第6章　計算機ネットワーク

今日の計算機システムにおいて、ネットワークは欠かせない機能である。それはなぜだろう? またネットワークによって、これまでにできなかったどんなことが可能になるのだろう? 本章では、ネットワークに特有の各種技術的課題とともにこれらの事柄について学ぼう。

6.1　計算機ネットワークの概念

6.1.1　計算機ネットワークとは?

まずそもそも、計算機ネットワークとは何だろう? 物理的には、複数の (たぶん多数の) 計算機システムが、互いに通信できるように (つまりデータがやりとりできるように) 接続されたものと考えてよいかも知れない。しかし、例えば (皆様がよくやるように)PC に端末ソフトを入れ、電話線などを通して計算機に接続してもそれは普通ネットワークとは言わない。計算機ネットワー

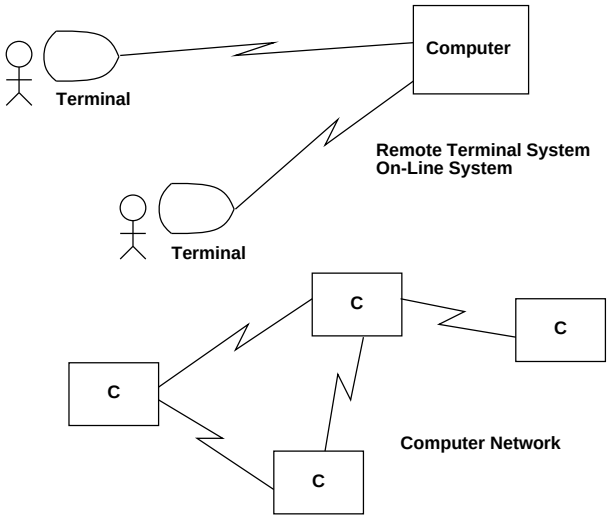

図 6.1: リモート端末システムとネットワークシステム

クと言った場合

- 接続された各計算機が
- 主従関係なしに、それぞれ自律的に動作し、
- 接続された他の計算機と協調動作しながら仕事を進める

ものを指すのが普通である。ということは、あるシステムが計算機ネットワークと呼ぶのに値するかどうかは、物理的な形態だけでなく、ソフトを含めてどのように運用されているかまで見ないと分からない。例えば電話線で PC と他の計算機をつないだシステムでも、PC 上で自律的に動作するソフトが動いていて、向こう側の計算機と協調して仕事をするようなシステムであれば立派な計算機ネットワークである。

では次に、**何のために**計算機ネットワークを作るのだろう? 「遠くの計算機システムに入っているデータを見たり操作したりするため」というのだったら×である (なぜなら、リモート端末システムでもそれはできるから)。タネンバウム [1] によれば、計算機ネットワークを構成する目的としては次のようなものがある:

a. 資源の共有 — 例えばある計算機に入っているデータを、その計算機で処理を行なう際だけでなく、別の計算機で処理を行なう際にも利用できるようにする、ある計算機の CPU 能力が不足したらデータの一部を別の計算機で処理する、など。

b. 信頼性 — 1 台の計算機であればそれが止まってしまえば処理は停止してしまうが、複数台の計算機をネットワークで結合したものであれば、1 台が壊れても残りで処理を進めて行くようにできる。

c. 経済性 — 大きな計算機 1 台で何もかもやらせるより、複数の PC や WS をネットワーク結合したシステムの方がコスト的に安い。

d. 段階的成長 — 1 台の計算機で能力が不足したら、より大きいマシンにリプレースするしかないが、ネットワークシステムなら何台かマシンを追加する形で成長して行ける。

e. 通信媒体 — 距離的に離れたシステムどうしを接続することにより、新しいタイプの応用が可能になる。

もちろん、どの目的を主とするかによって、ネットワークの形態は大幅に異なる。例えば信頼性や経済性のために 1 台の計算機に代えてネットワークシステムを構成するのなら、その各計算機間の距離は比較的近く、それらの間は高速な通信方式で結ばれることになるだろう (LAN – Local Area Network)。一方、地域的に離れたところにあるデータの共有や個人間の通信が目的なら、そのネットワークは長い距離を結ぶものになるわけである (WAN – Wide Area Network)。

[1] Tanenbaum, Computer Networks 2nd ed., Prentice-Hall, 1988.

6.1.2 2 点間リンクと放送

皆様のもっとも身近にある「ネットワーク」として「電話網」や「テレビ放送網」などが挙げられる。この 2 つはかなり違っているが、それはどういう点だろうか?

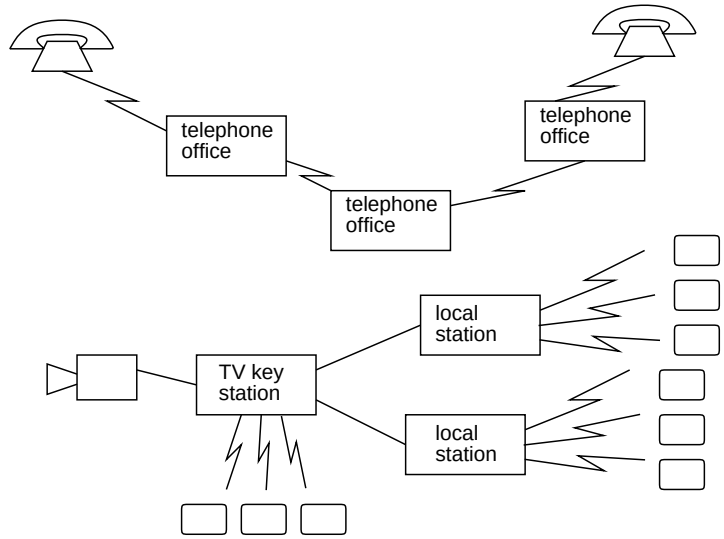

図 6.2: 電話 (2 点間リンク) とテレビ (放送)

