables --------------------------------------------------*/

/*-------------------------------------------------------------------------*/
int Q931PushKeypad (t_Byte *buf, const t_Keypad *keypad)
/*
 * DESC
 *      Creates a Q.931 keypad facility IE from the C structure
 * RET
 *      # of bytes occupied
 */
{
        int                     len = 0;
        int                     i;

/*** IE Type ***/
        buf[len++] = Q931_IE_KEYPAD_FACILITY;
/*** flush the len byte ***/
        buf[len++] = 0;
/*** contents ***/
        for (i=0; keypad->keypad[i] != '\0'; i++)
                buf[len++] = keypad->keypad[i] & 0x7f;
/*** set the len byte ***/
        buf[1] = len -2;
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PushKeypad */
/*-------------------------------------------------------------------------*/
int Q931PopKeypad (const t_Byte *buf, t_Keypad *keypad)
/*
 * DESC
 *      Decodes a Q.931 and converts it to the internal C structure.
 * RET
 *      # bytes consumed.
 */
{
        int                     len;
        int                     i;

        len = buf[1];
        for (i=0; i<len; i++)
                keypad->keypad[i] = buf[i] & 0x7f;
        keypad->keypad[i] = '\0';
        return len +2;
} /* Q931PopKeypad */
/*-------------------------------------------------------------------------*/
```

E.3.17 src/LowLayerComp.h

```
/*
 *      file:           LowLayerComp.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: LowLayerComp.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: LowLayerComp.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_LOW_LAYER_COMP
#define INCL_LOW_LAYER_COMP

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"

/*----- defines -----------------------------------------------------------*/
#define T_LOW_LAYER_COMP_SIZE   sizeof (t_LowLayerComp)
```

```
/*----- type definitions ----------------------------------------------------*/
typedef struct {
                /* FFS */
        int                     dummy;
} t_LowLayerComp;

/*----- global functions ----------------------------------------------------*/
extern  int     Q931PopLowLayerComp (const t_Byte *buf, t_LowLayerComp *llc);
extern  int     Q931PushLowLayerComp (t_Byte *buf, const t_LowLayerComp *llc);

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_LOW_LAYER_COMP */
```

E.3.18 src/LowLayerComp.c

```
/*
 *      file:           LowLayerComp.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: LowLayerComp.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: LowLayerComp.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: LowLayerComp.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- user includefiles ---------------------------------------------------*/
#include "Q931.h"
#include "LowLayerComp.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
int Q931PopLowLayerComp (const t_Byte *buf, t_LowLayerComp *llc)
{
        int                     i;
        int                     len;

        len = buf[1];
        i = 2;
/*** 3 ***/
        Q931_FFS ();
/*** return # of bytes consumed ***/
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopLowLayerComp */
/*---------------------------------------------------------------------------*/
int Q931PushLowLayerComp (t_Byte *buf, const t_LowLayerComp *llc)
{
        Q931_FFS ();
        return 0;
} /* Q931PushLowLayerComp */
/*---------------------------------------------------------------------------*/
```

E.3.19 src/NotifyInd.h

```
/*
 *      file:           NotifyInd.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: NotifyInd.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: NotifyInd.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_NOTIFY_IND
#define INCL_NOTIFY_IND

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/
```

```
/*----- defines -------------------------------------------------------------*/
#define T_NOTIFY_IND_SIZE       sizeof (t_NotifyInd)

/*----- type definitions ----------------------------------------------------*/
typedef enum {
        NI_D_USER_SUSPENDED     = 0x100,
        NI_D_USER_RESUMED       = 0x101,
        NT_D_BEARER_CHANGED     = 0x102
        /* ... Q.95x */
} t_NotifyIndDescr;

typedef struct {
        t_NotifyIndDescr        descr;
} t_NotifyInd;

/*----- global functions ----------------------------------------------------*/
extern  int     Q931PopNotifyInd (const t_Byte *buf, t_NotifyInd *ind);
extern  int     Q931PushNotifyInd (t_Byte *buf, const t_NotifyInd *ind);

/*----- global variables ----------------------------------------------------*/

#endif /* INCL_NOTIFY_IND */
```

E.3.20 src/NotifyInd.c

```
/*
 *      file:           NotifyInd.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: NotifyInd.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: NotifyInd.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: NotifyInd.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- user includefiles ---------------------------------------------------*/
#include "Q931.h"
#include "NotifyInd.h"

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
int Q931PopNotifyInd (const t_Byte *buf, t_NotifyInd *ind)
{
        int                     i;
        int                     len;

        len = buf[1];
        i = 2;
/*** 3 ***/
        Q931_FFS ();
/*** return # consumed bytes ***/
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopNotifyInd */
/*---------------------------------------------------------------------------*/
int Q931PushNotifyInd (t_Byte *buf, const t_NotifyInd *ind)
{
        Q931_FFS ();
        return 0;
} /* Q931PushNotifyInd */
/*---------------------------------------------------------------------------*/
```

E.3.21 src/ProgressInd.h

```
/*
 *      file:           ProgressInd.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: ProgressInd.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: ProgressInd.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
```

```
 *
 */

#ifndef INCL_PROGRESS_IND
#define INCL_PROGRESS_IND

/*----- standard includefiles ------------------------------------------------*/
#include <stdio.h>

/*----- local includefiles ---------------------------------------------------*/

/*----- defines --------------------------------------------------------------*/
#define T_PROGRESS_IND_SIZE     sizeof (t_ProgressInd)

/*----- type definitions -----------------------------------------------------*/
typedef enum {
        PR_L_USER               = 0x100,
        PR_L_PRIV_NET_LOC_USER  = 0x101,
        PR_L_PUB_NET_LOC_USER   = 0x102,
        PR_L_TRANSIT_NETWORK    = 0x103,
        PR_L_PUB_NET_REM_USER   = 0x104,
        PR_L_PRIV_NET_REM_USER  = 0x105,
        PR_L_INTERNETWORKING    = 0x10a
} t_ProgressIndLocation;

typedef enum {
        PR_D_NO_ISDN_ISDN       = 0x101,
        PR_D_DEST_NOT_ISDN      = 0x102,
        PR_D_ORIG_NOT_ISDN      = 0x103,
        PR_D_RETURNED           = 0x104,
        PR_D_INTERNETWORKING    = 0x105,
        PR_D_IN_BAND            = 0x108
} t_ProgressIndDescr;

typedef struct {
        t_ProgressIndLocation   location;
        t_ProgressIndDescr      descr;
} t_ProgressInd;

/*----- global functions -----------------------------------------------------*/
extern  void    Q931DumpProgressInd (FILE *f, const t_ProgressInd *ind);
extern  int     Q931PopProgressInd (const t_Byte *buf, t_ProgressInd *ind);
extern  int     Q931PushProgressInd (t_Byte *buf, const t_ProgressInd *ind);

/*----- global variables -----------------------------------------------------*/

#endif /* INCL_PROGRESS_IND */
```

E.3.22 src/ProgressInd.c

```
/*
 *      file:           Progress.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: ProgressInd.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: ProgressInd.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: ProgressInd.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ------------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles ----------------------------------------------------*/
#include "Q931.h"
#include "ProgressInd.h"
#include "Misc.h"

/*----- defines --------------------------------------------------------------*/

/*----- type definitions -----------------------------------------------------*/

/*----- local functions ------------------------------------------------------*/

/*----- global functions -----------------------------------------------------*/

/*----- local variables ------------------------------------------------------*/

/*----- global variables -----------------------------------------------------*/

/*--------------------------------------------------------------------------*/
void Q931DumpProgressInd (FILE *f, const t_ProgressInd *ind)
/*
 * DESC:
 *      Simply dumps the contents of the Progress Indicator IE in a human
 *      readable form.
 */
{
        fprintf (f, "x%02X: PROGRESS-IND: ", Q931_IE_PROGRESS_INDICATOR);
/*** location ***/
        fprintf (f, "LOC=");
```

```
        switch (ind->location) {
                case PR_L_USER:
                        fprintf (f, "USER"); break;
                case PR_L_PRIV_NET_LOC_USER:
                        fprintf (f, "PRIV_NET_SERV_LOC_USER"); break;
                case PR_L_PUB_NET_LOC_USER:
                        fprintf (f, "PUB_NET_SERV_LOC_USER"); break;
                case PR_L_TRANSIT_NETWORK:
                        fprintf (f, "TRANSIT_NET"); break;
                case PR_L_PUB_NET_REM_USER:
                        fprintf (f, "PUB_NET_SERV_REM_USER"); break;
                case PR_L_PRIV_NET_REM_USER:
                        fprintf (f, "PRIV_NET_SERV_REM_USER"); break;
                case PR_L_INTERNETWORKING:
                        fprintf (f, "NET_BEYOND_NET_POINT"); break;
                default:
                        fprintf (f, "UNKNOWN (%d)", ind->location); break;
        }
/*** description ***/
        fprintf (f, " DESC=");
        switch (ind->descr) {
                case PR_D_NO_ISDN_ISDN:
                        fprintf (f, "NO_ISDN_TO_ISDN"); break;
                case PR_D_DEST_NOT_ISDN:
                        fprintf (f, "DEST_NOT_ISDN"); break;
                case PR_D_ORIG_NOT_ISDN:
                        fprintf (f, "ORIG_NOT_ISDN"); break;
                case PR_D_RETURNED:
                        fprintf (f, "RETURNED_TO_ISDN"); break;
                case PR_D_INTERNETWORKING:
                        fprintf (f, "INTERNETWORKING"); break;
                case PR_D_IN_BAND:
                        fprintf (f, "IN_BAND_INFO_AVAIL"); break;
                default:
                        fprintf (f, "UNKNOWN (%d)", ind->descr); break;
        }
        fprintf (f, "\n");
} /* Q931DumpProgressInd */
/*------------------------------------------------------------------------*/
int Q931PopProgressInd (const t_Byte *buf, t_ProgressInd *ind)
{
        int                     i;
        int                     len;

        Q931_ZERO (*ind);
        len = buf[1];
        i = 2;
/*** 3 ***/
        if (GET_BITS_76 (buf[i]) != Q931_CODING_Q931)
                return len;
        ind->location = GET_BITS_4321 (buf[i]) | 0x100;
        Q931_SKIP_GROUP (buf, i);
/*** 4 ***/
        ind->descr = GET_BITS_7654321 (buf[i]) | 0x100;
/*** return # consumed bytes ***/
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PopProgressInd */
/*------------------------------------------------------------------------*/
int Q931PushProgressInd (t_Byte *buf, const t_ProgressInd *ind)
{
        Q931_FFS ();
        return 0;
} /* Q931PushProgressInd */
/*------------------------------------------------------------------------*/
```

E.4 Sonstige Q.931 Handling Routinen

In den folgenden C-Files sind sonstige Funktionen implementiert, die für die Funktion der Q.931 Implementierung von Bedeutung sind.

E.4.1 src/CallRef.h

```
/*
 *      file:           CallRef.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: CallRef.h,v 1.2 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: CallRef.h,v $
 * Revision 1.2  1996/09/30  19:34:31  czahl
 * *** empty log message ***
 *
 * Revision 1.1  1996/06/18  14:35:32  czahl
 * Initial revision
 *
 */
```

```
#ifndef INCL_CALL_REF
#define INCL_CALL_REF

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/
#include "Types.h"

/*----- defines -----------------------------------------------------------*/
#define T_CALL_REF_SIZE         sizeof (t_CallRef)
#define CALL_REF_ORIGIN         0
#define CALL_REF_DEST           1

/*----- type definitions --------------------------------------------------*/
typedef struct {
        int                     len;
        int                     flag;
        t_Byte                  value[3];
} t_CallRef;

/*----- global functions --------------------------------------------------*/
extern  int                     Q931PushCallRef (t_Byte *buf, const t_CallRef *callRef);
extern  int                     Q931PopCallRef (const t_Byte *buf, t_CallRef *callRef);
extern  int                     Q931AllocCallRef (t_CallRef *callRef);
extern  int                     Q931FreeCallRef (const t_CallRef *callRef);
extern  int                     Q931CompCallRef (const t_CallRef *callRef1,
                                const t_CallRef *callRef2);
extern  void                    Q931CopyCallRef (t_CallRef *dst, const t_CallRef *src);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_CALL_REF */
```

E.4.2 src/CallRef.c

```
/*
 *      file:           CallRef.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: CallRef.c,v 1.2 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: CallRef.c,v $
 * Revision 1.2  1996/09/30  19:34:13  czahl
 * *** empty log message ***
 *
 * Revision 1.1  1996/06/18  14:35:27  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: CallRef.c,v 1.2 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/

/*----- user includefiles -------------------------------------------------*/
#include "Types.h"
#include "CallRef.h"
#include "Misc.h"

/*----- defines -----------------------------------------------------------*/
#define CALL_REF_POOL_SIZE      127

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/
static  char                    callRefPool[127];
static  int                     lastCallRef = 0;

/*----- global variables --------------------------------------------------*/

/*-------------------------------------------------------------------------*/
int Q931PopCallRef (const t_Byte *buf, t_CallRef *callRef)
/*
 * DESC
 *      Reads the call reference out of the message in buf.
 *      Also toggles the call reference flag.
 * RET
 *      -1      value longer than 3
 *      or      # of bytes consumed
 */
{
        int                     len;
        int                     i;

        len = *buf++;
        if (len > 3)
                return -1;
        if (len == 0) {                                         /* dummy call ref */
                callRef->flag = 0;
                callRef->len = 0;
                callRef->value[0] = 0;
        } else {                                                /* normal call ref */
```

```
                callRef->flag = (*buf >> 7) & 1;
                callRef->len = len;
                for (i=0; i<len; i++)
                        if (i == 0)
                                callRef->value[i] = *buf++ & 0x7f;
                        else
                                callRef->value[i] = *buf++;
/*** toggle flag in incomming messages! ***/
                if (callRef->flag == CALL_REF_ORIGIN)
                        callRef->flag = CALL_REF_DEST;
                else
                        callRef->flag = CALL_REF_ORIGIN;
        }
        return len +1;
} /* Q931PopCallRef */
/*------------------------------------------------------------------------*/
int Q931PushCallRef (t_Byte *buf, const t_CallRef *callRef)
{
        int                     len = 0;
        int                     i;
        t_Byte                  byte;

/*** push the length ***/
        byte = 0;
        PUT_BITS_4321 (byte, callRef->len);
        buf[len++] = byte;
/*** push the reference ***/
        for (i=0; i<callRef->len; i++) {
                byte = 0;
                if (i == 0) {
                        PUT_BITS_8 (byte, callRef->flag);
                        PUT_BITS_7654321 (byte, callRef->value[i]);
                } else {
                        PUT_BITS_87654321 (byte, callRef->value[i]);
                }
                buf[len++] = byte;
        }
        return len;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Q931PushCallRef */
/*------------------------------------------------------------------------*/
int Q931AllocCallRef (t_CallRef *callRef)
/*
 * DESC
 *      Allocates a new call reference value out of the pool. The allocated
 *      value has to be freed after usage.
 *      The call reference will be marked as "ORIGIN". Also the value 0
 *      (the dummy call ref value) will never be allocated.
 * RET
 *      -1      no more call reference values available
 *      0       callRef allocated
 */
{
        int                     i;

        i = lastCallRef +1;
        while (i != lastCallRef) {
                if (callRefPool[i] == 0)
                        break;
                i++;
                if (i >= CALL_REF_POOL_SIZE)
                        i = 1;
        }
        if (i == lastCallRef)
                return -1;
        callRefPool[i] = 1;
        callRef->flag = CALL_REF_ORIGIN;
        callRef->len = 1;
        callRef->value[0] = i;
        lastCallRef++;
        if (lastCallRef >= CALL_REF_POOL_SIZE)
                lastCallRef = 1;
        return 0;
} /* Q931AllocCallRef */
/*------------------------------------------------------------------------*/
int Q931FreeCallRef (const t_CallRef *callRef)
/*
 * DESC
 *      Frees a prior allocated call reference value. If its flag is not
 *      ORIGIN, then the value was allocated by the network and will not
 *      be released (because it wasn't allocated).
 * RET
 *      0       allways
 */
{
        if (callRef->flag == CALL_REF_DEST)
                return 0;
        if (callRef->len != 1)
                return 0;
        if (callRef->value[0] > 127)
                return 0;
        callRefPool[callRef->value[0]] = 0;
        return 0;
} /* Q931FreeCallRef */
/*------------------------------------------------------------------------*/
int Q931CompCallRef (const t_CallRef *callRef1, const t_CallRef *callRef2)
/*
 * DESC
 *      Compares two call references.
```

