第 5 部

ISDN データリンク

第 1 章

序論

Ethernet などの通信媒体の発展に伴い、BSD UNIX のようなネットワークオペレーティングシステムが作成された。ネットワークオペレーティングシステムはネットワークを利用して分散した資源を有効に利用することができる。例えば、他の計算機が持つディスクをあたかも自システムにあるように見せかけるネットワークファイルシステムや他の計算機の資源を利用するための仮想端末、一台の計算機でネットワーク上の複数の計算機の管理を行なう集中管理システムなどのアプリケーションがある。ネットワークオペレーティングシステムによりネットワークを用いた分散環境の構築が可能となったが、この様なアプリケーションの多くは帯域幅の広い通信媒体を仮定しており、ローカルな環境での利用に限られることが多い。

一方、高速なデジタル専用回線の開発により広域での通信媒体の転送速度が大幅に向上し、この様なネットワークを利用したアプリケーションを広域で利用できるような研究が進められている。

現在ではデジタル交換網が開発され、回線利用料金や回線設備の数などによる制限を除けば、直接相手と通信することで経路制御や伝送遅延、利用可能な通信帯域などの問題が解決でき、専用線を用いた広域ネットワークより優れたネットワークを構築できる。しかし、現実には回線を利用することで料金の支払いが生じ、ユーザの要求全てを受け入れるだけの設備も揃っていない。そのために、専用線などと併用して利用する必要があり、回線の利用頻度から必要な通信帯域や最大伝送遅延を決定し、経済性から交換網の接続先や利用時間を決定して経路を選択し、経路制御を行なう必要がある。

本研究では広域分散環境を実現するために必要となる処理を抽象的に取り扱うアーキテクチャを設計し、専用線及び交換網を同時に扱うことのできるオペレーティングシステムを構築した。さらに実際に作成したオペレーティングシステムを用いて専用線や交換網を利用したネットワークを構築し問題点を明確にし、広域ネットワークの構築に必要な技術を確立した。

語句説明

計算機の世界で用いられる用語、特にネットワークでの用語には定義が明確でないものが多く、本論の解釈において誤解を招く可能性がある。そのため本文に入る前に、こ

れらの不明瞭な用語についての定義を行なう。

プロトコル

通信を行なうためには相手側が送ったデータを解釈する必要がある。送信するデータの作成方法と受信側での解釈が異なることで、送信されたデータが意味を持たなくなる。そのため両端で通信を行なうデータの解釈方法を事前に決定する必要がある。このデータの作成方法や解釈方法の規定をプロトコルと言う。

ネットワークアーキテクチャ

ネットワークアーキテクチャは物理的なデータの解釈方法からアプリケーションで用いる通信データの解釈方法まですべてのプロトコルを定義している。アプリケーションや通信媒体の違いに対応するために、ネットワークアーキテクチャの規定の方法として、プロトコルを役割に応じて階層的に分割して規定する方法が一般的である。考え方あるいは利用方法の違いからプロトコルの体系化の方法が異なるために、ネットワークアーキテクチャが複数存在する。ネットワークアーキテクチャには Internet で用いている Internet プロトコル、ゼロックスが提唱している XNS[1]や ISO[2]で定義されている OSI[3]の参照モデルに従った OSI プロトコルを用いたものなどがある。

OSI

OSI の参照モデルは物理、データリンク、ネットワーク、トランスポート、セッション、プレゼンテーションとアプリケーションの七層に分割されている。OSI プロトコルは参照モデルに従って各層に分類し、各層の規定に従ってプロトコルを定義している。

XNS

XNS はゼロックスが提唱しているネットワークアーキテクチャであり、ネットワーク層ではシステム間でのデータ通信を保証し、コントロールプロトコルとしてクーリエを規定している。クーリエはリモートプロシージャコールの考え方に基づくものであり、上位のアプリケーションのサービス提供に役立っている。

Internet プロトコル

Internet プロトコル体系では通信を行なう両システム間でのデータ転送するためのネットワーク層プロトコルとして IP[?] を定義し、両プロセス間での通信を行なうためのトランスポート層プロトコルとして TCP [?],UDP[?] などを定義している。アプリケーション層には仮想端末を実現する TELNET[?] やファイル転送を行なう FTP [?] などが規定されている。

---

[1] Xerox Network Systems

[2] International Standardization Organization

[3] Open Systems Interconnection

Internet

Internet プロトコルを用いて構築されているネットワークを Internet と言う。

サブネット

ネットワークでサブネットはネットワークの一部のネットワークを意味する。しかし、本論文では Internet プロトコルにおける IP プロトコルが定義しているサブネットを意味する。

データグラム通信と仮想サーキット通信

データグラム通信はデータグラム、つまりパケット毎に宛先を記述したデータグラムを送信することで通信を行なう方式で、仮想サーキットによる通信は通信を開始する際に相手先と仮想的に接続を行ない、接続後はサーキットに対してデータを書き込むことで通信を行なう方式である。

回線交換網とパケット交換網

交換網にはパケット単位で交換するパケット交換網と回線単位で交換する回線交換網がある。

point-to-point 型とブロードキャスト型

ネットワークを構築する際に必要な通信媒体には point-to-point 型、ブロードキャスト型がある。point-to-point 型は二つのノードを対抗して接続する形式を用いる通信媒体で、ブロードキャスト型はバス、スター、リング型などの通信媒体で一対多の通信が可能な通信媒体である。

DTE と DCE

物理的な接続を行なう場合の網との接点となる装置として DTE 及び DCE がある。DCE (Data Circuit-terminating Equipment) は網に対しての終端となる装置で、DTE (Data Terminal Equipment) は網の終端と接続している端末装置である。

アクセス方式

網内での接続やデータ転送を行なうための制御の手順をアクセス方式と言う。

同期通信と非同期通信

電気的なデータを転送する際に送信するデータに加えデータの位置を示す信号を転送する方式が同期による通信方式で、データのみを送信し受信側で送られたデータを順序に解釈する通信方式を非同期による通信と言う。

大規模性

ネットワークを利用することで分散した資源を有効に利用できる。しかし、資源の数が増加するに従って資源の管理が難しくなる。この数による制限の少ないシステムのことを大規模性のあるシステムと言う。

経路制御とゲートウェイ

ネットワークで複数の通信媒体を用いて接続する場合には、目的地に達するまでの経路が必要がある。この経路を制御する方法としては静的、動的の二つの方法がある。動的な経路制御は経路に関する情報を互いに交換することで実現される。経路上で二つの通信媒体を持ち、データを転送するノードのことをゲートウェイと呼ぶ。

BSD UNIX

BSD[4] UNIX はカリフォルニア大学バークレー分校 (UCB) のスタッフによつて UNIX に改良および機能追加したものである。現在我々が一般に利用しているバージョンは 4.3 BSD UNIX で、最新のバージョンとして 4.3 BSD reno が配布されている。

ネットワークインターフェイス

BSD UNIX では実際に通信を行なう通信媒体を取り扱うソフトウェア (ドライバ) をネットワークインターフェイスと呼ぶ。

WIDE プロジェクト

WIDE プロジェクトは広域に渡る大規模な分散環境を構築するための技術を実証的に確立することを目的として、Internet プロトコルを用いた実験基盤として WIDE Internet を構築している。WIDE Internet は国内に 6 箇所の基点を持ち、基点間を接続するバックボーンネットワークと、基点と各組織間を接続するネットワークから成る。また、国際接続には環太平洋地域での接続を行なっている PACCOM プロジェクトに参加することで米国などとの通信を可能としている。これらのネットワークは専用線を用いて構築されている。

stub

stub は二つのソフトウェアモジュール間に組み込むソフトウェアモジュールで、stub を組み込むことにより新しいインターフェイスを定義することができる。

[4] Berkeley Software Distribution

第 2 章

背景

2.1 BSD UNIX とネットワーク機能

BSD UNIX の主な特徴としては、仮想メモリサポート、高速ファイルシステム、ネットワーク機能、プロセス間通信機能などがあげられる。BSD UNIX のこれらの機能を用いて LAN 上の計算機や周辺機器を統合させた分散環境の構築が可能となる。

BSD UNIX の通信機能

BSD の通信モデル [?] は socket システムコールが通信の端点になっている。

s=socket(domain,type)

“s” は socket システムコールが作成した通信指向の記述子である。socket は UNIX のファイル記述子と同じ概念で扱うことができ、複写や close などのファイル記述子に対する操作が可能である。“domain” 引数は通信で用いるプロトコル体系を指定し、オペレーティングシステムがサポートしているプロトコル体系から選ぶことができる。4.2BSD では UNIX ファイルシステムと XNS と Internet プロトコルの 3 つをサポートしているが、4.3BSD reno 版では ISO,RMP[1]などもサポートされている。

プロトコル体系毎に “type” が定義されており、BSD では基本的な通信方式として

- データグラム用の SOCK_DGRAM
- 仮想サーキット用の SOCK_STREAM

などが定義されている。

サーバ (サービスを行う側) が socket を作成する場合には、そのサーバと通信するために他のプロセスが認識できるアドレス (名前) を与える必要がある。この socket アドレスは実際に socket にメッセージを送る場合や socket 間で接続を確立する場合に使われる。socket にアドレスを与えるのが

bind(s,name)

システムコールである。インターネット領域では、“name” は 32 ビットの IP アドレスと 16 ビットのポート番号などからなる。

[1] HP Remote Maintenance Protocol

簡単なデータグラムベースの通信は socket を作成し、sendto,recvfrom を用いることにより送受信を行なうことができる。

sendto(s,msg,to)

recvfrom(s,msg,from)

“s” は送受信を行なうための socket を示すもので、“to” は送り先の socket のアドレスである。データグラムは “to” で指定される socket に送られ、recvfrom によってデータグラムを受信することができる。データグラムの送信元は “from” 引数に格納される。

最も簡単なサーバ／クライアント関係では、サーバが socket を作り、その socket に対して bind を実行する。このサービスを呼び出したいクライアントでも socket を作成し、サーバに対して sendto を用いてメッセージを送る。サーバは recvfrom でメッセージを受け取り、“from” で得られるアドレスを使って応答を返すことができる。

仮想サーキット方式の通信を用いたサーバでは socket を作成しアドレスを bind した後、

listen(s,backlog)

システムコールを行なう。 listen は “s” で示される socket の待ち行列の数 “backlog” をシステムに対して宣言する。サーバはこの待ち行列から接続要求を取り出してからサービスを行う。待ち行列から接続要求を取り出すには、

s=accept(g,from)

を用いる。accept は接続待ち行列から要求を取り出すために繰り返して利用することができる。

サーバとの接続には、socket を作成しサーバを指定して connect を実行する。

connect(s,to)

“to” はサーバのアドレスである。

このようにして接続が確立したなら、通常の UNIX の read/write システムコールを socket に対して使用することができる。

一般に BSD 上では上記のデータグラム／仮想サーキットの二つの通信方法を用いてプロセス間での通信を行う。この二つのプロセス間通信の使用方法は一例に過ぎず、プロトコル体系やソケットの型を変更することで、高度な通信機能も利用できる。

BSD UNIX での通信機能の実現方法

BSD UNIX[?, ?] のオペレーティングシステム内部では socket システムコールによって作成された記述子に対して操作関数 fileops (Appendix 7 struct fileops 参照) を定義している。ユーザが read/write/close/ioctl などのファイル操作関数を起動することにより、fileops に記述された操作関数が起動される。

ファイル操作関数での操作がプロトコル体系や通信方式などによって異なるために、オペレーティングシステム内部に個々の socket に対する各々の属性を示す構造体 socket(Appendix 7 struct socket 参照) を持っている。この構造体はプロトコル体系と通信方式を示すフィールド so_proto を持ち、so_proto にはプロトコル体系と通信方式によって決められた操作

関数のリスト protosw(Appendix 7 struct protosw 参照) が格納されている。これらの操作関数には

- データの入出力を行なう pr_input,pr_output
- 上下層からの制御情報を伝達、処理する pr_ctlinput,pr_ctloutput
- ユーザからの特殊関数 (bind,listen,accept など) を処理する pr_usrreq
- 各プロトコルの初期化を行なう pr_init
- 定期的 (200ms,500ms) に起動される pr_fasttimo,pr_slowtimo

などがある。これらの操作関数はすべてのプロトコル体系の通信方式で定義されている。例えば、Internet プロトコル体系では通信方式に SOCK_DGRAM, SOCK_STREAM, SOCK_RAW を定義しており、各々の操作関数は以下のように定義されている。

```
{ SOCK_DGRAM,   &inetdomain,    IPPROTO_UDP,    PR_ATOMIC|PR_ADDR,
  udp_input,    0,              udp_ctlinput,   ip_ctloutput,
  udp_usrreq,
  udp_init,     0,              0,              0,
},
{ SOCK_STREAM,  &inetdomain,    IPPROTO_TCP,    PR_CONNREQUIRED|PR_WANTRCVD,
  tcp_input,    0,              tcp_ctlinput,   tcp_ctloutput,
  tcp_usrreq,
  tcp_init,     tcp_fasttimo,   tcp_slowtimo,   tcp_drain,
},
{ SOCK_RAW,     &inetdomain,    IPPROTO_RAW,    PR_ATOMIC|PR_ADDR,
  rip_input,    rip_output,     0,              rip_ctloutput,
  raw_usrreq,
  0,            0,              0,              0,
},
```

この様にネットワークアーキテクチャ毎にプロトコル体系を定義し、各々のプロトコル体系で利用できる通信方式の定義を行なっている。しかし、ネットワークアーキテクチャでのプロトコル階層化はネットワークアーキテクチャ毎に定義され、モジュール化されている。また、起動する関数や引数は静的に定められている。

データの入出力を実際の通信媒体に対して行なうモジュールも、通信媒体の種類やハードウェアの違いから操作関数を定義した構造体 ifnet (Appendix 7 struct ifnet 参照) を持っている。操作関数には

- ネットワークインターフェイスの初期化を行なう if_init
- データ出力を行なう if_output

- ネットワークインターフェイスに依存した特殊な制御を行なう if_ioctl
- バス上などでエラーが生じた際に再初期化を行なう if_reset
- 定期的 (1s) に起動される if_watchdog

が定義されている。また、ネットワークインターフェイスの属性は if_flags で示され、この if_flags には

- ブロードキャスト型のネットワークインターフェイスであることを示す IFF_BROADCAST
- point-to-point 型のネットワークインターフェイスであることを示す IFF_POINTOPOINT
- ネットワークインターフェイスが初期化されたことを示す IFF_UP
- ネットワークインターフェイスが動作可能なことを示す IFF_RUNNING

などが定義されている。

各プロトコル体系で作成されたデータは経路制御関数によって出力可能なネットワークインターフェイスが選択され、ネットワークインターフェイスの操作関数を定義している構造体 ifnet の中の出力関数 if_output に引き渡される。ネットワークインターフェイスがデータを入力した場合は上位に位置するプロトコル体系毎の入力用データバッファキューに保存する。複数のプロトコル体系のデータを同時に通信できる通信媒体ではデータを受信した際にそのデータを保存すべきプロトコル体系を判断することができる。しかし、この様な機能を持たない通信媒体では静的に決められたプロトコル体系の入力キューに保存するか、ネットワークインターフェイスに付けられたアドレス if_addrlist で示されるアドレスのプロトコル体系の入力キューに保存するかどちらかである。

これらのプロトコル操作を効率良く行なうために BSD UNIX では mbuf (Appendix 7 struct mbuf 参照) と呼ばれるデータ連鎖領域を用いている。各プロトコルでデータにヘッダやトレーラを追加する際には、新しい mbuf を確保し必要なプロトコルヘッダやトレーラを mbuf に書き込みデータと連鎖させる。連鎖の方法は構造体 mbuf 内の m_next に後続する mbuf を記述する。また、データからヘッダを削除する際には mbuf 中に格納されているデータのオフセットである m_off をヘッダ分だけ増加させる。データからトレーラを削除する際には mbuf 中に格納されているデータの長さを示す m_len をトレーラ分だけ減らす。この mbuf を用いてプロトコル処理を行ない、先頭の mbuf のアドレスを次のプロトコル操作関数に引き渡すことでプロトコル操作関数間でのデータ授受が行なえる。この様にデータのコピーを行なわずにプロトコル処理が可能なため、パフォーマンスの低下を防ぐことができる。

2.2　交換網

2.2.1　交換網の種類

交換網は大きく回線交換網とパケット交換網の二つの種類に分類できる。回線交換網の通信方式にはデジタル系とアナログ系がある。デジタル系の通信方式を用いることにより高速な通信が可能となり、回線多重化やプロトコル多重化などを実現するプロトコルを用いることで効率良い通信が可能となる。反対に低速なアナログ系の通信方式では簡単なプロトコルを用いて通信速度の低下を防ぐ必要がある。

交換網をアクセスするための手順は網の末端に位置する装置によって異なる。例えば、ISDN 網の場合では末端に位置する装置にはターミナルアダプタや直接計算機本体に組み込むボードなどが考えられる。計算機本体に組み込むボードにおいてはアクセスの手順が計算機やボードを作成したメーカにより異なり、ターミナルアダプタを利用する場合でも複数のアクセス手順 (X.21[?], V.25bis[?], .. など) が存在する。また、パケット交換網でも X.25[?] などのアクセス手順に従い通信を行なうことで仮想的に回線交換網として利用できる。

2.2.2　交換網の利用状況

BSD UNIX において交換網を利用するために開発されてきた機構には dialup-slip と CCITT のプロトコル体系を利用する二つの方法がある。dialup-slip は主にアナログ回線交換網を利用するために、CCITT のプロトコル体系は X.25 を用いたパケット交換網を利用するために開発された機構である。

dialup-slip

slip[?] プロトコルは非同期回線上で通信を行なうためのパケット化を行なうプロトコルである。slip プロトコルを用いることで非同期回線で直接計算機を接続した場合や専用回線でモデムを利用して計算機を接続した場合に互いの計算機間でのデータグラム通信が可能となる。slip プロトコルを用いて回線交換網を利用するために、カリフォルニア大学の Dan Dorough によって開発された dialup-slip やトロント大学の Rayan Zachariassen によって開発された sliplogin などがある。slip プロトコルを回線交換網で利用するためには、直接接続や専用線での接続を行なう場合とは異なり、接続や切断などの回線制御や接続した相手の認識や認証を行なう必要がある。

実現の方法は、まず、解釈する tty ドライバをオペレーティングシステムに追加し、データグラムを slip プロトコルに基づいて送受信するデータを作成する。受け入れ側では相手システムの login 名とパスワードに加え、login 後に起動するプログラムとして通常の tty ドライバから slip プロトコル用の tty ドライバに変更するコマンドを登録しておく。発信側では相手システム名と login 名、パスワード、電話番号を指定して回線接続プログラムを起動する。このプログラムは与えられた電話番号に基づき自動的にモデムの制御

を行ない回線を接続し、与えられた login 名とパスワードを用いて相手システムに login した後に自システムの tty ドライバを slip 用のドライバに切替える。

CCITT プロトコル体系

アクセス手順として X.25 を規定しているパケット交換網は論理的に複数の回線交換網として利用することができる。この論理回線を用いて通信を行なうためにブリティシュコロンビア大学で開発された方法は、新しいプロトコルファミリ CCITT を定義する方法である。CCITT のプロトコル体系でのプロトコル操作関数を示す構造体 protosw を以下に示す。

```
{ SOCK_STREAM, PF_CCITT, CCITTPROTO_X25, PR_CONNREQUIRED|PR_ATOMIC|PR_WANTRCVD,
  0,           0,        0,              pk_ctloutput,
  pk_usrreq,
  pk_init,     0,        pk_timer,       0,
}
```

CCITT のプロトコル体系では SOCK_STREAM のみが定義されており、アドレス体系は X.25 で用いる X.121[?] に基づいて定義されている。

この実現方法の特徴は一つのセッションを一つの論理回線に対応させていることである。一つのセッションは一つの STREAM 型の socket によって作られる。このためソケットの利用時間、送受信データ量を計測することで回線利用に関する情報が得られ、回線利用料金が計算できる。また、socket システムコールを起動したユーザの情報から回線を利用したユーザが分かり、課金対象となるユーザを特定することができる。この様に socket と論理回線を一対一対応させることで課金システムが実現できる。

このシステムをユーザが利用するためにはタイプを CCITT に設定してシステムコール socket を起動し、次に接続先のアドレスに対してシステムコール connect で接続する受け側では CCITT プロトコル体系で socket を起動しシステムコール accept を用いて受信を待つ。この様にシステム上でシステムコールを実行して始めて交換網の利用が可能となるため、他システムのユーザからの利用が不可能となる。

既存の交換網利用方法の問題点

dialup-slip での問題点は通信を開始するために発信側のシステムでコマンドを起動しなければならない。そのため既存のプログラムを用いて回線交換網上で通信を行なうためには事前にコマンドを起動し、利用が終了するとコマンドを用いて回線を切断しなければならない。これは回線制御コマンドを起動させることで、回線制御ができるユーザを限定しているためである。

CCITT のプロトコル体系を利用する方法では、新しく CCITT プロトコル体系を定義し新規に操作関数を定義しているため、既存のアプリケーションに対しての互換性がな

い。そのため既存のアプリケーションに変更を加える必要があり、アプリケーションによっては新しく作成し直す必要がある。

また、課金を実現するためにセッションと物理回線を一対一対応させることで同一相手に対する接続も複数の論理回線を必要とし通信効率が悪い。加えて、課金を行なう対象として自システムのユーザのみに利用者を限定しているためにゲートウェイとしての利用が不可能である。

上記の二つの方式ともに他の交換網を仮定しておらず、異なる交換網を利用するためには全く新しくソフトウェアの開発を行なう必要がある。利用の権限の限定や課金システムの実現を考えているために、回線制御や回線利用が限定され、ネットワーク上のすべてのユーザに対して利用を可能にすることができずゲートウェイとしての利用ができない。

これらの問題点から交換網を利用するため作成する交換網制御機構に対しての要求をまとめると、

1. 既存のアプリケーションに対して互換性を持つ交換網制御機構
2. 異なる種類の交換網を同時に扱うことのできる交換網制御機構
3. 回線を多重化し効率の良い通信を可能とする交換網制御機構
4. 回線を自動的に制御する交換網制御機構
5. 利用に関して制限のない形で実現し、必要に応じて制限を加える交換網制御機構

となり、システムを構築していく上での必要な条件となる。

第 3 章

回線交換網とデータグラム網の相互接続機構の設計

この章では交換網制御機構 (以下システムと呼ぶ) の設計について述べる。まず、3.1節で交換網を含む広域ネットワーク用の通信媒体の扱いに関する要求をまとめ、次に 3.2節においてシステムの全体的設計を行ない、回線交換網とデータグラム網の相互接続機構について述べる。

3.1 設計方針

自由度

回線交換網にはアナログ系とデジタル系の通信方式がある。アナログ通信とデジタル通信では速度や信頼性も異なり、網上で利用するプロトコルが異なる。また、パケット交換網を用いた場合でも X.25 プロトコルなどを利用して仮想的な回線交換網として利用できる。この様に交換網の種類が異なることにより利用するプロトコルが異なることに加えて、網の末端となる装置の種類が異なることで網へのアクセス方式が異なる。既存の交換網のみでも複数のプロトコルとアクセス方法が存在する。将来、より高速なデジタル交換網の開発によって、より多くの組合せが必要になると予想される。これらの組合せを一つ一つを実現すれば、システムが肥大化し、新しいプロトコルやハードウェアの開発による作業量も大きくなる。従って、システム全体に自由度を持たせ、他の組合せの実現が容易となるように設計を行なう必要がある。

大規模性

交換網を利用する目的の一つに不特定多数との接続が挙げられる。しかし、実際に接続作業を行なう際には接続相手毎に計算機資源（メモリや計算能力など）を必要とする。そのため、接続相手の増加に伴うパフォーマンスの低下や、接続相手の数が限界を越えた場合に利用不可能となる問題が生じる。大規模性の対応が充分でないシステムでは数による限界が早期に生じ、交換網の最も大きな特徴である不特定多数との接続が実現できなくなる。従って、分散した計算機資源を可能な限り有効に利用できるような大規模

性の対応が充分できるシステムを実現する必要がある。

セキュリティ

交換網を用いての接続を行なう場合には、セキュリティの確保が必要となる。交換網では接続する相手が複数存在し、特に公衆網では不特定他数の相手との接続が可能である。相手システムが故意に自システムを偽って接続した場合、すべての通信データを不当に得ることが可能となり、また、伝送上の障害により相手システムを誤認した場合などにはネットワーク内での混乱を引き起こすことにもなる。この様に接続している相手の認証を行なわずにデータ通信を開始することは、広域ネットワークでのデータ通信を不安定で危険なものとする。そのため接続時の相手の認証システムは交換網利用においては必要不可欠な機能である。相手の認証を行なう方法で最も確実な方法は、加入している網から接続相手に関する情報を得る方法である。この方法は ISDN 網でサービスされている発信者通知機能にあたる。しかし、一般のアナログ網ではこの様なサービスがないために通信帯域でデータ（パスワード）などの交換を行なって相手システムの認証を行なう必要がある。通信帯域での認証を行なう場合には、ユーザから保護されているオペレーティングシステム間で行なう方が通常のプロセス間で行なうより信頼性が高い。従って、第一に、網あるいは端末装置などのハードウェアからの相手システムに関する情報通知を信頼し、次にオペレーティングシステム間でのデータ通信による認証を信頼するようにし、セキュリティの確保を行なう。