- 電話は、特定の相手と 1 対 1 の通信を行なう。このような接続形態を 2 点間 (point-to-point) リンクと呼ぶ。
- テレビは、キー局から多数の受像機への 1 対多、片方向の通信を行なう。このような接続形態を放送 (broadcasting) と呼ぶ。

計算機ネットワークの場合にも、通信媒体に応じてどちらかの接続形態が使われることになる (図 6.3)。例えば 2 点間リンクとしては次のような媒体がある。

a. 銅線。RS-232C 規格などの信号ケーブルを自分で敷設することもできるし、既設の電話線などを利用することもできる。コストと速度に応じて多数の選択肢がある。

b. 光ファイバ。高速な 2 点間リンクを張ることができる。

c. マイクロ派や赤外線。道路をまたいだり、建物が違うなどしてケーブルを敷設するのが困難なときに有効。

一方、放送型の媒体には次のようなものがある。

d. 銅線。ただし、2 点間に張るのでなく、あちこちケーブルを引き回して、多数の計算機をつなぎ込む。同軸ケーブルを使用する Ethenet などがこの代表であり、LAN の標準的媒体でもある。

e. 通信衛星。パラボラアンテナさえ建てればどんな僻地でも使え、通信容量も多い (通信時間はやや掛かる)。

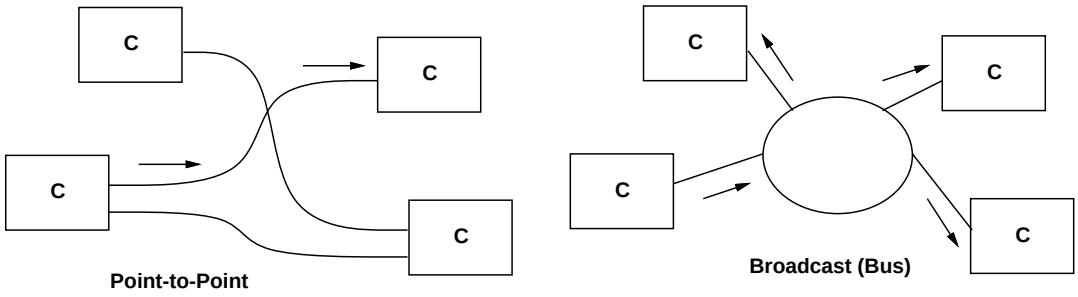

図 6.3: 2 点間リンクと放送 (バス) 型リンク

ただし、WAN の場合には自前で長距離のケーブルを張ったり衛星を持てる訳がなく、NTT など通信業者から回線を買うのが普通である。その場合、専用線を買えばそれは 2 点間にケーブルを張ったのと同じである (通信業者の中でどうやって通信を処理しているかは別問題)。また、衛星のチャネルを買えばそれは e. と同じわけである。

以上は物理的な接続形態だったが、ただし計算機ネットワークはソフトウェアによって自由に制御できるので、2 点間型の媒体を利用しても中継によって全員に同一データを放送したり、逆に放送型の媒体でも特定宛先以外のサイトではそのデータを無視することで、**論理的には**どちらの形態を利用することもできる。このような制御の柔軟性は計算機ネットワークならではといえる。

6.1.3　回線交換とパケット交換

電話やテレビなどのネットワークと計算機ネットワークには他にも違うところがある。それは、電話やテレビではまず通信経路が確保され (電話では最初にダイアルした時、テレビではテレビ局が電波の割り当てを受けて設備を用意した時)、通信中はその経路はずっと確保されている。話し中にちょっと沈黙している間とか、夜中の放送時間外のようにその経路にデータが流れていない場合でもそれを他人が有効活用するというわけには行かない。そのかわり、いつでも再度話し始めたり放送を開始することができる。このようなネットワークの接続形態を回線交換と呼ぶ。

計算機間の接続を回線交換で行なうことは、実はあまり得策ではない。というのは、計算機どうしの通信は「特に仕事がない」状態では何もやりとりするデータがなく、「座席予約」とか「ファイル転送」などの作業があるとその時だけどっと大量のデータが移動するという性質を持っている。

そこで、計算機の場合には通信したい内容をある範囲の大きさ (数十バイト〜数千バイト程度) の「パケット」と呼ばれるかたまりにまとめて、そのパケットをやりとりすることで通信を行なう。これをパケット交換と呼ぶ。回線交換が「電話」だとすれば、パケット交換は「葉書」ないし「小包み」に例えることができる。

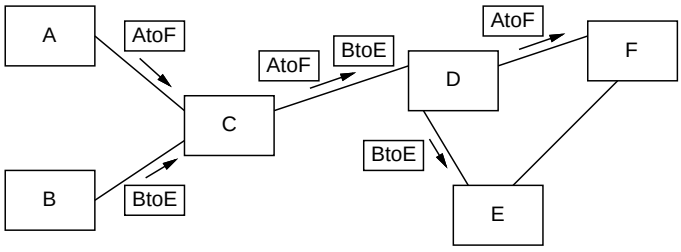

図 6.4: パケット交換の原理

たとえば、図 6.4 のようなネットワークで、A から F へのパケットはまず C に転送され、次に D に転送され、最後に目的地の F に付くことになる。各中継点でパケットは一旦受け取られて格納されることから、パケット交換ネットワークは「ストアアンドフォワード」であるとも呼ばれる。この方式は、通信のノイズなどのためにたとえば D → F の転送が失敗した場合でも、D に格納されたパケットをもう 1 度送るだけで済むといった利点も持つ。また、例えば D → F のリンクが壊れてしまったら、A から F へのパケットを代わりに D → E → F と転送することもできる。このようにパケットの転送経路を適切に制御する管理作業を経路制御と呼ぶ。

ところで、これと同時に B から E への通信もあったとすると、両者のパケットは C → D のところでは同じリンクに相乗りすることになる。そして、B から E への通信がちょっと中断している間は A から F へのパケットを目一杯流してもよい。つまりパケット交換では通信リンクが回線交換よりも柔軟に利用できる。その代わり、B から E への通信が現われたらこのリンクを両者で公平に使うような管理作業が必要である。