```
 * RET
 *      0       both are equal
 *      != 0    not equal
 */
{
        int                     i;

        if (callRef1->flag != callRef2->flag)
                return 1;
        if (callRef1->len != callRef2->len)
                return 1;
        for (i=0; i<callRef1->len; i++)
                if (callRef1->value[i] != callRef2->value[i])
                        return 1;
        return 0;
} /* Q931CompCallRef */
/*---------------------------------------------------------------------------*/
void Q931CopyCallRef (t_CallRef *dst, const t_CallRef *src)
{
        memcpy (dst, src, sizeof (t_CallRef));
} /* Q931CopyCallRef */
/*---------------------------------------------------------------------------*/
```

E.4.3 src/Q931.h

```
/*
 *      file:           Q931.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931.h,v 1.2 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931.h,v $
 * Revision 1.2  1996/09/30  19:34:31  czahl
 * *** empty log message ***
 *
 * Revision 1.1  1996/06/18  16:30:38  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931
#define INCL_Q931

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/
#include "Types.h"
#include "BearerCap.h"
#include "CallRef.h"
#include "CalledPartyNo.h"
#include "CalledPartySubAddr.h"
#include "CallingPartyNo.h"
#include "CallingPartySubAddr.h"
#include "Cause.h"
#include "ChannelId.h"
#include "HighLayerComp.h"
#include "Keypad.h"
#include "NotifyInd.h"
#include "LowLayerComp.h"
#include "ProgressInd.h"

/*----- defines -------------------------------------------------------------*/
        /* List of Q.931 message types. */
#define Q931_MT_ALERTING                0x01
#define Q931_MT_CALL_PROCEEDING         0x02
#define Q931_MT_CONGESTION_CONTROL      0x79
#define Q931_MT_CONNECT                 0x07
#define Q931_MT_CONNECT_ACK             0x0f
#define Q931_MT_DISCONNECT              0x45
#define Q931_MT_INFORMATION             0x7b
#define Q931_MT_NOTIFY                  0x6e
#define Q931_MT_PROGRESS                0x03
#define Q931_MT_RELEASE                 0x4d
#define Q931_MT_RELEASE_COMPLETE        0x5A
#define Q931_MT_RESTART                 0x46
#define Q931_MT_RESTART_ACK             0x4e
#define Q931_MT_RESUME                  0x26
#define Q931_MT_RESUME_ACK              0x2e
#define Q931_MT_RESUME_REJECT           0x22
#define Q931_MT_SEGMENT                 0x60
#define Q931_MT_SETUP                   0x05
#define Q931_MT_SETUP_ACK               0x0d
#define Q931_MT_STATUS                  0x7d
#define Q931_MT_STATUS_ENQUIRY          0x75
#define Q931_MT_SUSPEND                 0x25
#define Q931_MT_SUSPEND_ACK             0x2d
#define Q931_MT_SUSPEND_REJECT          0x21
#define Q931_MT_USER_INFO               0x20

        /*** Q.931 Information elements ***/
#define Q931_IE_BEARER_CAP              0x04
#define Q931_IE_CALLED_PARTY_NO         0x70
#define Q931_IE_CALLED_PARTY_SUB_ADDR   0x71
#define Q931_IE_CALLING_PARTY_NO        0x6c
#define Q931_IE_CALLING_PARTY_SUB_ADDR  0x6d
```

```
#define Q931_IE_CAUSE                   0x08
#define Q931_IE_CHANNEL_ID              0x18
#define Q931_IE_HIGHER_LAYER_CAP        0x7d
#define Q931_IE_KEYPAD_FACILITY         0x2c
#define Q931_IE_PROGRESS_INDICATOR      0x1e
#define Q931_IE_SENDING_COMPLETE        0xa1

        /*** Q.931 user states, according to clause Q.931 2.1.1 ***/
#define Q931_US_NULL                    0
#define Q931_US_CALL_INITIATED          1
#define Q931_US_OVERLAP_SENDING         2
#define Q931_US_OUT_CALL_PROCEEDING     3
#define Q931_US_CALL_DELIVERED          4
#define Q931_US_CALL_PRESENT            6
#define Q931_US_CALL_RECEIVED           7
#define Q931_US_CONNECT_REQUEST         8
#define Q931_US_IN_CALL_PROCEEDING      9
#define Q931_US_ACTIVE                  10
#define Q931_US_DISCONNECT_REQUEST      11
#define Q931_US_DISCONNECT_INDICATION   12
#define Q931_US_SUSPEND_REQUEST         15
#define Q931_US_RESUME_REQUEST          17
#define Q931_US_RELEASE_REQUEST         19
#define Q931_US_OVERLAP_RECEIVING       25

        /*** misc ***/
#define Q931_DISCRIMINATOR              0x08                    /* protocol discriminator (4.2) */
#define Q931_BC_MULTIRATE               0x18
#define Q931_CODING_Q931                0x00                    /* coding standard = Q.931 */

#define Q931_FFS()                      fprintf (stderr, "%s[%d]: \007FFS\n", __FILE__, __LINE__)
#define Q931_ZERO(buf)                  memset (&buf, '\0', sizeof (buf));
#define Q931_SKIP_GROUP(buf,iii)        {while (!(buf[iii] & 0x80)) iii++; iii++;}
#define Q931_SKIP_BYTE(buf,iii)         {iii++;}

/*----- type definitions ----------------------------------------------------*/
typedef struct _t_Q931Cxt {
        struct _t_Q931Cxt       *next;
        t_CallRef               callRef;
        int                     callId;
        int                     state;
        int                     timer[23+1];
        t_ChannelId             channelId;              /* describes the selected B-Channel */
        int                     fd;                     /* fd of B-Channel, -1 if closed */
} t_Q931Cxt;

/*----- global functions ----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/


#endif /* INCL_Q931 */
```

E.4.4 src/Q931Msg.h

```
/*
 *      file:           Q931Msg.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Msg.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Msg.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_MSG
#define INCL_Q931_MSG

/*----- standard includefiles -----------------------------------------------*/

/*----- local includefiles --------------------------------------------------*/

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/
typedef struct {
        int                     msgType;
        t_CallRef               callRef;
        t_BearerCap             bearerCap[2];
        t_CalledPartyNo         calledPartyNo;
        t_CalledPartySubAddr    calledPartySubAddr;
        t_CallingPartyNo        callingPartyNo;
        t_CallingPartySubAddr   callingPartySubAddr;
        t_Cause                 cause;
        t_ChannelId             channelId;
        t_Keypad                keypad;
        t_ProgressInd           progressInd[2];
} t_Q931Msg;

/*----- global functions ----------------------------------------------------*/
extern  void    Q931RecvMsg (void *dummy, int mask);
extern  int     Q931SendMsg (t_Byte *buf, int len);
```

```
/*----- global variables -------------------------------------------------*/

#endif /* INCL_Q931_MSG */
```

E.4.5 src/Q931Msg.c

```
/*
 *      file:           Q931Msg.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Msg.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Msg.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Msg.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles --------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles ------------------------------------------------*/
#include "Q931.h"
#include "Q931Main.h"
#include "Dlpi.h"
#include "Q931Msg.h"
#include "Q931Alerting.h"
#include "Q931CallProceeding.h"
#include "Q931Connect.h"
#include "Q931ConnectAck.h"
#include "Q931Disconnect.h"
#include "Q931Information.h"
#include "Q931Notify.h"
#include "Q931Progress.h"
#include "Q931Release.h"
#include "Q931Setup.h"
#include "Q931SetupAck.h"
#include "Debug.h"

/*----- defines ----------------------------------------------------------*/

/*----- type definitions -------------------------------------------------*/

/*----- local functions --------------------------------------------------*/

/*----- global functions -------------------------------------------------*/

/*----- local variables --------------------------------------------------*/

/*----- global variables -------------------------------------------------*/

/*------------------------------------------------------------------------*/
static void HandleQ931Msg (t_Byte *buf, int len)
/*
 * DESC
 *      Decodes an incomming message from the Q.921 DPLI interface and
 *      schedules it to the appriate handling function
 */
{
        int                     i = 0;
        int                     disc;
        t_Q931Msg               msg;
        t_Q931Cxt               *cxt;

        TRACE_ENTER (("HandleQ931Msg()"));
/*      Q931Dump (buf, len); */
        Q931_ZERO (msg);
/*** check if it is a Q.931 message ***/
        if ((disc = buf[i++]) != Q931_DISCRIMINATOR)
                return;
        i += Q931PopCallRef (buf+i, &msg.callRef);
        msg.msgType = buf[i++];
/*** get all the information elements ***/
        while (i < len) {
                switch (buf[i]) {
                        case Q931_IE_BEARER_CAP:
                                i += Q931PopBearerCap (buf+i, &msg.bearerCap[0]);
                                break;
                        case Q931_IE_CALLED_PARTY_NO:
                                i += Q931PopCalledPartyNo (buf+i, &msg.calledPartyNo);
                                break;
                        case Q931_IE_CALLING_PARTY_NO:
                                i += Q931PopCallingPartyNo (buf+i, &msg.callingPartyNo);
                                break;
                        case Q931_IE_CAUSE:
                                i += Q931PopCause (buf+i, &msg.cause);
                                break;
                        case Q931_IE_CHANNEL_ID:
                                i += Q931PopChannelId (buf+i, &msg.channelId);
                                break;
                        case Q931_IE_PROGRESS_INDICATOR:
                                i += Q931PopProgressInd (buf+i, &msg.progressInd[0]);
                                break;
```

```
                        default:
                                if (buf[i] & 0x80)
                                        i++;
                                else
                                        i += buf[i+1] +2;
                } /* switch */
        } /* while */
/*** post check the length ***/
        if (i > len) {
                fprintf (stderr, "HandleMsg: truncated!\n");
                return;
        }
/*** get the context ***/
        cxt = Q931SearchCallRef (&msg.callRef);
        if (!cxt)
                cxt = Q931AllocCxt (&msg.callRef);
/*** handle the message ***/
        switch (msg.msgType) {
                case Q931_MT_ALERTING:
                        Q931AlertingInd (cxt, &msg); break;
                case Q931_MT_CALL_PROCEEDING:
                        Q931CallProceedingInd (cxt, &msg); break;
                case Q931_MT_CONNECT:
                        Q931ConnectInd (cxt, &msg); break;
                case Q931_MT_CONNECT_ACK:
                        Q931ConnectAckInd (cxt, &msg); break;
                case Q931_MT_DISCONNECT:
                        Q931DisconnectInd (cxt, &msg); break;
                case Q931_MT_INFORMATION:
                        Q931InformationInd (cxt, &msg); break;
                case Q931_MT_NOTIFY:
                        Q931NotifyInd (cxt, &msg); break;
                case Q931_MT_PROGRESS:
                        Q931ProgressInd (cxt, &msg); break;
                case Q931_MT_RELEASE:
                        Q931ReleaseInd (cxt, &msg); break;
                case Q931_MT_RELEASE_COMPLETE:
                        Q931ReleaseCompleteInd (cxt, &msg); break;
                case Q931_MT_SETUP:
                        Q931SetupInd (cxt, &msg); break;
                case Q931_MT_SETUP_ACK:
                        Q931SetupAckInd (cxt, &msg); break;
                default:
                        fprintf (stderr, "HandleMsg: unknown Message Type (x%02X)\n", msg.msgType );
                        break;
        } /* switch */
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* HandleQ931Msg */
/*------------------------------------------------------------------------*/
void Q931RecvMsg (void *dummy, int mask)
/*
 * DESC
 *      This is the installed callback function for the D-channel (DLPI).
 *      It receives the message and handles it.
 * RET
 *      -
 */
{
        t_Q931DlpiMsg           msg;

        TRACE_ENTER (("Q931RecvMsg ()"));
        if (DlpiGet (&msg, q931DlpiFd) < 0) {
                perror ("Q931RecvDlpiMsg: DlpiGet fails !");
        } else if (msg.type == DL_ESTABLISH_CON) {
                TRACEX (DFLAG_Q921, ("DL_ESTABLISH_CON"));
        } else if (msg.type == DL_DATA_IND) {
                HandleQ931Msg (msg.buf, msg.len);
        } else {
                TRACE (("unknown primitive: %d\n", msg.type));
        }
        TRACE_EXIT;
} /* Q931RecvMsg */
/*------------------------------------------------------------------------*/
int Q931SendMsg (t_Byte *buf, int len)
{
        t_Q931DlpiMsg           msg;

        TRACE_ENTER (("Q931SendMsg(buf, %d)", len));
/*      Dump (buf, len); */
        msg.type = DL_DATA_REQ;
        msg.buf = buf;
        msg.len = len;
        if (DlpiPut (&msg, q931DlpiFd) < 0) {
                perror ("Q931SendMsg: DlpiPut fails !");
                return -1;
        }
        TRACE_EXIT;
        return 0;
} /* Q931SendMsg */
/*------------------------------------------------------------------------*/
```

E.4.6 src/Q931Main.h

```
/*
```

```
 *      file:           Q931Main.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Main.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Main.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_MAIN
#define INCL_Q931_MAIN

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- global functions --------------------------------------------------*/
extern  int             Q931Initialize (void);
extern  t_Q931Cxt       *Q931SearchCallId (int callId);
extern  t_Q931Cxt       *Q931SearchCallRef (const t_CallRef *callRef);
extern  t_Q931Cxt       *Q931AllocCxt (const t_CallRef *callRef);
extern  void            Q931FreeCxt (t_Q931Cxt *cxt);

/*----- global variables --------------------------------------------------*/
extern  t_Q931Cxt               *q931CxtList;
extern  int                     q931DlpiFd;

#endif /* INCL_Q931_MAIN */
```

E.4.7 src/Q931Main.c

```
/*
 *      file:           Q931Main.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Main.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Main.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Main.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdlib.h>
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Q931.h"
#include "Q931Msg.h"
#include "IsdnLib.h"
#include "Q931Main.h"
#include "Dlpi.h"
#include "Debug.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/
static  t_Q931Cxt               *cxtList = NULL;

/*----- global variables --------------------------------------------------*/
        int                     q931DlpiFd = -1;

/*---------------------------------------------------------------------------*/
t_Q931Cxt *Q931SearchCallId (int callId)
{
        t_Q931Cxt               *cxt;

        TRACE_ENTER (("Q931SearchCallId()"));
        cxt = cxtList;
        while (cxt != NULL)
                if (cxt->callId == callId)
                        break;
                else
                        cxt = cxt->next;
        if (cxt == NULL)
                TRACEX (DFLAG_CALLID, ("callId not found!"))
        else
                TRACEX (DFLAG_CALLID, ("cxt=%p, callRefLen=%d callRef=%02d, state=%d", cxt, cxt->callRef.len, cxt->callRef.value[0], cxt->state))
        TRACE_EXIT;
        return cxt;
```