回線が接続されてからの通信路上のデータに関するセキュリティ確保は広域網一般に存在する問題で、これを解決するには送信するデータの暗号化が必要となる。しかし、送信するすべてのデータを暗号化し、受信したすべてデータを解読することによる大幅なパフォーマンス低下が予測されるために、一般的には信頼性が著しく低い場合のみ通信データの暗号化を行なうようにすることが望ましい。従って、必要とした場合のみ暗号化が利用できるような設計を行なう必要がある。

効率

プロトコル操作にはヘッダやトレーラの作成と処理が必要となる。これらの処理は送受信するデータすべてに必要であるため、多少の処理時間の増加もセッション当たりに換算すると大きなオーバヘッドとなる。オーバヘッドを減少させるためにはヘッダやトレーラの処理を高い優先度を持つオペレーティングシステム内部で行なうことが望ましい。

有効利用

デジタル交換網はアナログ交換網に比べて広い帯域幅を提供する。ファイル転送のような大量なデータ転送以外のデータ転送では、一つのセッションにおいて網の帯域すべてを利用するこは殆どない。デジタル交換網のような高速な回線を利用する場合はセッションやプロトコルの多重化を行ない、回線の有効利用をはかるべきである。

安定性

アナログ交換網では網アクセスの際に発生した障害から復帰するためのタイムアウト値は秒単位であるが、デジタル交換網では高速なデータ通信が可能であるためにミリ秒単位でタイムアウト値が設定されている。UNIX のような非リアルタイムオペレーティングシステムでは、ユーザプロセスでミリ秒単位の操作を実現することは非常に困難である。あえてユーザプロセスでの実現を行なった場合には接続に不安定さを生じさせ、回線接続を繰り返すことによりオーバヘッドを生じさせる可能性がある。正確な回線制御を行ない不必要なリトライを減少させる必要がある。

移植性

ネットワークで利用するソフトウェアを設計する際には高い移植性が必要となる。ネットワークを利用し相手システムと通信するためには、相手システムでも同じプロトコルを解釈するソフトウェアが必要となり、全く同じソフトウェアが動作することが望ましい。交換網制御のための通信ハードウェアやプロトコル処理を行なう際のデータ表記などは利用するハードウェアに依存し、利用できるシステム関数やハードウェアの操作関数などは利用するオペレーティングシステムに依存して異なる。ハードウェアやオペレーティングシステムに依存する部分を極力減少させることにより移植性を高くする必要がある。さらに、ハードウェアやオペレーティングシステムに依存する部分については、移植の際に修正が必要な部分を明確にするために独立させ、入れ換えを容易に実現できるように設計する必要がある。

互換性

交換網の利用を可能にするまったく新しいネットワークアーキテクチャを導入した場合には、新たなネットワークアーキテクチャに適したアプリケーションの開発が必要となる。また既存のネットワークアーキテクチャを利用た場合でも交換網を利用するために特殊な手続きを必要であれば、広域ネットワークで利用してきたアプリケーションすべてに変更を加えなければならない。交換網利用のためにこれらの作業を行なうことは現実的で内ため、既存の網の利用と互換性のある手続きにより交換網の利用を可能とする必要がある。

ネットワーク管理

ネットワークを構築し運営して行く上で最も重要なことはネットワーク管理である。ネットワーク管理を行なうには、障害が発生する前に情報を収集しネットワークの状態を予測すること、ネットワーク内で発生した障害を早急に検知し原因を究明し対処することが必要である。現在のネットワーク状況の把握、障害が発生したネットワークの原因究明においても回線の状態を知る必要がある。そのためには、回線上にモニタ装置を設置

し、収集した情報をネットワークに影響なくネットワーク管理者に伝達する必要がある。しかし、実際にすべての回線上にモニタ装置を配置しネットワークに影響なく情報の送受を行なうことは不可能である。簡単かつ有効に回線の状態を知る方法は、データの送受信を行なっている計算機上で送受信を行なったデータをモニタする方法である。回線上に障害が生じて相手側の状態が不明な場合でも、送信したデータと受信したデータから相手の状態や回線の状態を推測することが可能である。また、複数のプロトコル階層を利用している場合では、各層において送受信しているデータをモニタできるならば一層正確な予想が可能になる。従って、ネットワークの状態予測やネットワーク上での障害検知を行いネットワークの管理を実現するために、送受信を行なう計算機上の各プロトコル階層での回線の状態をモニタする機能が必要となる。

3.2　回線交換網とデータグラム網の相互接続

3.2.1　システム設計

前節では広域ネットワークで用いられる通信媒体をサポートするためのシステムが必要とする考え方について述べた。この考え方と 2.2.2項で述べた交換網を利用するためのシステムへの要求を合わせシステム全体の設計行ない、その機能について述べる。

ネットワークインターフェイス stub

OSI や Internet プロトコルなどのネットワークアーキテクチャなど多くのネットワークアーキテクチャは役割毎に分割した階層的な構造を持っている (表 3.1, 表 3.2参照)。一部のネットワークアーキテクチャでは ISDN などの交換網利用手順を規定している。ネットワークアーキテクチャが規定している交換網利用手順に従い、ネットワークアーキテクチャ内の階層構造の一部として交換網制御機構を実現することで交換網利用が可能となる。しかし、ネットワークアーキテクチャにより階層構造の分け方が異なり、交換網利用の考え方も異なる。そのために、ネットワークアーキテクチャに従った交換網利用手順をネットワークアーキテクチャ毎に実現しなければならない。一つの交換網に対してネットワークアーキテクチャ毎にモジュールが存在することはオペレーティングシステム自体を巨大なものにすることになり、複数のネットワークアーキテクチャで同時に交換網を利用するためには複雑な構造を必要とする。さらに、交換網利用を規定していないネットワークアーキテクチャでの交換網利用が不可能となる。

複数のネットワークアーキテクチャでの利用を考えると、交換網を利用するために統一したインターフェイスを定義する必要がある。各ネットワークアーキテクチャを実現するプロトコル体系では、この定義にしたがって交換網を利用するように変更を加えることで実現できる。しかし、交換網を利用するためにプロトコル体系に変更を加えることでネットワークアーキテクチャに矛盾が生じたり、あるいは、既存のアプリケーションに変更を加える必要がある場合もあり、互換性のないシステムとなる可能性がある。ネット

表 3.1: OSI の参照モデル

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Datalink
1	Physical

表 3.2: Internet プロトコル

プロセス	アプリケーション	
システムコール	socket インターフェイス	
オペレーティング	TCP	UDP
	IP	
システム	Ethernet	専用線

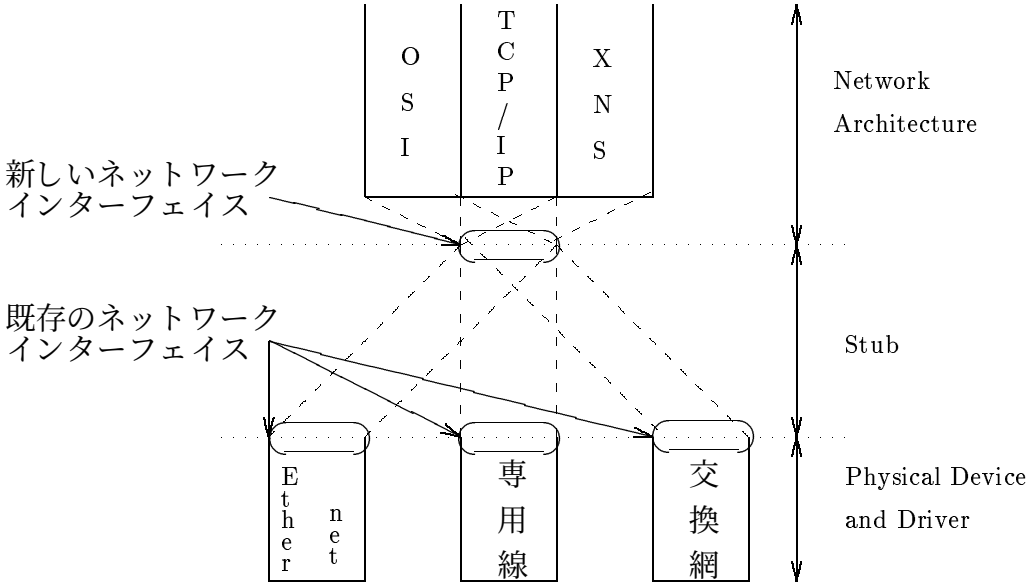


図 3.1: ネットワークインターフェイス stub

ワークアーキテクチャを実現するプロトコル体系に一切の変更を加えず交換網を利用するためには、交換網が既存のネットワークインターフェイスと同等に扱える必要がある。そのために、図 3.1で示すようにネットワークアーキテクチャと既存のネットワークインターフェイスの間に “stub” を定義し、各ネットワークインターフェイスの違いを stub で吸収し、既存のアプリケーションやネットワークアーキテクチャに変更を加えることなく、複数のネットワークアーキテクチャからの利用を可能にする新しいネットワークインターフェイスを定義する。

ネットワークインターフェイス stub 機能

stub の概念を導入することで各々のネットワークインターフェイス独自、また、すべてのネットワークインターフェイスに共通して機能を付加させることができる。しかし、

設計の方針で述べたような設計を行なうために stub の作成にあったて以下の点を厳守しなければならない。

- 動的にプロトコルやハードウェアの入れ換えを可能とする。

  プロトコル規定以外の理由で上下のプロトコルやハードウェアを静的に設定したり、上下のプロトコルやハードウェアの特殊な機能を仮定してはならない。

- ネットワークの管理で重要なネットワークインターフェイスの情報を提供する。

  物理インターフェイスは最低一つ以上のネットワークインターフェイスとし、ネットワークインターフェイスの情報を調べることで、物理インターフェイスの情報を得ることができなければならない。

これらの規定に基づき stub には以下に述べるデータリンク、point-to-point プロトコル及び回線制御機能を持せることとする。

電気的な通信をデータとして解釈するためにデータリンクプロトコルが定義されている。非同期通信と異なり同期通信では通信路上の電気信号の解釈にはビット同期、外部同期などがあり、これらの同期方法やデータ形式が異なることでデータとしての解釈ができなくなる。従って、この様なデータリンクプロトコルを解釈するモジュールが必要となる。

交換網などの回線を効率良く利用するためには論理的に回線を分割する多重化機能が必要となる。多重化には、ネットワークアーキテクチャ毎での回線多重化と複数のネットワークアーキテクチャを同時に利用するためのプロトコル多重化がある。ネットワークアーキテクチャ毎での回線多重化は各ネットワークアーキテクチャが提供する機能であり、ネットワークインターフェイスが提供する機能ではない。そのためプロトコル多重化機能のみを提供する。プロトコル多重化機能の実現の模範となるのが Ethernet[?] である。Ethernet は複数の上位プロトコルのデータが共存できるネットワークインターフェイスである。Ethernet 上のデータは表 3.3で示される形を取り、上位に位置するプロトコルの区別を可能としている。実際に Ethernet は OSI、TCP/IP や XNS でネットワークインターフェイスとして利用されている。Ethernet と同様の機能を提供することにより、一つの交換網を複数の上位プロトコルでの利用が可能となる。専用線などで利用されてきた point-to-point プロトコルの多くもまた送信するデータに上位のプロトコルを示すプロトコルフィールドを持つ。この様な point-to-point プロトコルを利用することにより一つの交換網を複数のネットワークアーキテクチャで利用することが可能となる。

交換網を利用するためには、まず交換網のアクセス手順を実現するモジュールが必要となる。さらにデータグラム転送を行なうネットワークアーキテクチャで回線交換網や論理的な接続を規定しているパケット交換網などを利用するためには、何らかの方法を用いて回線を自動的に制御する必要がある。従って、これらの交換網を利用するにあたって必要な網アクセス手順や自動回線制御モジュールが必要となる。

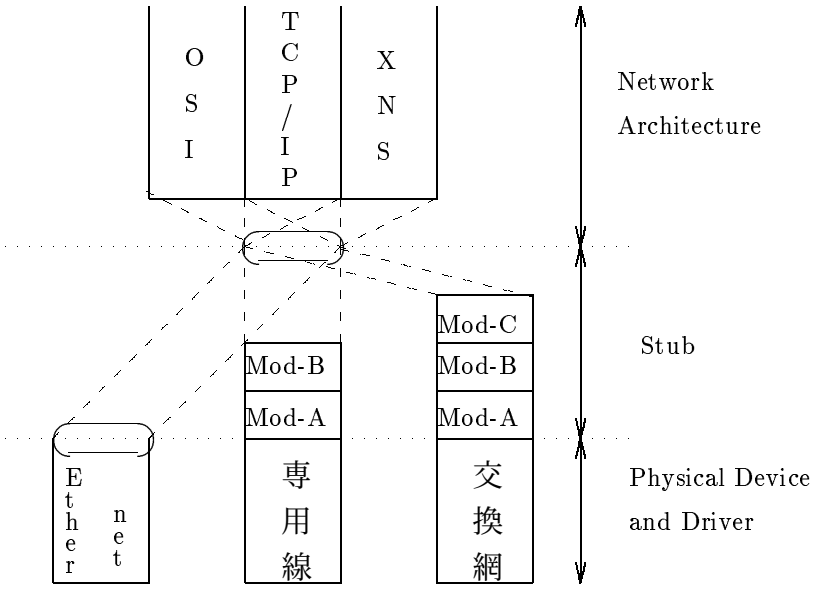
表 3.3: Ethernet データフォーマット

	0	1	2	3	4	5	6	7
0 Byte	送信先アドレス							
1 Byte								
2 Byte								
3 Byte								
4 Byte								
5 Byte								
6 Byte	送信元アドレス							
7 Byte								
8 Byte								
9 Byte								
10 Byte								
11 Byte								
12 Byte	上位プロトコル タイプ							
13 Byte								

モジュール階層

複数のプロトコルやハードウェアのサポートを容易とす自由度を持ったシステムとし、高い移植性を保持するためにハードウェアやオペレーティングシステムに依存した部分が独立したシステムとするため、stub を役割毎にモジュールに分割し、モジュールの組合せにより実現する方法を用いる (図 3.2参照)。

モジュールを組み合わせる方法には静的、動的に組合せを行なう二種類の方法がある。回線が固定されている場合には通信プロトコルは回線速度や信頼性などによりほぼ一意的に決定できるため、静的な組合せ方法によって実現できる。しかし、接続する通信ポート、通信機器や回線などはコネクタ等の差し替えによって容易に変更され得るため、動的な組合せを必要とする。また、ネットワーク管理のためのモニタリング機能なども必要に応じてモジュール間に組み込むことが可能でなければならない。そのため、役割毎に統一されたインターフェイスを持つモジュールに分割し、モジュールを動的に組み合わせることで交換網の制御と通信を可能にする。動的なモジュールの組合せを実現するためには、モジュール間のインターフェイスが統一されており、透過性のあるモジュールである必要がある。もし、モジュール間インターフェイスが統一されていないシステムで動的な組合せを実現すれば、他のモジュールの動作が保証されず完全なシステムを作成することができない。



Mod-A: データリンクプロトコルモジュール
Mod-B: point-to-point プロトコルモジュール
Mod-C: 回線制御モジュール

図 3.2: stub でのモジュール階層

3.2.2 階層化モジュール

モジュール内関数

モジュールの動的な組み合わせを可能とするネットワークインターフェイスを正常に動作させるためには、モジュールに対してある手続きを行った場合に確実に必要な機能が動作することを保証しなければならない。そのためには、モジュール間のインターフェイスを統一し、手続きに基づく動作機能の定義を行なう必要がある。手続きの方法については次項で述べ、この項ではどの様な機能が必要であるかを述べる。

BSD UNIX はネットワークインターフェイス操作関数として、2.1節で述べたように if_init, if_output, if_ioctl, if_reset, if_watchdog を定義している。モジュールを重ね合わせて利用する際には出力データ処理だけではなく入力データ処理の関数も必要である。モジュールを重ね合わせることを仮定していない BSD UNIX では、入力されたすべてのデータは上位に位置する各々のネットワークアーキテクチャのネットワークプロトコルを解釈するモジュール用のデータキューに格納する。しかし、モジュールを重ね合せる場合には、再上位のモジュール以外は上位に位置するネットワークインターフェイスの入力関数にデータを引き渡す必要がある。そのために、上記のネットワークインターフェイス操作関数群に対してデータを入力する関数 if_input を追加する。この入力関数は必要な処理を行なったデータを上位にネットワークインターフェイスが存在する場合には上位のネットワークインターフェイスの入力関数に引き渡し、存在しない場合にはデータに応じたネットワークアーキテクチャのネットワークプロトコルモジュールの入力用のデータキューに格納する関数と定義する。

動的にモジュールの入れ換えを行なうには、現在利用しているモジュールの機能を停止し新しいモジュールと交換し、交換が終了した後に新しいモジュールの初期化を行なって入れ換え作業が終了する。従って、モジュールの初期化とともに終了処理が必要となる。モジュールの終了処理を行なう関数 if_halt を追加する。

現在まで我々が用いてきた Ethernet や専用線は常時回線が接続されているために初期化が終了すればデータ転送が可能である。しかし、交換網の場合は初期化が終了した時点では回線接続が可能な状態であるのみで、実際にデータ通信が行なうことができるわけではない。従って、交換網を利用するためには回線制御関数を定義する必要がある。この回線制御関数の詳細に付いては、3.2.3項で述べる。

モジュールの重ね合わせ方法

モジュール化の方法には stream 型、デバイスドライバ型がある。stream 型とデバイスドライバ型の大きな違いはモジュールへのアクセスの入口の数である。stream 型はモジュールへのアクセスの入口を一つに絞り内部の構造をブラックボックス化する。一般にはブラックボックス化することにより内部での構造を知ることなく利用できるために、利用者が簡単に利用できるようになる。これに対してデバイスドライバ型は、最小限必要な機能を定義し必要の機能分のアクセスの入口を利用する。そのため、各機能がどの

様な役割をするかある程度を知っている必要がある。

データ入出力でエラーが発生した場合などは強制的に回線を切断する必要がある。この際にデータ入出力キュー内のデータを処理するためにデータ入出力を続けると繰り返しエラーが発生するため、データ入出力より優先して回線制御を行なう必要がある。しかし、正常に回線を切断する場合は、データ入出力キュー内のデータをすべて処理してから回線制御を行なわなければならない。この様に特殊な場合のみ入出力関数より回線制御関数を優先させるには、stream 型ではすべての要求に優先度を持たせ、モジュール内部で処理を行なう際に必ず後続する要求の中で優先度の高いものがないことを確認して処理を継続する必要がある。デバイスドライバ型では二つの関数が独立しているために二つの関数間でのみ同期を取ることで優先度を持たせることができる。

操作性において互いの優劣を考えた場合、モジュールは一般のユーザが作成するものではないために操作性が極めて簡単である必要はない。また、ブラックボックス化された内部での動作が予想と異なることによる誤動作を招く可能性もある。処理の順序操作を行なう面から考えると、簡単な同期だけで実現できるデバイスドライバ型の方がネットワークインターフェイスをモジュール化して階層的に組み合わす場合には適していると判断できる。従って、モジュールの重ね合わせを実現するためには関数間で同期を取る デバイスドライバ型を用いる。この際には関数間で同期を取る方法が問題となる。

3.2.3　回線制御機構

オペレーティングシステム内部で役割毎にモジュールに分割し、各々のモジュールの役割と作成方法を決定した。ここで始めて回線交換網とデータグラム網の相互接続が可能となる。しかし、我々が用いていた既存のネットワークインターフェイス Ethernet や専用線には回線の接続切断などの回線制御の概念がない。従って、回線交換網とデータグラム網の相互接続を行なう前に、ネットワークインターフェイスに回線制御機構を加える必要がある。この項では回線制御機構について述べ、前項で述べたデバイスドライバ型の問題点である関数間同期の機構について述べる。

モジュール内回線制御機構

回線交換網を利用するに当たって回線制御を行なう必要がある。しかし、前述したように役割毎にモジュールに分割し、モジュールを重ね合わすことで交換網を利用するため様々な組合せが考えられる。例えば、重ね合わせるモジュールとしては、プロトコル多重化を行なう point-to-point プロトコルやデータリンクレベルでの制御を行なうデータリンクプロトコルなどのモジュールなどが挙げられる。これらの階層的に重ね合わせる point-to-point プロトコルやデータリンクプロトコルなどにもコネクション型の接続を行なうプロトコルが多く存在する。そのため、物理的な回線の制御に加え、各プロトコルモジュールでも論理的に回線制御を行なう必要がある。しかし、回線交換網が接続されていない状態で上位に位置するプロトコル間で接続作業を開始しても無意味である。接続作業を行なう順序は最下層からで、切断の処理を行なうのは上位層から行なわなけれ

ばならない。このため、各層での接続や切断の処理が終了したことを他の層に伝達する必要がある。伝達するためにはどの層が回線制御を必要とするかを知っている必要がある。動的なモジュール階層を考慮すると、伝達する度にモジュールが回線制御機能を有しているかどうかを調べ伝達しなければならない。接続切断の処理終了を伝達する度に回線制御機構の有無を調べることによるオーバヘッドを回避するため、各層で回線制御機能を持たせ、回線制御機能を必要としないモジュールでは単に上位や下位に対して処理の終了を伝達するように設定しておく。回線制御の要求、処理や終了の伝達方法の詳細を次項で述べる。

回線制御の要求と結果の伝達方法

接続を要求したシステムでは発信を、要求されたシステムでは着信を行なう。切断を要求したシステムでは切断に必要な手続きを行ない、接続先のシステムでは切断を検知し必要な処理を行なう。自システムから接続を要求する場合は上位層から接続の準備を行ない、下位層に対して接続を要求し、下位層が接続を確立した後に接続作業を行なう。また、自システムから切断を要求する場合は上位層から切断処理を開始し、下位層に対して切断を依頼する。下位層の切断作業が終了した後に必要となる終了処理を行ない、切断が完了する。相手システムから接続や切断の要求があった場合は下位層から必要な作業を行ない、その結果を上位層に伝達していく。この様に自システムから要求を出す場合と相手システムから要求があった場合では必要となる手続きが異なる。そのため要求を行なう場合と要求を受け入れる場合の二つに分けて考える。

接続や切断を完了するには下位層での処理の終了を待つ必要がある。この間オペレーティングシステムがブロック状態になり、非効率的である。従って、要求のみを伝達し、接続や切断の作業が完全に終了するのを待たずに制御を一旦返すことでブロック状態を防ぎ、下位層からの接続や切断の成功又は失敗の通知を受けることで処理を再開する。そのため、下位層においては接続や切断の要求があった場合は、必ずその結果を上位に通知する (図 3.3 (a),(b) 参照)。

相手側から接続や切断の要求が発生した時は、接続を開始する際に必要な作業を行ない、相手との接続を開始する。互いの条件が一致し、接続や切断の作業が完了した際に接続や切断を上位に通知する（図 3.4 (a) 参照)。相手からの接続要求を拒否する場合には、下位層に対して切断の要求を行なうことで実現する (図 3.4 (b) 参照)。

自システムから接続や切断の要求を行なう際には、最下層から必要な手続きを終えてその結果を上位に伝達する。相手システムから接続や切断の要求があった場合には、最下層から必要な手続きを行ない、その結果が上位に伝達される。下層からの接続や切断の処理終了の通知により、自システムの要求か相手システムからの要求かは、事前に上位層からの要求が発せられたかどうかで区別できる。この区別を確実に行なうためには、上位からの要求が生じた場合は成功や失敗に関わらず、必ず上位に対して結果を通知する必要がある。