あるホストから別のホストへ大きなファイルを転送するような場合を考えると、ファイル全体を 1 つのパケットに入れるのは無理だから多数の連続したパケットを用いて転送を行うことになる。このとき、予め「どこからどこへ転送を行う」という準備をしておくことで転送効率をあげたりエラー回復に備えることができる。この方式を、転送自体はパケット交換だが「仮想的

に」回線のような効果を持たせることから「仮想回線」と呼ぶ。

6.1.4　アドレスとポート

パケットにせよ仮想回線にせよ、データを送る場合には「送り先」を指定しなければならない。この送り先を指定する情報を「ネットワークアドレス」ないし単に「アドレス」と呼ぶ。アドレスは各パケットに格納する必要があり、しかも送り先と送り元の両方が必要なのであまり長い形式では処理効率が悪く、しかしネットワーク中の任意の行き先を指定できるだけのビット数が必要である。

多くの場合アドレスは「どのホスト」という情報と「そのホストの中のどこ」という情報を併せた形になっている。「どこ」というのは、具体的にはパケットを送り出したり受け取ったりするための「窓口」のようなものであり、「ポート」と呼ばれることが多い (Unix では「ソケット」とも呼ばれる)。「ポート」は「ファイル」や「プロセス」などと同様、実際には存在しないが OS によって作り出される仮想的な「もの」である。

6.1.5　簡単なネットワークプログラム

そろそろお話ばかりで飽きたと思うので、ここでごく簡単なネットワークプログラムを示そう。このプログラムはあるホストから別のホストへパケットを使って文字を送るというもので、送る側と受け取る側に分かれている。

```
/* recv.c -- packet receiving example. */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

main() {
  int fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
  char buf[100];
  int len, fromlen;
  struct sockaddr_in adr;
  adr.sin_family = AF_INET;
  adr.sin_addr.s_addr = inet_addr("192.50.17.51"); /**/
  adr.sin_port = 10020;                                    /**/
  if(bind(fd, (struct sockaddr*)&adr, sizeof(adr)) < 0) {
    perror("bind: "); exit(1); }
  while(1) {
```

```
        len = recvfrom(fd, buf, 100, 0, 0, &fromlen);
        if(len < 0) {
          perror("recvfrom: "); exit(1); }
        write(1, buf, len); }
    }
```

まず、#include ... とあるのはネットワーク関係のデータ構造定義をとりこむ「おなまじない」である。次に main に入って最初の socket システムコールでソケット (ポート) を作り、そのディスクリプタ番号を変数 fd に入れる。buf はパケットデータを格納するバッファ、len と fromlen は整数変数である。その次にネットワークアドレス型のレコード変数 adr を割り当て、そのホスト部に「192.50.17.51」というアドレス、ポート番号に 10020 を入れ、bind システムコールにより先に作ったソケットのアドレスを上記の値にする (失敗したらメッセージを出して終わる)。

ここまでの所はこれ以上詳しく説明しても頭が痛いだけなので、「おまじない」だと思ってそのまま使っていただくが、要は OS に頼んでポートないしソケットを準備しているのだと理解して頂ければ十分である。ただ、アドレスについては説明しておこう。ここで使う方式 (TCP/IP) ではホスト指定は 0〜255 の数値を 4 個「.」で区切って並べたものである。皆様がふだん使うホストのアドレスは次の通り。

```
smb   --- 192.50.17.254
smd   --- 192.50.17.51
smf   --- 192.50.17.53
smg   --- 192.50.17.54
utogw --- 192.50.17.2
```

どれも途中まで同じであるが、その辺の事情は後で述べる。そして、ポートは 0〜65535 の数値である。後で実習していただく時に複数の人が同じポートを使うと混乱するので、ここでは「10000 + UID」を使って頂く。(自分の UID は「echo $UID」でわかる。) ともあれ、「/**/」の部分は適宜書き換えて使うことを忘れないように。

さて、ポートが準備できたらあとは recvfrom というシステムコールにより、パケットの到着を待つよう OS に頼む。recvfrom が終わった時には配列 buf にパケットデータが入り、そのバイト数が recvfrom の戻り値として帰されるので、それを write により標準出力に書き出している。以上である。

では次に送り側のプログラムを示す。

```
/* send.c -- packet sending example. */

#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>

main() {
  int fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
  char buf[20];
  int len;
  struct sockaddr_in adr;
  adr.sin_family = AF_INET;
  adr.sin_port = 10020;                                        /**/
  adr.sin_addr.S_un.S_addr = inet_addr("192.50.17.254"); /**/
  if(bind(fd, (struct sockaddr*)&adr, sizeof(adr)) < 0) {
    perror("bind: "); exit(1); }
  adr.sin_addr.S_un.S_addr = inet_addr("192.50.17.51");  /**/
  while((len = read(0, buf, 20)) > 0) {
    if(sendto(fd, buf, len, 0, (struct sockaddr*)&adr, sizeof(adr)) < 0) {
      perror("sendto: "); exit(1); }
  }
}
```

こちらは送るためのポートを作るところまでは先と同様である。次に、アドレス型レコードを送り先指定にも使うため、ホストアドレスの部分を相手のアドレスに書き換える。その上で、あとは入力からデータを読んでは sendto でパケットとして送るわけである。ではこれを使ってファイルを送ってみよう。

```
smd% gcc recv.c
smd% mv a.out recv
smd% recv
(データ待ちになる)
```

なお、send と recv がどっちも a.out になると不便だから、名前をつけかえている。

```
%smb gcc send.c -lsocket -lnsl
%smb mv a.out send       ↑ smb ではオプションが余計にいる。
%smb more t
aaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbb
ccccccccccccccccccc
ddddddddddddddddddd
aaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbb
...
```

```
% send <t
```