```
/*** to make gcc happy ***/
        *rcsid = *rcsid;
} /* Q931SearchCallId */
/*------------------------------------------------------------------------*/
t_Q931Cxt *Q931SearchCallRef (const t_CallRef *callRef)
{
        t_Q931Cxt               *cxt;

        TRACE_ENTER (("Q931SearchCallRef()"));
        cxt = cxtList;
        while (cxt != NULL)
                if (Q931CompCallRef (&cxt->callRef, callRef) == 0)
                        break;
                else
                        cxt = cxt->next;
        if (cxt != NULL)
                TRACEX (DFLAG_CALLID, ("cxt=%p, callRefLen=%d callRef=%02d, state=%d", cxt, cxt->callRef.len, cxt->callRef.value[0], cxt->s
        TRACE_EXIT;
        return cxt;
} /* Q931SearchCallRef */
/*------------------------------------------------------------------------*/
t_Q931Cxt *Q931AllocCxt (const t_CallRef *callRef)
{
        t_Q931Cxt               *cxt;

        TRACE_ENTER (("Q931AllocCxt (%p)", callRef));
        if ((cxt = malloc (sizeof (t_Q931Cxt))) == NULL)
                return NULL;
        Q931_ZERO (*cxt);
        Q931CopyCallRef (&cxt->callRef, callRef);
        cxt->state = Q931_US_NULL;
        cxt->next = cxtList;
        cxt->fd = -1;
        cxtList = cxt;
        TRACE_EXIT;
        return cxt;
} /* Q931AllocCxt */
/*------------------------------------------------------------------------*/
void Q931FreeCxt (t_Q931Cxt *cxt)
{
        t_Q931Cxt               *tmp;
        t_Q931Cxt               *last;

        TRACE_ENTER (("Q931FreeCxt()"));
        last = NULL;
        tmp = cxtList;
        while (tmp) {
                if (tmp == cxt)
                        break;
                last = tmp;
                tmp = tmp->next;
        }
        if (tmp) {
                if (last == NULL)
                        cxtList = tmp->next;
                else
                        last->next = tmp->next;
                free (tmp);
        }
        TRACE_EXIT;
} /* Q931FreeCxt */
/*------------------------------------------------------------------------*/
int Q931Initialize ()
/*
 * DESC
 *      Initializes the whole module. It also opens the Q.921 DLPI interface
 *      and installs the file handler callback for incomming messages.
 * RET
 *      0       OK
 *      -1      indicates an error
 */
{
        t_Q931DlpiMsg           msg;

        if ((q931DlpiFd = DlpiOpen (NULL)) < 0) {
                return -1;
        }
        msg.type = DL_ESTABLISH_REQ;
        msg.buf = NULL;
        msg.len = 0;
        if (DlpiPut (&msg, q931DlpiFd) < 0) {
                perror ("Q931Initialize");
                DlpiClose (q931DlpiFd);
                return -1;
        }
        IsdnCreateFileHandler (q931DlpiFd, ISDN_READABLE, Q931RecvMsg, NULL);
        cxtList = NULL;
        return 0;
} /* Q931Initialize */
/*------------------------------------------------------------------------*/
```

E.4.8 src/Q931Timer.h

```
/*
 *      file:           Q931Timer.h
```

```
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Q931Timer.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Q931Timer.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_Q931_TIMER
#define INCL_Q931_TIMER

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/
        /*** timer ids ***/
#define Q931_TIMER_T300                 0               /* private */
#define Q931_TIMER_T301                 1
#define Q931_TIMER_T302                 2
#define Q931_TIMER_T303                 3
#define Q931_TIMER_T304                 4
#define Q931_TIMER_T305                 5
#define Q931_TIMER_T308                 8
#define Q931_TIMER_T310                 10
#define Q931_TIMER_T313                 13
#define Q931_TIMER_T318                 18
#define Q931_TIMER_T319                 19

/*----- type definitions --------------------------------------------------*/

/*----- global functions --------------------------------------------------*/
extern  void    Q931StartTimer (t_Q931Cxt *cxt, const int timer);
extern  void    Q931StopTimer (t_Q931Cxt *cxt, const int timer);
extern  void    Q931StopAllTimers (t_Q931Cxt *cxt);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_Q931_TIMER */
```

E.4.9 src/Q931Timer.c

```
/*
 *      file:           Q931Timer.c
 *      written by:     Christian Zahl
 *      description:    Handles the various timers and timeouts
 *
 * $Id: Q931Timer.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Q931Timer.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Q931Timer.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <string.h>

/*----- user includefiles -------------------------------------------------*/
#include "Types.h"
#include "Q931.h"
#include "Q931Msg.h"
#include "Q931Disconnect.h"
#include "Q931Release.h"
#include "Q931Timer.h"
#include "Q931Main.h"
#include "IsdnLib.h"
#include "Debug.h"

/*----- defines -----------------------------------------------------------*/

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/
static  void    Q931TimerT300 (t_Q931Cxt *cxt);
static  void    Q931TimerT302 (t_Q931Cxt *cxt);
static  void    Q931TimerT303 (t_Q931Cxt *cxt);
static  void    Q931TimerT304 (t_Q931Cxt *cxt);
static  void    Q931TimerT305 (t_Q931Cxt *cxt);
static  void    Q931TimerT308 (t_Q931Cxt *cxt);
static  void    Q931TimerT310 (t_Q931Cxt *cxt);
static  void    Q931TimerT313 (t_Q931Cxt *cxt);
static  void    Q931TimerT318 (t_Q931Cxt *cxt);
static  void    Q931TimerT319 (t_Q931Cxt *cxt);

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/
static  int                     timeout[] = {
                                  8000,         /* dummy T300 */
                                180000,         /* T301 */
```

```
                                15000,          /* T302 */
                                 4000,          /* T303 */
                                30000,          /* T304 */
                                30000,          /* T305 */
                                    0,          /* T306 n/a */
                                    0,          /* T307 n/a */
                                 4000,          /* T308 */
                                90000,          /* T309 */
                                60000,          /* T310 */
                                    0,          /* T311 n/a */
                                    0,          /* T312 n/a */
                                 4000,          /* T313 */
                                 4000,          /* T314 */
                                    0,          /* T315 n/a */
                               120000,          /* T316 */
                                60000,          /* T317 */
                                 4000,          /* T318 */
                                 4000,          /* T319 */
                                    0,          /* T320 n/a */
                                30000,          /* T321 */
                                 4000,          /* T322 */
                               };
static  void                   (*timerCb[])() = {
                               Q931TimerT300,
                               NULL,
                               Q931TimerT302,
                               Q931TimerT303,
                               Q931TimerT304,
                               Q931TimerT305,
                               NULL,
                               NULL,
                               Q931TimerT308,
                               NULL,
                               Q931TimerT310,
                               NULL,
                               NULL,
                               Q931TimerT313,
                               NULL,
                               NULL,
                               NULL,
                               NULL,
                               Q931TimerT318,
                               Q931TimerT319,
                               NULL,
                               NULL,
                               NULL,
                               NULL};

/*----- global variables -------------------------------------------------*/

/*------------------------------------------------------------------------*/
void SetupComplError (t_Q931Cxt *cxt)
{
        TRACE_ENTER (("SetupComplError (%p)", cxt));
        Q931_FFS ();
        TRACE_EXIT;
        return;
        *rcsid = *rcsid;               /* to make gcc happy */
}
/*------------------------------------------------------------------------*/
void ResumeConfirmError (t_Q931Cxt *cxt)
{
        TRACE_ENTER (("ResumeConfirmError (%p)", cxt));
        Q931_FFS ();
        TRACE_EXIT;
}
/*------------------------------------------------------------------------*/
void SuspendConfirmError (t_Q931Cxt *cxt)
{
        TRACE_ENTER (("SuspendConfirmError (%p)", cxt));
        Q931_FFS ();
        TRACE_EXIT;
}
/*------------------------------------------------------------------------*/
void SetupConfError (t_Q931Cxt *cxt)
/*
 * The SETUP message send to the network was not answered. To inform the
 * calling user, we will generate a RELEASE indication.
 */
{
        t_IsdnInd              ind;

        TRACE_ENTER (("SetupConfError (%p)", cxt));
        TRACEX (DFLAG_Q931_INTERNAL, ("SETUP-CONF-ERROR (RELEASE) INDICATION"));
        Q931_ZERO (ind);
        ind.ind.indType = ISDN_RELEASE_IND;
        ind.ind.callId = cxt->callId;
/*** cause ***/
        ind.releaseInd.cause.location = CA_L_PRIV_NET_LOC_USER;
        ind.releaseInd.cause.value = CA_V_TIMER_EXPIRED;
        IsdnCallback (&ind);
        TRACE_EXIT;
} /* SetupConfError */
/*------------------------------------------------------------------------*/
static void Q931TimerT300 (t_Q931Cxt *cxt)
/*
 * NOTE:
 *      This timer does NOT exists in Q.931, but it is necessary for
 *      incomming calls. When two SETUPs has been received, and noone
```

```
 *      has answered the call within 4s, then the network cancels the
 *      call. But there is NO RELEASE message or something like this
 *      send. So we use this timer to trigger the calcelation.
 */
{
        t_IsdnInd               ind;

        TRACE_ENTER (("Q931TimerT300 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T300 expired!"));
        if (cxt->timer[Q931_TIMER_T300] <= 0) {
                TRACEX (DFLAG_WARN |DFLAG_TIMER, ("Timer not running!"));
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T300] = 0;
        if (cxt->state == Q931_US_CALL_PRESENT) {               /* 6 */
/*** NOTE: not Q.931 conform ***/
                Q931FreeCallRef (&cxt->callRef);
                cxt->state = Q931_US_NULL;
                Q931FreeCxt (cxt);
                Q931_ZERO (ind);
/*** fake an RELEASE ***/
                ind.ind.indType = ISDN_RELEASE_IND;
                ind.ind.callId = cxt->callId;
                IsdnCallback (&ind);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT300 */
/*---------------------------------------------------------------------------*/
static void Q931TimerT302 (t_Q931Cxt *cxt)
{
        TRACE_ENTER (("Q931TimerT302 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T302 expired!"));
        if (cxt->timer[Q931_TIMER_T302] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T302] = 0;
        if (cxt->state == Q931_US_OVERLAP_RECEIVING) {          /* 25 */
#ifdef FFS
                timeout ind ...
#endif
                cxt->state = Q931_US_OVERLAP_RECEIVING;         /* 25 */
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT302 */
/*---------------------------------------------------------------------------*/
static void Q931TimerT303 (t_Q931Cxt *cxt)
/*
 * NOTE:
 *      No support for two immediate timeouts!
 */
{
        t_IsdnInd               ind;

        TRACE_ENTER (("Q931TimerT303 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T303 expired!"));
        if (cxt->timer[Q931_TIMER_T303] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T303] = 0;
        if (cxt->state == Q931_US_CALL_INITIATED) {             /* 1 */
                SetupConfError (cxt);
                Q931FreeCallRef (&cxt->callRef);
                cxt->state = Q931_US_NULL;
                Q931FreeCxt (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT303 */
/*---------------------------------------------------------------------------*/
static void Q931TimerT304 (t_Q931Cxt *cxt)
{
        t_Q931DisconnectReq     req;

        TRACE_ENTER (("Q931TimerT304 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T304 expired!"));
        if (cxt->timer[Q931_TIMER_T304] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T304] = 0;
        if (cxt->state == Q931_US_OVERLAP_SENDING) {            /* 2 */
                Q931_ZERO (req);
                req.cause.value = CA_V_TIMER_EXPIRED;
                Q931DisconnectReq (cxt, &req);
                SetupConfError (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
```

```
        }
        TRACE_EXIT;
} /* Q931TimerT304 */
/*------------------------------------------------------------------------*/
static void Q931TimerT305 (t_Q931Cxt *cxt)
{
        t_Q931ReleaseReq        req;

        TRACE_ENTER (("Q931TimerT305 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T305 expired!"));
        if (cxt->timer[Q931_TIMER_T305] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T305] = 0;
        if (cxt->state == Q931_US_DISCONNECT_REQUEST) {         /* 11 */
                Q931_ZERO (req);
                Q931SendRelease (cxt, &req);
                Q931StartTimer (cxt, Q931_TIMER_T308);
                cxt->state = Q931_US_RELEASE_REQUEST;           /* 19 */
                SetupConfError (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT305 */
/*------------------------------------------------------------------------*/
static void Q931TimerT308 (t_Q931Cxt *cxt)
{
        t_Q931ReleaseReq        req;

        TRACE_ENTER (("Q931TimerT308 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T308 expired!"));
        if (cxt->timer[Q931_TIMER_T308] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T308] = 0;
        if (cxt->state == Q931_US_RELEASE_REQUEST) {            /* 19 */
/* NOTE: currently implemented as endless-loop */
                Q931_ZERO (req);
                Q931SendRelease (cxt, &req);
                Q931StartTimer (cxt, Q931_TIMER_T308);
                cxt->state = Q931_US_RELEASE_REQUEST;
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT308 */
/*------------------------------------------------------------------------*/
static void Q931TimerT310 (t_Q931Cxt *cxt)
/*
 * DESC
 *      Handles the timeout of timer T310
 */
{
        t_Q931DisconnectReq     req;

        TRACE_ENTER (("Q931TimerT310 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T310 expired!"));
        if (cxt->timer[Q931_TIMER_T310] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T310] = 0;
        if (cxt->state == Q931_US_OUT_CALL_PROCEEDING) {        /* 3 */
                Q931_ZERO (req);
                req.cause.value = CA_V_TIMER_EXPIRED;
                Q931DisconnectReq (cxt, &req);
                SetupConfError (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT310 */
/*------------------------------------------------------------------------*/
static void Q931TimerT313 (t_Q931Cxt *cxt)
{
        t_Q931DisconnectReq     req;

        TRACE_ENTER (("Q931TimerT313 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T313 expired!"));
        if (cxt->timer[Q931_TIMER_T313] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T313] = 0;
        if (cxt->state == Q931_US_CONNECT_REQUEST) {            /* 8 */
                Q931_ZERO (req);
                req.cause.value = CA_V_TIMER_EXPIRED;
                Q931DisconnectReq (cxt, &req);
                SetupComplError (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
        }
```

```
        TRACE_EXIT;
} /* Q931TimerT313 */
/*------------------------------------------------------------------------*/
static void Q931TimerT318 (t_Q931Cxt *cxt)
{
        t_Q931ReleaseReq        req;

        TRACE_ENTER (("Q931TimerT318 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T318 expired!"));
        if (cxt->timer[Q931_TIMER_T318] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T318] = 0;
        if (cxt->state == Q931_US_RESUME_REQUEST) {             /* 17 */
                Q931_ZERO (req);
                Q931SendRelease (cxt, &req);
                Q931StartTimer (cxt, Q931_TIMER_T308);
                cxt->state = Q931_US_RELEASE_REQUEST;
                ResumeConfirmError (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT318 */
/*------------------------------------------------------------------------*/
static void Q931TimerT319 (t_Q931Cxt *cxt)
{
        TRACE_ENTER (("Q931TimerT319 (%p)", cxt));
        TRACEX (DFLAG_TIMER, ("T319 expired!"));
        if (cxt->timer[Q931_TIMER_T319] <= 0) {
                TRACE_EXIT;
                return;
        }
        cxt->timer[Q931_TIMER_T319] = 0;
        if (cxt->state == Q931_US_SUSPEND_REQUEST) {            /* 15 */
                cxt->state = Q931_US_ACTIVE;
                SuspendConfirmError (cxt);
        } else {
                /* handle error */
                Q931_FFS ();
        }
        TRACE_EXIT;
} /* Q931TimerT319 */
/*------------------------------------------------------------------------*/
void Q931StartTimer (t_Q931Cxt *cxt, const int timer)
/*
 * DESC
 *      Starts the requested timer
 */
{
        TRACE_ENTER (("Q931StartTimer (%p, T%d)", cxt, timer +300));
        cxt->timer[timer] = IsdnCreateTimerHandler (timeout[timer], timerCb[timer], cxt);
        TRACEX (DFLAG_TIMER, ("T%d started", timer +300));
        TRACE_EXIT;
} /* Q931StartTimer */
/*------------------------------------------------------------------------*/
void Q931StopTimer (t_Q931Cxt *cxt, const int timer)
/*
 * DESC
 *      Stops a privously started timer. If the timer isn't active any more,
 *      then nothing happens.
 */
{
        TRACE_ENTER (("Q931StopTimer (cxt, T%d)", timer +300));
        TRACEX (DFLAG_TIMER, ("T%d stopped", timer +300));
        if (cxt->timer[timer]) {
                IsdnDeleteTimerHandler (cxt->timer[timer]);
                cxt->timer[timer] = 0;
        } else {
                TRACEX (DFLAG_WARN |DFLAG_TIMER, ("Timer was not running!"));
        }
        TRACE_EXIT;
} /* Q931StopTimer */
/*------------------------------------------------------------------------*/
void Q931StopAllTimers (t_Q931Cxt *cxt)
/*
 * DESC
 *      Stops all currently running timers.
 */
{
        int                     i;