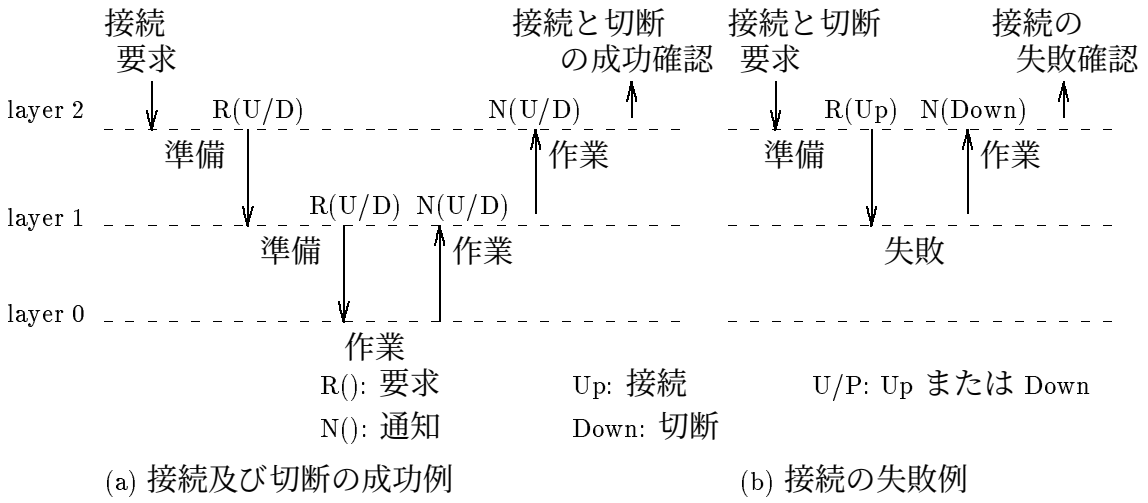


図 3.3: 自システムからの回線接続や切断要求

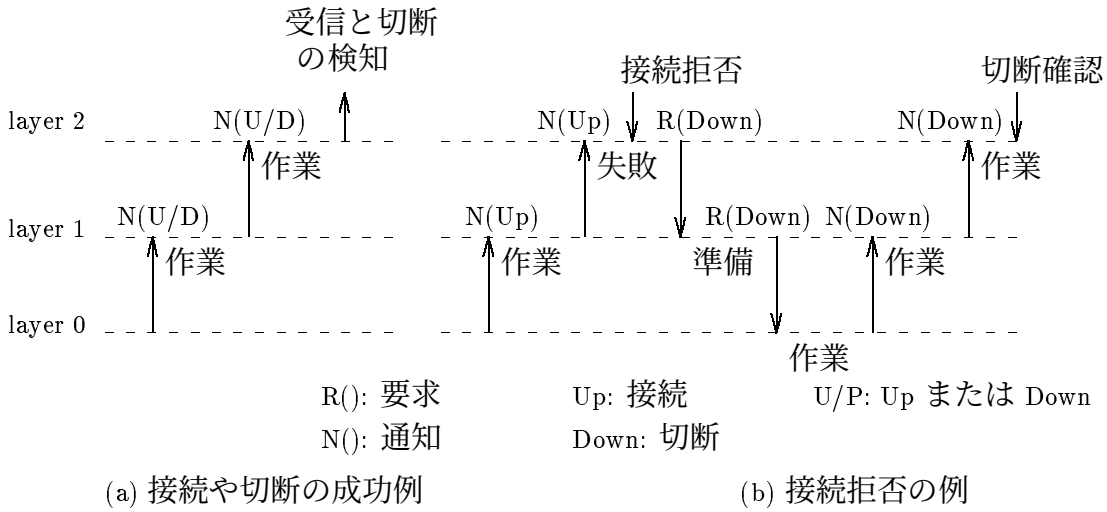


図 3.4: 相手側からの回線接続や切断要求

入出力関数との同期

回線制御関数とデータ入出力関数は非同期に動作する。非同期に動作することで、既に送出したデータのキャンセルや緊急回線切断などが実現できる。しかし、その反面回線が非接続の状態においてデータの入出力操作が行なわれる可能性などがある。回線制御と入出力関数が同時に動作して問題が生じる場合は、制御を要求し制御が完了するまでの間にデータ入出力関数が起動された時のみである。この問題を解決するには、各モジュールに状態変数を持たせ、状態が推移している間はデータ入出力を行なわずにデータを保存し、回線接続後に送出する。回線接続が失敗した場合は保存したデータはすべて破棄する。また、接続が確立さている間のみデータの入力作業を行ない、非接続の際にデータ入力があった場合にはそのデータを破棄する。

初期化、終了処理関数などと制御関数の同期は初期化処理が実行され終了処理が実行されていない時に限り制御関数が動作するものとする。

3.2.4 自動回線制御機構

自動回線制御機構は自動的に回線の接続や切断などの回線制御を行なう機構である。回線制御機能をモジュールに持たせたことで自動回線制御機構を一つのモジュールとして実現できる。自動回線制御モジュールでは回線の接続や切断の指示を与えるだけである。この節ではどの様な機構で回線制御を指示するかについて述べる。

自動回線制御モジュール

上位で利用するネットワークアーキテクチャがコネクション型の通信を行なう場合は、上位でのコネクションであるセッションと物理回線を一対一対応させることで回線の自動制御を行なうことができる。上位でのコネクションと物理回線を同期させセッション確立作業を行なう前に回線の接続を自動的に行ない、セッションの切断作業が終了したら回線を自動的に切断する。この方法での問題点は、同じ計算機に対して同時に二つのセッションを確立する場合においても二回線必要となり、同一の計算機に対して複数のセッションを利用することが不可能となることである。この問題を解決するためには同一相手に対する複数のセッションを一つの物理回線を対応させて利用する必要がある。しかし、この方法においても問題点がある。セッションを確立するのはトランスポート層で、相手計算機にデータを転送する機能はネットワーク内での経路制御を行なうネットワーク層である。他の計算機から交換網を持つ計算機を経由して交換網を利用する場合には、トランスポート層でのコネクションを上位に位置するネットワーク層で検知し回線制御を行なう必要がある。これは階層的な管理を行なうネットワークアーキテクチャの考え方に反するもので、この実現を行なうためにはネットワークアーキテクチャの考え方をまったく変える必要がある。従って、セッションと物理回線の一対一対応を用いた場合は自システムのみからの利用のみ可能である。

上位で利用するネットワークアーキテクチャがデータグラム転送に基づいている場合

はデータグラム転送に回線の概念がないために、何らかの方法を用いて “データグラム網と回線交換網を相互に接続する” 必要がある。データグラム転送はデータ毎に相手を指定して転送している。そこで、同一相手へのデータグラムをまとめて仮想的に一つのコネクションを確立し、仮想的なコネクションと物理的な回線を同期させる方法がある。仮想的なコネクションが確立している間は、同一相手へのデータを転送する。しかし、最初にコネクションを確立する際と、最後にコネクションを切断する際のタイミングと方法が問題となる。

上位で利用するネットワークアーキテクチャがコネクション型の通信を行なう場合においても上位のコネクションにこだわらず、一つ一つのデータグラムによる通信と考えることができる。そのため、上位で利用するネットワークアーキテクチャがデータグラム転送に基づく場合に対応することで、コネクション型の通信を行なうネットワークアーキテクチャにおいても同じ機構が利用可能である。従って、データグラムを集めた仮想的コネクションの確立と切断のタイミングを決定して、仮想的なコネクションと物理回線を同期させて自動回線制御を行なう。

回線の接続に関してはネットワークインターフェイスにデータが出力された時点で回線の接続を開始する。これは、交換網を利用する権限があるデータのみがネットワークインターフェイスに対して出力されると仮定することができるためである。送信するデータのための出力ネットワークインターフェイス (交換網あるいは他の通信媒体) は経路制御機構によってが選択される。

データグラム転送は一つのパケットによって構成されているため、通信の終了を検知することは不可能である。そのため、通信の終了を検知するのではなく上位層で推測し回線の自動切断を行なう。推測を行なうための唯一の情報となるのは回線の利用状況である。回線の利用状況を表すためには、送信されているデータと利用されている時間の二つの要因があり、回線の利用状況の判断は単純ではない。最も確実な方法は送信されているデータを元に回線利用率が低下した時を検知し、その後、利用されていない時間から通信の終了を推測する方法である。この方法での問題点は、通信を行なっている両端の計算機の距離が遠い時にデータ送信の間隔にばらつきが生じ、送信されているデータからの判断が難しくなることである。特に、TCP プロトコルが用いている Congestion Control[?] などにより、データ送信間隔を故意に制御している場合などには、通信の終了の推測が非常に難しくなる。送信されているデータから判断を行なうためには複雑な解析を必要とし、簡単な解析方法を用いた場合は誤算となる可能性が高い。複雑な解析方法を用いることによるパフォーマンスの低下を避けるために、利用されている時間のみで通信終了を推測する簡単な方法を用いる。利用時間から通信終了を推測するために、ある一定間隔以上転送するデータがない場合に通信終了と見なしコネクション（回線）の切断を行なう方法をとる。時間間隔の測定方法としては、データ転送を行なう度にタイマをリセットし、タイムアウトが生じた際に回線の切断を行なう方法が考えられる。この方法では、転送されるデータの数が多い場合はタイマの設定、解除の回数が多くなりオーバヘッドとなる。従って、ここでは一定間隔で回線接続からの入出力データ総数を測定し、入出力データ数に変化がなくなった時点で通信が終了したと判断し、コネクショ

ン（回線）を切断する方法を用いる。この方法を用いることで、オーバヘッドを減少させることができる。

自動回線制御モジュールの配置場所

実際にネットワーク上で障害が生じた場合、その障害がどこで、なぜ起きたかを知ることは非常に難しい。利用者にとっては何が原因で障害が起きたかより、どこで障害が発生したかが重要である。利用している回線上で障害の発生したことを知る方法として折り返し検査を用いることが多い。折り返し検査とは、相手に対して適当なデータを送信し、データを受信した相手は送信されたデータを送信元に対して送り返す。送信側では、送り返されたデータのエラー率や戻ってくるまでの時間などから、利用している回線に障害が生じたことを推測する方法である。

折り返し検査は障害が発生したと推測される時に行なう方法と、一定間隔で行なう方法がある。前者の方法を用いれば不必要なデータ転送を行なわないためにオーバヘッドが少ないが、送信するデータがない場合には障害を推測することが不可能となる。後者の方法では、不必要なデータ送信を行なうためのオーバヘッドはあるものの、一定間隔で障害を検知することが保証されている。折り返し検査を行なう多くのプロトコルは、後者の一定間隔での折り返し検査を用いている。

一定間隔で折り返し検査を行なうことで定期的にデータのやり取りが発生し、前述した自動切断のアルゴリズムでは入出力総数の測定時間によって、回線が切断されなくなったり、切断後すぐに接続を繰り返すことになる。しかし、現在では折り返し検査による障害の推測方法が最も有効であるために、交換網上での障害検知方法として折り返し検査を用いないわけにはいかない。

折り返し検査を用いる理由は、どこで何が起きたかを知る方法がない場合に用いる方法である。ハードウェアが正常に作動している場合にソフトウェアで障害が起きた際には障害を知る方法はある。しかし、ハードウェアで障害が生じた場合は、障害の原因や状態をソフトウェアで知ることは不可能である。従って、主にハードウェアで障害が生じた際に折り返し検査が必要となる。通信で用いるハードウェアは、通信回線と回線端末装置である。これらのハードウェアに関する規定を行なうのは物理、データリンク層であるために、この二つの層での折り返し検査が最も重要な意味を持つ。この折り返し検査を行なう物理、データリンク層より上位に自動回線制御モジュールを位置させることで、自動回線モジュールでは折り返し検査のデータを計測せず、回線の切断を行なうことができる。自動回線制御モジュールをデータリンクプロトコルや point-to-point プロトコルの上位に位置させるために、ネットワークインターフェイス stub の最上位に位置させる。

3.2.5 アドレス変換

データグラムの宛先から自動的に発信を行なうには、利用する網でのアドレスが必要となる。また、着信の際に発信者通知機能を用いて認証を行なうには、網のアドレスか

らネットワークインターフェイスにつけるアドレスに変換する必要がある。

Ethernet では、"Ethernet Address Resolution Protocol"[?](以下では ARP と略す) プロトコルを用いて上位のネットワークアーキテクチャのアドレスと Ethernet アドレスを関連づけている。ARP は自分の Ethernet アドレスと上位のネットワークアーキテクチャアドレスを自主的に申告することで二つのアドレスの関連を他の計算機に伝える。また、相手のアドレスが不明な場合は要求を送出し答を得る。交換網では ARP のような方式を取るわけにはいかない。なぜなら、Ethernet のような閉じた世界とは異なり、交換網、特に公衆交換網では不特定他数の相手が対象となり、自主的な申告を信用することはできない。また、Ethernet はブロードキャスト型ネットワークであるのでネットワーク上のすべての計算機に対してブロードキャストを用いて要求を出すことが可能であるが、交換網、特に回線交換網ではブロードキャストやすべての計算機に要求を出すことは不可能である。そのために、ARP とは異なったアドレス変換方式を用いる必要がある。利用するアドレス変換方式においては、

- 変換したアドレスが十分信用できること。
- 問い合わせを行なう場合は事前に問い合わせ先が明らかであること。

を満たしている必要がある。

加えて、データグラムの宛先やネットワークインターフェイスに付けるアドレスは、ネットワークアーキテクチャ毎に異なる。例えば、TCP/IP では IP アドレス、XNS では Ethernet アドレスとなる。網でのアドレスは、ISDN 網では I.331 [?] で、アナログ網では E.163[?] により網内でのアドレスフォーマット（電話番号）が決まっている。変換する両アドレスともに複数のアドレス体系を扱える必要がある。また、複数のプロトコルで同一の回線を利用するためには、異なるネットワークアーキテクチャでのアドレスが同一であるかを調べなければならない。例えば、最初に Internet プロトコル体系で利用し次に XNS プロトコル体系で利用する場合は、XNS プロトコル体系での相手アドレスが最初に回線を接続した Internet プロトコル体系での相手アドレスと同じであることを確認する必要があり、プロトコル体系間でのアドレス変換機能が必要になる。従って、ネットワークアーキテクチャでのアドレスと網でのアドレスのような上下関係を持ったアドレス体系の変換だけではなく、一般的な二つのアドレス体系のアドレスを変換する機構を確立する必要がある。

交換網を利用する場合、特に、公衆交換網の場合は回線利用に対して料金を支払わなければならない。支払いの責任が生じることにより、他のユーザのデータのために料金を支払う可能性がある。そのために、自動着信のみを行ない自動発信を行なわない機能が必要となる。自動発信を行なうか行なわないかの選択機能は回線単位ではなく、通信相手単位の設定ができる必要があるために、これを相手アドレスの属性として持つ必要がある。

しかし、前述したようにこのアドレス変換機能で扱う二つのアドレス体系にはネットワークアーキテクチャでのアドレスと網でのアドレスの区別がなく、一般的な二つのアドレス体系として定義した。そのために、着信や発信などの特殊な属性を持たせること

ができない。着信や発信などの属性を持たせるためには、着信や発信を二つのアドレスの変換方向として定義しなければならない。そこで、着信を行なうには設定する二つのアドレスの網でのアドレスからネットワークアーキテクチャでのアドレスに変換することを許可し、発信を行なうにはネットワークアーキテクチャのアドレスから網でのアドレスへの変換を許可するように設定する。

アドレス変換を実現する方法には、事前に情報をテーブルに設定しておきテーブル内を検索する方法と情報を持つサーバに対して問い合わせを行ない変換を行なう方法の二種類がある。前者の方法では高速にかつ必ず結果が得られる。しかし、網に加入しているすべてのアドレスに関しての情報を一つのシステムで管理することは不可能であり、大規模性を失うことになる。後者の方法を用いると情報をネットワーク上のシステムに分散させることですべての情報を得ることができ、大規模性を持たせることができる。しかし、ネットワークを利用して結果を得るために、結果を得るまでの時間がかかり、結果を得られない可能性もある。大規模性を持たせた上でできるだけ高速にアドレス変換を行なうために、二つの方法を合わせて利用する必要がある。すなわち、最初にテーブルを検索し、結果が得られない場合にのみサーバに問い合わせを行なう。ここで問題となるのが問い合わせを行なっている間、システムがブロック状態になることである。ブロックさせないためにはまず、情報の検索に成功した場合と失敗した場合に起動する関数を指定して検索を依頼する。テーブル内部での情報検索が成功した場合はすぐに結果を返す (図 3.5パターン A 参照)。テーブル内部に必要な情報がない時は問い合わせ要求を出し、一旦制御を元の関数に返す。この際に結果が得られなかった場合を考慮してタイマを設定しておく。結果が得られた時点で、検索依頼の際に指定された検索成功の場合の関数を起動する (図 3.5パターン B 参照)。タイマによりタイムアウトを生じた場合は、検索依頼の際に指定された検索失敗の場合の関数を起動する (図 3.5パターン C 参照)。

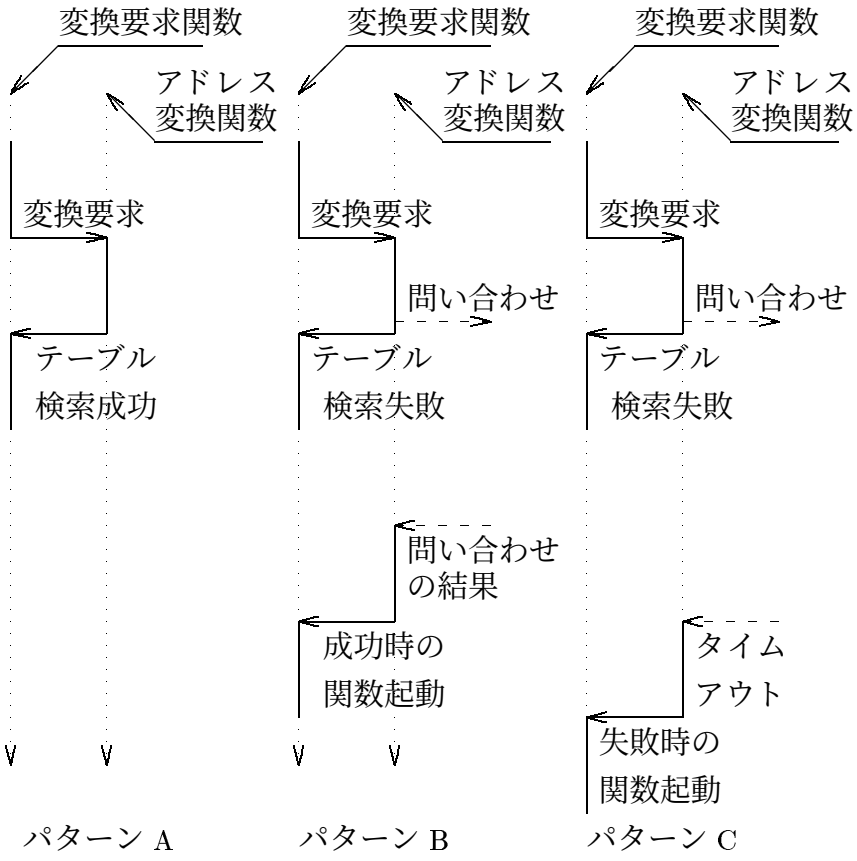


図 3.5: アドレス変換の制御推移

第 4 章

回線交換網とデータグラム網の相互接続機構の実現

この章では、前章の設計に基づき回線交換網とデータグラム網の相互接続を実現するための実現方法について述べる。

4.1 モジュール階層

BSD UNIX ではネットワークインターフェイスを 3.2.2項で述べた機能を持つデバイスドライバ型で実現している。これらの関数は各ネットワークインターフェイスの属性を示す構造体 ifnet によって管理されている。この構造体に、入力データ処理、終了処理、回線制御などの関数を新たに加え、上下のモジュールの構造体を示すメンバーを追加する。

システムが立ち上がる際に各モジュールの初期化関数が起動され、初期化関数内で自分自身の操作関数をネットワークインターフェイスの属性を示す構造体に記述する。モジュールを組み合わせるためには iflayer というコマンドを用いて上下のモジュールが持つネットワークインターフェイスの属性を示す構造体に互いのモジュールを登録する。

また、モジュール上下間での入出力データの際やモジュール内部でのヘッダやトレーラ処理の際にデータのコピーによるオーバヘッドを減少させるために、モジュール上下間ではデータが格納されている位置のみを引き渡し、モジュール内部では 2.1節で述べた mbuf を用いてヘッダやトレーラ処理にデータのコピーを行なわず実現している。

4.2 回線制御機構

3.2.2項で示した関数群に加え、要求を伝達する関数 if_ctlout(ifp, flag, dst) と状態変化を伝達する関数 if_ctlin(ifp, flag, dst) を追加する。ifp はネットワークインターフェイスの属性を示す構造体を、flag は要求する状態や変化した状態、すなわち接続や切断を、dst は接続相手を示す。要求を伝達する関数は上位から下位に対して起動し、起動された関数では必要な手続きを開始し、すべての処理の終了を待たずに制御を戻す。状態変化を伝達する関数は下位から上位に対して起動し、起動された関数は必要な手続きを行ない関数を終了する。

2.1節で述べたようにBSD UNIXではネットワークインターフェイスの状態を示す変数if_flagsにはIF_UP,IF_RUNNIGなどがある。IF_UPはネットワークインターフェイスが動作可能であることを示し、IF_RUNNINGはデータ転送が可能であることを示す。常時接続されているEthernetや専用線などのネットワークインターフェイスでは動作可能であることはデータ転送可能であることを意味するために、IF_UP,IF_RUNNINGは同じ意味を持ち、二つの変数を区別する必要がなかった。しかし、回線制御機能を持ち、常時接続されているわけではないネットワークインターフェイスにおいてはIF_UPは初期化されていることを、IF_RUNNINGは回線が接続されていることを意味する。終了処理が行なわれた場合は変数からIF_UPとIF_RUNNINGを取り除く。この状態変数の値により、データ入力、初期化、終了処理関数との同期を確立した。

4.3　アドレス変換

I.331,E.164,X.121(X.25網でのアドレス)などすべての網でのアドレス、並びに、ネットワークプロトコルのすべてのアドレス体系を扱うことができるように、問い合わせを行なう際には変換するアドレスのタイプを指定する。問い合わせのデータ形式を表4.1に示す。

変換テーブルは動的にオペレーティングシステム内部に格納し、オペレーティングシステム外部への問い合わせはキャラクタデバイスファイルを用いて実現した。ユーザプロセスがこのデバイスファイルを読むことでオペレーティングシステムからの問い合わせ要求を受け入れることができ、問い合わせの結果を書き込むことで問い合わせに対して答を返すことができる。

4.4　作成したモジュール群

if_duc(自動回線制御機構)

if_ducは回線交換網を制御するモジュールである。制御の方法は3.2.4項に示した。回線切断を行なうためのタイマの値は90秒に設定されているが、必要に応じて変更することができる。

if_ducは一般のネットワークインターフェイスとは異なり、非接続の状態や接続相手が変わるなどの特殊な状態を持つ。そのために、通常のネットワークインターフェイスでの接続相手先を示す変数に工夫が必要となる。接続相手先を示す変数は、非接続状態では相手先のアドレスを未設定とし、接続が完了すると接続した相手のアドレスを自動的に設定する。

表 4.1: アドレス変換のパケットフォーマット

0 Byte	問い合わせ識別子
1 Byte	ID
2 Byte	パケットタイプ
3 Byte	フラグ
4 Byte	変換元
5 Byte	アドレスタイプ
6 Byte	変換元
7 Byte	アドレス長
8 Byte	変換先
9 Byte	アドレスタイプ
10 Byte	変換先
11 Byte	アドレス長
12 Byte 〜 (不定長)	変換元アドレス
必要に 応じて	変換後アドレス

表 4.2: XSP のパケットフォーマット

	0	1	2	3	4	5	6	7
0 Byte	パケットタイプ							
1 Byte	コントロールタイプ						リザーブ	LTA

表 4.3: XSP のコントロールパケットの種類

値	コントロールタイプ	内容
1	Null	意味なし
2	Terminate Request	終了要求
3	Terminate Reply	終了確認
4	Are You There	回線チェック要求
5	I Am Here	回線チェック確認
10	My Options	オプション通知
11	Options Ack	オプション確認
12	Version Reject	バージョン拒否
13	Class Reject	クラス拒否
14	Address Reject	アドレス拒否
15	Size Reject	最大転送サイズ拒否

if_xsp(point-to-point プロトコル)

if_xsp は、”Xerox Synchronous point-to-point Protocol” [?](以下では XSP と略す。) を実現したモジュールである。XSP は専用線や回線交換網を用いた際に仮想的に Ethernet と等価に扱える様にするために Xerox が規定したプロトコルであり、XNS で専用線を用いる際には必要不可欠なプロトコルである。XSP の下位層になるデータリンクプロトコルにはビット同期を仮定しており、XSP はプロトコル多重化と回線の状態のチェックなどを可能な限り簡単に実現したプロトコルである。

XSP は、表 4.2で示されるパケットフォーマットを用いて通信を行なう。表 4.3で示される制御データを交換することで回線の制御を行ない、表 4.4に基づく情報交換により接続の際のシステム間協議を行なう。接続から切断までの一連の手続きを図 4.1に示す。