すると、データを待っていた recv が再開される。

```
(先の続き)
aaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbb
ccccccccccccccccccc
ddddddddddddddddddd
aaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbb
...
aaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbb  ←あれ?
ccccccccccccccccccc
ccccccccccccccccccc
aaaaaaaaaaaaaaaaaaa
ddddddddddddddddddd
bbbbbbbbbbbbbbbbbbb
ddddddddddddddddddd
^C
smd%
```

大体よさそうだが…一部データが抜け落ちている??? これは、送り側の smb の方が受け側の smd より速いため smd の方で受け取りが間に合わなくなって取りこぼしが起きているのである。このほかにも、ノイズなどによりパケットが失われたり、転送経路の切り替えのため送ったのと違った順序でパケットが到着したりすることもある。このような障害を乗り越えて正しくデータを送ることは容易でない、ということは少なくともおわかり頂けると思う。次節でそのような容易ならざる仕事をこなすネットワークソフトウェアの機能と構造について考えてみる。

6.2 ネットワークソフトウェアとプロトコル

6.2.1 ネットワークソフトの階層構造

そもそもネットワークソフトウェアに備わっているべき機能としてはどんなものがあるだろうか? Top-Down に考えてみよう。例えばあなたが遠くにある計算機システムから自分の計算機システムに必要なデータの入ったファイルを転送したいものとする。すると....

a. まず、あなたは相手計算機の名前 (アドレス?) とファイル名を指定して「ファイル転送」という指令を起動する。すると、その指令は相手計算機の「ファイル転送機能」と接続して必要な情報を知らせ、相手計算機の「ファイル転送機能」は指定されたファイルの中身を最初から 1 バイトずつ順番に送ってくる。こちらの指令は送られてきた中身をこちらのファイルに順番に格納していく。

b. さて、中身を順番に送ってくる、と言ったが、送っている途中でデータの一部が欠落したり、順番が入れ代わったりしては困ることになる。そこで、こちらから 1 バイトずつ送ったものが向うでも確かに欠落なく、その順番で受けとられるように制御する、という機能が必要である。

c. さらに、もし相手計算機が USA の計算機だったりすると、各データはまず向うの計算機から日米の橋渡しになっている計算機へやってきて、橋渡しに衛星を経由し、こっちがわの橋渡し計算機から出て自分の計算機に着く (実際にはもっとずっと沢山の中継点がある) ことになるので、そういう通過経路を制御する、という機能が必要である。

d. そして、ある中継点 (または出発点) から次の中継点 (または最終目的地) への転送に当たっては、当然パケットにデータを格納して送る、という作業が必要である。

e. もちろん、一番下ではネットワーク入出力装置と媒体が個々のパケットを運ぶわけである。

こうしてみると、ネットワークのソフトウェアというのは一番上の我々が利用したい操作 (ファイルを転送する、etc.) から一番下のハードウェアまで多数の階層が積み重なって作られていることが分かる。

6.2.2　OSI 参照モデル

ISO ではネットワークの標準規格 (OSI — Open System Interconnect) を制定するに当たって、このような階層化の標準モデルを提案している。これは 7 つの階層から成り、OSI 参照モデルとか 7 層モデルとか呼ばれている。その構成を図 47 に示す。ここで左側に先の各機能がどこに入るか、を示した。a. については OSI モデルではさらに細かく分けられているが、そこまで細かく考える必要はないと思うので先の例ではまとめたものである。

ところで、このモデルで同じレベル (層) にある機能どうしがやりとりするのに使う約束ごとを「プロトコル」と呼ぶ。例えば先のファイル転送機能では「まず相手側にファイル名の長さを送り、続いてファイル名を送る。すると相手側はファイルの長さを送り、続いてその長さぶんだけファイル本体を送って終る。もし指定のファイルがなければファイル長さとして-1 を送る。」

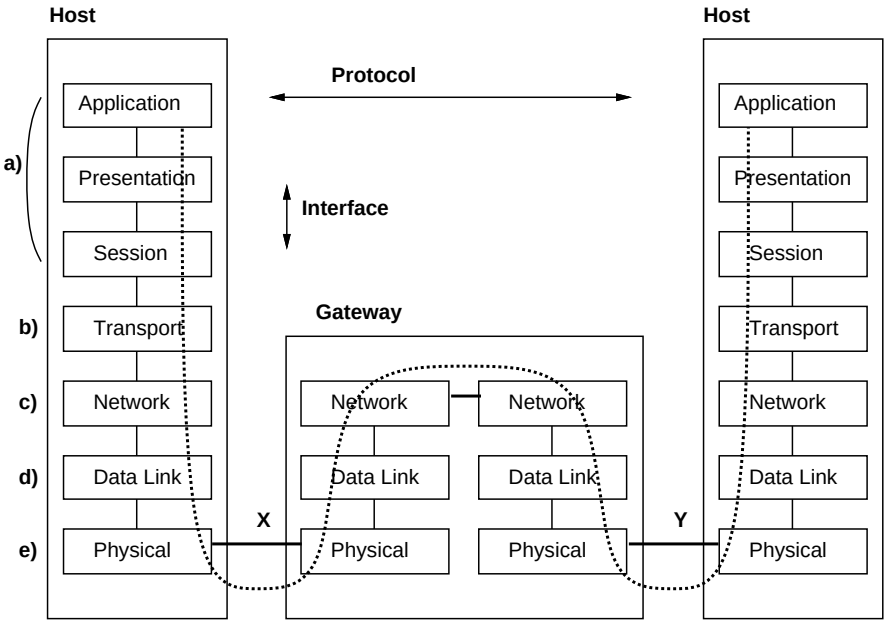

図 6.5: OSI 参照モデル

という約束ごと (つまりファイル転送プロトコル、ですね) を使っているかも知れない。

また、各層が上の層に提供する機能の集まりを「インタフェース」と呼ぶ。例えば、伝達 (Transport) 層は「ホストアドレスとサービス (例えば「ファイル転送」) を指定すると、そのホストの指定されたサービスを行なっている相手を探し、接続番号を返す」「接続番号、バイト数、バッファアドレス、読み/書きの別、を指定すると、その接続に対して指定バイト数の転送を行なう」などのインタフェースを持つのが普通である。