        TRACE_ENTER (("932StopAllTimers (%p)", cxt));
        for (i=Q931_TIMER_T301; i<=Q931_TIMER_T319; i++)
                if (cxt->timer[i])
                        Q931StopTimer (cxt, i);
        TRACE_EXIT;
} /* Q931StopAllTimers */
/*------------------------------------------------------------------------*/
```

E.5 DLPI Interface

Das Interface zum DLPI Streams Modul, welches das Q.921 Protokoll implementiert, wird durch die folgenden C-Files realisiert.

E.5.1 src/Dlpi.h

```
/*
 *      file:           Dlpi.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Dlpi.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Dlpi.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_DLPI
#define INCL_DLPI

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/
        /* q931 primitives */
#define DL_ESTABLISH_REQ        1
#define DL_ESTABLISH_CON        2
#define DL_RELEASE_REQ          3
#define DL_RELEASE_IND          4
#define DL_RELEASE_CON          5
#define DL_DATA_REQ             6
#define DL_DATA_IND             7
#define DL_UNITDATA_INDICATION  8

/*----- type definitions --------------------------------------------------*/
typedef struct {
        int             type;                   /* message type (aka primitive) */
        int             len;                    /* length of buf */
        unsigned char   *buf;                   /* the buffer */
} t_Q931DlpiMsg;

/*----- global functions --------------------------------------------------*/
extern  int     DlpiOpen (char *dlpiDriver);
extern  int     DlpiClose (int fd);
extern  int     DlpiGet (t_Q931DlpiMsg *ptr, int fd);
extern  int     DlpiPut (t_Q931DlpiMsg *ptr, int fd);

/*----- global variables --------------------------------------------------*/

#endif /* INCL_DLPI */
```

E.5.2 src/Dlpi.c

```
/*

                        *** dlpi_open.c ***

                   TIK-76.115 INDIVIDUAL PROJECT
                          GROUP: SunShine
                         AUTHOR: Bengt Sahlin

*/
/*
 *      file:           dlpi.c
 *      written by:     Bengt Sahlin
 *      modified by:    Christian Zahl
 *      description:    This module communicates with the lower datalink
 *                      driver. In general, this would be the q921
 *                      driver.
 *
 * $Id: Dlpi.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Dlpi.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */

static  char                    rcsid[] = "$Id: Dlpi.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include <stdlib.h>
#include <stropts.h>
#include <unistd.h>

/*----- user includefiles ----------------------------------------------------*/
#include "dlpi.h"
#include "Dlpi.h"
#include "Debug.h"

/*----- defines --------------------------------------------------------------*/
#define DEBUG(xxx)      {xxx}
#define DEFAULT_DLPI_DRIVER     "/dev/q921"

/*----- type definitions -----------------------------------------------------*/

/*----- local functions ------------------------------------------------------*/

/*----- global functions -----------------------------------------------------*/

/*----- local variables ------------------------------------------------------*/

/*----- global variables -----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
void Dump (unsigned char *buf, int len)
{
        int                     i;

        while (len > 0) {
                printf ("\t");
                for (i=0; i<16; i++)
                        if (i < len)
                                printf ("%02X ", buf[i]);
                        else
                                printf ("   ");
                printf ("--");
                for (i=0; i<16; i++)
                        if (i < len)
                                if (buf[i] >= ' ' && buf[i] < 127)
                                        printf ("%c", buf[i]);
                                else
                                        printf ("?");
                printf ("\n");
                len -= 16;
                buf += 16;
        }
        return;
        *rcsid = *rcsid;                /* to make gcc happy */
} /* Dump */
/*---------------------------------------------------------------------------*/
int DlpiOpen (char *dlpiDriver)
/*
 * DESC
 *      DlpiOpen opens a stream to the specified datalink driver dlpiDriver.
 *      If dlpiDriver is NULL, then the default q921 driver will be
 *      used.
 *
 * RETURNS
 *      -1      unable to connect to the datalink driver
 *      >= 0    the fd to / from the datalink driver
 */
{
        int                     fd;

        TRACE_ENTER (("DlpiOpen()"));
        if (dlpiDriver == NULL)
                dlpiDriver = DEFAULT_DLPI_DRIVER;
        if ((fd = open (dlpiDriver, O_RDWR)) < 0) {
                perror ("DlpiOpen: unable to open");
                return -1;
        }
        TRACE_EXIT;
        return fd;
} /* DlpiOpen */
/*---------------------------------------------------------------------------*/
int DlpiClose (int fd)
/*
 * DESC
 *      DlpiClose closes the stream to the datalink driver. Before that
 *      it has to perform local management functions (unbinding and
 *      detaching).
 * RETURNS
 *      -1      error occured
 *      0       OK
 */
{
        TRACE_ENTER (("DlpiClose()"));
        if (close (fd) < 0) {
                perror ("DlpiClose: close failed");
                return -1;
        }
        TRACE_EXIT;
        return 0;
} /* DlpiClose */
/*---------------------------------------------------------------------------*/
int DlpiGet (t_Q931DlpiMsg *ptr, int fd)
/*
 *   dlpi_get gets messages from the q921_module. It translates the
 *   DLPI-primitives to q931-primtives. Four different messages can
```

```
 *   occur. DL_CONNECT_CON confirms the connection to the telephone
 *   switch, DL_DISCONNECT_CON confirms a release of the connection.
 *   Data from a peer entity can come through connection-oriented data
 *   transfer (no DLPI - primitive) or with the DL_UNITDATA_IND-primi-
 *   tive.
 * RETURNS
 *      RC      0 on success, -1 otherwise
 *      *ptr    filled with appriate data
 */
{
        struct strbuf           ctrlMsg;
        union dl_primitives     ctrlBuf;
        struct strbuf           dataMsg;
        unsigned char           dataBuf[2000];
        int                     flags = 0;

        TRACE_ENTER (("DlpiGet()"));
        ctrlMsg.len = 0;
        ctrlMsg.maxlen = sizeof (union dl_primitives);
        ctrlMsg.buf = (char *) &ctrlBuf;
        dataMsg.len = 0;
        dataMsg.maxlen = sizeof (dataBuf);
        dataMsg.buf = dataBuf;
        if (getmsg (fd, &ctrlMsg, &dataMsg, &flags) != 0) {
                perror ("DlpiGet: getmsg failed");
                TRACE_EXIT;
                return -1;
        }
        TRACEX (DFLAG_Q921, ("DlpiGet, message received (ctrLen=%d, dataLen=%d)", ctrlMsg.len, dataMsg.len))
/*** check if we don't get both message types ***/
        if (ctrlMsg.len > 0 && dataMsg.len > 0) {
                fprintf (stderr, "DlpiGet: received data and control message at the same time\n");
                TRACE_EXIT;
                return -1;
        }
/*** check for any control message data (M_PROTO data) ***/
        if (ctrlMsg.len > 0) {
                DEBUGX (DFLAG_FCALLS, Dump (ctrlMsg.buf, ctrlMsg.len);)
                switch (ctrlBuf.dl_primitive) {
                        case DL_CONNECT_CON:
                                TRACEX (DFLAG_Q921, ("CONNECTION CONFIRMATION"))
                                ptr->type = DL_ESTABLISH_CON;
                                ptr->len = 0;
                                ptr->buf = NULL;
                                TRACE_EXIT;
                                return 0;
                        case DL_DISCONNECT_IND:
                                TRACEX (DFLAG_Q921, ("DISCONNECT INDICATION"))
                                ptr->type = DL_RELEASE_IND;
                                ptr->len = 0;
                                ptr->buf = NULL;
                                TRACE_EXIT;
                                return 0;
                        case DL_UNITDATA_IND:
                                TRACEX (DFLAG_Q921, ("UNITDATA INDICATION"))
                                DEBUGX (DFLAG_Q921, Dump (dataMsg.buf, dataMsg.len);)
                                ptr->type = DL_UNITDATA_INDICATION;
                                ptr->len = dataMsg.len;
                                ptr->buf = malloc (dataMsg.len);
                                if (ptr->buf == NULL) {
                                        perror ("DlpiGet: malloc failed");
                                        TRACE_EXIT;
                                        return -1;
                                }
                                memcpy (ptr->buf, dataMsg.buf, dataMsg.len);
                                TRACE_EXIT;
                                return 0;
                        default:
                                fprintf (stderr, "DlpiGet: message (%ld) is not valid\n", ctrlBuf.dl_primitive);
                                TRACE_EXIT;
                                return -1;
                } /* switch */
        } /* if */
/*** check for any "normal" message data (M_DATA) ***/
        if (dataMsg.len > 0) {
                TRACEX (DFLAG_Q921, ("DATA INDICATION:"))
                DEBUGX (DFLAG_Q921, Dump (dataMsg.buf, dataMsg.len);)
                ptr->type = DL_DATA_IND;
                ptr->len = dataMsg.len;
                ptr->buf = malloc (dataMsg.len);
                if (ptr->buf == NULL) {
                        perror ("DlpiGet: malloc failed");
                        TRACE_EXIT;
                        return -1;
                }
                memcpy (ptr->buf, dataMsg.buf, dataMsg.len);
                TRACE_EXIT;
                return 0;
        }
/*** when this point is reached, then no message was received ?***/
        TRACE_EXIT;
        return -1;
} /* DlpiGet */
/*------------------------------------------------------------------------*/
int DlpiPut (t_Q931DlpiMsg *ptr, int fd)
/*
 *   dlpi_put gets messages from the q931-module. It translates the
 *   q931-primitives to dlpi-primitives and sends down the message
 *   with putmsg to the q921-module. Three requests can be done.
```

```
 *   DL_CONNECT_REQ sends a request to the q921-module to establish
 *   a connection to the telephone switch, DL_DISCONNECT_REQ to release
 *   the link. DL_DATA_REQ is used to send information to another q931-module.
 */
{
        struct strbuf           ctrlMsg;
        struct strbuf           dataMsg;
        dl_connect_req_t        connectReq;
        dl_disconnect_req_t     disconnectReq;

        TRACE_ENTER (("DlpiPut()"));
        switch (ptr->type) {
                case DL_ESTABLISH_REQ:
                        TRACEX (DFLAG_Q921, ("ESTABLISH_REQ"))
                        connectReq.dl_primitive = DL_CONNECT_REQ;
                        connectReq.dl_dest_addr_length = 0;
                        connectReq.dl_dest_addr_offset = 0;
                        connectReq.dl_qos_length = 0;
                        connectReq.dl_qos_offset = 0;
                        connectReq.dl_growth = 0;
                        ctrlMsg.len = sizeof (union dl_primitives);
                        ctrlMsg.buf = (char *) &connectReq;
                        if (putmsg (fd, &ctrlMsg, NULL, 0) != 0) {
                                perror ("DlpiPut: putmsg failed");
                                TRACE_EXIT;
                                return -1;
                        }
                        TRACE_EXIT;
                        return 0;
                case DL_RELEASE_REQ:
                        TRACEX (DFLAG_Q921, ("RELEASE_REQ"))
                        disconnectReq.dl_primitive = DL_DISCONNECT_REQ;
                        disconnectReq.dl_reason = 0;
                        disconnectReq.dl_correlation = 0;
                        ctrlMsg.len = sizeof (union dl_primitives);
                        ctrlMsg.buf = (char *) &disconnectReq;
                        if (putmsg (fd, &ctrlMsg, NULL, 0) != 0) {
                                perror ("DlpiPut: putmsg failed");
                                TRACE_EXIT;
                                return -1;
                        }
                        TRACE_EXIT;
                        return 0;
                case DL_DATA_REQ:
                        TRACEX (DFLAG_Q921, ("DATA_REQ"))
                        dataMsg.len = ptr->len;
                        dataMsg.buf = ptr->buf;
                        DEBUGX (DFLAG_Q921, Dump (dataMsg.buf, dataMsg.len);)
                        if (putmsg (fd, NULL, &dataMsg, 0) != 0) {
                                perror ("DlpiPut: putmsg failed");
                                TRACE_EXIT;
                                return -1;
                        }
                        TRACE_EXIT;
                        return 0;
                default:
                        fprintf(stderr, "Not a valid primitive (%d)\n", ptr->type);
                        TRACE_EXIT;
                        return -1;
        } /* switch */
} /* DlpiPut */
/*------------------------------------------------------------------------*/
```

E.5.3 src/dlpi.h

```
/*
 * dlpi.h: header file for Data Link Provider Interface
 * for more information about DLPI, look at:
 * gopher://ftp.std.com:70/11/obi/book/Standards/DLPI
 *
 */

#define DL_CURRENT_VERSION 0x02
#define DL_VERSION_2 0x02

/*
   Primitives for Local Management Services
*/

#define DL_INFO_REQ 0x00
#define DL_INFO_ACK 0x03
#define DL_ATTACH_REQ 0x0b
#define DL_DETACH_REQ 0x0c
#define DL_BIND_REQ 0x01
#define DL_BIND_ACK 0x04
#define DL_UNBIND_ACK 0x02
#define DL_OK_ACK 0x06
#define DL_ERROR_ACK 0x05

/*
   Primitives used for Connectionless Service
*/

#define DL_UNITDATA_IND 0x07
```

```
/*

   Primitives used for Connection-Oriented Service
*/

#define DL_CONNECT_REQ 0x0d
#define DL_CONNECT_IND 0x0e
#define DL_CONNECT_CON 0x10

#define DL_DISCONNECT_REQ 0x13
#define DL_DISCONNECT_IND 0x14


/*
   Primitives used for Data Transfer Service (This is not in the
   standard, it is used in our solution in the function dlpi_get.
*/
#define DL_NO_PRIMITIVE 1000

/*
   DLPI provider service supported
*/

#define DL_CODLS 0x01
#define DL_CLDLS 0x02
#define DL_ACLDLS 0x04

/*
 *
 *  DLPI INTERFACE PRIMITIVE DEFINITIONS
 *
 */

/*
 *  LOCAL MANAGEMENT SERVICE PRIMITIVES
 */


/*
   DL_INFO_REQ, M_PCPROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
} dl_info_req_t;


/*
   DL_INFO_ACK, M_PCPROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_max_sdu;
  unsigned long dl_min_sdu;
  unsigned long dl_addr_length;
  unsigned long dl_mac_type;
  unsigned long dl_reserved;
  unsigned long dl_current_state;
  unsigned long dl_sap_length;
  unsigned long dl_service_mode;
  unsigned long dl_qos_length;
  unsigned long dl_qos_offset;
  unsigned long dl_qos_range_length;
  unsigned long dl_qos_range_offset;
  unsigned long dl_version;
  unsigned long dl_brdcst_addr_length;
  unsigned long dl_brdcst_addr_offset;
  unsigned long dl_growth;
} dl_info_ack_t;