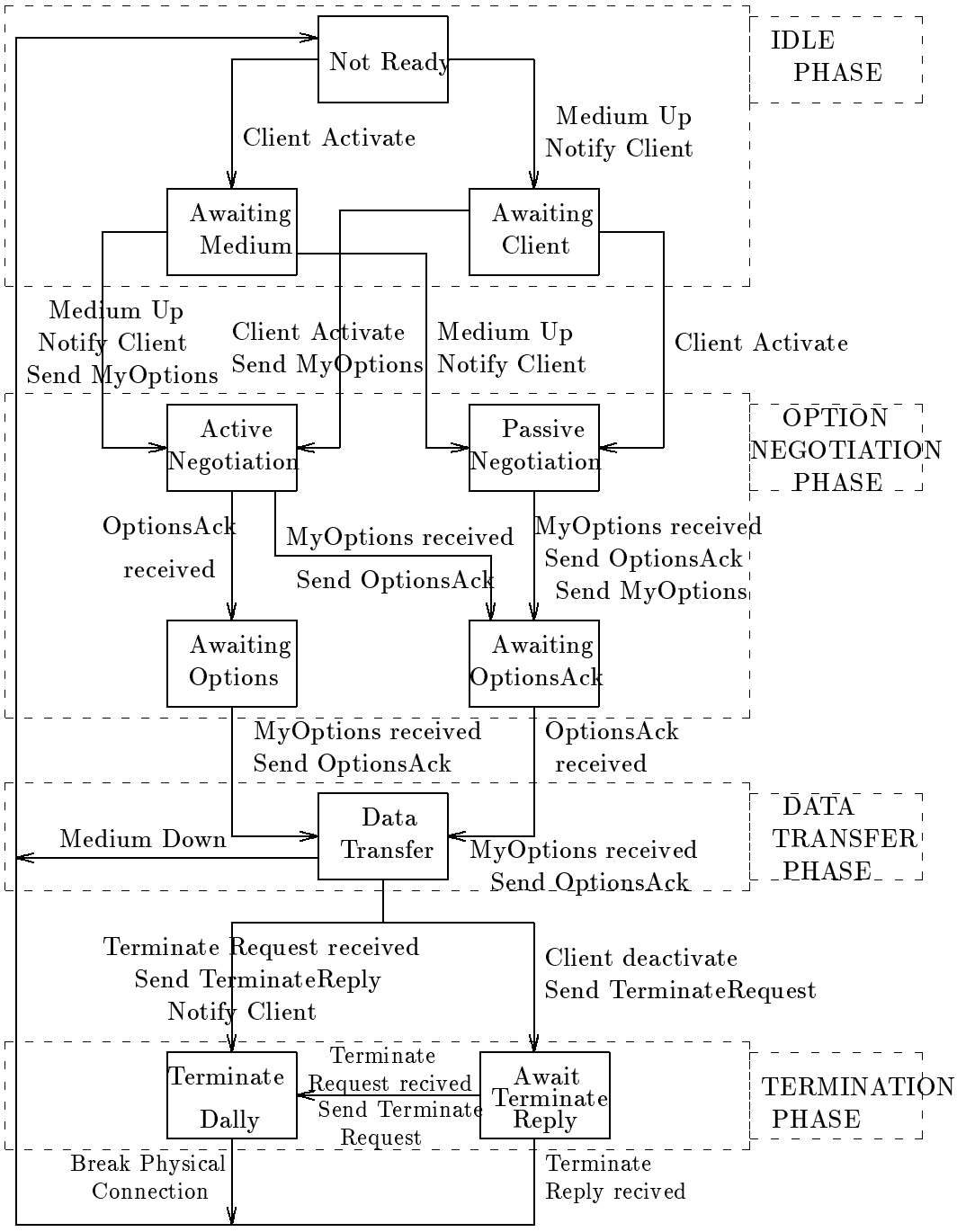


図 4.1: XSP の接続手順

表 4.4: XSP の接続時のデータフォーマット

0 Byte	対応可能な最新バージョン番号
1 Byte	対応可能な最古バージョン番号
2 Byte	送信先アドレス
3 Byte	
4 Byte	
5 Byte	
6 Byte	
7 Byte	
8 Byte	対応可能な Entity クラス
9 Byte	転送可能な一パケット
10 Byte	の最大データ長

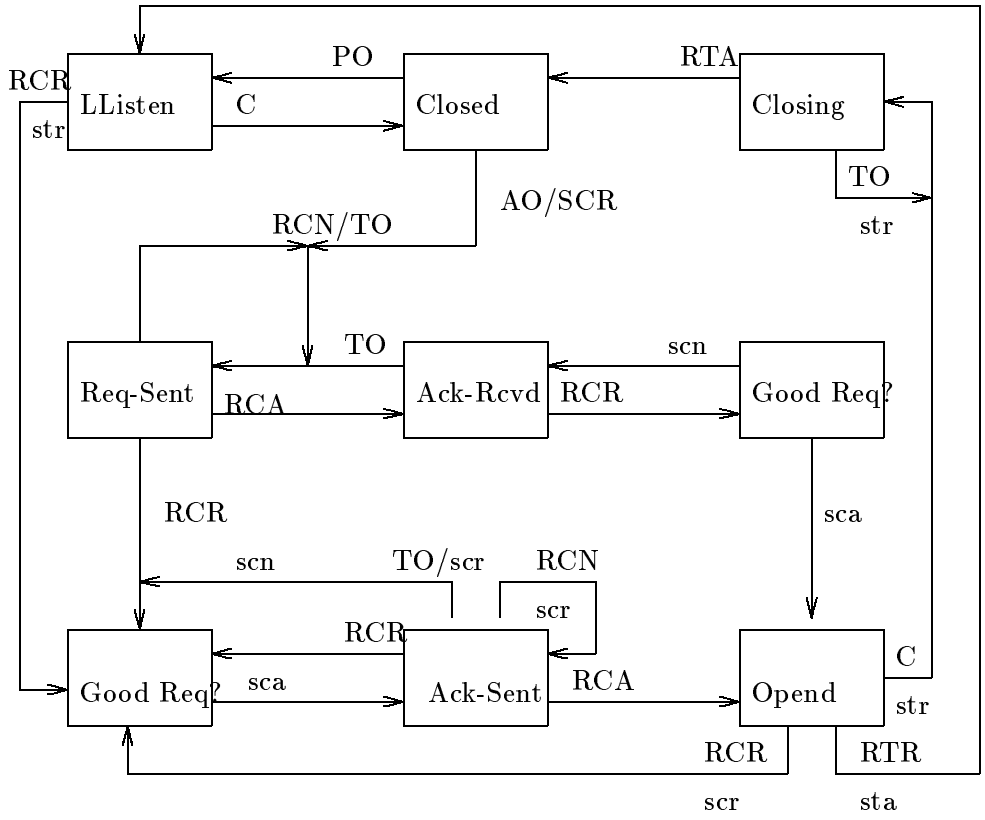
if_lcp と if_ipcp(point-to-point プロトコル)

if_lcp と if_ipcp は IETF[1]のドラフトである”Point to Point Protocol”[?](以下では PPP と略す。) を実現したモジュールである。PPP は下位層に HDLC の UI フレームを用いた通信を仮定している。また、非同期通信路上での利用を考慮し HDLC フレームへの変換方式も定義している。PPP では、Link Control Protocol(以下では LCP と略す) と Network Control Protocols(以下では NCP と略す) の二つからなる。LCP は呼制御を目的とし、呼設定のオプションにはパケット長、認証方法、回線品質検査条件、プロトコル圧縮、UI フレームフィールド圧縮などがある。LCP の制御手順を図 4.2に示す。NCP は利用するネットワークプロトコル毎に規定されており IP プロトコルを利用する際には”The PPP Internet Progocol Control Protocol”[?](以下では IPCP と略す) を用いる。IPCP では IP アドレスの設定と確認、IP プロトコルヘッダ圧縮などのオプションが接続の際に設定できる。PPP での接続手順はまず、LCP が接続を行ない、LCP オプションの認証機能が設定されている時は”The PPP Authentication Protocols”[?] に従って認証された場合のみ、NCP プロトコルで接続手順を行なう。この二つのモジュール if_lcp と if_ipcp は if_ipcp を上位にとして、常に一組として用いる。

if_xcs(ISDN 網制御機能)

if_xcs は X.21 プロトコルの呼制御機能のみを実現したモジュールである。X.21 プロトコルは回線交換網における DTE-DCE インターフェイスであり、回線交換網に接続するための電気的、物理的条件及び、呼制御機能が主な規定内容である。X.21 プロトコルは専用線にも適用可能である。その場合呼制御機能は不要であり、電気的、物理的条件の

[1]Internet Engineering Task Force



		Events			Actions
RCR	-	Receive-Configure-Request	scr	-	Send Configure-Request
RCA	-	Receive-Configure-Ack	sca	-	Send Configure-Ack
RCN	-	Receive-Configure-Nak or Reject	scn	-	Send Configure-Nak
RTR	-	Receive-Terminate-Req			or Reject
RTA	-	Receive-Terminate-Ack	str	-	Send Terminate-Req
AO	-	Active-Open	sta	-	Sent Terminate-Ack
PO	-	Passive-Open			
C	-	Close			
TO	-	Timeout			

図 4.2: LCP の接続手順

みの規定となる。

X.21 では物理インターフェイスの機械的条件として IS 4903 で規定してる 15pin コネクタを使用し、電気的条件としては X.27[?] に従い、9.6Kbps 以下の場合は X.26[?] を使用することも許可している。回線交換網を利用する際の呼制御手順は制御データを IS 4903 で規定している T,R 回路に、制御信号を C,I 回路に送信するこで実現している。T,R 回路上のデータフォーマットは、二つ以上の SYNC キャラクタを先頭にしたビット同期で行なわれる。回線交換網の呼制御手順を図 4.3, 図 4.4に示す。

if_lapb と if_ui(データリンク制御機能)

if_lapb は、X.25 で用いられる Link Access Procedure Balanced (以下では LAPB と略す）を実現したモジュールで、LAPB は下位層にビット同期を仮定している。通信を行なう際のデータのフレーム構成を表 4.5に示す。制御フィールドにはコマンド、レスポンスあるいはフレームの番号が必要に応じて設定される。制御フィールドのフォーマットには I,S,U の三種類のフォーマットがある。I フォーマットは情報転送に使用するもので、送受信したフレームの番号を含む。フレーム番号は、0 から受信確認なしに送信できる最大フレーム数のモジュラス値までの値で示される。通常の場合モジュラス値は 8 で、衛星など伝送遅延の大きい通信媒体での利用を考慮して拡張機能として 128 も定義されている。S フォーマットはフロー制御やフレーム再転送要求などの使用される。U フォーマットは呼の設定や終了に用いられる。FCS はフレームの伝送エラーを検出するための Cyclic Redundancy Check に用いる。

LAPB の接続手順の一例を図 4.5に示す。LAPB は DTE-DCE 間プロトコルであるが、T.90[?] は DTE-DTE 間で LAPB を用いるための規定を行なっている。DCE が LAPB を解釈する場合は LAPB を用いて、DCE が LAPB を解釈しない場合は T.90 に基づいて DTE 間で LAPB を用いて通信を行なうことで、データリンク層でデータの補償を行なうことができる。従って、設定によって DTE-DCE, DTE-DTE 間プロトコルとなるように実現した。

if_ui は前述の LAPB と同様のフレームフォーマットを利用するが呼の概念がなく、送受信するフレームにも番号を付けず、データ補償は行なわない。つまり、フレーム化を行ない伝送エラーのみをチェックするプロトコルである。

if_vjc(プロトコルヘッダ圧縮機能)

if_vjc は”Compressing TCP/IP headers for low-speed serial link” [?](以下では、Van Jacobson Compression method から VJC と略す。) を実現したモジュールである。VJC は、TCP/IP のヘッダ部分を圧縮するための規定である。仮想端末などの利用により転送されるデータは 1Byte づつであることが多い。1Byte のデータに対して必要な TCP/IP のヘッダは 40Byte で実際に転送されるデータは 41Byte となる。VJC では TCP/IP のヘッダ部分を圧縮し約 5Byte 前後にすることでプロトコルオーバヘッドを減少させるこ

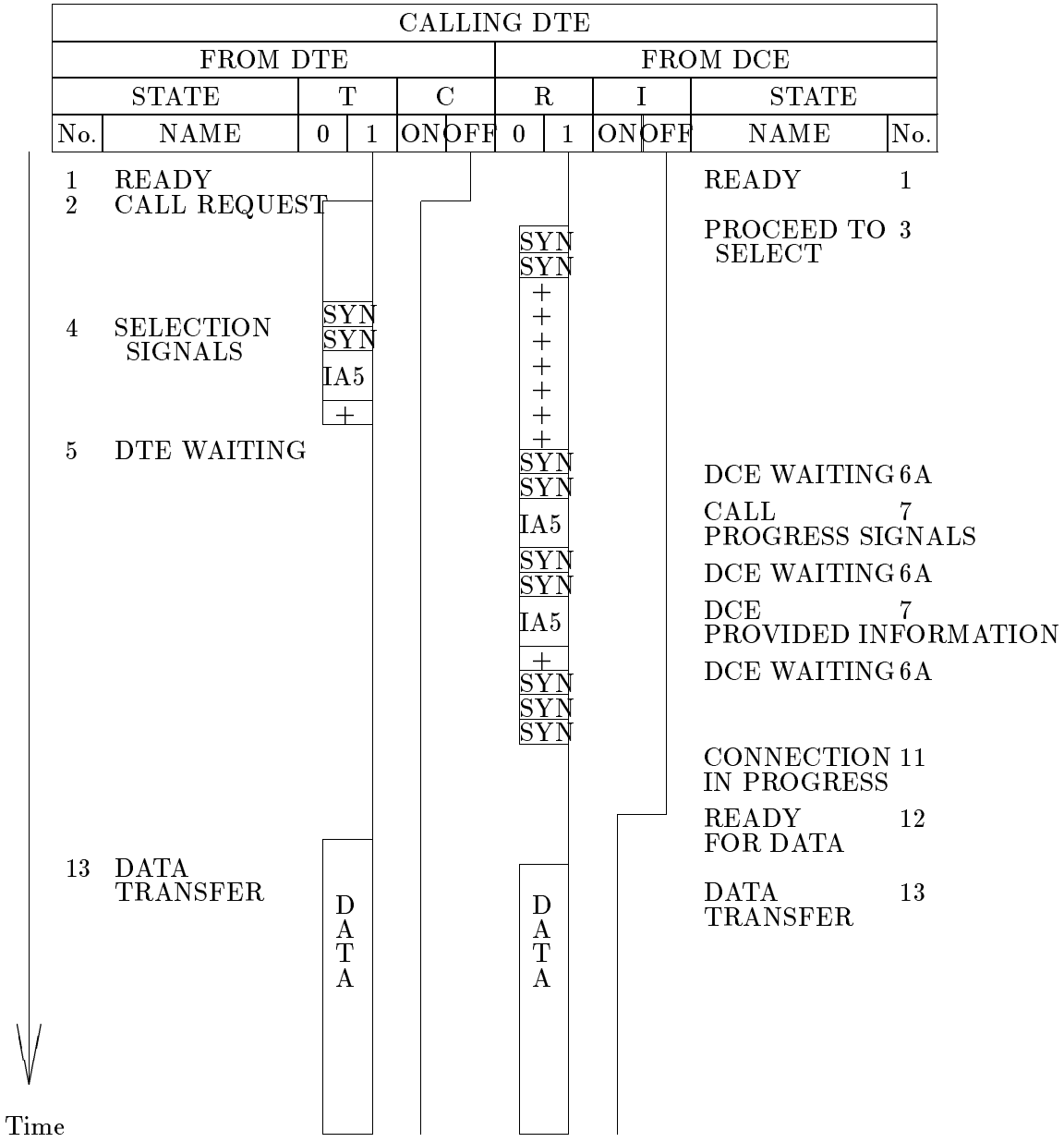


図 4.3: X.21 プロトコルの呼制御手順における呼の設定

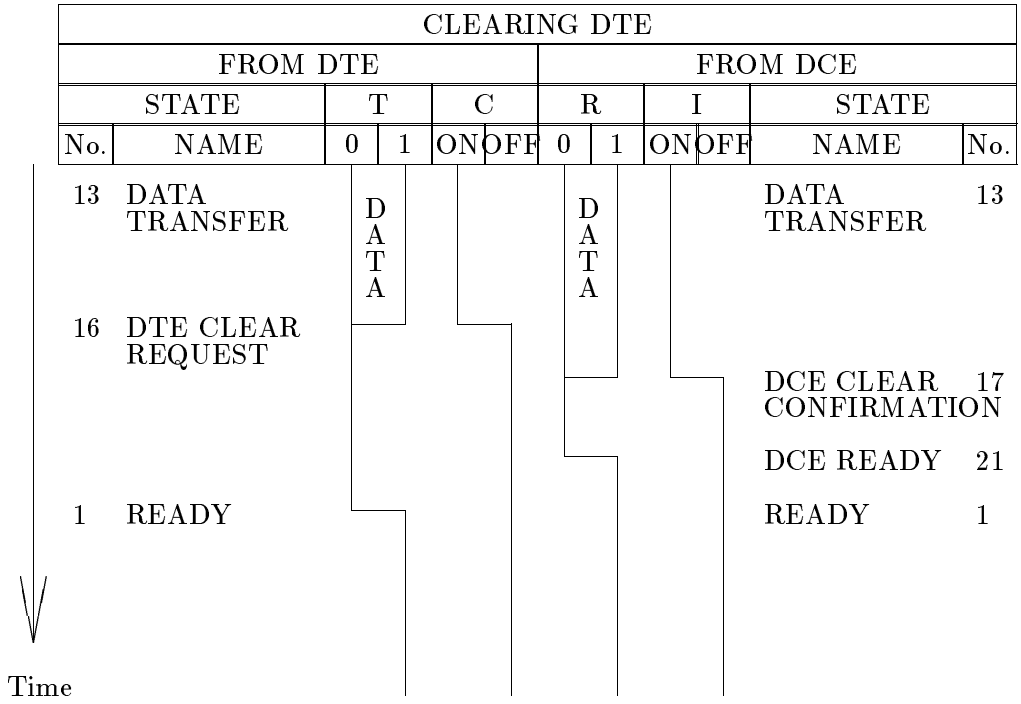


図 4.4: X.21 プロトコルの呼制御手順における呼の終了

とができる。低速な通信路で TCP/IP プロトコルを用いる場合には非常に有効な通信手段である。

if_slip と tty_slip(非同期通信機能)

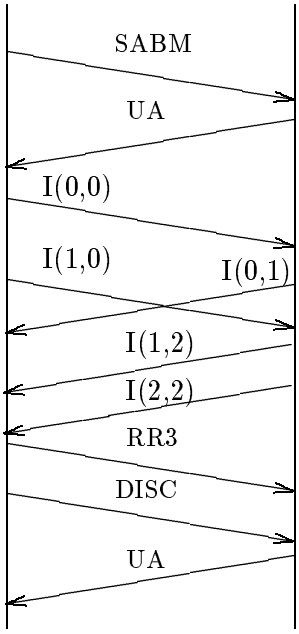
if_slip は BSD UNIX 上で、tty_slip は SunOS 4 上で"Nonstandard for transmission of IP datagrams over serial line: SLIP"(以下では SLIP と略す) を実現したモジュールである。SLIP は下位層に非同期による通信を仮定しており、キャラクタオリエンテッドな通信媒体上にデータグラム通信を行なうためのパケット化機能を持っている。パケット化の方法は送信するデータ中の 0xc0,0xdc を 0xdb+0xdc,0xdb+0xdd に置き換えて送信し、エンドマークとして 0xc0 を送信する方法である。

if_pppa と tty_pppa(非同期通信機能)

if_pppa は BSD UNIX 上で、tty_pppa は SunOS 4 上で PPP で規定している非同期通信路上での HDLC フレーム変換方式を実現したモジュールである。

表 4.5: LAPB のフレーム構成

	0	1	2	3	4	5	6	7
0 Byte	01111110(開始フラグ)							
1 Byte	アドレス							
2 Byte	制御タイプ							
3 Byte 〜 N-3 Byte	データ							
N-2 Byte	FCS							
N-1 Byte								
N Byte	01111110(終了フラグ)							



SABM: Set Asynchronous Balanced Mode
UA: Unnumbered Acknowledge
RR: Receive Ready
DISC: Disconnect
I(x,y): Data (x: Sequence y: Acknowledgment)

図 4.5: LAPB の接続手順の一例

表 4.6: 同期通信の際のフレームフォーマット

	0	1	2	3	4	5	6	7
0 Byte	01111110(開始フラグ)							
1 Byte 〜 N-3 Byte	データ							
N-2 Byte	FCS							
N-1 Byte								
N Byte	01111110(終了フラグ)							

N: データ長

同期ドライバ

mono-sync は一つ以上の SYNC キャラクタを先頭にしてビット同期で通信する方法である。bi-sync は二つ以上の SYNC キャラクタを先頭にしたビット同期通信であるが、一般に bi-sync と呼ばれるものは IBM が規定してる通信方式で、二つ以上の SYNC キャラクタを先頭としてデータ IN,OUT などやヘッダなどの情報を加えて送信する。しかし、ここで用いる bi-sync は単純に二つ以上の SYNC キャラクタを先頭にした同期通信と定義する。この bi-sync 通信は X.21 プロトコルがデータリンク層のプロトコルとして定義している通信方式である。

また、mono-sync や bi-sync モードでの通信はキャラクタオリエンテッドであるために、データグラム通信を行なうためにはビット同期モードでパケット化を行うプロトコルが必要となる。ここでは LAPB などに用いるフレームを構成するためのフレーミング方式を用いた。フレームのフォーマットを表 4.6に示す。開始、終了フラグに 01111110 を用いるために、送信するデータで 1 が 5 つ以上連続する場合には 0 を挿入して送信する。

このような bi-sync やフレーミング方法は利用するハードウェア、特にデータ出力用のチップに依存する部分が大きい。従って、同期通信を行なうモジュールは利用する計算機毎に作成されている。

if_ifd と if_ntd (ネットワークインターフェイスモニタ機能)

if_ifd はネットワークインターフェイスの動作確認を行なうために、if_ntd はネットワーク上のデータをモニタするためのモジュールである。

if_ifd はファイルシステム上の特殊ファイルであるキャラクタデバイス ifd へのデータをネットワークインターフェイスのデータ形式に変換し入出力を行なうものである。網終端装置をテストする場合には終端装置とのデータを転送に必要なデータリンク規定に基づいたモジュールを下位層に位置させ、終端装置を制御するためのデータをファイル ifd に書き込むことにより終端装置の操作ができ、終端装置からのデータはファイル ifd

を読むことによって得られる。ファイル ifd へのデータの入出力は UNIX の入出力関数を用いているために、時間のずれが生じる。そのため、網終端装置とのやり取りでのタイミングが問題となる場合には利用できない。

if_ntd はモジュール間データをコピーしファイルシステム上のキャラクタデバイス ntd に出力するモジュールである。このモジュールをモニタしたいモジュール間に挿入することにより、実際に入出力しているデータをすべてモニタできる。しかし、実際にネットワークの状態を把握するには送受信されるすべてのデータをモニタする必要はない。なぜなら、両端間で利用しているプロトコルのヘッダやトレーラをモニタすることで、ネットワークやプロトコル上でのエラーなどの状況を把握することができるからである。多くのプロトコルでは高速な処理を可能にするためにプロトコル情報はヘッダとして付加する。トレーラとして付加される情報の多くは転送したデータのチェックを行なうためのものであり、データリンク層でよく用いる FCS などがこれに当たる。データリンク層での FCS エラーは伝送上のエラーか送受信を行なうハードウェアによるエラーの可能性が非常に高く、普通は通信チップを操作しているデバイスドライバがエラーを検知し、その情報を保持している。従って、ヘッダをモニタすることにより、ネットワークでの状態を把握することができる。加えて、ヘッダ部分だけをコピーしファイル ntd に出力することにより、コピーによるデータ送受信のパフォーマンス低下やプライバシの侵害などを避けることができる。モニタする先頭からのバイト数は利用するプロトコルのヘッダのサイズに依存するため自由に設定可能である。

4.5 ISDN 回線交換網を利用した実現

ハードウェア環境

ISDN 回線交換網を利用する方法として、一つには個々の計算機で利用可能な ISDN ボードを用いる方法がある。しかし、計算機に直接 ISDN ボードを挿入し利用することにより、計算機やボードに依存する部分が大きくなり移植性を失ってしまう。ここでは ISDN ボードを使用せずに ISDN 網の終端装置であるターミナルアダプタを用いること前提とした。ターミナルアダプタは X.21 プロトコルにより ISDN 網を制御できるものを選び、ターミナルアダプタと計算機間は RS232 又は RS422 で接続を行なった（図 4.6参照）。実際に利用したハードウェア構成を表 4.7に示す。

RS232C の規格は伝送速度は 20Kbps 以下を規定しており、RS423 では 300Kbps 以下を規定している。表 4.7に示す多くの計算機は RS232C と上位互換である RS423 を利用している。しかし、レベル変換器を含め幾つかの計算機は RS232C に基づいているために、RS232C を用いる場合は電気的な障害が起きる可能性があり、電気的に充分な配慮が必要となることを注意しなければならない。