各層は自分が提供する機能を実現するのに、相手ホストの対になっている層とプロトコルを用いてやりとりするが、そのやりとりのためには自分の下にある層に対してインタフェースを通じて頼むことを行なうわけである。例えば伝達層は相手ホストのトランスポート層から送られてくるデータに抜けがないことをチェックするために、「各データには 1 から始まる 16 ビットの 1 連番号を先頭に付加する」という「プロトコル」を使うかも知れない。一方、データを受けとるには例えばネットワーク層の「自分あてに来たデータの 1 かたまりを指定したバッファに入れ、同時に送ってきた相手のアドレスも通知する」という「インタフェース」を利用することになるだろう。言い替えれば、各層は直接相手の同レベルの層と話している「つもりになっている」が、それは幻想で実際のデータは下へ下へと送られ、物理層で相手へ渡って、そこから上へ上へと上ってくる、という大迂回を生じているのであった。

もちろん、物理的な媒体を通ることなしに他のホストへデータが行けるわけがないので、当たり前ではある。

ところで、先に出てきた「中継点」ではどんなことが起きているだろう。当然、送っているファイルを一旦全部中継点に蓄えてしまう、というのは賢くない (そんな余分なディスクがどこにある?)。一方、中継点では「米から衛星で来たデータ 1 かたまりを見ると、それは GSSM 宛である。では東京方面への専用線に送り出そう」という制御は必要である。だから中継点ではネットワーク層が動いている必要はある。ときに、中継点の両側では物理層、データリンク層はそれぞれ別ものであることに注意。というのは X は専用線、Y は衛星なのだからその媒体も制御のしかたも全然異なるからである。というわけで、中継点まで考えるとデータそのものの流れは点線のようになるわけである。

6.2.3　様々なプロトコル群

さて、ネットワークの目的は複数の計算機間で互いに通信することにあるわけだが、そのためには通信し合うモジュールどうしでプロトコルが同じでないと通信ができない (あたりまえ)。そして、ある層だけ共通のプロトコルだがその下は全然別、というのは不自然なので、結局図 6.5 にあるような各階層にわたってそれぞれが共通のプロトコルを喋る計算機間で通信する、というのが普通なわけである。そのような**プロトコル群**としては、例えば次のようなものがある。

- SNA プロトコル群。IBM のシステムで使われている。もちろん、IBM 互換各社でもそれぞれ FNA とか HNA とかいうのをやっている。
- Decnet プロトコル群。DEC のシステムで使われている。これもわりと古くからある。
- Internet プロトコル群。TCP/IP とも呼ばれている。もともとは米国 ARPANET というネットワークの実験 (広域ネットワークの元祖) に起源するが、4.2bsd Unix の標準付属品となったので、その結果 Unix の広まりとともに世の中で多く使われるようになった。Unix そのものと同様、多くのベンダーで共通に使われているのが特徴。
- MAP/TOP。GM が最初に開発したもので、Factory Automation / Office Automation のため各種機器を接続することを目的としている。その方面の分野では多く使われるようになっている。
- OSI。これは前述のように ISO が規格として標準化を進めているもので、まだ開発途上にあるが、今後採用が増えるものと思われている。

- Appletalk, Netware, LanManager — PC/LAN レベルのプロトコルとしてアップル、ノベル、マイクロソフトなどがそれぞれ推進している。

6.2.4 TCP/IP

さて、抽象的なお話はそれくらいにして、具体的な話へ進もう。我々のところではプロトコル群として TCP/IP を採用しているので、以下では当然これを題材に取り上げることにする。プロトコル群は多数のプロトコルの集まりだと先に述べたが、TCP/IP の場合は図 6.6 のような構成になっている。

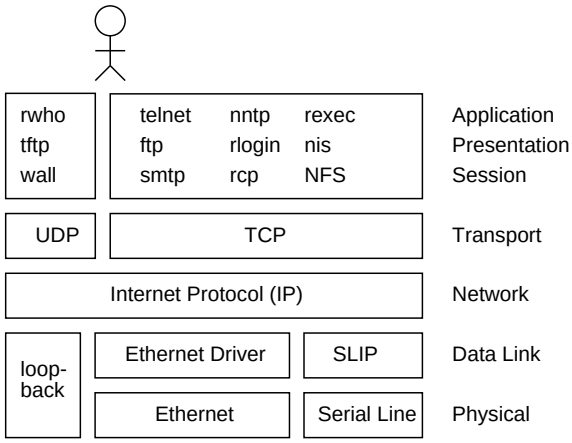

図 6.6: TCP/IP のプロトコル構成

まず、我々の手元では LAN の媒体として Ethernet を使用しているので (廊下を這っている赤と黄色のケーブルがそうです)、物理層がこれで、それを制御するデータリンク層もそのためのものが OS に組み込まれている。また、遠隔地との接続は専用線 (シリアルライン) を使っていて、その制御は SLIP と呼ばれるモジュールによっている。ところで、**1 つのマシンの上で**あるプロセスとあるプロセスが通信したい場合もありますよね? Ethenet や専用線はそこを通ると他へ行ってしまうので、自分あての通信には使えない。そこで、ループバックという特別なデバイスがあるつもりになって、そこへデータを流すと自分自身に返ってくる、というのもおまけとして用意してある。ループバックの場合には「媒体」は要らないので物理層はからっぽである。

さて、それらの上にあるネットワーク層プロトコルは IP と呼ばれ、これから上ではどのような媒体を使用しているかに関わらず同様の機能が提供される。その上の伝達層には UDP(User Datagram Protocol) と TCP(Transmission Control Protocol) という 2 種類のプロトコルがあり、それぞれ違う種類のサービスを提供している。その上は各種応用ごとに様々なプロトコルがあるが、代表的なものを挙げると次のようなものがある。