/*
   DL_ATTACH_REQ, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_ppa;
} dl_attach_req_t;

/*
   DL_DETACH_REQ, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
} dl_detach_req_t;

/*
   DL_BIND_REQ, M_PROTO type
*/

typedef struct
{
```

```
  unsigned long dl_primitive;
  unsigned long dl_sap;
  unsigned long dl_max_conind;
  unsigned short dl_service_mode;
  unsigned short dl_conn_mgmt;
  unsigned long dl_xidtest_flag;
} dl_bind_req_t;

/*
   DL_BIND_ACK, M_PCROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_sap;
  unsigned long dl_addr_length;
  unsigned long dl_addr_offset;
  unsigned long dl_max_conind;
  unsigned long dl_xidtest_flag;
} dl_bind_ack_t;


/*
   DL_UNBIND_REQ, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
} dl_unbind_req_t;


/*
   DL_OK_ACK, M_PCROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_correct_primitive;
} dl_ok_ack_t;


/*
   DL_ERROR_ACK, M_PCROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_error_primitive;
  unsigned long dl_errno;
  unsigned long dl_unix_errno;
} dl_error_ack_t;


/*
 *  CONNECTION-ORIENTED SERVICE PRIMITIVES
 */

/*
   DL_CONNECT_REQ, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_dest_addr_length;
  unsigned long dl_dest_addr_offset;
  unsigned long dl_qos_length;
  unsigned long dl_qos_offset;
  unsigned long dl_growth;
} dl_connect_req_t;

/*
   DL_CONNECT_IND, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_correlation;
  unsigned long dl_called_addr_length;
  unsigned long dl_called_addr_offset;
  unsigned long dl_calling_addr_length;
  unsigned long dl_calling_addr_offset;
  unsigned long dl_qos_length;
  unsigned long dl_qos_offset;
  unsigned long dl_growth;
} dl_connect_ind_t;

/*
   DL_CONNECT_CON, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
```

```
  unsigned long dl_correlation;
  unsigned long dl_resp_addr_length;
  unsigned long dl_resp_addr_offset;
  unsigned long dl_qos_length;
  unsigned long dl_qos_offset;
  unsigned long dl_growth;
} dl_connect_con_t;

/*
   DL_DISCONNECT_REQ, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_reason;
  unsigned long dl_correlation;
} dl_disconnect_req_t;

/*
   DL_DISCONNECT_IND, M_PROTO type
*/

typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_originator;
  unsigned long dl_reason;
  unsigned long dl_correlation;
} dl_disconnect_ind_t;

/*
 *   CONNECTIONLESS SERVICE PRIMITIVES
 */

/*
   DL_UNITDATA_IND, M_PROTO type, with M_DATA block(s)
*/


typedef struct
{
  unsigned long dl_primitive;
  unsigned long dl_dest_addr_length;
  unsigned long dl_dest_addr_offset;
  unsigned long dl_src_addr_length;
  unsigned long dl_src_addr_offset;
  unsigned long dl_group_address;
} dl_unitdata_ind_t;


/*
 * union of all dlpi-primitives
 */

union dl_primitives
{
  unsigned long dl_primitive;
  dl_info_req_t info_req;
  dl_info_ack_t info_ack;
  dl_attach_req_t attach_req;
  dl_detach_req_t detach_req;
  dl_bind_req_t bind_req;
  dl_bind_ack_t bind_ack;
  dl_unbind_req_t unbind_req;
  dl_ok_ack_t ok_ack;
  dl_error_ack_t error_ack;
  dl_connect_req_t connect_req;
  dl_connect_ind_t connect_ind;
  dl_connect_con_t connect_con;
  dl_disconnect_req_t disconnect_req;
  dl_disconnect_ind_t disconnect_ind;
  dl_unitdata_ind_t unitdata_ind;
};
```

E.6 Der Scheduler

Der unabhängige Scheduler, der auch für andere Dinge verwendet werden kann.

E.6.1 src/Scheduler.h

```
/*
 *      file:           Scheduler.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Scheduler.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Scheduler.h,v $
```

```
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_SCHEDULER
#define INCL_SCHEDULER

/*----- standard includefiles ------------------------------------------------*/

/*----- local includefiles ---------------------------------------------------*/

/*----- defines --------------------------------------------------------------*/
       /* NOTE: these values HAVE to be the same as Tcl_xxx */
#define SCHEDULER_READABLE      (1<<1)
#define SCHEDULER_WRITABLE      (1<<2)
#define SCHEDULER_EXCEPTION     (1<<3)

/*----- type definitions -----------------------------------------------------*/

/*----- global functions -----------------------------------------------------*/
extern  int     SchedulerInitialize ();
extern  int     SchedulerCreateFileHandler (int fd, int mask, void (*callBack) (), void *data);
extern  void    SchedulerDeleteFileHandler (int fd);
extern  int     SchedulerCreateTimerHandler (int milliseconds, void (*cb) (), void *data);
extern  void    SchedulerDeleteTimerHandler (int timerId);
extern  void    SchedulerMainLoop ();

/*----- global variables -----------------------------------------------------*/

#endif /* INCL_SCHEDULER */
```

E.6.2 src/Scheduler.c

```
/*
 *      file:           Scheduler.c
 *      written by:     Christian Zahl
 *      description:    This module implements the receiption and schedulling
 *                      of messages.
 *
 * $Id: Scheduler.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Scheduler.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Scheduler.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/poll.h>
#include <sys/time.h>
#include <stropts.h>

/*----- user includefiles ----------------------------------------------------*/
#include "Scheduler.h"
#include "Debug.h"

/*----- defines --------------------------------------------------------------*/
#define CBLIST_LEN              20

/*----- type definitions -----------------------------------------------------*/
typedef struct {                                        /* cb list */
        void                    *data;                  /* pointer to any kind of data */
        void                    (*cb) ();               /* pointer to cb function */
        int                     pollListIdx;            /* index in pollList */
} t_CbList;

typedef struct _t_TimerList {
        int                     timeoutSec;
        int                     timeoutUsec;
        void                    (*cb) ();
        void                    *data;
        int                     timerId;                /* unique ID for this timer */
        struct _t_TimerList     *next;
} t_TimerList;

/*----- local functions ------------------------------------------------------*/

/*----- global functions -----------------------------------------------------*/

/*----- local variables ------------------------------------------------------*/
static  t_CbList                cbList[CBLIST_LEN];
static  struct pollfd           pollList[CBLIST_LEN];
static  int                     pollListLen;            /* # used pollList structs */
static  t_TimerList             *timerList;             /* ordered list of timers (top = shortest) */
static  int                     timerId = 0;

/*----- global variables -----------------------------------------------------*/
        int                     schedulerStop = 0;

/*--------------------------------------------------------------------------*/
int SchedulerInitialize ()
```

```
/*
 * DESC
 *      Initializes the scheduler.
 * RET
 *      0 = OK, error otherwise
 */
{
        int                     i;

        for (i=0; i<CBLIST_LEN; i++) {
                cbList[i].cb = NULL;
                cbList[i].data = NULL;
                cbList[i].pollListIdx = -1;
                pollList[i].fd = -1;
        }
        pollListLen = 0;
        schedulerStop = 0;
        timerList = NULL;
        return 0;
/*** to make gcc happy ***/
        *rcsid = *rcsid;
} /* SchedulerInitialize */
/*---------------------------------------------------------------------------*/
int SchedulerCreateFileHandler (int fd, int mask, void (*cb) (), void *data)
/*
 * DESC
 *      Installes a cb handler for the given filedescriptor. Also the
 *      fd will be added to the list of monitored fds.
 * NOTE
 *      Has to be compatible with Tcl_CreateFileHandler!
 */
{
        if (fd >= CBLIST_LEN)
                return -1;
        if (fd < 0)
                return -1;
        SchedulerDeleteFileHandler (fd);
        cbList[fd].cb = cb;
        cbList[fd].data = data;
        cbList[fd].pollListIdx = pollListLen;
        pollList[pollListLen].fd = fd;
        pollList[pollListLen].events = 0;
        if (mask & SCHEDULER_READABLE)
                pollList[pollListLen].events |= POLLIN;
        if (mask & SCHEDULER_WRITABLE)
                pollList[pollListLen].events |= POLLOUT;
        if (mask & SCHEDULER_EXCEPTION)
                pollList[pollListLen].events |= POLLPRI;
        pollListLen++;
        return 0;
} /* CreateFileHandler */
/*---------------------------------------------------------------------------*/
void SchedulerDeleteFileHandler (int fd)
/*
 * DESC
 *      Removes a previously created filehandler.
 * NOTE
 *      Has to be compatible with Tcl_DeleteFileHandler!
 */
{
        int                     i;
        int                     idx;

        if (fd >= CBLIST_LEN)
                return;
        if (fd < 0)
                return;
        if (cbList[fd].cb == NULL)
                return;
        idx = cbList[fd].pollListIdx;
        cbList[fd].cb = NULL;
        cbList[fd].data = NULL;
        cbList[fd].pollListIdx = 0;;
        pollListLen--;
        for (i=idx; i<pollListLen; i++) {
                memcpy (&pollList[i], &pollList[i+1], sizeof (struct pollfd));
                cbList[pollList[i].fd].pollListIdx = i;
        }
        return;
} /* SchedulerDeleteFileHandler */
/*---------------------------------------------------------------------------*/
int SchedulerCreateTimerHandler (int milliseconds, void (*cb) (), void *data)
/*
 * DESC
 *      Installs a timer cb for the given timeout. The timeout is given
 * PARAMS
 *      timeoutMsec
 *              Timeout in milliseconds
 *      cb
 *              The cb function
 *      data
 *              Timer related context
 * RET
 *      rc
 *              Timer ID (used when uninstalling a timer). -1 in cast of
 *              an error.
 * NOTE
 *      Has to be compatible with Tcl_CreateTimerHandler!
 */
```

```
{
        struct timeval          tv;
        int                     timeoutSec;
        int                     timeoutUsec;
        t_TimerList             *new;
        t_TimerList             *tmp;
        t_TimerList             *last;

        TRACE_ENTER (("SchedulerCreateTimerHandler(%d, %p, %p)", milliseconds, cb, data));
        if ((new = malloc (sizeof (t_TimerList))) == NULL) {
                TRACE_EXIT;
                return -1;
        }
/*** determine the real timeout time ***/
        gettimeofday (&tv, NULL);
        timeoutUsec = tv.tv_usec + (milliseconds % 1000) * 1000;
        timeoutSec = tv.tv_sec + milliseconds / 1000;
        timeoutSec += (timeoutUsec % 1000000) / 1000000;
        timeoutUsec = timeoutUsec % 1000000;
        new->timeoutUsec = timeoutUsec;
        new->timeoutSec = timeoutSec;
        new->cb = cb;
        new->data = data;
        new->timerId = ++timerId;
/*** insert new timer into the ordered list ***/
        tmp = timerList;
        last = NULL;
        while (tmp != NULL) {
                if (tmp->timeoutSec > new->timeoutSec)
                        break;
                if (tmp->timeoutSec == new->timeoutSec)
                        if (tmp->timeoutUsec > new->timeoutUsec)
                                break;
                last = tmp;
                tmp = tmp->next;
        }
        if (last == NULL)
                timerList = new;
        else
                last->next = new;
        new->next = tmp;
/*** return the timer ID ***/
        TRACE_EXIT;
        return new->timerId;
} /* SchedulerCreateTimerHandler */
/*---------------------------------------------------------------------------*/
void SchedulerDeleteTimerHandler (int timerId)
/*
 * DESC
 *      Stops a given timer (identified by the timerId) and removes it.
 * PARAMS
 *      timerId
 *              The unique ID for the timer to be stopped (returned from
 *              SchedulerTimerStart ())
 * RET
 *      -
 * NOTE
 *      Has to be compatible with Tcl_DeleteTimerHandler!
 */
{
        t_TimerList             *tmp;
        t_TimerList             *last;

        TRACE_ENTER (("SchedulerDeleteTimerHandler (%d)", timerId));
/*** search the timerList element ***/
        tmp = timerList;
        last = NULL;
        while (tmp != NULL) {
                if (tmp->timerId == timerId)
                        break;
                last = tmp;
                tmp = tmp->next;
        }
        if (tmp == NULL)
                return;
/*** remove the entry from the list ***/
        if (last == NULL)
                timerList = tmp->next;
        else
                last->next = tmp->next;
        free (tmp);
        TRACE_EXIT;
} /* SchedulerDeleteTimerHandler */
/*---------------------------------------------------------------------------*/
static int GetTimeout ()
/*
 * DESC
 *      Determines the timeout in miliseconds out of the current timerlist.
 * RET
 *      The timeout in milliseconds. If no timer is running, -1 will be
 *      returned. If any timer has been passed, then 0 will be returned.
 */
{
        struct timeval          tv;
        int                     timeout;

        TRACE_ENTER (("GetTimeout()"));
        if (timerList == NULL)
                timeout = -1;
```

```
        else {
                gettimeofday (&tv, NULL);
                timeout = (timerList->timeoutUsec - tv.tv_usec) / 1000 +
                        (timerList->timeoutSec - tv.tv_sec) * 1000;
                if (timeout < 0)
                        timeout = 0;
        }
        TRACEX (DFLAG_FCALLS, ("GetTimeout: timeout=%d", timeout))
        TRACE_EXIT;
        return timeout;
} /* GetTimeout */
/*---------------------------------------------------------------------------*/
static void HandleTimeout ()
/*
 * DESC
 *      Handles one timeout. Calles the cb function of for the given
 *      timer with the data and the timerId as the argument.
 * NOTE
 *      Because the list of Timers is ordered ascending of the timeout
 *      times, we only have to take the first element in the list.
 */
{
        t_TimerList             *old;

        TRACE_ENTER (("HandleTimeout()"));
        if (!timerList) {
                TRACE_EXIT;
                return;
        }
/*** first remove the timer element from the list ***/
        old = timerList;
        timerList = timerList->next;
/*** then call the callback routine ***/
        (*old->cb) (old->data);
/*** and finaly remove the timer element ***/
        free (old);
        TRACE_EXIT;
} /* HandleTimeout */
/*---------------------------------------------------------------------------*/
static void HandlePoll (struct pollfd *pollFd)
/*
 * DESC
 *      Handles any POOL event for the given fd. In fact the installed
 *      cb function will be called with the assigned data and fd.
 */
{
        int                     mask = 0;

        TRACE_ENTER (("HandlePoll()"));
        if (pollFd->revents & POLLIN)
                mask |= SCHEDULER_READABLE;
        if (pollFd->revents & POLLOUT)
                mask |= SCHEDULER_WRITABLE;
        if (pollFd->revents & POLLPRI)
                mask |= SCHEDULER_EXCEPTION;
        (*cbList[pollFd->fd].cb) (cbList[pollFd->fd].data, mask);
        TRACE_EXIT;
} /* HandlePoll */
/*---------------------------------------------------------------------------*/
void SchedulerMainLoop ()
/*
 * DESC
 *      This is the mail-loop of the whole module. It implements the
 *      scheduler, which handles the availability of incomming data and
 *      all the timers.
 *      The scheduler runs until no POLLIN events are
 *      requested, or until it is stopped (schedulerStop).
 */
{
        int                     i;
        int                     n;