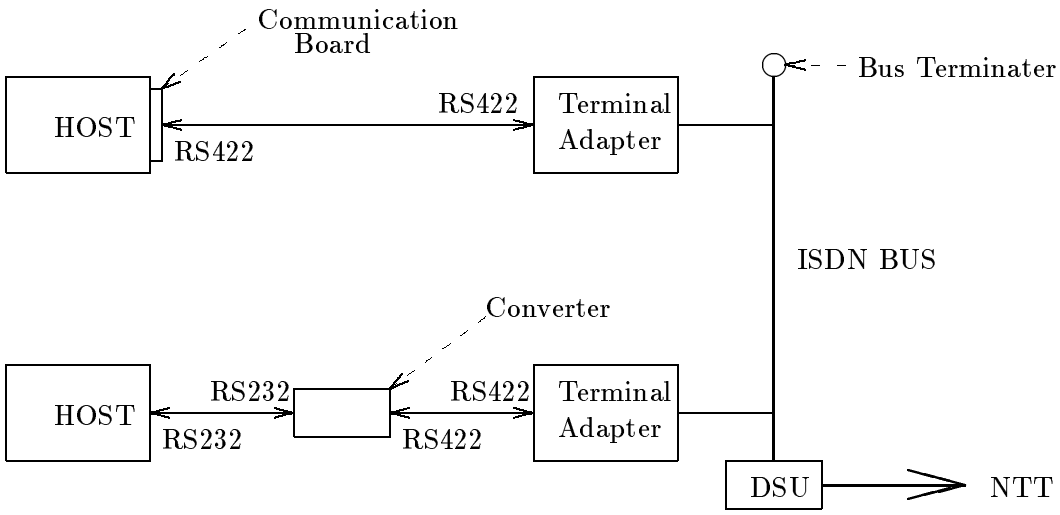


図 4.6: ISDN 回線交換網を利用するための機器と接続方法

表 4.7: ISDN 回線交換網を利用するためのハードウェア構成

計算機	Sun Sparc Station I	Sun3/60	Sun3/160	OMRON LUNA
通信ボード	-	-	Sun MCP	GPX Board
レベル変換器	RS232C-RS422	RS232C-RS422	-	RS232C-RS422
ターミナルアダプタ	NH-5101-1	Aterm 112	ISPTC	ISPTC
ISDN 回線	INS 64	INS 64	INS 64	INS 64
回線数	1 B channel	1 B channel	2 B channels	1 B channel
回線速度	64K bps	64K bps	64K bps	64K bps

表 4.8: ISDN 回線交換網を利用するためのソフトウェア構成

パターン	1	2	3	4
モジュール		if_duc		if_duc
			if_xcs	if_xcs
	同期ドライバ	同期ドライバ	同期ドライバ	同期ドライバ

パターン 1: 相手固定接続で手動接続／切断
パターン 2: 相手固定接続で自動接続／切断
パターン 3: 不特定相手接続で手動接続／切断
パターン 4: 不特定相手接続で自動接続／切断

ソフトウェア構成

ISDN の回線交換網とターミナルアダプタを用いて可能となる接続形態には、相手固定接続と不特定相手接続がある。相手固定接続は事前に設定した相手とのみ接続し通信を可能とする接続形態で、不特定相手接続は一般の公衆回線の様に必要な相手との通信を可能とする接続形態である。これらの組合せに対して自動接続／切断を行なうか、手動による回線制御を行なうかの選択が加わる。これらの組合せを実現するために必要なソフトウェア構成を表 4.8 に示す。表 4.8で示されるパターン 1 は手動による接続／切断を行なう相手固定接続で、専用線を利用する場合と同じである。

計算機の通信ポートに高速な (回線速度が 64Kbps 以上) 回線を接続すると、計算機の能力の限界や伝送路上のノイズなどにより送受信するデータが破壊される危険性がある。しかし、データにエラーが生じても補償を行なう機能があれば問題はない。一般的に、この補償は送信したデータに対しての応答が返送されずに一定間隔が過ぎた場合に再転送を行なう方法で実現されている。上位層で補償を行なわない場合や、エラーが頻繁に生じるが上位での再転送の間隔が長く上位層での補償がパフォーマンスの低下につながる場合は、データリンク層で補償する必要がある。lapb プロトコルなどを同期ドライバの上位に位置させることにより、データリンク層でのデータの信頼性を高めることができる。

一つの専用線や回線交換網などの point-to-point リンクを複数のプロトコルで利用するためには、プロトコル多重化を行なう必要がある。プロトコル多重化は送受信するすべてのデータのヘッダにプロトコルを示すフィールドを追加し、追加したフィールドに上位層を明記することにより実現できる。また、専用線や回線交換網などの point-to-point リンクでは定期的に折り返し検査を行なうことで、回線や相手の異常を検知することができる。プロトコル多重化や異常検知機能は、point-to-point プロトコルである XSP や PPP に実装されている機能である。これらの point-to-point プロトコルを利用することでより確実な接続が可能になる。

我々が現在利用している計算機では送受信の際にデータが破損する確率や ISDN などのデジタル回線上でのデータの破損確率は非常に低いために、データの補償を行なう if_lapb

表 4.9: WIDE での ISDN 回線交換網の利用

if_duc
if_xsp
if_xcs
同期ドライバ

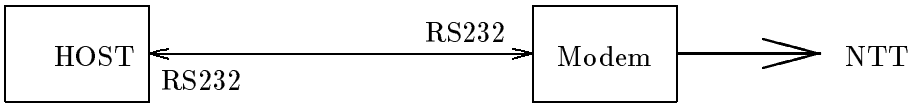


図 4.7: アナログ回線交換網を利用するための機器と接続方法

などを用いる必要がない。逆に、if_lapb などを用いることにより、ヘッダ部分が増加することによるパフォーマンスの低下につながる。従って、ISDN 回線を用いる場合には if_lapb などのデータ補償を行なうプロトコルは利用しない。また、公衆回線交換網では回線や相手側に異常が生じた場合は即時に回線を切断し、不必要な料金の支払いを避ける必要がある。加えて、広域網で回線交換網を利用するためには複数のプロトコルのデータを伝送する必要がある。従って、プロトコル多重化と異常検知機能を持つ point-to-point プロトコルを利用する。PPP は XSP に比べ、相手の認証、プロトコルヘッダの圧縮機能などの付加機能を持つ。ISDN では発信者番号通知機能が利用できるために相手の認証の必要性は低く、高速なデジタル通信においてはプロトコルヘッダ部分の圧縮機能の必要性も低い。必要性の低い機能を実装している PPP を利用することによりパフォーマンスの低下を導くことになるために、point-to-point プロトコルとしては XSP を利用する。

一般的に我々が ISDN 回線交換網を用いるために利用しているモジュール構成を表 4.9 に示す。

4.6　アナログ回線交換網を利用した実現

ハードウェア環境

アナログ回線を利用する場合には、モデムを用いて計算機の通信ポートに接続する (図 4.7参照)。実際に利用した機器を表 4.10に示す。

表 4.10: アナログ回線交換網を利用するためのハードウェア構成

オペレーティングシステム	SunOS 4.0.3	4.3BSD
計算機	Sun Sparc Station 330	OMRON LUNA
モデム	TrailBlazer	TrailBlazer
モデム間速度	9.6Kbps	9.6Kbps

ソフトウェア構成

アナログ回線を用いて通信を行なう場合には回線やモデムにより同期、非同期の両方の通信方式が利用できるが、一般的に非同期通信が利用されている。同期通信を行なうためには 4.5節で述べた方法で回線制御モジュールをモデムにあったモジュールに入れ換えることで利用できる。ここでは非同期通信を行なうための方法について述べる。非同期通信路でデータグラム転送を行なうための方法は、SLIP プロトコルを用いるか PPP が定義している”Asynchronous HDLC” を用いる方法の二通りがある。

まず、PPP が定義している”Asynchronous HDLC” を用いて PPP を実現するためには表 4.11に示すモジュール構造で利用する。アナログ回線では発信者通知機能がないために通信を行なう前に相手の認証 (識別) を行なう必要がある。従って、PPP のオプション機能である相手の認証を利用しなければならない。表 4.11の auth は相手の認証行ない、認証を受けるためのデータを作成するためのモジュールである。

auth_chk(type, peer, data)

を用いて、type に認証の方式、peer には相手システムを data に相手が認証のために申告したデータを与えることで相手の認証を行うことができる。また、

auth_make(type, peer, data)

を起動することで相手システムに認証を得るために申告するデータを得ることができる。この auth モジュールの関数が扱うデータはすべてオペレーティングシステム内部に事前に設定されたもののみで、関数は設定されたテーブルを検索し認証又はデータ作成を行ない関数を終了する。もし、データがない場合はすぐにエラーを返して終了する。

アナログ回線での PPP の利用は、PPP のオプション機能である認証を用いること、データリンク層のモジュールが非同期用である点を除けば 4.5節での利用方法と同じである。

次に、SLIP プロトコルを用いる場合について述べる。SLIP プロトコルは直接計算機同士を接続する場合やモデムを用いて専用回線上で計算機同士を接続するために開発されたプロトコルである。そのために認証の機能がない。アナログ交換網で相手の認証を行なわずに接続すると相手の区別がつかず、セキュリティの面において問題となる。従って、相手の認証と認証に必要な手続きを行なうモジュール if_uath が必要となる。このモジュールは dialup-slip などと互換性を持たせるために通常の UNIX の認証方式である login 名とパスワードの入力、検査をそのまま実現している。自システムが発信側の場合

表 4.11: アナログ回線での PPP の利用

if_duc	
if_ipcp	
if_lcp	auth
if_at	
if_pppa/pppa_tty	

表 4.12: アナログ回線での SLIP の利用

if_duc	
if_at	
if_uauth	auth
if_slip/slip_tty	

表 4.13: WIDE でのアナログ回線の利用

if_duc	
if_vjc	
if_at	
if_uauth	auth
if_slip/slip_tty	

には login 名とパスワードを出力し、着信側である場合は入力を auth モジュールを用いて確認する。この認証の終了した後にデータリンク層を slip モジュールに切替え通信を開始する。SLIP プロトコルを用いて通信する場合のモジュール階層を表 4.12に示す。

PPP を用いることで複数のネットワークアーキテクチャで同時に回線を利用することができる。しかし、低速なアナログ回線では複数のネットワークアーキテクチャで利用することにより回線が飽和状態に達する可能性がある。また、複数のネットワークアーキテクチャの利用を可能とするためのプロトコルフィールドやデータリンク層でのエラー検知機能のための FCS などヘッダやトレーラが多いために実質的な通信速度を低下させる。この様な理由から一般的に SLIP プロトコルを用い、より高速なデータ通信を行なうために VJC のプロトコルヘッダ圧縮を加えて用いる (表 4.13参照)。

第 5 章

IP ネットワークのための ISDN ゲートウェイ

この章では作成した交換網制御機構を用いて実際にネットワークを確立することで、交換網を用いたネットワークの構築に必要な技術の確立を行なう。

実験ネットワークを構築する際に同一環境に保つために一般的な環境として TCP/IP プロトコルを用いて、回線には ISDN の回線交換網である高速な INS64 を用いて実験を行なった。

5.1 回線交換網を利用したネットワークのモデル

交換網を実際に利用する際に最も問題となる点が接続切断のタイミングと接続先の決定方法である。実際に利用するにあたってはセキュリティや回線利用料金などの面で接続相手がある程度限定される。また、接続のタイミングや接続時間に関しては利用料金と利用するサービスの重要度によって決められる。サービスの重要度が高ければ伝送遅延や利用可能な通信帯域幅を重視し、低ければ利用料金を考慮する。この様に利用料金やセキュリティなどの制限と利用するサービスの種類による要求のトレードオフで接続の時間、タイミングや接続相手などが決定される。また、接続相手と接続時間などにより経路を変更し経路制御を行なう必要がある。

これらのトレードオフを実際のすべてネットワークで実験し、接続相手、接続のタイミングや時間を決定方法を考え、経路制御を行い、ネットワーク構築に必要な技術の確立を行なうわけにはいかない。そこで、交換網を利用したネットワークをモデル化し、モデルに従って実際のネットワークを分類する。分類されたモデルの典型的なネットワークを作成し、作成したすべてのネットワークでネットワーク構築に必要な技術を確立し、作成した実験ネットワークにおいて制限や要求のトレードオフを考える。

接続相手や接続時間、タイミングに加えて経路とその制御方法などの要素によって実際に構築するネットワークの形態が変わることが予想される。これらの要素は交換網を利用するための制限と要求によって決定される。従って、この制限や要求を元にモデル化を行なう。交換網利用の制限にはセキュリティ、設備の数や経済的問題などが挙げられる。また、交換網を利用する際の要求には高速な応答、広い通信帯域や直接相手と接続することによる確実性などがあげられる。個人的なサービスのために交換網を利用する場合は要求を重視して制限を受け入れる形となる。しかし、システムサービスのため

には制限を厳守して要求を満たすように利用する。制限と要求を同じレベルで考えて利用する際はすべてのサービスを提供する場合である。個人的なサービスとシステムサービスの二つのサービスの種類で、両立させる場合を加えて三つのモデルに分類して、実験ネットワーク構築に必要な技術を確立し、有効性を検証する。

	要求と制限の優先度の関係
個人的サービス提供が目的の場合	要求 $\gg$ 制限
システムサービス提供が目的の場合	要求 $\ll$ 制限
全てのサービス提供が目的の場合	要求 $\equiv$ 制限

接続する二つの計算機の Internet での位置が異なることにより、交換網を利用するための要求と制限が異なる可能性がある。両計算機が Internet 外に位置する場合は、どのモデルにおいても要求に応じて交換網を利用すれば良く、そのために必要な機能は前章に述べた通り実装済みである。従って、接続する両計算機のネットワークでの位置が交換網を利用する際の要求と制限に影響を与えるモデルでは、既に Internet に接続されている場合と、交換網を用いてのみ Internet に接続される場合にモデルを分離して実験を行なう。

5.2　個人的利用

利用者にとって個人的に回線を利用する際に必要なことは、利用したい時に回線の利用でき、利用が終了すると回線が自動的に切断されることが保証されていれることである。このために必要な機能は前章で述べた通り実現されている。しかし、個人的利用を受け入れる相手側では大規模性を持たせるために、個人的な利用による計算機資源やネットワーク資源を必要以上に消費することを避けなければならない。前章で述べた回線制御機構を用いることで回線の接続切断時に計算機に与える影響は極めて少ない。しかし、回線を新しく接続するために Internet では、回線に対してネットワークアドレスを取得し、ネットワーク上の交換網を利用できるすべての計算機への経路情報を登録する必要がある。この様なネットワーク資源を交換網にどのように割り当てるかが問題となる。

交換網を利用できる計算機の数を N とすると、ネットワークアドレスを接続可能なすべての計算機の組合せに割り振るためには${}_NC_2$通りのネットワークアドレスが必要となる。交換網を利用する計算機が増えることにより、ネットワークアドレスの割り当てが不可能になる。

また、経路情報に関しては、経路を制御するための情報を静的に設定する場合にはすべての組合せ${}_NC_2$を設定必要があり、経路情報を維持するためのコストが莫大なものとなり、検索にかかる時間が増大し、通信を行なう上でのオーバヘッドとなる。経路情報を交換し動的に経路情報を更新していくことで、現在接続されている経路のみを扱うため、経路情報の低いコストでの維持、検索が可能となる。しかし、交換網、特に、デジタル系の交換網では秒単位以下で接続や切断が行なうことができるために、経路情報の更新が頻繁に起こることが予想される。交換網を利用できる計算機が増え、接続や切断が頻

繁に行なわれることでネットワーク上で伝達される経路情報が莫大なものとなり、ネットワークの負荷を増大させることになる。

個人的な利用を可能にするためには、この様なネットワークアドレスの割当や経路制御をいかに扱うが大きな問題となる。この節では、このネットワーク資源の扱い方を重視して、個人的なサービス提供を目的としたネットワークを構築する。接続する二つの計算機の位置が異なることで利用者の要求度合が異なり、構築するネットワークの形態が異なる。従って、交換網を利用することのみで Internet に接続する場合と、既に Internet 内部に接続されている場合に分けて実験ネットワークを構築する。

5.2.1 交換網のみにより Internet と接続する場合

設計方針

point-to-point リンクにおいてネットワークアドレスを一つ利用することにより、自システム、相手システムとブロードキャストアドレスの三つのアドレスが必要になり、ホストアドレス部には最低 2 ビット必要となる。交換網を持つ計算機間のリンクすべてにネットワークアドレスを割り当てるのではなく、接続の可能性のある計算機とのリンクのみを考えた場合、組織が受け入れる計算機を N とすると、外部からの交換網などを利用して接続する計算機に対して $N * 2^2$通りのアドレスを事前に割り当てる必要がある。必要となるアドレスを最低限の N 通りにするにはネットワークの存在自体を隠蔽する必要がある。XNS などでは専用線にネットワークアドレスを用いず両端の計算機を一つのゲートウェイとして考えている。この考え方を用いることで論理的に point-to-point リンクを隠蔽することができる。実現の方法は point-to-point リンクを unnumbered リンクとし、point-to-point リンクにネットワークアドレスを付けず両端の計算機では既に取得しているアドレスを付ける。この方法を用いることで、組織が受け入れる計算機の数 N だけアドレスを用意すれば良いことになる。

経路情報が設定されていない計算機を通過して通信を行なうことはできない。しかし、交換網を利用できる計算機が増えた場合、交換網で接続される全ての経路情報の設定を行なうことは不可能である。そこで、経路情報が必要となる場合は、新たなアドレスを使用することによるものであることに注目する。既に利用されているネットワークアドレスの一部であるサブネットを用いることで、組織外部に対して経路情報を流す必要がなくなる。ただし、組織内部に対しては新しいサブネットを利用するために経路情報を流さなければならない。新たな経路情報を全く流さずに接続を行なう方法は、接続されている計算機を既存のネットワーク上に位置させるのみである。

事前に必要となるアドレスを接続する計算機の数 N と同数とし、新たな経路情報を流さないためには、point-to-point リンクを隠蔽し、接続した計算機を既存のネットワーク上に位置しているように見せかける (図 5.1 参照) 方法を取る必要がある。この方法を実現するためには、接続先の計算機が接続しているネットワークが、

- 複数の計算機が接続できるネットワークであること。

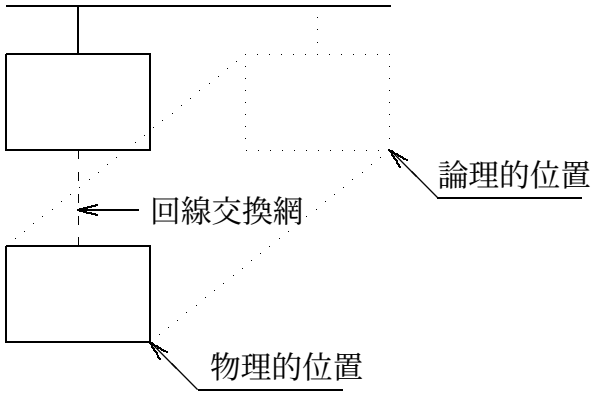


図 5.1: ネットワークアドレスと経路情報を必要としない接続

- ソフトウェアによってハードウェアアドレスと上位のネットワークアドレス (IP アドレス) を関連付けているネットワークであること。

が必要となる。この条件を満たしているネットワーク上で、交換網の接続先の IP アドレスを自分であるように広報し、集まったデータは交換網を通し転送することで、論理的に接続先のネットワーク上にある様に見せることができる。

実験ネットワーク

実験は WIDE プロジェクトのバックボーン上の計算機と自宅の計算機を接続して行なった。接続の方法を以下に述べる (図 5.2参照)。接続に利用した機器は、HOST A に Sun SparcStation 330 を、HOST B に Luna SX9100DT を、交換網には ISDN 網の回線交換サービス INS64 を 64Kbps で使用した。

Internet 側の計算機を A とし相手の計算機を B とする。事前に B は A と同じネットワークアドレスのアドレスを取得する。次に、A は B への経路を交換網が接続されているネットワークインターフェイスに設定する。最後に、proxyarp を用いて A が接続されているイーサネットに対して B の IP アドレスが自分であることを広報する。proxyarp はイーサネットのアドレス変換プロトコル ARP に基づいて、あるシステムの Internet アドレスと自システムのイーサネットアドレスをネットワーク上のすべての計算機に広報するためのアプリケーションである。B が A に接続する際には、交換網に接続されている B のネットワークインターフェイスのアドレスを A と同じネットワークアドレスを持つアドレスに設定することで利用きる。

オペレーティングシステム内部では unnumbered リンクに接続されている相手のシステムを識別することでネットワークインターフェイスを区別する。交換網のように接続が完了するまで相手システムが未定の場合には、オペレーティングシステム内でネットワークインターフェイスの区別がつかなくなる。例えば、図 5.2において A のすべての

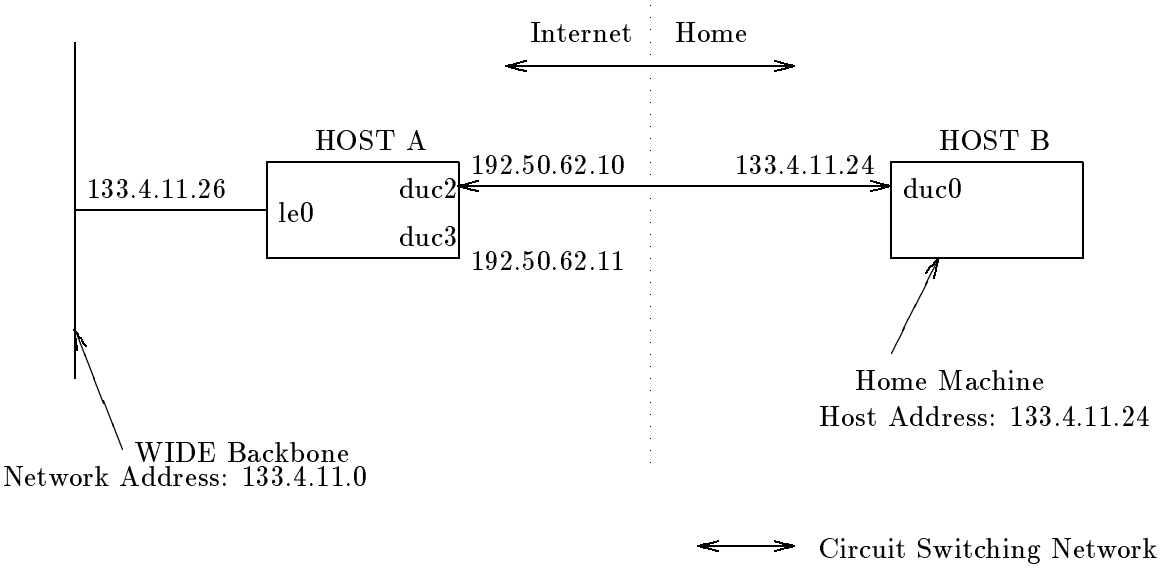


図 5.2: 自宅との接続

ネットワークインターフェイスに同じアドレスが付いていると、A が B への経路を設定する際にオペレーティングシステム内部ではどのネットワークインターフェイス上に B が存在するかの区別がつかない。これは BSD UNIX では経路の設定をゲートウェイと相手の組合せで行なっているためであり、出力ネットワークインターフェイスを指定するのではないためである。従って、経路設定の方法を変えない場合は、図 5.2での A の交換網に接続されているネットワークインターフェイスのアドレスのように、他との区別ができるアドレスを設定する必要がある。そのため異なったアドレスを unnumbered リンクに割り当てる。しかし、B はネットワークインターフェイスを一つしか持たないために、交換網に接続されいるネットワークインターフェイスに対して B のアドレスをつけることが可能である。

5.2.2 既に Internet と接続している場合

設計方針

Internet 内部の計算機同士での接続で交換網を利用する目的は、交換網を用いて既存の経路に対してのバイパス経路を作成することにある。既存の経路ではトラフィックが多く伝送遅延が大きい場合や経路上の計算機や回線の信頼性が低い場合などに、バイパス経路が必要となる。この様なバイパスを目的とする場合は、バイパス経路によって早く転送したいデータや確実に転送したいデータを持つ特殊なサービスのみ交換網を利用するように設定する必要がある。