- telnet — 他の計算機に login するためのプロトコル。
- rlogin — 同じだが、Unix マシンどうし専用。その分機能も多い。
- ftp — 他の計算機との間でのファイル転送プロトコル。
- rcp — 同じく、ただし Unix 専用でより簡便。
- smtp — 電子メール搬送用プロトコル。
- nntp — 電子ニュース搬送用プロトコル。
- NFS — 他のマシンのファイルシステムを読み書きする。
- rwho — 誰がどのマシンにいるかの情報を交換し合う。
- rwall — マシン停止などのお知らせメッセージ伝達用。

では、以下で物理層から始めてもう少し具体的に見てみよう。

6.2.5　物理層とデータリンク層 — Ethernet

我々の所で LAN のために使用している媒体は Ethernet である。その原理は次のようなものである。まず、1 本の同軸ケーブルに多数の計算機を接続する。各接続点では、計算機からケーブルに信号 (というのは、電圧の変化ですね) を送り出すことと、ケーブル上の信号を感知することができる。ケーブルは 1 本だから、誰かがケーブル上に信号を出せば (つまりデータフレームを送信すると)、それは他の全員に感知できる (つまりデータフレームが受信できる)。もちろん、実際には特定の相手だけに送信したいのだから、12.2 節で説明したようにフレームには相手のアドレスが入っている。[2]

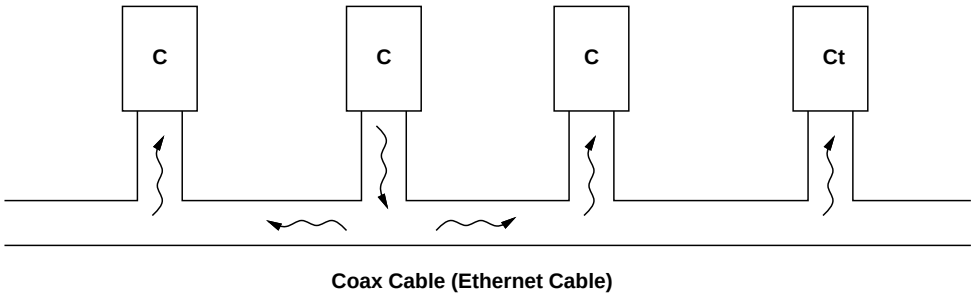

図 6.7: Ethernet の原理

ただここで問題なのは、たまたま 2 台の計算機が同時に信号を送ろうとすると、信号が同軸ケーブル上でぶつかって (衝突)、でたらめな値になる、と

[2] このアドレスは 48 ビットの数値で、世の中にある全ての Ethernet 機器は別のアドレスになるように、上 24 ビットが製造業者固有番号、その下が業者内部の固有番号にしてある。だからどっかで買ってきた 2 つの Ethenet 機器をつなげたらアドレスがおんなじで混乱する、という心配はない。

いうことである。そこでどうするかというと、送る計算機は送るのと同時にケーブル上の信号の感知も行ない、衝突が起きたのが分かったら送るのを止めてランダムな時間だけ待ち、改めて送り始める。2 つの計算機上の乱数が一致することはまずないから、今度は衝突せずに送れる可能性が高いわけである。もちろん、Ethernet がひどく混んでいる時などは再送のときまた衝突が起きることもあるが、その時はランダム時間を計算してその倍待ってから再試行する。また衝突したらさらに倍... とする。こうすると、衝突が起きている限りは各計算機の送信確率は半分ずつに減っていくから、確率的に見ていつかは必ず送れるようになるわけである。

6.2.6 ネットワーク層

IP と IP アドレス

ネットワーク層の役割は「〜と通信したい」と言われたら、そのデータをできる限りうまく指定の相手に向かって送ることである。では、その「指定」はどのようにしたら行なえるだろうか? 例えば郵便であれば住所と氏名、ということになるが、計算機ネットワークの場合にはいちいちそういう長い文字列を引きずって歩きたくない。そこで IP では 32 ビット (4 バイト) の数値をアドレスとして使用している。このうち、上の何バイトかはネットワーク番号、残りがホスト番号、と言う風に分かれている。例えば我々のところは上 3 バイトがネットワーク番号であり、赤と黄色の Ethernet ケーブルがそれぞれ

```
192.50.17.
192.50.18.
```

という番号を持っている (この書き方は、各バイトをそれぞれ 10 進数だと思って読んで、それを . でつなげる、という流儀で、IP アドレスの表記は伝統的にこの流儀によっている)。それぞれにつながっているホストのアドレスはネットワーク部分は同じで最後のホスト部分で区別される。例えば

```
192.50.17.2       utogw
192.50.17.51      smd
192.50.17.254     smb
```

といった具合である。

これからも分かるように、ネットワーク番号というのはそれぞれ 1 つの物理的なネットワーク (例えば Ethenet ケーブル) に対応している。そして、世の中には同じネットワークアドレスは 2 つとないように割り当てを (人手で) 管理しているので、IP の世界では世界中どこからでも `192.50.17.2` といえばうちの `utogw`、ということになるわけである。

ところで、実は `utogw` は黄色のケーブルにもつながっているが、`192.50.17.` というのは赤色のケーブルの方のアドレスだから、黄色のケーブルにつながっている部分 (これを Unix 用語でネットワークインタフェースと呼ぶ。先のプロトコル層のインタフェースと混同しないでね) には

```
192.50.18.21      utogw-gw
```

という別のアドレスがつけてある。この辺りは IP のちょっと分かりにくい部分ではある。なお、より大きな組織でホストを多数持っている場合には、ホスト番号の部分にもっと多くのビット数が割り当てられ、そのぶんネットワーク部分のビット数は少なくなっている。