        TRACE_ENTER (("SchedulerMainLoop()"));
        while (1) {
                if (pollListLen <= 0 || schedulerStop)
                        break;
                TRACEX (DFLAG_SCHEDULER, ("_____________________________"))
                n = poll (pollList, pollListLen, GetTimeout ());
                if (n == 0) {                                   /* timeout */
                        HandleTimeout ();
                        continue;
                }
                for (i=0; i<pollListLen; i++)
                        if (pollList[i].revents)
                                HandlePoll (&pollList[i]);
        } /* while */
        TRACE_EXIT;
} /* SchedulerMainLoop */
/*---------------------------------------------------------------------------*/
```

E.7 Allgemeine Routinen

Die folgenden Files enthalten Funktionen und Definitionen, für die allgemeine Benutzung.

E.7.1 src/Misc.h

```
/*
 *      file:           Misc.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Misc.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Misc.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_MISC
#define INCL_MISC

/*----- standard includefiles ---------------------------------------------*/

/*----- local includefiles ------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/
#define GET_BITS_8(buf)                 ((buf >> 7) & 0x01)
#define GET_BITS_7(buf)                 ((buf >> 6) & 0x01)
#define GET_BITS_76(buf)                ((buf >> 5) & 0x03)
#define GET_BITS_765(buf)               ((buf >> 4) & 0x07)
#define GET_BITS_7654(buf)              ((buf >> 3) & 0x0f)
#define GET_BITS_76543(buf)             ((buf >> 2) & 0x1f)
#define GET_BITS_765432(buf)            ((buf >> 1) & 0x3f)
#define GET_BITS_7654321(buf)           ((buf >> 0) & 0x7f)
#define GET_BITS_6(buf)                 ((buf >> 5) & 0x01)
#define GET_BITS_65(buf)                ((buf >> 4) & 0x03)
#define GET_BITS_654(buf)               ((buf >> 3) & 0x07)
#define GET_BITS_6543(buf)              ((buf >> 2) & 0x0f)
#define GET_BITS_65432(buf)             ((buf >> 1) & 0x1f)
#define GET_BITS_654321(buf)            ((buf >> 0) & 0x3f)
#define GET_BITS_5(buf)                 ((buf >> 4) & 0x01)
#define GET_BITS_54(buf)                ((buf >> 3) & 0x03)
#define GET_BITS_543(buf)               ((buf >> 2) & 0x07)
#define GET_BITS_5432(buf)              ((buf >> 1) & 0x0f)
#define GET_BITS_54321(buf)             ((buf >> 0) & 0x1f)
#define GET_BITS_4(buf)                 ((buf >> 3) & 0x01)
#define GET_BITS_43(buf)                ((buf >> 2) & 0x03)
#define GET_BITS_432(buf)               ((buf >> 1) & 0x07)
#define GET_BITS_4321(buf)              ((buf >> 0) & 0x0f)
#define GET_BITS_3(buf)                 ((buf >> 2) & 0x01)
#define GET_BITS_32(buf)                ((buf >> 1) & 0x03)
#define GET_BITS_321(buf)               ((buf >> 0) & 0x07)
#define GET_BITS_2(buf)                 ((buf >> 1) & 0x01)
#define GET_BITS_21(buf)                ((buf >> 0) & 0x03)
#define GET_BITS_1(buf)                 ((buf >> 0) & 0x01)

#define PUT_BITS_8(buf,val)             buf |= ((val) << 7) & 0x80
#define PUT_BITS_87(buf,val)            buf |= ((val) << 6) & 0xc0
#define PUT_BITS_876(buf,val)           buf |= ((val) << 5) & 0xe0
#define PUT_BITS_8765(buf,val)          buf |= ((val) << 4) & 0xf0
#define PUT_BITS_87654(buf,val)         buf |= ((val) << 3) & 0xf8
#define PUT_BITS_876543(buf,val)        buf |= ((val) << 2) & 0xfc
#define PUT_BITS_8765432(buf,val)       buf |= ((val) << 1) & 0xfe
#define PUT_BITS_87654321(buf,val)      buf |= ((val) << 0) & 0xff
#define PUT_BITS_7(buf,val)             buf |= ((val) << 6) & 0x40
#define PUT_BITS_76(buf,val)            buf |= ((val) << 5) & 0x60
#define PUT_BITS_765(buf,val)           buf |= ((val) << 4) & 0x70
#define PUT_BITS_7654(buf,val)          buf |= ((val) << 3) & 0x78
#define PUT_BITS_76543(buf,val)         buf |= ((val) << 2) & 0x7c
#define PUT_BITS_765432(buf,val)        buf |= ((val) << 1) & 0x7e
#define PUT_BITS_7654321(buf,val)       buf |= ((val) << 0) & 0x7f
#define PUT_BITS_6(buf,val)             buf |= ((val) << 5) & 0x20
#define PUT_BITS_65(buf,val)            buf |= ((val) << 4) & 0x30
#define PUT_BITS_654(buf,val)           buf |= ((val) << 3) & 0x38
#define PUT_BITS_6543(buf,val)          buf |= ((val) << 2) & 0x3c
#define PUT_BITS_65432(buf,val)         buf |= ((val) << 1) & 0x3e
#define PUT_BITS_654321(buf,val)        buf |= ((val) << 0) & 0x3f
#define PUT_BITS_5(buf,val)             buf |= ((val) << 4) & 0x10
#define PUT_BITS_54(buf,val)            buf |= ((val) << 3) & 0x18
#define PUT_BITS_543(buf,val)           buf |= ((val) << 2) & 0x1c
#define PUT_BITS_5432(buf,val)          buf |= ((val) << 1) & 0x1e
#define PUT_BITS_54321(buf,val)         buf |= ((val) << 0) & 0x1f
#define PUT_BITS_4(buf,val)             buf |= ((val) << 3) & 0x08
#define PUT_BITS_43(buf,val)            buf |= ((val) << 2) & 0x0c
#define PUT_BITS_432(buf,val)           buf |= ((val) << 1) & 0x0e
#define PUT_BITS_4321(buf,val)          buf |= ((val) << 0) & 0x0f
#define PUT_BITS_3(buf,val)             buf |= ((val) << 2) & 0x04
#define PUT_BITS_32(buf,val)            buf |= ((val) << 1) & 0x06
#define PUT_BITS_321(buf,val)           buf |= ((val) << 0) & 0x07
#define PUT_BITS_2(buf,val)             buf |= ((val) << 1) & 0x02
#define PUT_BITS_21(buf,val)            buf |= ((val) << 0) & 0x03
#define PUT_BITS_1(buf,val)             buf |= ((val) << 0) & 0x01

#define CLR_BITS_8(buf)                 buf &= 0x7f

#define SKIP_BYTE(buf,iii)      iii++
#define SKIP_GROUP(buf,iii)     {while (!(buf[iii] & 0x80)) iii++; iii++;}

/*----- type definitions --------------------------------------------------*/

/*----- global functions --------------------------------------------------*/
```

```
/*----- global variables ----------------------------------------------------*/

#endif /* INCL_MISC */
```

E.7.2 src/Types.h

```
#ifndef INCL_TYPES
#define INCL_TYPES

#ifndef NULL
#define NULL                    ((void*)0)
#endif

typedef unsigned char           t_Byte;
typedef unsigned short          t_16Bit;
typedef unsigned int            t_32Bit;

/*
typedef unsigned char           uchar;
typedef unsigned short          ushort;
typedef unsigned int            uint;
*/

#define LOCAL                   static
#define PRIVATE
#define PUBLIC

#endif
```

E.8 Debugging Stuff

Immer wieder nützlich...

E.8.1 src/Dump.h

```
void Dump (unsigned char *buf, int len);
```

E.8.2 src/Dump.c

```
/*
 *      file:           Dump.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Dump.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Dump.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Dump.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- user includefiles ---------------------------------------------------*/

/*----- defines -------------------------------------------------------------*/

/*----- type definitions ----------------------------------------------------*/

/*----- local functions -----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
void Dump (unsigned char *buf, int len)
{
        int                     i;

        while (len > 0) {
                for (i=0; i<16; i++)
                        if (i < len)
                                printf ("%02X ", buf[i]);
                        else
                                printf ("   ");
                printf ("--");
```

```
                for (i=0; i<16; i++)
                        if (i < len)
                                if (buf[i] >= ' ' && buf[i] < 127)
                                        printf ("%c", buf[i]);
                                else
                                        printf ("?");
                printf ("\n");
                len -= 16;
                buf += 16;
        }
} /* Dump */
/*---------------------------------------------------------------------------*/
```

E.8.3 src/Debug.h

```
/*
 *      file:           Debug.h
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Debug.h,v 1.1 1996/09/30 19:34:31 czahl Exp czahl $
 *
 * $Log: Debug.h,v $
 * Revision 1.1  1996/09/30  19:34:31  czahl
 * Initial revision
 *
 */

#ifndef INCL_DEBUG
#define INCL_DEBUG

/*----- standard includefiles -----------------------------------------------*/
#include <stdio.h>

/*----- local includefiles --------------------------------------------------*/

/*----- defines -------------------------------------------------------------*/
#define DFLAG_SCHEDULER         0x0001          /* scheduler infos */
#define DFLAG_CALLID            0x0002          /* callID related */
#define DFLAG_TIMER             0x0008          /* timer handling */
#define DFLAG_Q931              0x0010          /* Q.931 message */
#define DFLAG_Q921              0x0020          /* Q.921 message */
#define DFLAG_Q931_INTERNAL     0x0040          /* internal Q.931 REQ's and IND's */
#define DFLAG_RETURNS           0x1000          /* function returns */
#define DFLAG_FCALLS            0x2000          /* function calls */
#define DFLAG_WARN              0x4000
#define DFLAG_ERR               0x8000
#define DFLAG_ALL               0xffff

#ifdef WITH_DEBUG
#  define DEBUG(xxx)            {xxx}
#  define DEBUGX(mask,xxx)      { if (__debugMask & (mask)) {xxx} }
#  define TRACE_ENTER(xxx)      { DebugLineNo (__LINE__); DebugEnter xxx; }
#  define TRACE_EXIT            { DebugLineNo (__LINE__); DebugExit (); }
#  define TRACE(xxx)            { DebugLineNo (__LINE__); DebugMsg xxx; }
#  define TRACEX(mask,xxx)      { if (__debugMask & (mask)) TRACE (xxx); }
#else
#  define TRACE(xxx)            {}
#  define TRACEX(mask,xxx)      {}
#  define DEBUG(xxx)            {}
#  define DEBUG_ENTER(xxx)      {}
#  define DEBUG_EXIT            {}
#endif

#ifdef ____WITH_DEBUG_OLD
#  define TRACE(xxx) \
        { \
                printf ("%15s.%d: ", __FILE__, __LINE__); \
                xxx; \
                printf ("\n"); \
        }
#  define TRACEX(level,xxx) \
        { \
                if (__debugMask & level) { \
                        printf ("%15s.%d: ", __FILE__, __LINE__); \
                        xxx; \
                        printf ("\n");\
                } \
        }
#  define DEBUG(xxx)            {xxx}
#  define DEBUG_ENTER(xxx) \
        char *__func = xxx; \
        TRACEX (TRACE_FCALLS, printf ("%s enter", __func);)
#  define DEBUG_EXIT            {TRACEX (TRACE_FCALLS, printf ("%s exit", __func);)}
#endif

/*----- type definitions ----------------------------------------------------*/

/*----- global functions ----------------------------------------------------*/
extern  void    DebugInitialize (const char *envVar);
extern  void    DebugLineNo (int lineNo);
extern  void    DebugEnter (const char *fmt, ...);
extern  void    DebugMsg (const char *fmt, ...);
extern  void    DebugExit (void);

/*----- global variables ----------------------------------------------------*/
```

```
extern  unsigned        __debugMask;

#endif /* INCL_DEBUG */
```

E.8.4 src/Debug.c

```
/*
 *      file:           Debug.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: Debug.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $
 *
 * $Log: Debug.c,v $
 * Revision 1.1  1996/09/30  19:34:13  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: Debug.c,v 1.1 1996/09/30 19:34:13 czahl Exp czahl $\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdlib.h>
#include <stdarg.h>
#include <time.h>
#include <sys/time.h>

/*----- user includefiles --------------------------------------------------*/
#include "Debug.h"

/*----- defines ------------------------------------------------------------*/

/*----- type definitions ---------------------------------------------------*/

/*----- local functions ----------------------------------------------------*/

/*----- global functions ---------------------------------------------------*/

/*----- local variables ----------------------------------------------------*/

/*----- global variables ---------------------------------------------------*/
        unsigned                __debugMask;
static  int                     debugLineNo = -1;
static  int                     _my_nest_level = 0;
static  char                    buf[100];
static  int                     n;

/*--------------------------------------------------------------------------*/
void Timestamp ()
{
static  int                     lastSec = 0;
static  int                     lastUsec = 0;
        struct timeval          tv;
        struct tm               *tm;
        int                     delta;

        gettimeofday (&tv, NULL);
        tm = localtime (&tv.tv_sec);
        printf ("%02d:%02d:%02d.%03d", tm->tm_hour, tm->tm_min, tm->tm_sec, tv.tv_usec / 1000);
        if (lastSec == 0) {
                lastSec = tv.tv_sec;
                lastUsec = tv.tv_usec;
        }
        delta = (tv.tv_sec - lastSec) * 1000 + (tv.tv_usec - lastUsec) / 1000;
        printf (" +%2d.%03d ", delta/1000, delta %1000);
        lastSec = tv.tv_sec;
        lastUsec = tv.tv_usec;
}
/*--------------------------------------------------------------------------*/
static void TimeStamp ()
{
static  int                     startSec = 0;
static  int                     lastSec = 0;
static  int                     startUsec = 0;
static  int                     lastUsec = 0;
        struct timeval          tv;
        int                     delta;

        Timestamp ();
        return;
        gettimeofday (&tv, NULL);
        if (startSec == 0) {
                startSec = tv.tv_sec;
                startUsec = tv.tv_usec;
                lastSec = tv.tv_sec;
                lastUsec = tv.tv_usec;
        }
        delta = (tv.tv_sec - startSec) * 1000 + (tv.tv_usec - startUsec) / 1000;
        printf ("%2d.%03d", delta/1000, delta %1000);
        delta = (tv.tv_sec - lastSec) * 1000 + (tv.tv_usec - lastUsec) / 1000;
        printf ("+%2d.%03d ", delta/1000, delta %1000);
        lastSec = tv.tv_sec;
        lastUsec = tv.tv_usec;
} /* TimeStamp */
/*--------------------------------------------------------------------------*/
void DebugInitialize (const char *envVar)
{
```

```
        char                    *v;
        int                     i;

        if (envVar)
                v = getenv (envVar);
        else
                v = getenv ("DEBUG");
        __debugMask = 0;
        if (v) {
                if (v[0] == '0' && v[1] == 'x') {
                        for (i=2; v[i] != '\0'; i++) {
                                __debugMask = __debugMask * 16;
                                if (v[i] >= '0' && v[i] <= '9')
                                        __debugMask += v[i] -'0';
                                else if (v[i] >= 'A' && v[i] <= 'F')
                                        __debugMask += v[i] -'A' +10;
                                else if (v[i] >= 'a' && v[i] <= 'f')
                                        __debugMask += v[i] -'a' +10;
                        }
                } else {
                        __debugMask = atoi (v);
                }
        }
        TRACE (("DEBUG-MASK = x%x", __debugMask))
/*** make gcc happy ***/
        *rcsid = *rcsid;
} /* DebugInitialize */
/*---------------------------------------------------------------------------*/
void DebugLineNo (int lineNo)
{
        debugLineNo = lineNo;
} /* DebugLineNo */
/*---------------------------------------------------------------------------*/
void DebugEnter (const char *fmt, ...)
{
        va_list         va;
        int             i;

        va_start (va, fmt);
        if (!(__debugMask & DFLAG_FCALLS))
                return;
        n = 0;
        for (i=0; i<_my_nest_level-1; i++) {
                buf[n++] = '|';
                buf[n++] = ' ';
        }
        for (; i<_my_nest_level; i++) {
                buf[n++] = '+';
                buf[n++] = '>';
        }
        TimeStamp ();
        vsprintf (buf+n, fmt, va);
        printf ("%s [%3d]\n", buf, debugLineNo);
        _my_nest_level++;
        va_end (va);
} /* DebugEnter */
/*---------------------------------------------------------------------------*/
void DebugMsg (const char *fmt, ...)
{
        va_list         va;
        int             i;

        va_start (va, fmt);
        n = 0;
        for (i=0; i<_my_nest_level-1; i++) {
                buf[n++] = '|';
                buf[n++] = ' ';
        }
        for (; i<_my_nest_level; i++) {
                buf[n++] = '+';
                buf[n++] = '-';
        }
        TimeStamp ();
        vsprintf (buf+n, fmt, va);
        printf ("%s [%3d]\n", buf, debugLineNo);
        va_end (va);
} /* DebugMsg */
/*---------------------------------------------------------------------------*/
void DebugExit ()
{
        int                     i;

        if (!(__debugMask & DFLAG_FCALLS))
                return;
        if ((__debugMask & DFLAG_RETURNS)) {
                n = 0;
                for (i=0; i<_my_nest_level-1; i++) {
                        buf[n++] = '|';
                        buf[n++] = ' ';
                }
                for (; i<_my_nest_level; i++) {
                        buf[n++] = '<';
                        buf[n++] = '<';
                }
                buf[n] = '\0';
                TimeStamp ();
                printf ("%s [%3d]\n", buf, debugLineNo);
        }
        _my_nest_level--;
```

```
} /* DebugExit */
/*------------------------------------------------------------------------*/
```

E.9 Für das Allgemeine

E.9.1 README, eine allgemeine Einführung

```
WELCOME
=======
Welcome to the IsdnLib, a library which implements the Q.931 stack and
gives the user an easer way to communicate with the ISDN card in a SUN.