交換網用いてのみ Internet と接続される場合と異なることは特殊なサービスのみのデー

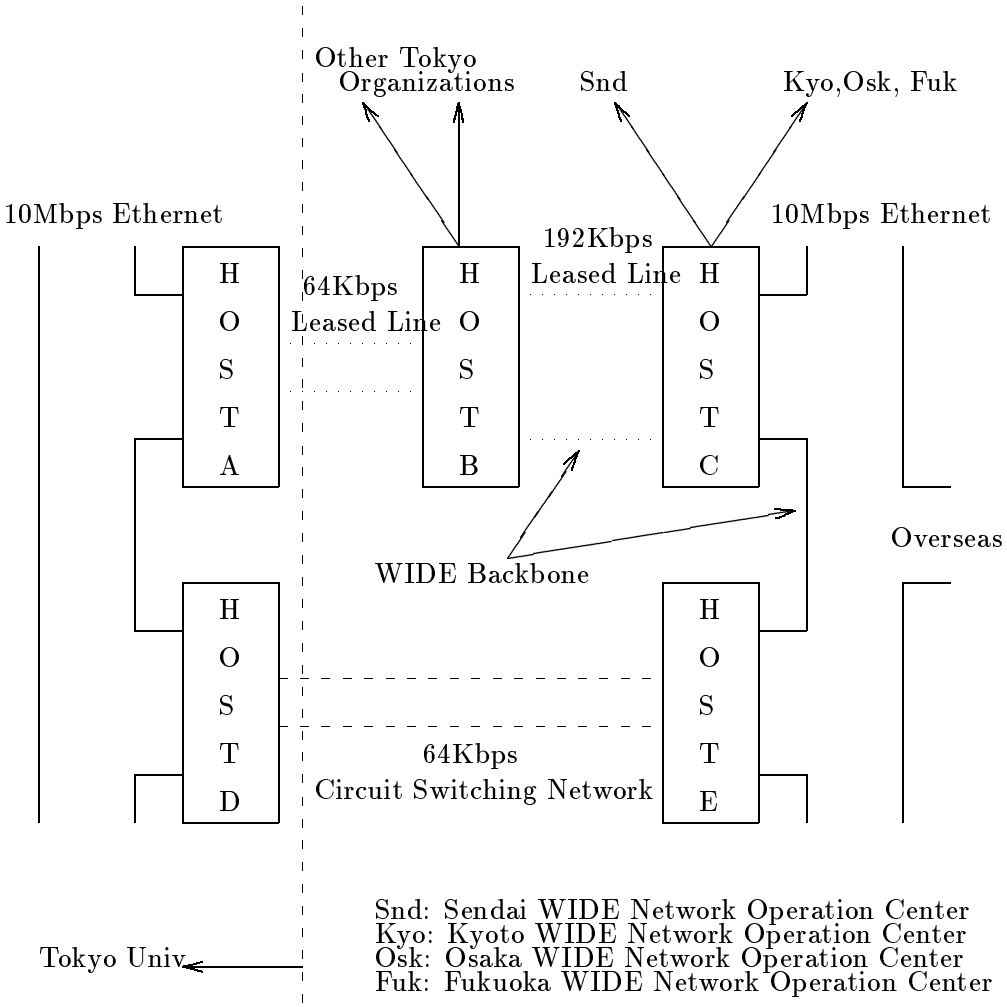
タを交換網で転送し、他のサービスは既存の回線を利用する点である。従って、交換網のみで Internet と接続される場合のように経路を交換網に設定せずに、限られたサービスのみが交換網を利用するように設定する必要がある。通常の IP プロトコルでの経路決定のアルゴリズムは各計算機が目的地と送信すべき次のゲートウェイのリストを持ち、データを送信する際は目的地から次のゲートウェイをを探し、ゲートウェイに対してデータを送信する。各計算機がこれを繰り返すことでデータが目的地に転送される。この経路制御方法に加えて IP プロトコルには、オプション機能として始点経路制御を提供している。始点経路制御とは経路上の計算機が動的に経路を決定するのではなく、経由する計算機を指定しデータを送信し、受信したゲートウェイは指定された経路に従って転送する方法である。IP プロトコルの始点経路制御には二種類あり、経由するすべての計算機を順番に明記する厳密な始点経路制御、経由する計算機の一部を順序通り明記し厳密でない始点経路制御がある厳密な始点経路制御を用いるためには経路上のすべての計算機の位置を把握していなければならない。しかし、厳密でない始点経路制御を用いれば交換網の両端の計算機のみを指定することで通常の経路を通らずにデータの転送が可能になる。この厳密でない始点経路制御を特定のサービスのみに適応することで限られたサービスのみが交換網を利用するような設定を可能とする。

実験ネットワーク

この実験は東京大学計算機センターと WIDE プロジェクトのバックボーンネットワーク間で行なった。実験に利用した機器や回線は図 5.3に示す。

東京大学のユーザが他組織の計算機にアクセスする際に WIDE プロジェクトのバックボーン (図 5.3の HOSTA,B 間) を通ることが多いために東京大学とバックボーン間のトラフィックは非常に多い。そのため他のユーザと回線利用の競合を引き起こし、Round Trip Time(RTT) が遅く、データ転送効率も低い。東京大学と WIDE プロジェクトのバックボーン間にバイパス回路 (図 5.3の HOST D,E 間) を作成することで一般のユーザと競合せずに通信が行なうことが可能となる。加えて、図 5.3のように HOST A,B 間の回線より HOST B,C 間の回線方が帯域幅が広いために、HOST A を経由して他の東京にある組織への通信も、バイパス回路を通る経路 (図 5.3での HOST D,E,C,B の経路) でデータ通信を行なうことも可能である。

図 5.3の HOST D からバイパス回路を経由してデータ転送を行なうためには、HOST D 上に HOST E へのバイパス回路の経路を設定し、IP のオプション機能の厳密でない始点経路制御を用い、HOST E を経由ゲートウェイに指定してデータを転送することで実現できる。HOST D 以外の計算機からの通信は HOST D,E を経由ゲートウェイに指定してデータを転送する。ここで問題となるのが、相手システムからのデータ転送（返送）である。上位層に TCP プロトコルを用いる場合でデータに始点経路制御のオプションが指定されている場合は、相手システムにより自動的に指定された経路を逆にして送信される。しかし、他のコネクションレスのプロトコル (UDP) の場合は、パケット単位でのデータ転送であるためにこの様なサービスはない。従って、一般的に利用できるの



HOST A	:	Proteon
HOST B	:	Sun Sun4/260
HOST C	:	Sun Sun4/260
HOST D	:	Sun Sun4/110
HOST E	:	Sun SparcStation 330
64Kbps 専用線	:	NTT Super Digital 64
192Kbps 専用線	:	NTT Super Digital 192
64Kbps 回線交換	:	NTT INS-64

図 5.3: 始点経路制御を用いた交換網利用

は TCP プロトコルを利用したサービスに限られる。

実際に利用したシステムでは始点経路指定のデータ処理に誤りがあり返送の機構が正常に働かなかったために、受信したデータが経路指定で送信されている場合は指定されている経路を逆に指定し送信するように修正したアプリケーションプログラムを利用することで実現した。また、経路上に存在するシステムがすべて正常に IP オプション (始点経路制御) のサービスを提供していることを確かめた上で実験を行なった。

5.3　システムサービスのための利用

システムサービスとして回線を接続する主な目的は、情報交換システムの利用にある。現在利用されている情報交換システムには、電子メイル、電子掲示版（ニュースシステム）などがある。情報交換を行なう際には正確に、かつ迅速に情報を伝達する必要がある。目的とする相手に対して複数の経路が存在すると、転送するデータの経路にループが生じたり、接続可能なゲートウェイの一つに対してデータを転送してもゲートウェイと組織間の接続が行なわれずにデータが転送されないことなどが生じ、目的地へのデータ転送が保証されない可能性が大きくなる。安定して目的地にデータを転送するためには経路を限定する必要があり、接続する相手を限定することが望ましい。従って、システムサービスとして交換網を利用する際には限られた相手とのみ接続する。

また、迅速に情報の伝達を行なうためにはデータが到達するとすぐに回線の接続を行ない、データを配送することが望ましい。しかし、情報毎に回線を接続するとコストがかさみ、場合によっては専用線を用いた方が安価となり交換網を用いる理由がなくなる。従って、ある程度の情報が集まった時点で回線を接続し転送を行なう必要がある。通信のデータ単価を最小とするには料金体系における時間と回線の伝送速度を乗じた量の送信データが集まった時点で回線を接続する方法である。しかし、転送に必要なデータ量が集まるまでの時間が不定であり、伝送遅延が大きくなり過ぎてしまう可能性がある。安定して情報伝達を行なうには最大伝送遅延をある値に収束させる必要があり、転送するデータ量が少なくとも定期的に回線を接続しデータ転送を行なう必要がある。

実験ネットワーク

実験は青山学院大学とテレマティーク国際研究所間で図 5.4に示すネットワークで実験を行なった。

両組織間での情報交換に UUCP-T プロトコルを用いた。UUCP-T プロトコルは TCP/IP 上で UNIX の通常のログイン手続きを行ないファイル転送を行なうものである。UUCP では転送すべきファイルは一旦自システム上に保存され、相手システムへのアクセスを行なった際に保存されていたファイルが転送される。従って、相手システムへのアクセスを定期的に行なうことで安定したファイル転送が可能となる。定期的にアクセスを行なう時期を調整することで経済性を向上させることができるが、逆に保存されている時間が伸びることにより伝送遅延が大きくなる。

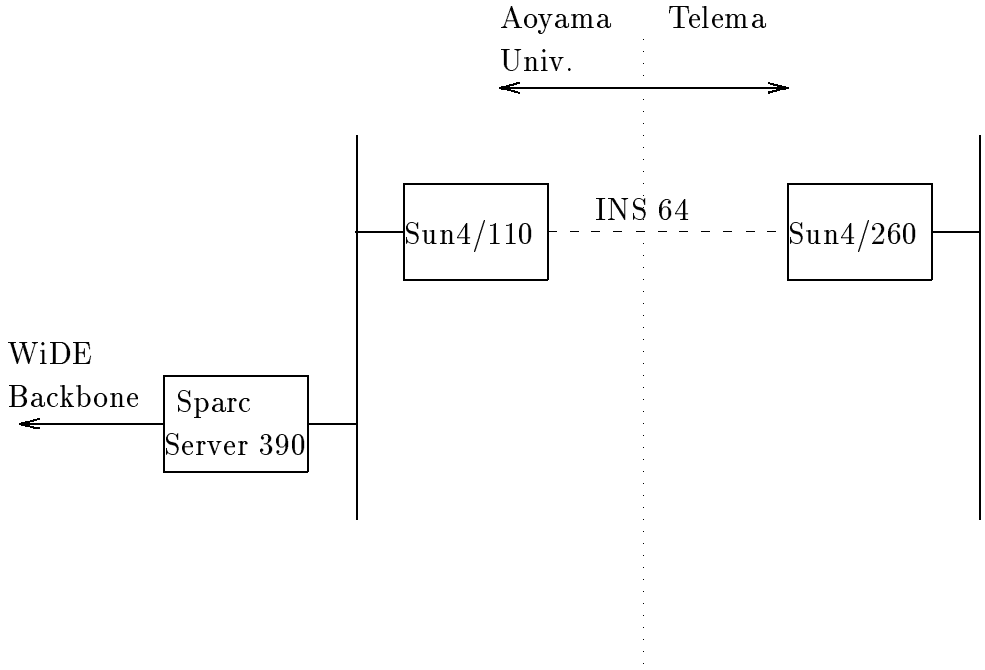


図 5.4: UUCP-T プロトコルを用いた組織間情報交換

もちろん、局間距離や料金体系の変更により計算は異なるが、回線料金と伝送遅延のトレードオフである。伝送遅延のみを考えると最も低価格な専用線の利用が有効な手段であるが、コストなどの面から接続時間を最小限にするために交換網が必要となる。この実験での最も重要な点はこのトレードオフの限界点を調べることである。

また、回線を利用することで利用料金を支払う責任があり、一般のユーザや組織外部のユーザが不必要に回線を利用することで莫大な料金の支払うことになる。そのために回線の利用に制限をつけ、回線を保護する必要がある。保護の方法には消極的な保護と積極的な保護の二つの方法を用いた。回線を接続し相手との通信を行なうためには相手への経路を知っている必要がある。この経路情報をゲートウェイが広報せず、かつ、自動による回線接続を行なわないことで消極的に保護を行なう。積極的な保護は回線交換網を持つ計算機でゲートウェイの機能を遮断する方法である。実際には IP のデータ転送機能を止めることで実現でき、その計算機を通過するには一旦その計算機にログインし再び目的地へ向かうことでのみ通過することができる。消極的な保護は青山学院大学側で行ない、積極的な保護はテレマティーク国際研究所側で行なった。

5.4　すべてのサービスでの利用

すべてのサービスを交換網で提供することで、回線上を流れるデータ量が増加し、回線を切断することができなくなる可能性が生じる。常時接続される状態となると交換網を利用するより専用線を利用した方が経済性の面で有利となる。対策として利用時間を限ることで、専用線に対する優位性を保つことができる。一日あるいは一定時間内での利用時間が限られている場合には交換網利用が適応できる。

接続する二つの計算機の位置が異なることで利用者の要求度合が異なり、構築するネットワークの形態が異なる。従って、交換網を利用することのみで Internet に接続する場合と、既に Internet 内部に接続されている場合に分けて実験ネットワークを構築する。

5.4.1　交換網のみにより Internet と接続する場合

設計方針

Internet 外部の計算機から利用時間を限って利用するのは学会の研究会や展示会など昼の時間帯のみ利用する場合などが挙げられる。研究会や展示会などでの利用は開催時間のみで、日中 (9:00AM-5:00PM) を中心に限られた時間のみに集中してデータ転送が行なわれる。また、限られた条件でのみ利用される場合には既存の回線との負荷分散がある。

実験ネットワーク

実験として以下の三つの場合において実験ネットワークを構築した。

1. 1990 年 12 月に新宿 NS ビルで開催された展示会 JUS[1] Unix Fair'90 と WIDE バックボーンを接続し実験を行なった。アドレスは JUS が取得した展示会用のアドレスを用い、接続中は gated による経路制御情報 (RIP[2]) の交換を行ない経路を伝達した。

   この展示会ではネットワークでの接続実験などを行なっており、開催中、外部とのデータ転送量が多いことが予想された。そのため、接続や切断に必要な時間がオーバヘッドとなり、料金面でも基本時間内での接続切断の繰り返しによる料金の増加などが起こる可能性が高いために、fair 開催中は常時接続する方法、つまり、自動回線制御を用いず手動による回線制御を行なった。

2. 1991 年 7 月に北九州で開催された Joint Workshop for Computer Communication(JWCC) の際に端末ルームを設置し、この端末ルームと WIDE プロジェクトのバックボーンを接続して実験を行なった。接続の方法と接続に用いた機器を図 5.5に示す。アドレスは WIDE バックボーンのサブアドレスを利用し、gated による経路制御情報 (RIP) の交換を行ない経路を伝達した。

[1] 日本ユニックスユーザ会
[2] Routing Information Protocol

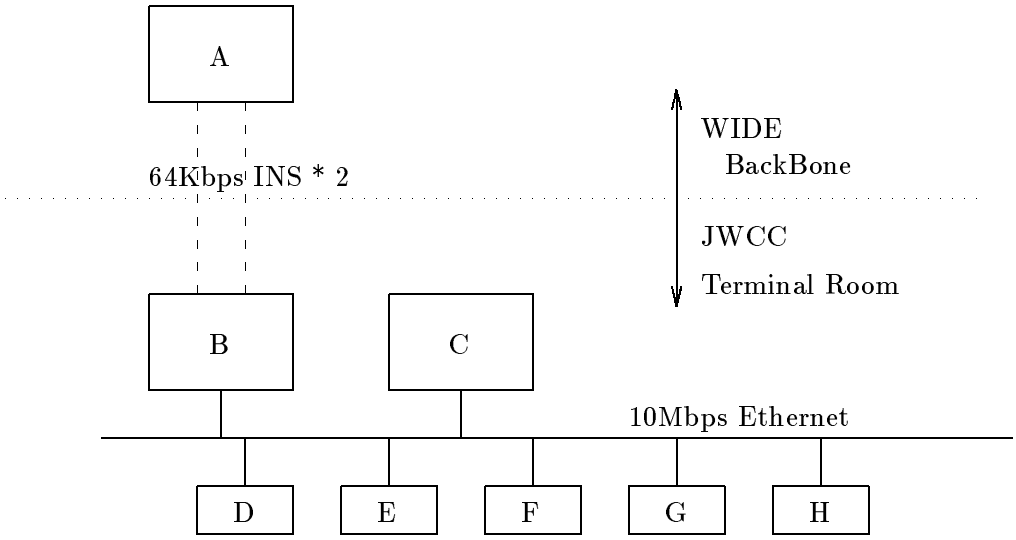


図 5.5: JWCC での接続実験

この会議はアジア地域でのコンピュータネットワークの会議で、主にネットワークの管理にたずさわる出席者が多く、会議開催中も各地のネットワークへのアクセスを可能にする必要があったために、自動回線制御を行なわず手動による接続切断で会議開催中は常時接続状態に保った。

この実験では会議の休憩時間におけるトラフィックの急激な増加に対応するために、回線を二回線同時に利用することによる負荷分散の実験を行なった [?]。二本の回線を両端で同一の計算機に接続し、作成した負荷分散ネットワークインターフェイスを用いて負荷分散を行なった。負荷分散ネットワークインターフェイスは出力できるネットワークインターフェイスを下位層に位置させ、複数のネットワークインターフェイスに出力データバイト数が均等になるように出力した。

3. 1991 年 10 月に河口湖で行なわれた WIDE プロジェクトの合宿と上智大学を接続して実験を行なった。接続の方法と接続に用いた機器を図 5.6に示す。アドレスは上智大学のアドレスを利用し、gated による経路制御情報 (RIP) の交換を行ない経路を伝達した。

   WIDE 合宿の参加者は各ネットワークを管理運営している技術者が主であるために各組織へのアクセスを可能とする必要がある。もし、新しいネットワークアドレスを用いれば、新たな経路情報を広報する必要があり、フィルタリングなどを行なうために静的に経路を設定しているゲートウェイを経由することができない。従って事前に広報されているアドレスで使用されていないアドレスを用いる必要がある。この時既に広報されており利用されていないアドレスは WIDE のバックボーンのサブアドレスのみであった。バックボーンのサブアドレスを利用するためにはバックボー

ンに接続されている計算機に直接接続するか、サブネットの経路情報を広報する方法のどちらかである。後者の場合経路変更を必要とし、ネットワーク管理者が殆んど集まっている状態で遠隔地に位置する計算機の経路設定を変更することが危険であると判断したために利用できなかった。前者の方法については東京近辺で WIDE バックボーンに直接接続されている計算機で ISDN の接続が可能な計算機がなかったために利用できなかった。この様な理由から上智大学のサブネットを利用した。

5.4.2　既に Internet と接続している場合

設計方針

Internet 内に存在する組織がすべてのトラフィックを交換網に対して送出する理由には、専用線などの既存の回線のバックアップと既存の回線容量を越えたトラフィックが生じた際の負荷分散の二つがあげられる。負荷分散に関しては 5.5節の 2で実験しているため、ここではバックアップ機能のみを実験する。

既存の回線のバックアップとして交換網を利用するのは、Internet に接続されている既存の回線に障害が生じ、Internet から隔離されてしまった場合である。ここでの問題点は、障害の自動検知と自動復旧である。自動的に障害の発生や復旧を検知するための方法は、

消極的バックアップ 回線を持つ計算機の申告による方法

積極的バックアップ 折り返し検査による方法

の二つがあげられる。前者の方法を用いる場合は計算機や回線の負荷による遅延を考えておく必要があるために早急な対処ができない。後者の方法を用いる場合は折り返し検査によりその時点での回線の状態が把握できるが、折り返し検査によるトラフィック増加につながる。この実験では両端で異なる方法を用いて実験を行なった。

実験ネットワーク

実験は WIDE プロジェクトのバックボーンと上智大学間で行なった。接続の方法と接続に用いた機器を図 5.7に示す。

WIDE プロジェクトのバックボーン側では、gated を用いて経路制御情報の交換によりバックアップシステムへのデータ転送を行なった。WIDE 側では交換網と専用線が接続されている計算機が同じネットワーク上にあるために、交換網を持つ計算機は専用線を持つ計算機より多いメトリックの経路情報をネットワークに流し続ける。専用線に障害が生じた場合や、専用線を持つ計算機に障害が生じた場合は、専用線を通る上智大学への経路情報が流れなくなり、高いメトリックの交換網を利用した経路が優先される。障害が復帰した場合は、専用線を持つ計算機から低いメトリックの経路情報が流れ、この情報が交換網を持つ計算機を通る経路より優先され、すべてのデータは専用線を経由して転送される。

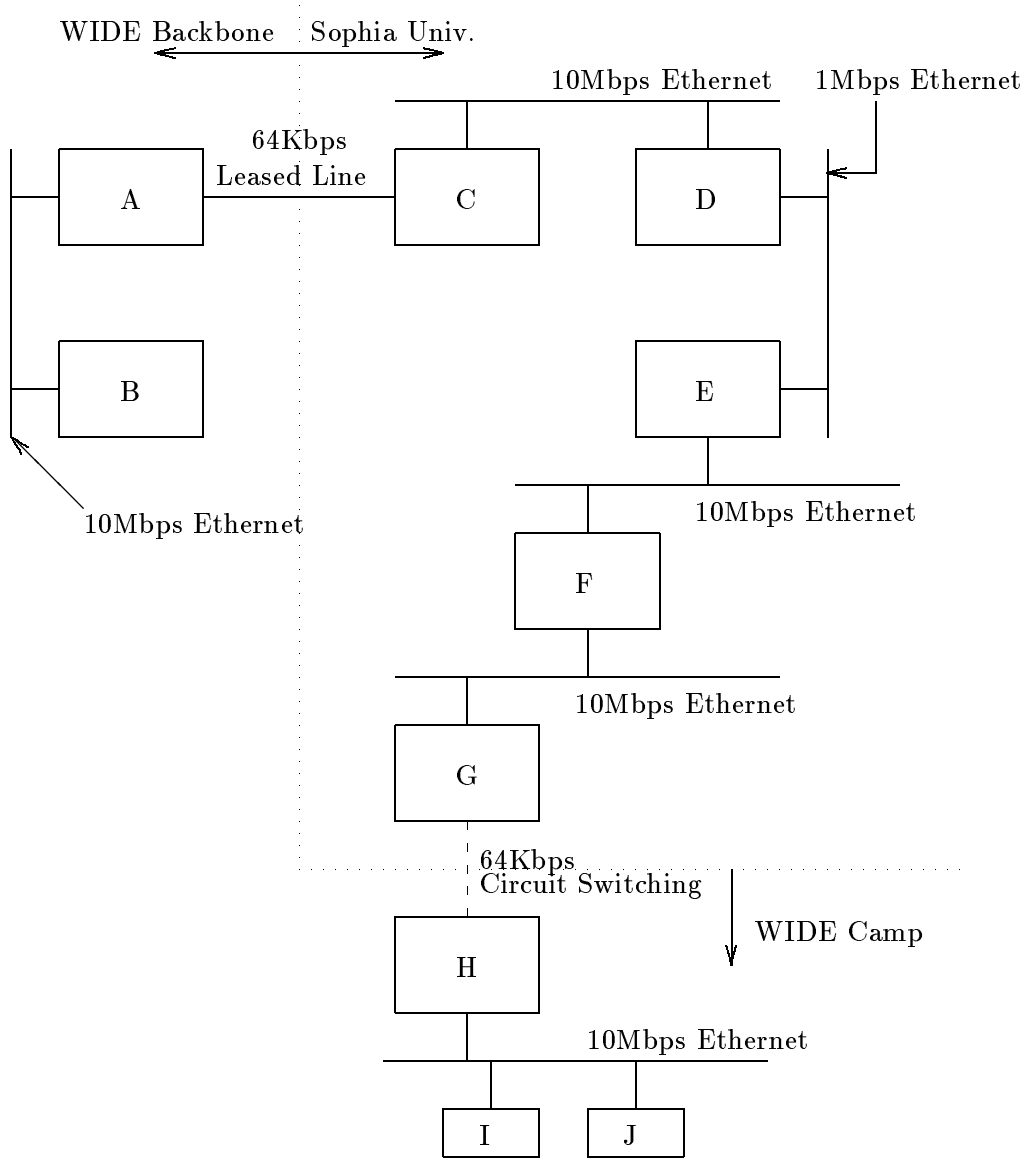


図 5.6: WIDE 合宿での接続実験

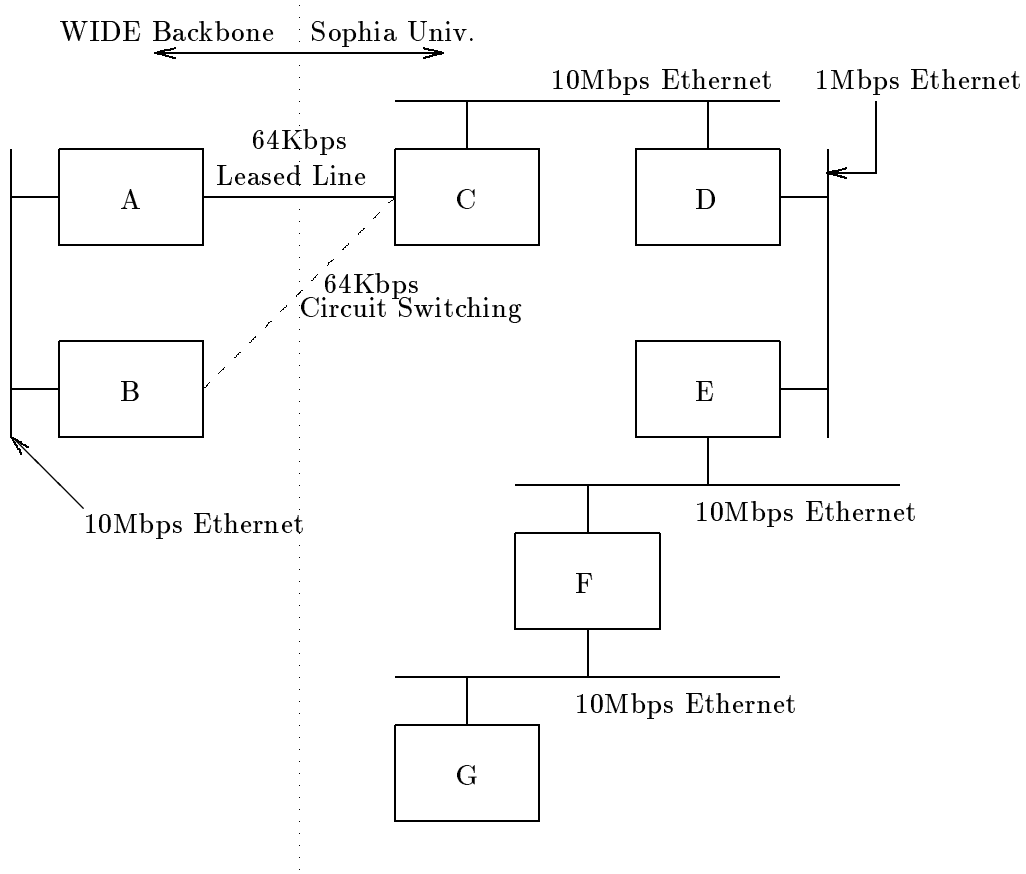


図 5.7: 上智大学と WIDE バックボーン間でのバックアップシステム

上智大学側では同じ計算機に交換網と専用線を接続している。よって、交換網を利用する場合でも、専用線を利用する場合での上智内でのデータはすべて上智のゲートウェイである HOST C に集まるため、経路制御による経路変更は意味を持たない。ここで有効となる方法は、インターフェイスのメトリックを利用する方法と折り返し検査を行なう方法の二通りがある。上智大学側では申告による方法を用いず、積極的に回線の状態を調べる折り返し検査の方法を用いた。まず、HOST C から A に対して折り返し検査を行なう。折り返し検査で障害を検知した場合、通常のデータが転送される経路を HOST C 上で B に向けて変更する。この間 HOST C,A の折り返し検査は継続する。しかし、通常の折り返し検査を行なえば、C,B,A を通り折り返し検査が成功してしまう。そのため C,A 間の折り返し検査は C,A 間を必ず通るように始点経路制御を用いて行った。