IP の経路制御

さて、それではこのアドレスを使ってどうやって「うまく相手に送る」のかを見てみよう。IP ではデータのかたまり (パケット、と呼ぶ) ごとにその頭の部分 (ヘッダ、と呼ぶ) に送り先、送り元の IP アドレス、その他の制御情報を入れておく。そこで、あるホスト A に図 6.8 のような具合に IP パケットがやってきたものとする。このパケットの行き先を見ると、`192.50.21.31` とある。

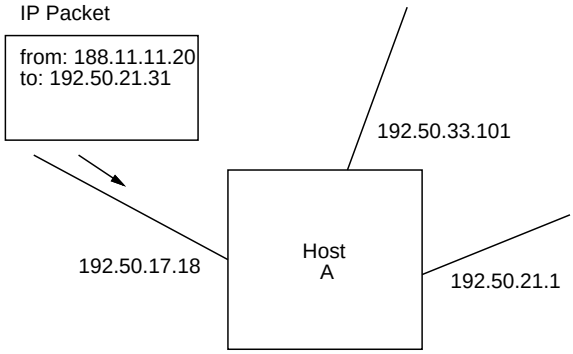

図 6.8: IP の経路制御 (1)

すると、ネットワーク番号は `192.50.21.` ということになる。ときに、このホストは 3 つネットワークインタフェースを持っていて、その中に `192.50.21.1` というのがある。ということは、このインタフェースは `192.50.21.` のネットワークにつながっているのだから、ここへパケットを送り出せば目指すホストに行きつくに違いない! [3]

このように、IP の体系では「ネットワーク番号が同じ」ならば「隣接している」ことになるので、ホストがどこにあるかを個別に勘案しなくてもネットワーク番号だけ見て経路制御を行なうことができる。もちろん、上の例はたまたま目的地の「すぐ手前の」ホストにパケットが来たからこれで済むのだが、一般の場合には「どのネットワーク番号ならどっちへ送る」という表 (ルーティングテーブル) を持っている必要がある。例えば図 6.9 のように、ホスト B にさっきと同じようなパケットが来た場合にはテーブルを見てインタフェース le0 経由でホスト A へ送ればよい、とあるのでその通りにする (と、

[3]実際はネットワークが Ethernet であれば、Ethernet アドレスと IP アドレスの対応表が別にあって、それを見て目的地にパケットを転送するだけのことである。

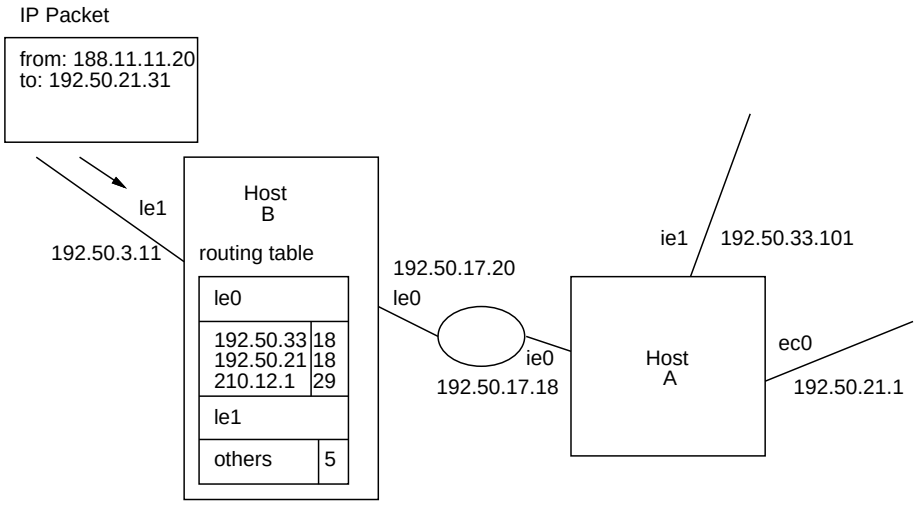

図 6.9: IP の経路制御 (2)

A はさっきと同様にして送ることができる) わけである。当然、これらのテーブルはどうやって用意されるのか、という質問が出ると思うが、答えは 2 通りある。1 つはシステム管理者が手で用意する、というもので、これは比較的小規模のサイトで行なう方法である。もう 1 つは、隣合うホストどうしでテーブルの情報を交換することを繰り返して、どちらへ行けばどのネットワークへ通じるかの情報を自動的に伝達する方法で、これは多数のネットワークの接続経路に当たるホストで用いられる。

ところで、B のルーティングテーブルには「`192.50.33, 192.50.21, 210.12.1` については `le0`、それ以外は `le1`」と書いてある。これはちょうど、函館駅のルーティングテーブル (?) に「千歳線と室蘭本線と宗谷本線と...(以下北海道各線の名前) 行きは函館本線経由、その他は全部津軽海峡線経由」と書いてあるようなもので、表の大きさを減らす効果がある。

さらに、場合によっては「わざと経路を書かない」ことがある。これは、経路への経路が表に入っていなければそのマシンへ外部からアクセスすることができず (なぜならパケットが到着しても返事が戻って行かないから)、従って外部からの侵入に対してより安全となる。我々のところでは現状ではメール/ニュースのサーバをやっている `utogw` のみ外部への経路を設定してある。

アドレスと経路関係の情報

Unix ではこれらの情報を調べるのに次の指令が使える。

- ホスト名とホストアドレスの対応表 (ローカル) — ファイル `/etc/hosts` に格納されている。

- ホスト名とホストアドレスの対応表 (広域) — 「nslookup ホスト名」により調べられる (utogw で)。
- 各ホストに備わっているネットワークインタフェースの一覧とパケット数の累積 — 「netstat -i」で調べられる。
- 各インタフェースを通過するパケット数の観察 — 「netstat -I インタフェース名 秒数」で指定した秒数間隔ごとに通過したパケット数が調べられる。
- 各ホストでの経路制御表 — 「netstat -r」で調べられる。
- 各ホストから別のホストへの経路追跡 — 「traceroute IP アドレス」または「traceroute ホスト名」で調べられる。(utogw/smd/smf/smg のみ。)