The very first version of an Q.931 module was written by Bengt Olof Sahlin
(bos@mail.cs.hut.fi). This is a total rewrite of the first version and
looks very different. It was written by (Christian Zahl, zahl@fokus.gmd.de)
as an part of my master-thesis.

TABLE OF CONTENTS:
==================
        1 OVERVIEW OF THE PACKAGE
                1.1 ISDN-LIB INTERFACE
                1.2 HANDLING OF IE's
                1.3 HANDLING OF Q.931 MESSAGES
                1.4 SCHEDULER
                1.5 MISCELLANEOUS
                1.6 DLPI INTERFACE
        2 COMPILING
                2.1 DEBUGGING
        3 RUNNING
        4 FURTHER WORK
                4.1 PROBLEMS / QUESTSIONS
                4.2 TODO
        5 ACKNOWLEDGEMENTS
        6 DOCUMENTATION
        7 AUTHOR, FINAL WORDS
-------------------------------------------------------------------------------

1 OVERVIEW OF THE PACKAGE
=========================
The package is devided into several logical areas:
        o the IsdnLib interface
        o handling of Information Elements (IE's)
        o handling of Q.931 messages
        o the scheduler
        o misc files
        o the DLPI interface to the Q.921 module

The files and their contents is described in 00FILES. Below is a brief
description of the parts mentioned above.

1.1 ISDN-LIB INTERFACE
----------------------
The IsdnLib is a library for using an ISDN interface in a more comfortable
way. Because SUN does not provides an appriate API for it's ISDN card, we
decided to create such a beast.

The main problem is, that because ot the nature of the ISDN, the whole
library must be event driven. So the library implements an scheduler,
which is the core of the whole lib. It is also possible to "overwrite"
the default scheduler for using another one, e.g. to make use of it
in tcl/tk.

Because messages from the ISDN can arrive at any time, there is a way to
"register" your-self. This is done by calling IsdnInitialize(), with a
callback function as parameter. Whenever a new message from the ISDN
arrives, the callback function is called with an argument, a pointer
to the arrived message. The callback function then can handle the message
and can make the right decissions. E.g. it can call other IsdnLib function
to send another message out.

Because there can be several messages for different and new (not yet
established) connection, we need an identifier to decide to which connection
the given message belongs. This is done a "CallId", which will be send
with every message and indication. This CallId can be understood as a
filedescriptor for the connection.

For a detailed description of the IsdnLib interface, refer to the man-page
IsdnLib (3).

1.2 HANDLING OF IE's
--------------------
Because the Q.931 representation of the IE's is very complex and bit oriented,
we have decided to use another representation for the internal handling.
There is one .c and .h file for each supported IE. In the .h file, the struct
for representing the IE in C is defined (e.g. t_Cause). The .c files contains
two functions, one for converting the Q.931 representation into the interal
C struct, named Q931Pop###() (e.g. Q931PopCause), and one for the other direction,
named Q931Push###() (e.g. Q931PushCause).

Currently not all possible IE's are implemented, but it should be very easy
to extend the set of IE's. There is one thing you should keep in mind when
doing so. For an easy and fast conversion, the internal values in the
members of each struct is the value in the Q.931 recommendation, plus 0x100!
```

So every valid value for the internal fields is in the range of 0x100 to 0x1ff. If an filed is missing in the Q.931 message (can happen VERY often), then the value SHALL be set to 0. By doing so, the user can easily decide, if the field was present or not.

File	IE
BearerCap.[ch]	Bearer Copability
CallRef.[ch]	Call Reference Value
CallingPartyNo.[ch]	Calling Party Number
CalledPartyNo.[ch]	Called Party Number
Cause.[ch]	Cause
ChannelId.[ch]	Channel Indication
HighLayerComp.[ch]	High Layer Compatibility
Keypad.[ch]	Keypad
LowLayerComp.[ch]	Low Layer Compatibility
NotifyInd.[ch]	Notification Indicator
ProgressInd.[ch]	Progress Indicator

1.3 HANDLING OF Q.931 MESSAGES

To keep the files small and fine, there is one .c and one .h file for each possible Q.931 message. The .h file defines the struct, which is the internal representation of the message (e.g. t_Q931SetupReq). Unused fields are filled with 0, acording to the rules described above. The .c file implements (in most cases) four function.

Q931###Req() is called by the library, when there is a message to be send to the network (e.g. Q931SetupReq). This function is responsible to handle the internal state machine, according to Q.931.

It then calls the second function, Q931Send###(), which converts the internal message into a valid Q.931 message (e.g. Q931SendReq).

The third function is called Q931xxxInd() and is called whenever this message is received from the network (e.g. Q931SetupInd). It handles the internal Q.931 state machine, handles the CallReferenceValues and the timers.

It then calls the fourth function, xxxInd(), which generates the internal indication and calls the users callback routine.

Files:

File	Q.931 message
Q931Alerting.[ch]	ALERTING
Q931CallProceeding.[ch]	CALL-PROCEEDING
Q931Connect.[ch]	CONNECT
Q931ConnectAck.[ch]	CONNECT-ACKNOWLEDGE
Q931Disconnect.[ch]	DISCONNECT
Q931Information.[ch]	INFORMATION
Q931Notify.[ch]	NOTIFY
Q931Progress.[ch]	PROGRESS
Q931Release.[ch]	RELEASE + RELEASE-COMPLETE
Q931Setup.[ch]	SETUP
Q931SetupAck.[ch]	SETUP-ACKNOWLEDGE

The other Message Types are not implemented yet:

- RESUME
- RESUME ACKNOWLEDGE
- SUSPEND
- SUSPEND ACKNOWLEDGE
- SUSPEND REJECT
- USER INFORMATION
- RESTART
- RESTART ACKNOWLEDGE
- SEGMENT
- CONGESTION CONTROL
- STATUS
- STATUS ENQUIRY

1.4 SCHEDULER

The scheduler is a main-loop, which waits for any incomming events. When an event occurs, the appropiate handler will be called. It is possible to register so called "FileHandler" and "TimerHandler". A FileHandler can be installed for a given fd and an event to occur:

SCHEDULER_READABLE	signals when fd becomes readable
SCHEDULER_WRITABLE	signals when fd becomes writable
SCHEDULER_EXECPTION	signals any other exception

A TimerHandler can be installed for any given timeout value (in ms). When any of the handlers has to signal an event, it calls the installed callback routine for the given fd or timer (identified by a TimerId).

The parameters are the same as for the tcl/tk interface. To make use of another scheduler, e.g. the one from tcl/tk, simply write six function, which calls the right funtions:

- SchedulerInitialize ()
- SchedulerCreateFileHanlder ()
- SchedulerDeleteFilehandler ()
- SchedulerCreateTimerHandler ()
- SchedulerDeleteTimerHandler ()
- SchedulerMainLoop ()

NOTE: to "overwrite" the funtions has not been tested up to now :-O

1.5 MISCELANEOUS

Of course, there are some other impotant files. They are responsable for

```
receiving and scheduling incomming messages to the right functions, handle
all the various Q.931 timers and give some debugging support.

Files:
        Q931Main.[ch]
        Q931Msg.[ch]
        Q931Timer.[ch]
        Debug.[ch]
        Dump.[ch]

1.6 DLPI INTERFACE
------------------
The Q.921 module we are using has been implemented by Bengt Olof Sahlin as
an DLPI interface. He also wrote some routines for easy handling of DLPI
messages, which was simply included in the distribution.

Files:
        Dlpi.[ch]

------------------------------------------------------------------------------
2. COMPILING
============
There is a Makefile.in available, which is used as a source for "make depend".
The Makefile will then be recreated, so don't make any changes in the Makefile
itself! To create the library, do the following:
        o Make any necessarry changes in Makefile.in
        o run "make depend" to create the correct Makefile
        o run "make" to create the IsdnLib

This will result in a library called "src/libisdn.a", which is the IsdnLib.
There are some test programms available in the "test" subdirectory:
        test1   Dumps any Q.931 message received by the IsdnLib through the
                callback routine. It doesn't send any message by itself!


2.1 DEBUGGING
-------------
I have implemented some kind of debugging, which can be totaly turned off
in the Makefile. When "WITH_DEBUG" is defined, debugging support is inclueded,
otherwise not. You can turn on and off various kinds of debugging by setting
the environment variable "ISDN_DEBUG" to a specific value. It's an good idea
to use the haxadecimal representation (e.g. "0x1234"). So, to turn on all
debugging, so the following:
        ksh:    export ISDN_DEBUG=0xffff
        csh:    setenv ISDN_DEBUG 0xffff

To turn off debugging without recompiling:
        ksh:    export ISDN_DEBUG=0
        csh:    setenv ISDN_DEBUG 0

The see which values are available, look into "src/debug.h". All debugging
output is send to stdout.

------------------------------------------------------------------------------
3. RUNNING
==========
        # cd driver
        # make install
        # insertmux open
        # cd ..
        # cd q931/NEW
        # test/test1
        # test/test2

------------------------------------------------------------------------------
4. FURTHER WORK
===============

4.1 PROBLEMS / QUESTIONS
------------------------
o called party no in overlap sending in keypad or calledPartyNo???

4.2 TODO
--------
There is still a lot of stuff that can be done. E.g. implementing the
handling of other messages like AOC[DE]. Also using more than one ISDN
card and using a PRI card can be added. There are also some IE's and
messages not fully implemented.

        o Check if all IE's will be "received" and handled
        o Check, if all the IE's which can occur more than one, are handled
          in the right way

------------------------------------------------------------------------------
5. ACKNOWLEDGEMENTS
===================
Many thanks to Bengt Olof Sahlin, who implemented the first Q.931 module.
He also implements the Q.921 module, which will be used by this package.

------------------------------------------------------------------------------
6. DOCUMENTATION
================
It is planed to describe the whole library and all functions in several
man pages. Because my time is limited, this has not been done up to now.

To get an idea of how to use the library, refer to the file "DOC".

------------------------------------------------------------------------------
7. AUTHOR, FINAL WORDS
```

```
======================
Well, the whole library and all the related files were written by me,
Christian Zahl (zahl@fokus.gmd.de). As also noted, this is one
(unfortunately not the only) part of my master-thesis.

Currently (Oct-96) I will stop with any further work on the IsdnLib. It is
planed that Frank Oertel will use the IsdnLib in his master-thesis in the
first real application. Bengt Olof Sahlin will also use the library for the
ongoing SunShine project in Finnland.

I stoped with any further work, because I have also to make the remaining
part of my master-thesis. Anyway, any bugs found can be send to me:
        zahl@fokus.gmd.de

Please don't expect quick responses, because of the remaining stuff; there
is allways toooo less time :-(
```

E.9.2 DOC, allgemeine Beschreibung

```
0 HELLO
=======
IsdnLib -- a library for using the ISDN device in a SUN

This file gives a overview of the IsdnLib, how it works and how it can be
used. There should be also the man-pages available for all the requests
and indication, but currently they are in an incomplete and old state.
And on the other side, the man-pages don't give you an good explaination
of how the library can be used.

1.0 INTRODUCTION
================
The main idea was to design an interface to the ISDN world, which is much
simpler than using ISDN directly. The first attempt was to write some
funktion, so that the interface is very simmilar to the socket interface.
Because everyone (or not :-) knows the socket interface, it should be
easy to use the library.

The problem was, that it should still be possible to use the library for
_EVERY_ kind of ISDN application. Because of this, I had to make a compromise.
Using some kind of socket interface will not give us the wanted results.
We need an event driven interface, because events can occur on every time,
like in the X-interface.

Secondly, it should be possible to combine tcl/tk with the library. This
results in the interface which currently exists.

2 DESIGN
==========
The core of the library (called IsdnLib) is a scheduler function, which
reveices ISDN and other messages and handles them in a (hopefully) correct
way. The application have to install filehandlers for any input or
output device, it is using. When a given filedescriptor has someting
to signal, the application provided callback function is called. This
callback function then can handle the event, and have to return (relative
quickly, because of the nature of the ISDN). So, let's say that the
callback function shouldn't be active for more than 1s!

The application also installs a callback function for ISDN events. This
function will be called, when (interesting, not all) messages arrives
from the ISDN. The IsdnLib translates the ISDN messages into indications,
which will be signalled via the callback.

When the application wants to make use of ISDN features, it has to call
one of the various requests to issue the right ISDN messages.

Below is a very simple prototype of an application source file, when
using the IsdnLib:
        ...
        #include "IsdnLib.h"
        ...
        void IsdnCb (t_IsdnInd *ind)
        {
                switch (ind) {
                        case ISDN_CALL_IND:
                                printf ("incomming call from...");
                                ...
                        case ISDN_DISC_IND:
                                ...
                        ...
                }
        }
        ControlCb (int mask, void *data)
        {
                /* read from the control fd and handle the request */
        }
        ...
        main ()
        {
                ...
                IsdnInitialize (IsdnCb);
                ...
                IsdnCreateFileHandler (controlFd, ISDN_READABLE, ControlCb, NULL);
                ...
                IsdnMainLopp ();
                IsdnFini ();
                ...
```

```
        }
```

Every application has to follow the prototype above.

```
3. REQUESTS AND INDICATION
==========================
As mentioned above, there are a lot of requests and indications available.
The following is a brief description of them. First let's see how to issue
a call and which indications will arrive.

3.1 IsdnCallReq (t_IsdnCallReq *req)
------------------
Whenever the user wants to issue a call through the ISDN, it calls this
function. The req specifies all the necessarry informations. E.g. the
number to be called can be specified, but because of the nature of the
ISDN don't have to be specified (see IsdnDialReq).

On return, the "callId" field in req was set to the unique callId given
for this (not yet established) connection. This callId has to be used in
all the other requests, and will be used in other indications belonging
to the same connection.