第 6 章

評価と課題

6.1 結果と評価

自動回線制御機構

ISDN 回線交換網を利用した場合 (表 4.9参照) にモニタモジュールを加えて表 6.1で示す構造で自動回線制御をモニタした。この結果より回線接続に関するデータをまとめて表 6.2に示す。NNT が提供する 64Kbps の専用線 (Super Digital 64) を用いて、同期通信のみのプロトコルを利用してデータ転送を行なった場合の ftp によるファイル転送も 7.4KB/s で ping による RTT も 30ms である。ISDN 回線が接続し、XSP プロトコルが両システムでの協議が完了しデータを転送し始めるまでの時間の比率も極めて低く、XSP などの point-to-point プロトコルの利用によるオーバヘッドも少ない。

また、モジュール分割や統一したインターフェイス機構のオーバヘッドを測定するために、回線に INS64 の回線交換網を用い、計算機に Sun Sparc Station 330 を用いて、モジュールに分割せずデータグラムのフレーム化、point-to-point プロトコルの解釈を一つのネットワークインターフェイスとしてオペレーティングシステム内部に実現した場合とユーザプロセス空間で実現した場合、モジュールに分割しオペレーティングシステム内部で実現した場合の計測を行なった。この測定結果を表 6.3と図 6.1に示す。図 6.1、表 6.3でのデータ出力に必要な時間は、データが回線上にすべて送出されるまでの時間計測である。データ入力に必要な時間は DMA を用いて転送を行なったために DMA が終了

表 6.1: モニタモジュールを加えて ISDN 自動回線接続

ntd0
duc0
xsp0
ntd1
xcs0
同期モジュール

表 6.2: 図 4.2 を用いた場合の接続結果

	同局内	東京藤沢間
INS 回線接続時間	0.946 s	2.908 s
データ通信が可能になるまでの時間	0.2 - 0.3 s	
ftp によるファイル転送	7.4KB/s	
ping による RTT	30ms	

表 6.3: 64Kbps の ISDN 回線でのデータ入出力に必要な時間

データ長 (Byte)	102	150	500	1000	1500	2000
出力	0.010488	0.019390	0.062100	0.138432	0.154589	0.194890
入力	0.000210	0.000582	0.000680	0.000280	0.000550	0.000360

オペレーティングシステム内部でモジュールに分割しない場合のデータ処理時間 (s)

データ長 (Byte)	102	150	500	1000	1500	2000
出力	0.010943	0.020858	0.065490	0.140000	0.168490	0.198904
入力	0.000344	0.000980	0.000534	0.000458	0.000434	0.000540

オペレーティングシステム内部でモジュールに分割した場合のデータ処理時間 (s)

データ長 (Byte)	102	150	500	1000	1500	2000
出力	0.755878	0.818284	0.867008	0.978274	1.052631	1.137280
入力	0.734449	0.784821	0.815060	0.839183	0.893840	0.932850

ユーザプロセスでモジュールに分割しない場合のデータ処理時間 (s)

した後からデータ処理に必要な時間のみを計測したため、データのプロトコル処理に必要な時間を表す。図をみて明らかなようにユーザプロセスでの実現に比べてオペレーティングシステム内部での実現の方がパフォーマンスが良く、オペレーティングシステム内部での実現でモジュールに分割した場合としない場合を比べてもプロトコルデータ処理時間にほとんど差がなく、モジュール分割によるオーバヘッドもみられない。

相手の網内でのアドレスがテーブル内部になく、ユーザプロセスに対して問い合わせを行なう場合は表 6.2の結果に最小で約１秒加える必要がある。しかし、頻繁に接続を行なう相手の情報をテーブル内に保存しておくことでこのオーバヘッドを避けることができ、通常利用しない相手のみがこのオーバヘッドが発生する対象となるが、数秒の接続の遅延のみでネットワーク上に登録されているすべての相手との接続が可能になり大規模性の充分あるシステムとなる。

加えて、XSP プロトコルを用いることで自由度を持たせることができ、OSI, XNS, TCP/IP などの複数のプロトコルで利用が可能となり、自動回線制御機構を用いること

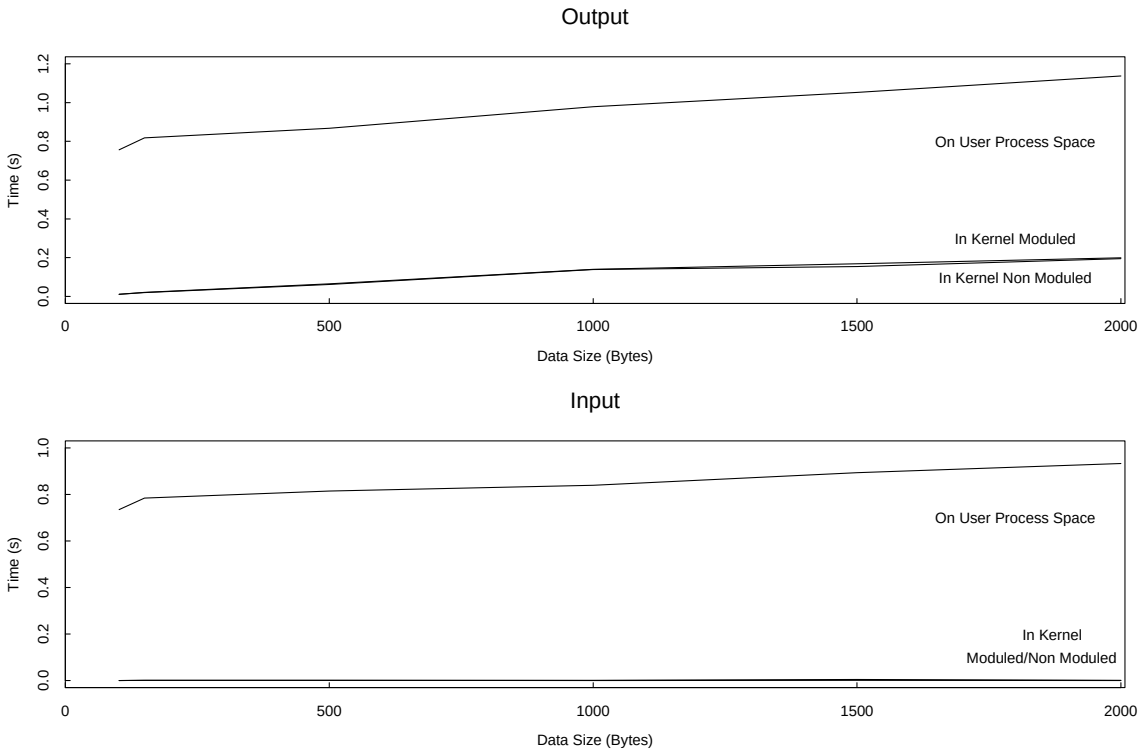


図 6.1: 64Kbps の ISDN 回線でのデータ入出力に必要な時間

で既存のネットワークインターフェイスと互換性を持ちアプリケーションに一切変更を加えず利用できるシステムとすることができた。

個人的利用を目的としたネットワーク

個人的な利用を目的とし Internet 外の場合の結果は表 6.2と全く同じである。これは経路情報の伝達、設定など接続後の処理が一切不必要であるために、接続した瞬間からすべてのサービスが利用できるためである。

Internet 内の場合は比較的回線が空いていると予想される年末の 12 月 28 日に、図 5.3で示されるネットワークで RTT と利用されている回線状態を計測した。図 5.3の D,A,B,C 間、D,E,C 間、D,A,B 間、D,E,C,B 間で RTT を計測した結果を図 6.2に示す。D,C 間でバイパス回路経由の場合は最大 60ms でばらつきがほとんどないのに比べ通常の経路を通る場合は最大で 3.5s でばらつきも激しい。D,B 間でもバイパス回路を経由する場合は最大で 1s、平均 60ms であるのに対して通常の経路では最大で 3.5s、平均 82ms でばらつきも大きい。A,B の二地点間のトラフィックを traffic(A,B) とすると、D,C 間ではバイパス回路が 64Kbps をすべて利用可能であるのに対して、通常の経路では 64Kbps - traffic(A,B) と 192Kbps - traffic(B,C) の少ない方の帯域が利用可能である。A,B 間のトラフィックは最大で回線能力の 64Kbps に達しているため、この間に利用できる通信帯域は 0 となり、明らかにバイパス回路を利用する方が利用可能な帯域幅は広い。D,B 間ではバイパス回路では 192Kbps - traffic(B,C) と 64Kbps の小さい方で通常の経路では 64Kbps - traffic(A,B) である。traffic(A,B) では 64Kbps の帯域のほぼ全部を利用している時間帯があるにも関わらず、B,C 間では 100Kbps 以上利用していることがなく残りの 90Kbps は利用可能であり、D,E 間でも 64Kbps が利用可能である。従って、D,B 間でもバイパス回路を利用した方が広い通信帯域を確保することができる。

この様にバイパス回路を作成し利用することで他のユーザと競合せずに回線を利用できるため、高速な通信や高速な応答を得ることができる。また、ネットワークの状態によっては目的地より遠い地点とバイパス回路を作成しても高速で応答の早い通信が可能になる。

システムサービスを提供するためのネットワーク

Internet 外の組織間との接続に関しての１日の接続時間と転送データ量を表 6.4に示す。この実験では自動切断のタイマを 100 秒に設定し、午前 8 時から午後 8 時までは一時間に二回、その他の時間帯は一時間一回の接続を行なった。

支払いを片側に集中させるために、片側からのみの接続を行なった。従って、自システムから転送するデータが存在しなくとも相手側と接続を行ない、相手側から転送すべきデータかどうかの確認を行なう必要があるために、転送データが 0Byte の接続が生じた。もし着信課金が可能であれば、転送すべきデータがある時のみ接続を行ない、相手システムでは接続がなかった場合で送信すべきデータが存在する場合のみ着信課金サービス

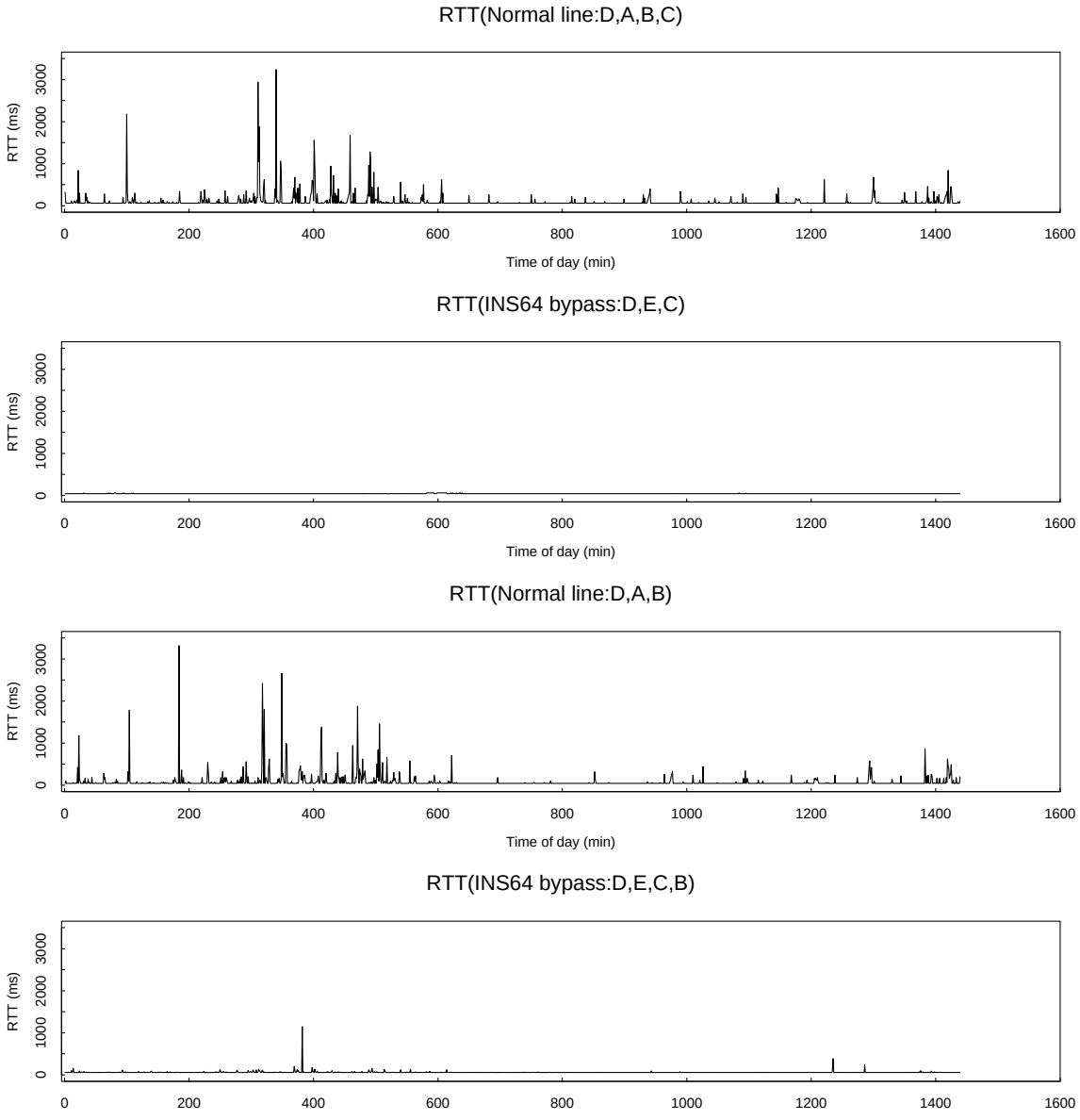


図 6.2: 図 4.3 における RTT の計測

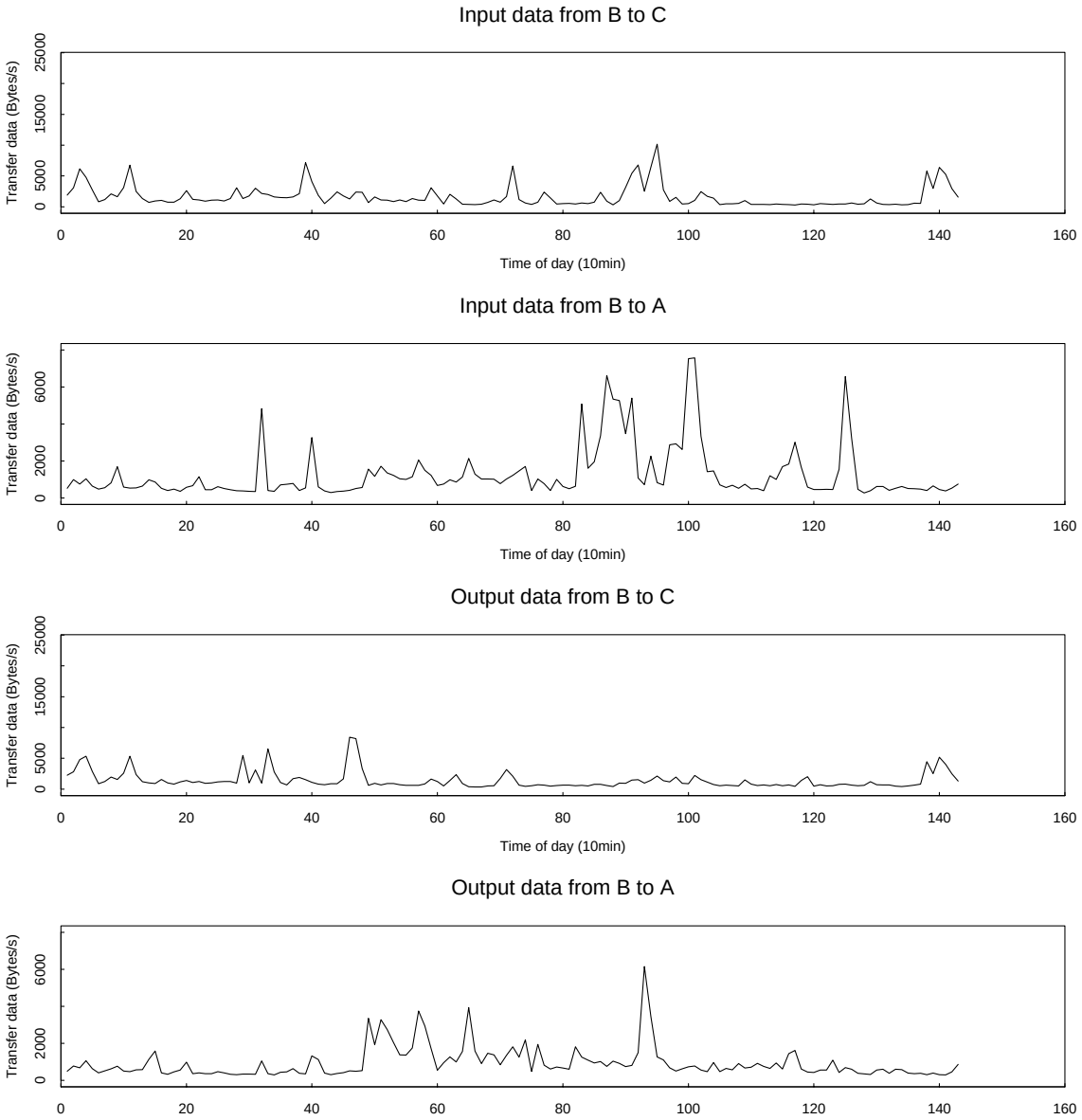


図 6.3: 図 4.3 におけるデータ転送量の計測

表 6.4: UUCP-T プロトコルを用いた一日の接続結果

接続開始時間	回線切断時間	転送データ量 (Bytes)
Feb 25 00:20:21	Feb 25 00:22:00	11248
Feb 25 01:20:03	Feb 25 01:21:42	7482
Feb 25 02:20:02	Feb 25 02:21:42	7075
Feb 25 03:20:05	Feb 25 03:21:44	43714
Feb 25 04:20:03	Feb 25 04:21:42	30104
Feb 25 05:20:03	Feb 25 05:21:42	7832
Feb 25 06:20:02	Feb 25 06:21:41	0
Feb 25 07:20:02	Feb 25 07:23:21	101674
Feb 25 08:20:02	Feb 25 08:21:42	8234
Feb 25 08:50:03	Feb 25 08:51:43	0
Feb 25 09:20:02	Feb 25 09:21:42	29858
Feb 25 09:50:03	Feb 25 09:51:43	0
Feb 25 10:20:03	Feb 25 10:21:42	7596
Feb 25 10:50:02	Feb 25 10:51:42	0
Feb 25 11:20:03	Feb 25 11:22:32	1453199
Feb 25 11:50:03	Feb 25 11:51:43	5292
Feb 25 12:20:02	Feb 25 12:24:12	1076613
Feb 25 12:50:04	Feb 25 12:51:44	381530
Feb 25 13:20:02	Feb 25 13:24:12	908857
Feb 25 13:50:03	Feb 25 13:51:43	599200
Feb 25 14:20:02	Feb 25 14:21:42	7462
Feb 25 14:50:03	Feb 25 14:51:43	0
Feb 25 15:20:02	Feb 25 15:20:52	0
Feb 25 15:50:03	Feb 25 15:51:42	15032
Feb 25 16:20:02	Feb 25 16:21:42	7743
Feb 25 16:50:03	Feb 25 16:51:43	0
Feb 25 17:20:20	Feb 25 17:22:00	27741
Feb 25 17:50:03	Feb 25 17:51:43	2319
Feb 25 18:20:02	Feb 25 18:21:42	12035
Feb 25 18:50:03	Feb 25 18:51:43	680230
Feb 25 19:20:02	Feb 25 19:21:41	28824
Feb 25 19:50:20	Feb 25 19:52:00	7100
Feb 25 20:20:02	Feb 25 20:23:22	136202
Feb 25 20:50:03	Feb 25 20:51:42	0
Feb 25 21:20:02	Feb 25 21:21:41	8684
Feb 25 22:20:02	Feb 25 22:22:32	52040
Feb 25 23:20:03	Feb 25 23:21:42	52363

表 6.5: UUCP-T を用いて効率改善を行なった場合

	料金 (円)	式 6.1	式 6.2
1 回/1 時間	270	0.183	0.101
1 回/2 時間	150	0.287	0.174
1 回/3 時間	100	0.309	0.170

を利用して接続を行なうことができる。

一回の接続でかかる費用は、

$$\text{接続時間} = \frac{\text{情報転送量}}{\text{回線速度}}$$

$$\text{回線料金} = \text{切り上げ}\,(\frac{\text{接続時間}}{\text{基本時間}}) * \text{基本料金}$$

である。表 6.4で 10 円/180 秒の回線料金で一日当たりの回線料金を計算すると 380 円となる。着信課金サービスを利用した場合だと 80 円安くなり一日当たり 300 円となる。

一日一回の接続では、

$$\text{回線料金} = \text{切り上げ}\,((6MB/64Kbps)/180s) * 10\ \text{円}$$

$$= \text{切り上げ}\,(((6 * 10^6 * 8)/(64 * 10^3))/180) * 10$$

$$= 50\ \text{円/1 日}$$

となり低価格での転送が実現できる。通信効率や料金効率を表 6.4を元に計算すると、

$$\text{平均}\,(\frac{\text{通信時間}}{\text{接続時間}}) = 0.135 \tag{6.1}$$

$$\text{平均}\,(\frac{\text{通信時間}}{(\text{切り上げ}\,(\text{接続時間}/\text{単位時間})) * \text{単位時間}}) = 0.093 \tag{6.2}$$

となる。式 6.1が接続時間を元に通信効率を計算した場合で、式 6.2が料金を元に通信効率を計算した場合である。式 6.2を見ても明らかなように 1 日当たり 6MB のデータでは日中に 2 回/1 時間、その他の時間帯には 1 回/1 時間の接続は非効率的である。表 6.4の転送データを元に 1 回/1 時間の接続、1 回/2 時間の接続と 1 回/3 時間の接続を行なった場合の料金と式 6.1, 6.2の値を計算してみると表 6.5のようになる。64Kbps の回線を用い 180 秒が基本単位の場合には、基本単位時間で転送できるデータ量は 1.4MB となり、6MB を転送するためには約 4 回基本時間接続すれば良い。しかし、通信するデータ量のばらつきが大きいために、1 回/2 時間と 1 回/3 時間にはほとんど差が見られない (表 6.5 参照)。従って、二時間に一回の接続が効率と伝送遅延の両方の要求を最も満たしていると思われる。

実験は Internet 外部の組織との間で行なった。この結果を Internet 内部の組織に適応し、上智大学と WIDE プロジェクト間に当てはめて考えてみる。一般に転送される一通の電子メイルのデータ量は少ない。また、電子メイルの伝送遅延はユーザにとって重要な問題である。従って電子メイルでの通信は既存の回線を利用し、電子掲示版など伝送遅延を生じても問題の少ないデータのみをまとめて交換網で転送する。

12 月 27 日の上智大学と WIDE プロジェクト間の入出力データを図 6.4に示す。回線が常時接続されいる状態では電子掲示版のデータを転送するために NNTP プロトコルを用いる。NNTP は対話型のアプリケーションで相手システムが持つ最新のデータを転送することができる。図に示される様に電子掲示版のデータ転送を行なう NNTP プロトコルのデータを除くことで送受信ともに 1,2 回の例外を除き定常的なデータ転送となる。電子掲示版の情報は上智大学側より転送するデータが 4.3MByte で、上智大学側へのデータが 33.6MByte である。仮に、このデータを一日一回で 64Kbps の交換網を利用して転送したとすると約一時間利用することになる。NNTP プロトコルのデータを除くことにより既存の回線の利用度が著しく減少する。NNTP プロトコルでのデータのみを交換網を用いて転送することにより、専用線のクラスを下げ、安価に回線の維持ができる。

しかし、実際に NNTP プロトコルで電子掲示版の情報の転送を行なえば、一日分の保存されたデータを転送する間に転送すべき新しい情報が次々と生じ、回線の切断がなかなか行なわれないことになる。従って、NNTP プロトコルの様な LAN 上での情報転送方式ではなく、UUCP-T の様に情報を一旦保存し、既に保存されていた情報のみを転送するプロトコルを利用する必要がある。

すべてのサービス提供するためのネットワーク

学会の研究会や展示会などのように一時的に Internet と接続を行なう場合では利用時間が限られているために専用線を用いるより安価に接続できる。また、開催期間が短ければ臨時設備の工事料金や基本料金も専用線に比べて公衆回線交換網の方が安くなる。

負荷分散を行なうために二本の回線を接続して出力データ量を同じにする様に分散させる簡単な手法を用いた場合でも、ftp によるファイル転送が 15KB/s と一回線を利用する際のほぼ二倍の転送効率が得られた。

既存の回線に障害が生じバックアップを行なうために交換網を利用した実験では、回線異常の申告によるバックアップを用いた場合は回線に異常が生じてから約 1 分後にバックアップ回線の経路が有効となり通信が可能になる。折り返し検査により回線の障害を検知する方法では、10 秒間隔で折り返し検査を行ない 3 回以上連続して失敗した場合にバックアップ回線を接続し経路を変更した。そのために異常発生後 30 秒で通信が可能となった。しかし、この折り返し検査の失敗とは折り返し検査のデータを出力して 10 秒間以内に結果が戻らない場合であるため、連続した大量のデータが回線を流れている場合などで、たまたま折り返し検査に失敗し、回線や相手側の計算機に障害が生じていない場合にも、バックアップ回線の接続が行なわれた。

図 5.7での C,B 間でのバックアップではなく、G,B 間でバックアップを行なう実験を行

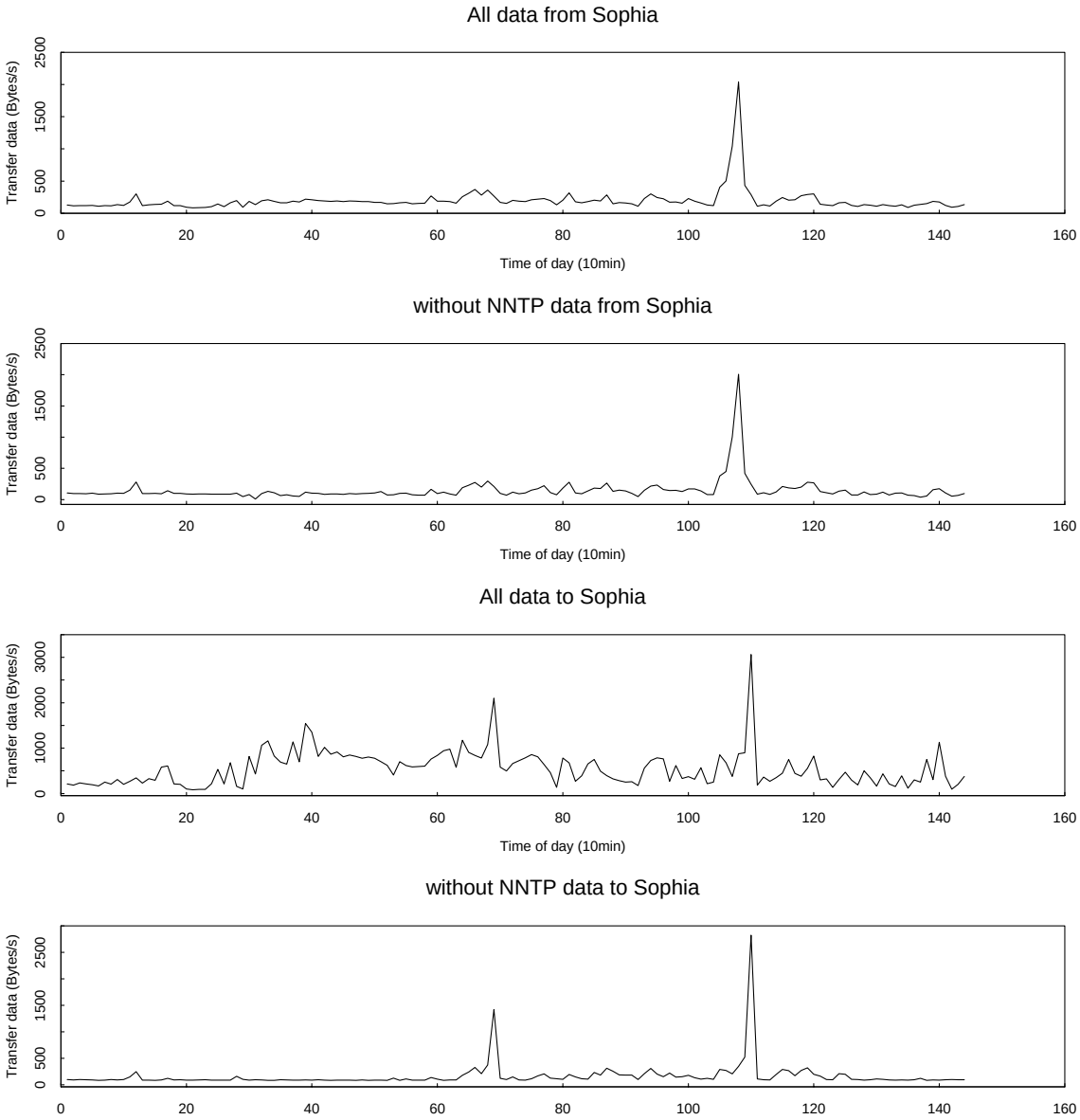


図 6.4: 上智大学と WIDE プロジェクト間のデータ転送量

なった。この実験ではバックアップ回線が接続されるまでの時間は 30 秒で同じであったが、新しく作成されたバックアップ回線への経路情報が上智大学内部に伝達されるまでに時間がかかり、計算機 C では約 3 分後に通信が再開できた。この様に既存の回線とバックアップ回線を持つ計算機の間が離れていることで経路情報の伝達が遅れ、バックアップ回線が接続されたにも関わらず利用されない状態が長く続くことになる。

6.2　課題

自動回線制御機構

一定間隔でデータの入出力を監視し変化が生じなければ回線を切断する自動切断機構を用いた。この間隔の設定は動的に行なえるが、接続している両システムでこの自動切断の機能が働いている場合は両システムの間隔が短い方が優先され、設定値を少くすることは有効であるが、設定値を大きくしても相手システムの設定値も同時に大きくしない限り有効とはならない。実験を行なう際は受け側となるシステムの値を大きくし、発信側で制御する方法を用いた。しかし、これは本質的な解決方法ではなく、接続を行なう際に自動切断を行なうための間隔を互いのシステム間で協議すべきである。

自動切断のもう一つの問題点は、折り返し検査や分散型データベースの扱いである。定期的に折り返し検査を行ない相手システムの状態を調べるアプリケーションなどが存在することで自動切断が不可能となる。また、データ更新の度に分散したデータベースを更新するのではなく、定期的に情報を交換することで分散データベースを最新に保つ分散型システムを利用した場合にも回線が切断されなくなる。この様な定期的な折り返し検査やデータベース更新は LAN 上でのアプリケーションが主に用いる方法で、広域ネットワークではトラフィックの増加につながり、回線障害を引き起こす可能性もある。交換網のみならず、広域なネットワークではこの様な不必要に定期的なデータ交換を行なうことは避けなければならず、アプリケーションに修正が必要である。

アドレス変換での課題は、アドレスの変換を行なう際に自動発信を行なうか自動着信を行なうかをフラグを用いて決定したことである。一般的にはこの様な静的な設定による方法で十分である。しかし、長距離に発信する場合などは夜間のみ発信するなど、条件つきの自動発信や着信が考えられる。この様な場合には静的な設定では対応できない。従って、フラグによる制御ではなく関数による制御が望ましい。

モジュールを重ね合わせることで異なるプロトコル階層での利用が容易になった。しかし、プロトコルの重ね合わせが異なれば両端での通信ができない。相手との通信を行なう以前に相手の識別が可能である場合は着信後すぐに相手に従ったプロトコル階層に入れ換えることで通信ができるが、通信以前に相手の識別を行なう方法がない場合は、相手が識別できず相手によりプロトコル階層を変えることも不可能である。この様に全く異なるプロトコル階層の相手と通信するためには、通信プロトコルの最下層で上位のプロトコル階層を相手に示す必要がある。しかし、この方法でも電気的な信号からデータグラムへの変換方式が異なることで通信できない可能性が生じる。従って、異なるプロ

トコル階層を用いた相手との通信を行なうためには、受信した電気信号から相手のプロトコル階層を推測するモジュールを付け加える方法しかなく、今後、複数のプロトコル階層を区別するための識別子などが利用できることを期待するのみである。

個人的利用を目的としたネットワーク

最近ではポータブルな計算機が増加している。自分の計算機を持ち運び移動先で交換網に接続し利用することが予測される。移動先から接続を行なう相手として最もふさわしい接続相手は最寄りの組織である。この様な時に図 5.2に示す接続方法を用いるには、事前に移動先から最寄りとなる組織に対して、移動先の電話番号を申請し移動先の最寄りとなる組織からアドレスを取得する必要がある。接続を行なう際に用いる point-to-point プロトコルに PPP を用いることでパスワードによる認証が可能となり、移動先の電話番号を申請せずとも認証を受けることができる。認証を受けた後に動的に接続先のシステムから未使用のアドレスを与えられることで、事前にアドレスを取得する必要もなくなる。従って、今後は PPP プロトコルでの動的なアドレス割当を可能とする必要がある。

自動発信機能を持たすことで広い範囲での利用が可能となった。しかし、回線を接続し相手システムで送出するデータが生じるようにしておき、回線を切断すると、相手システムは発生したデータを出力するために回線接続を行なう。Call Back 機能の実現である。個人的な利用のために Call Back が行なわれると、相手組織が料金の殆どを負担しなくてはならない。これを防ぐためには着信課金サービスを利用している相手のみ自動発信を行ない、その他のシステムは自動発信を行なわないように設定しておく必要がある。ここで問題となるのが自動発信を拒否されたデータの対処である。送信元に何の通知もせずにデータを破棄する方法と、エラーとして送信元に送り返す方法の二通りある。データを破棄すれば送信元は状態が把握できず、再転送を繰り返すことになる。エラーを返すためには受信したデータを発信作業が終了するまですべて保持しておく必要がある。また、送信元では発信作業が失敗に終りエラーが返ってくる間、送信した送信したデータに関する情報を保持しておく必要がある。データ転送速度が高速な場合は入出力用のバッファ領域を確保することが事実上不可能となる。そのため、個々のデータグラムに対するエラーを返すのではなく、データの経路に関するエラーを返すことで実現する必要がある。

Internet 内部の計算機間で交換網を個人的に利用する場合は、通常のデータ通信を行なう経路を変更せず、限られたサービスのみ交換網を利用するようにしなければならない。このためには、送信するデータに経由するゲートウェイが明記されている場合はその経路にしたがってデータを配送しなければならない。しかし、この機能は IP プロトコルのオプション機能であるために実現されていないシステムや正常に動作しないシステムが存在する。IP オプションを正しくサポートしていないシステムが経路上にあることで Internet 内での交換網などの個人利用が不可能となるため、正しい IP オプションの制御が可能なシステムのみを利用する必要がある。

すべてのサービス提供するためのネットワーク

今回行なった負荷分散の実験では同じ計算機に対して、伝送遅延や転送速度などの回線品質が同じ回線を用いて行なった。伝送遅延と回線転送速度によって相手計算機までのデータ転送率が決定されるために、伝送遅延や転送速度が異なることで分散させるためのアルゴリズムが複雑になることが予想される。また、同じ計算機に接続されている回線を用いず、異なる計算機に接続されている回線を負荷分散用に用いる場合は、実験の様にネットワークインターフェイスモジュールでは実現できず、経路制御機構として実現する必要がある。従って、今後より有効な負荷分散を実現するためには、回線上での遅延と回線転送速度から実際のデータ転送速度の計算方法と経路制御による負荷分散機構を確立する必要がある。

折り返し検査を用いて回線のバックアップを行なう場合で既存の回線上に大量のデータが連続して出力されると折り返し検査に失敗し、障害が発生したと判断してバックアップを行なってしまい無駄が生じる。しかし、実際には大量のデータの転送のために、他の用途での回線を利用できない状態になっており、バックアップではなく負荷分散の必要がある。そのために、バックアップ回線を接続して直後は負荷分散の形でバックアップ回線を利用し、一定時間以上状態が変わらない場合のみ負荷分散の機能を停止しバックアップ回線として利用する方法が最も効率良いバックアップとなる。今後、回線状態からバックアップと負荷分散を自動的に使い分けるシステムが必要となる。

第 7 章

結論

今後の広域ネットワークの拡充の核となる交換網の効果的な利用を実現するため、ネットワークオペレーティングシステムである UNIX オペレーティングシステムに改良を加え、交換網制御機構等を含むネットワークインターフェイスを実装し、実験を行なった。交換網制御機構は、役割毎にモジュールに分割することにより高度な自由度を有し、システムに対して異なる通信プロトコルや交換網アクセス方式を容易に提供することができた。加えて、ハードウェアやオペレーティングシステムに依存する部分を独立したモジュールとすることで移植性を高め、SunOS や 4.3BSD UNIX などでの実装が可能となった。

各々の役割を持つモジュールは、モジュール間インターフェイスが統一され、階層化されたアーキテクチャにおいて自由に重ね合わせて利用することができる。さらにモジュールの動的な入れ換えを実現することで、網の端末装置の設定、端末装置との接続や相手に応じた通信プロトコルの選択などを可能とした。また、モニタモジュールを動的に各モジュール間に挿入することで、現在の回線上やプロトコル上の利用状況やエラー検知などを可能となる。さらに、交換網アクセスを必要とする最初のデータグラムを検知して自動発呼し、一定時間通信が存在しないことを検知し、自動切断を行なう自動回線制御モジュールにより、汎用的な交換網利用を実現した。

これらの機能をネットワークインターフェイスとして実現することで、複数のネットワークアーキテクチャからの利用を可能とした。ネットワークインターフェイス stub の考え方を導入することにより、交換網や専用線を利用する際には point-to-point プロトコルモジュールやデータリンクプロトコルモジュールを加えることができ、Ethernet、専用線や交換網などの性質の異なる通信媒体を統一的に扱うことができた。そのため既存のネットワークアーキテクチャやそのアプリケーションに一切の変更を加えず利用できた。

実用にあたって最も問題となるパフォーマンスは、モジュール間でデータコピーを行なわないため、図 6.1に示されるようにオペレーティングシステム内部でモジュールに分割しない場合と比べほとんど差がない。

交換網利用の最も大きな利点である一時的な利用を実現するため、作成した交換網制御機構を用いて実際のネットワークを構築して実験を行なった。

- 個人的な利用では、交換網制御機構の大規模性に加えネットワークアドレスや経路制御情報などのネットワーク資源利用を極小にすることで、個人的利用の要求を満たすことのできる大規模性を持ったネットワーク構築が可能となった。

- システム間での定期的なデータ転送では一日あたりの接続回数を調整することで経済性を向上させることができた。
- 個人的な利用を目的とするバイパス経路や既存の回線に対する負荷分散を行なう場合では、物理的なネットワークインターフェイスを一つ以上のネットワークインターフェイス stub で表すことで、利用目的に合わせて物理的な回線を区別し、負荷分散を行なうことができた。
- 既存の回線のバックアップ回線として交換網を用いる場合では、point-to-point プロトコルの折り返し検査による回線障害検知機構を用いることで回線の障害を検知し、交換網を接続することで、回線接続の信頼性の向上が実現できた。

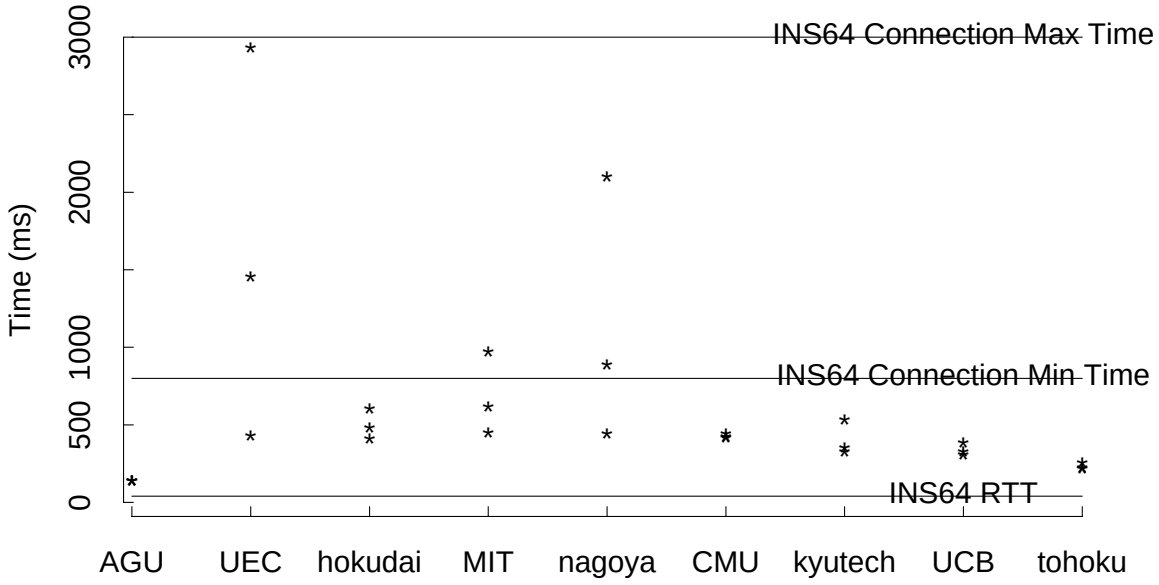
現状の広域ネットワークでは既存の回線の利用度により、データが目的地まで到達するまでの時間にばらつきが生じ、一つのデータグラム転送に必要な時間が長くなることがある。例えば、上智大学から適当な目的地までの RTT は図 7.1に示されるように目的地によっては、回線交換網として INS64 を用いた場合の接続に必要な時間 (最小で 800ms、最大で 3s) を上回る。加えて、回線交換網接続後の RTT は非常に小さいために、通信データが多い場合は交換網を用いた方が優位となり、広域ネットワークでの交換網利用が大きな役割を持つことになる。

この様に現在の広域ネットワークで高速なデジタル交換網は大きな役割を果している。作成した交換網制御機構や実験ネットワークでのネットワーク構築技術を WIDE プロジェクトでは、

- 組織と自宅の接続
- 回線や計算機の状態を調べるためのバイパス経路
- 緊急な処理を行なうためのバイパス経路
- 回線や計算機の障害による WIDE バックボーンのバックアップ

に利用している。

開発した交換網制御機能や作成した実験ネットワークでの経験は ISDN 回線交換網のためのゲートウェイを実現するばかりでなく、データグラムネットワークをパケットや交換網を用いて相互接続する際に一般的に利用でき、新しい通信媒体である FDDI と高帯域 ISDN の接続などへの有効性が期待できる。



10 回の計測の最小値、平均値、最大値

AGU	:	青山学院大学
CMU	:	Carnegie Mellon University
hokudai	:	北海道大学
kyutech	:	九州工業大学
nagoya	:	名古屋大学
MIT	:	Massachusetts Institute of Technology
tohoku	:	東北大学
UCB	:	University of California, Berkeley
UEC	:	電気通信大学

図 7.1: INS64 の接続時間と上智大学から他大学への RTT

付録: BSD UNIX 内部で用いている構造体

```
File: /usr/include/sys/file.h
struct fileops {
        int     (*fo_rw)();
        int     (*fo_ioctl)();
        int     (*fo_select)();
        int     (*fo_close)();
} socketops = { soo_rw, soo_ioctl, soo_select, soo_close };

File: /usr/include/sys/socket.h
struct socket {
        short   so_type;                /* generic type, see socket.h */
        short   so_options;             /* from socket call, see socket.h */
        short   so_linger;              /* time to linger while closing */
        short   so_state;               /* internal state flags SS_*, below */
        caddr_t so_pcb;                 /* protocol control block */
        struct  protosw *so_proto;      /* protocol handle */
        struct  socket *so_head;        /* back pointer to accept socket */
        struct  socket *so_q0;          /* queue of partial connections */
        short   so_q0len;               /* partials on so_q0 */
        struct  socket *so_q;           /* queue of incoming connections */
        short   so_qlen;                /* number of connections on so_q */
        short   so_qlimit;              /* max number queued connections */
        struct  sockbuf  so_rcv, so_snd;
        short   so_timeo;               /* connection timeout */
        u_short so_error;               /* error affecting connection */
        short   so_oobmark;             /* chars to oob mark */
        short   so_pgrp;                /* pgrp for signals */
};

File: /usr/include/sys/protosw.h
struct protosw {
        short   pr_type;                /* socket type used for */
        short   pr_family;              /* protocol family */
```

```
        short   pr_protocol;            /* protocol number */
        short   pr_flags;               /* see below */
        int     (*pr_input)();          /* input to protocol (from below) */
        int     (*pr_output)();         /* output to protocol (from above) */
        int     (*pr_ctlinput)();       /* control input (from below) */
        int     (*pr_ctloutput)();      /* control output (from above) */
        int     (*pr_usrreq)();         /* user request: see list below */
        int     (*pr_init)();           /* initialization hook */
        int     (*pr_fasttimo)();       /* fast timeout (200ms) */
        int     (*pr_slowtimo)();       /* slow timeout (500ms) */
        int     (*pr_drain)();          /* flush any excess space possible */
};

File: /usr/include/net/if.h
struct ifnet {
         char   *if_name;               /* name, e.g. ``en'' or ``lo'' */
         short  if_unit;                /* sub-unit for lower level driver */
         short  if_mtu;                 /* maximum transmission unit */
         short  if_flags;               /* up/down, broadcast, etc. */
         short  if_timer;               /* time 'til if_watchdog called */
         int    if_metric;              /* routing metric (external only) */
         struct ifaddr *if_addrlist;    /* linked list of addresses per if */
         struct ifqueue if_snd;         /* output queue */
         int    (*if_init)();           /* init routine */
         int    (*if_output)();         /* output routine */
         int    (*if_ioctl)();          /* ioctl routine */
         int    (*if_reset)();          /* bus reset routine */
         int    (*if_watchdog)();       /* timer routine */
         int    if_ipackets;            /* packets received on interface */
         int    if_ierrors;             /* input errors on interface */
         int    if_opackets;            /* packets sent on interface */
         int    if_oerrors;             /* output errors on interface */
         int    if_collisions;          /* collisions on csma interfaces */
         struct ifnet *if_next;
};

File: /usr/include/sys/mbuf.h
struct mbuf {
        struct  mbuf *m_next;           /* next buffer in chain */
```

```
        u_long  m_off;                  /* offset of data */
        short   m_len;                  /* amount of data in this mbuf */
        short   m_type;                 /* mbuf type (0 == free) */
        u_char  m_dat[MLEN];            /* data storage */
        struct  mbuf *m_act;            /* link in higher-level mbuf list */
};
```