Note, that this requests only issues a call request. The call has NOT been
established when the function returns. Other indications will arrive
via the IsdnCb function (see above).

The IsdnCallReq results in an outgoing SETUP message send to the ISDN.

3.2 IsdnChannelInd
------------------
Normaly, the ISDN responds with an SETUP-ACKNOWLEDGE. When this message
arrives, the IsdnLib calls the IsdnCb function with an IsdnChannelInd
as parameter. The callId in the parameter is set to the same as returned
by the IsdnCallReq().

This indication informs the application that a B-chnannel has been assigend
and opened now. The filedescriptor of the opened fd is included in the
indication (IsdnChannelInd.fd). The application can use this fd to send the
received data to an audio hardware. Normaly, you will hear some kind of
tone from the ISDN network, specially when no number to be dialed has been
given.

Again, the connection has not been established up to now!

3.3 IsdnProgressInd
-------------------
At any time IsdnProgressInd can arrive, for an active call and for those
currently initiated or terminated. This simply gives the application an
idea of the current state of the connection. Specially, the tones available
in most ISDN environments, which are intended to be understood by the human
user, will be described via this indication.

3.4 IsdnDialReq ()
------------------
Analog to the old analog telephne, you have still the chance the dial the
number after an SETUP has been send to the ISDN. This can be done by
using the IsdnDialReq(). The request can be called multiple times with
one ore more "digits" included.

3.4 IsdnAlertInd
----------------
When the number to be dialed is complete, the calling user will be informed.
If he is able and willing to receive the call, he will answer by sending
an ISDN ALERTING message. This message will be send to the caller (to us
in this example). The application is then informed by invoking the IsdnCb
with an IsdnAlertInd as the parameter.

After this indication has been reveiced, no further IsdnDialReq() will be
accepted (according to the Q.931 spec).

3.5 IsdnConnectInd ()
---------------------
When the called user accepts the call and sends an ISDN CONNECT message
(the same as putting the phone off-hook), then the call is connected.
The application is informed by the invokation of IsdnCb with an IsdnConnectInd.
At this point a circuit connection has been established between the
calling and the called user. All B-Channel data will be transmitted in both
directions from now on.

3.6 IsdnReleaseReq ()
---------------------
When the calling user wish to terminate the call, it should issue an
IsdnReleaseReq. The connection will be terminated then and the application
will be informaed of this onve again (IsdnReleaseInd).

Because of the nature of the ISDN, this will be done in a two phase
way. First, (in this case) an ISDN DISCONNECT message will be send, and
the network will answer with an ISDN RELEASE. The IsdnLib then sends an
ISDN RELEASE COMPLETE message and sends a IsdnReleaseInd ().

3.7 IsdnReleaseInd ()
---------------------
When the ISDN network sends a RELEASE message, the IsdnLib will inform the
application by calling the IsdnCb with an IsdnReleaseInd. This informs
the application to release all resources associated with the call indicated
by callId.

After return from the IsdnCb, the IsdnLib also closes any open B-channel for
the given context and released ISDN resources. So NEVER use the B-Channel fd
```

```
after the receiption of an IsdnReleaseInd!

Also note that this indication can occur at any time for any given callId!
This can happen when the ISDN resets, the call is beeing aborted etc.
In any case, after receipt of the IsdnReleaseInd, free ALL resources
for the associated call!

3.8 IsdnDisconnectInd ()
------------------------
Of course, it can also happen that not you, but the other party will terminate
the call. The ISDN then sends an ISDN DISCONNECT message to inform the user.
I decided to keep this message, because additional messages / tone can be
available via the B-Channel.

When the user no longer needs the B-Channel for any messages, it has to call
the IsdnReleaseReq(). If the users doesn't do this, the ISDN will generate
an RELEASE message after some timeout.

The IsdnDisconnectInd can also occur when the (not yet connected) call will
be terminated. So as with the IsdnReleaseInd, you should expect that this
indication can arrive at any time for a given callId!

3.9 IsdnCallReq ()
------------------
After we took a look at the outgoing call, let's see the other direction,
incomming calls. At any time, an IsdnCallInd can arrive. Included in the
indication are all the necessary informations for deciding what to do.
E.g. the called party number is included, so that the application can decide
if it is responsible for this number. Also the bearer capabilities etc.
are included.

If the application cannot handle the call, there are two possiblities.
First, it can simply ignore the indication (an IsdnReleaseInd will be
send after some timeout), or can call the IsdnReleaseReq(). The latter
can be used to inform the ISDN network of the cause, why the application
will not accept the call (e.g. "User is busy"). Depending of the kind of
ISDN interface (singlepoint of multipoint environment), the call will be
rejected by the ISDN.

3.10 IsdnAllertingReq ()
------------------------
When the application want's to signal it's interest in the call, it shall
anser with an IsdnAllertReq() This signals the ISDN network, that there
is someone who is able to accept the call. The calling party then gets
the well-known ringing tone.

Note that you have to issue this request relative fast after the receipt
of the IsdnCallInd. There is a timeout defined by the ISDN network, after
which an unanswered call will be rejected. This timeout is aprox. 4s,
so don't use this time for the "normal ringing procedure of a telephone".


3.11 IsdnConnectReq ()
----------------------
After an incomming call has been signaled, and the user wishes to accept
the call, the application should call this request. This requests the
ISDN network to establish the ciruit switched connection to the calling
user. The connection is not installed at this time, you have to wait
for the IsdnChannelInd() and IsdnConnectInd () (see above).

When the network is not able to create the connection, an IsdnReleaseInd
or IsdnDisconnectInd will be invoked. This can happen e.g. when there is
a collision (calling user gives up, called user takes the phone).

3.12 IsdnKeypadReq ()
---------------------
The ISDN network provides the ability to issue other ISDN related procedures.
These can be invoked via this request. The meaning of the contents of the
keypad IE, and if the network supports this, depends on the provinding
ISDN network. In my test environment I never used this request :-)

3.13 IsdnChargeInd
------------------
Of course, billing is also an interesting information. The ISDN provides
the ability to generate several kinds of AOC informations (AOCD, AOCE, AOCR).
Whenever such an message assives, it is translated into an IsdnCargeInd,
which will be send to the user.

Unfortunately this indication is not implemented at the moment. The problem
is that I haven't understood the ASN.1 encoding of such messages :-(

------------------------------------------------------------------------------
4 SUMMARY
=========
Well I hope that the above description gives an overview of what you can
do with the IsdnLib and how it works. For more informations refer to the
test programms in test/test*.c.
```

E.9.3 HISTORY, Beschreibung des zeitlichen Verlaufes

```
Short overwiev of the history of the IsdnLib

End 1995
        - searched for an API for using ISDN
        - found SunShine
```

```
Mar 1996
        - decided to use SunShine
        - decided to rewrite the Q.931 part of SunShine

1-Oct-1996
        - first alpha version
```

E.9.4 Makefile

```
all:
        (cd src; make all)
        (cd test; make all)
```

E.9.5 config, Konfiguration für make

```
#CC=c89
CC=gcc
CFLAGS=-I$(TOP)/src -g
ISDN_LIB=$(TOP)/src/libisdn.a
```

Literaturverzeichnis

[CJ96] Stephen Casner and Van Jacobson. Compressing ip/udp/rtp headers for low-speed serial links. Internet Draft, Internet Engineering Task Force, November 1996. Work in progress.

[CP97] Yingwei Chen and Kavitha Parthasarathy. Bottleneck bandwidth estimation. Philips research report, Philips Corp., 1997.

[Dee89] S. Deering. Host extensions for IP multicasting. Request for Comments (Standard) STD 5, RFC 1112, Internet Engineering Task Force, August 1989. (Obsoletes RFC0988).

[DEF+96] Steven Deering, Deborah Estrin, Dino Farinacci, Mark Handley Ahmed Helmy, Van Jacobson, Chinggung Liu, Puneet Sharma, David Thaler, and Liming Wei. Protocol independent multicast-sparse mode (PIM-SM): Motivation and architecture. Internet Draft, Internet Engineering Task Force, October 1996. Work in progress.

[DH96] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. Request for Comments (Proposed Standard) RFC 1883, Internet Engineering Task Force, January 1996.

[DPW88] S. Deering, C. Partridge, and D. Waitzman. Distance vector multicast routing protocol. Request for Comments (Experimental) RFC 1075, Internet Engineering Task Force, November 1988.

[Eur] European Telecommunication Standards Institute. Group special mobile. Provisonal Standard GSM 06.10, prI-ETS 300 036, ETSI.

[Han96a] Mark Handley. SAP: Session announcement protocol. Internet Draft, Internet Engineering Task Force, November 1996. Work in progress.

[Han96b] Mark Handley. SDP: Session description protocol. Internet Draft, Internet Engineering Task Force, November 1996. Work in progress.

[Int90] International Telecommunication Union. Data compression procedures for data circuit terminating equipment (DCE) using error correction procedures. Recommendation V.42 bis, Telecommunication Standardization Sector of ITU, January 1990.

[Int92a] International Telecommunication Union. Pulse code modulation (PCM) of voice frequencies. Recommendation G.711, Telecommunication Standardization Sector of ITU, June 1992.

[Int92b] International Telecommunication Union. Support by an ISDN of data terminal equipment with v-series type intercaces with provision for statistival multiplexing. Recommendation V.120, Telecommunication Standardization Sector of ITU, September 1992.

[Int93a] International Telecommunication Union. Basic user-network interface – layer 1 specification. Recommendation I.430, Telecommunication Standardization Sector of ITU, March 1993.

[Int93b] International Telecommunication Union. CCITT specification and description language (SDL). Recommendation Z.100, Telecommunication Standardization Sector of ITU, March 1993.

[Int93c] International Telecommunication Union. Digital subscriber signalling systen No. 1 (DDS 1) - ISDN user-network interface layer 3 - general aspects. Recommendation Q.930, Telecommunication Standardization Sector of ITU, March 1993.

[Int93d] International Telecommunication Union. Digital subscriber signalling systen No. 1 (DDS 1) - ISDN user-network interface layer 3 - specification for basic call control. Recommendation Q.931, Telecommunication Standardization Sector of ITU, March 1993. (also known as ITU-T I.451).

[Int93e] International Telecommunication Union. Digital subscriber signalling systen No. 1 (DDS 1) - generic procedures for the control of ISDN supplementary services. Recommendation Q.932, Telecommunication Standardization Sector of ITU, March 1993. (also known as ITU-T I.452).

[Int93f] International Telecommunication Union. Digital subsriber signalling system No. 1 (DSS 1 – ISDN user-network interface data link layer – general aspects note. Recommendation Q.920, Telecommunication Standardization Sector of ITU, March 1993.

[Int93g] International Telecommunication Union. Error-correcting procedures for DCEs using asynchronous to synchronous conversion. Recommendation V.42, Telecommunication Standardization Sector of ITU, March 1993.

[Int93h] International Telecommunication Union. A family of 2-wire, duplex modems operating at data signalling rates of up to 9600 bit/s for use on the general switched telephone networks and on leased telephone-type circuits. Recommendation V.32, Telecommunication Standardization Sector of ITU, March 1993.

[Int93i] International Telecommunication Union. Intereated services digital networks (ISDNs). Recommendation I.120, Telecommunication Standardization Sector of ITU, March 1993.

[Int93j] International Telecommunication Union. ISDN user-network interface - data link layer specification. Recommendation Q.921, Telecommunication Standardization Sector of ITU, March 1993.

[Int93k] International Telecommunication Union. ISDN user-network interface data link layer - general aspects. Recommendation I.440, Telecommunication Standardization Sector of ITU, March 1993. (see ITU-T Q.920).

[Int93l] International Telecommunication Union. ISDN user-network interface data link layer specification. Recommendation I.441, Telecommunication Standardization Sector of ITU, March 1993. (see ITU-T Q.921).

[Int93m] International Telecommunication Union. ISDN user-network interface layer 3 - general aspects. Recommendation I.450, Telecommunication Standardization Sector of ITU, March 1993. (see ITU-T Q.930).

[Int93n] International Telecommunication Union. ISDN user-network interface layer 3 - specification of basic call control. Recommendation I.451, Telecommunication Standardization Sector of ITU, March 1993. (see ITU-T Q.931).

[Int93o] International Telecommunication Union. Transmission of start-stop characters over synchronous bearer channels. Recommendation V.14, Telecommunication Standardization Sector of ITU, March 1993.

[Int93p] International Telecommunication Union. Usage of cause and location in the digital subscriber signalling system No. 1 and the signalling system No. 7 ISDN user part. Recommendation Q.850, Telecommunication Standardization Sector of ITU, March 1993.

[Int95] International Telecommunication Union. Inegrated services digital network ISDN - stage 3 description for for charging supplementary services using DSS 1. Recommendation Q.956, Telecommunication Standardization Sector of ITU, October 1995.

[Int96] International Organization For Standardization. Information technology telecommunications and informations exchange between systems – high data link control (HDLC) procedure. Standard ISO/IEC 3309:1991 (E), ISO, June 1996.

[Jac90] Van Jacobson. Compression TCP/IP header for low-speed serial links. Request for Comments (Standard) RFC 1144, Internet Engineering Task Force, February 1990.

[Kes94] Srinivasan Keshav. Packet-pair flow control. submitted for publication, 1994.

[McG92] G. McGregor. The PPP internet protocol control protocol (IPCP). Request for Comments (Proposed Standard) RFC 1332, Internet Engineering Task Force, May 1992. (Obsoletes RFC1172).

[Mic94a] Sun Microsystems. *XTL Administrator's Guide*. Sun Microsystems, 2550 Garcia Avenue Mountain View, California 94043-100 U.S.A., a edition, December 1994. URL=http://www.sun.com:80/products-n-solutions/sw/SunXTL/xtl_man.tar.Z.

[Mic94b] Sun Microsystems. *XTL Application Programmer's Guide*. Sun Microsystems, 2550 Garcia Avenue Mountain View, California 94043-100 U.S.A., a edition, December 1994. URL=http://www.sun.com:80/products-n-solutions/sw/SunXTL/xtl_man.tar.Z.

[Mic94c] Sun Microsystems. *XTL Architecture Guide*. Sun Microsystems, 2550 Garcia Avenue Mountain View, California 94043-100 U.S.A., a edition, December 1994. URL=http://www.sun.com:80/products-n-solutions/sw/SunXTL/xtl_man.tar.Z.

[Mic94d] Sun Microsystems. *XTL Provider Programmer's Guide*. Sun Microsystems, 2550 Garcia Avenue Mountain View, California 94043-100 U.S.A., a edition, December 1994. URL=http://www.sun.com:80/products-n-solutions/sw/SunXTL/xtl_man.tar.Z.

[Pos80] J. Postel. User datagram protocol. Request for Comments (Standard) STD 6, RFC 768, Internet Engineering Task Force, August 1980.

[Pos81] J. Postel. Internet protocol. Request for Comments (Standard) RFC 791, Internet Engineering Task Force, September 1981. (Obsoletes RFC0760).

[Rom88] J. Romkey. Nonstandard for transmission of IP datagrams over serial lines: SLIP. Request for Comments (Standard) RFC 1055, Internet Engineering Task Force, June 1988.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. Request for Comments (Proposed Standard) RFC 1889, Internet Engineering Task Force, January 1996.

[Sim94a] W. Simpson. The point-to-point protocol (PPP). Request for Comments (Standard) STD 51, RFC 1661, Internet Engineering Task Force, July 1994. (Obsoletes RFC1548).

[Sim94b] W. Simpson. PPP in HDLC-like framing. Request for Comments (Standard) STD 51, RFC 1662, Internet Engineering Task Force, July 1994. (Obsoletes RFC1549).

[Ste94] W. Richard Stevens. *TCP/IP Illustrated*. Volume 1 edition, 1994.