6.2.7　伝達層

TCP と UDP

伝達層のプロトコルには TCP と UDP の 2 つがある、と述べたが、その違いについてまず説明しておこう。まず、TCP は先のファイル転送の例に出てきたように、接続元と相手先の間に仮想的な「回線」を用意する機能を持つ。一旦「回線」がつながれば、片方がそこに送り込んだデータはその順番で重複や損失なく他方が受けとることが保証される。このようなサービスを (本当に線を敷設するわけではないから)「仮想回線」と呼ぶのだった。

一方、UDP は送り先アドレスと送りたいデータを指定すると、そのアドレスにデータを転送する、という機能のみを持つ。ただし、途中の経路に障害があったり混雑があった場合にはそのデータは着かないかも知れないが、ともかくできる範囲で努力する (best effort)。その分 TCP よりオーバーヘッドは小さい。このようなサービスを「データグラム」と呼ぶ。[4]

TCP の各種機能

UDP の方はほとんど IP パケットそのものに相当し、機能も単純であるが、エラーなどにはほとんど対処してくれない。これに対し、 TCP の提供する仮想回線は次のような機能を持っている。

- フロー制御 — 受け取り側の速度が遅くて受信が間に合わない場合、それに応じて送り側を待たせることにより、受け取り側の「とりこぼし」を防ぐ。

[4]「着かないかも知れない」サービスが役に立つのか、と思われるかも知れないが、たとえば音声をディジタル転送して電話のように使う場合には、時々データが落ちてノイズが聞こえても、時間的遅れがない方が望ましいからデータグラムが向いている。要は適材適所である。

- エラー検出と再送 — 一連のパケットは伝達途中で失われてしまったり、内容の一部が書き変わってしまうことがある。そのようなことが起きた場合にそれを検出し、失われたり壊れたりしたパケットを再度送り直してもらう。
- 順序の保存 — 一連のパケットは経路の状況により送り出したのと違った順序で到着することがある。そこで受け取り側に渡す手前で順序をチェックし、正しい順序で渡す。

これらを実現するには、原理的には次のような方法を用いる。

- 一連番号 — パケットには一連番号を振る。これによって、順序の入れ替わりを検出し並べ変える。
- チェックサム — パケットの内容を数値と見て一定の演算を行ない、その結果をパケットの一部と照合する。照合が一致しなければパケットに誤りがあったものとして捨てる。
- 到着確認 — パケットが着くごとに、何番のパケットまで正しく着いたかを送信側に送り返す。これにより、どこまで正しく着いたかが制御できる。
- ウィンドウ制御 — まだ到着確認がなされていないパケットは最大 W 個までしか送らない。これにより、フロー制御が行なえる。
- タイムアウト — 到着確認が失われた場合に備えて、ある時間経っても確認が着かないパケットは再送する。

これらの技術により、TCP はエラーのない信頼できる仮想回線をユーザに提供しているわけである。

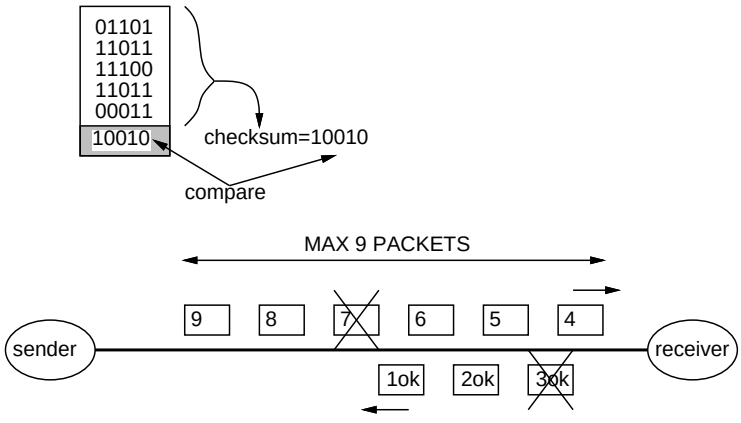

図 6.10: TCP の各種制御

ポート番号

ところで、IP アドレスはあくまでもホスト単位につくのであった。実際には各ホストで沢山のプログラムが動いているだろうから、ネットワークを使って仕事をするには「どのホストの、どのプログラム」という指定ができないと役に立たない。TCP/IP ではこの「どのプログラム」というのも IP アドレスと同様の考えに基づき、16 ビットの数値 (ポート番号) で表している。この、ポート番号の機能を提供するのも UDP や TCP の重要な役割である。ポート番号には、予めサービスの種類によって決められているものと、プログラムを動かしていて必要になった時適当に割り当ててもらうものの 2 種類がある。予め決められているポート番号の情報はファイル

```
/etc/services
```

に格納されていて見ることができる。また、現在活きている TCP ポートの一覧は

```
netstat -f inet
```

で見ることができる。

6.3　各種応用プロトコルとネットワークサービス

伝達層より上の層については、ネットワーク伝送そのものは TCP や UDP に頼んでしまえるため、各プロトコルそのものは比較的単純である。たとえばリモートログイン (ネットワーク経由で他のマシンに login する) プロトコルの場合、TCP の接続を行なってから、まず最初に login するユーザの名前を転送するが、その後パスワードを答えて入ってしまえば、あとはキーボードから打ち込んだものを相手先に転送し、向うのプログラムの出力をこちらへ転送してくるだけなので、ほとんど TCP の機能そのものに過ぎない。というわけで、以下では応用プロトコルというよりは、Unix の場合を例に取り、どのようなネットワークサービスがあるか、について紹介していく。

6.3.1　遠隔ログイン — telnet/rlogin

最初に述べたように、計算機ネットワーク以前から、手元の端末から遠隔地にある計算機に通信回線を通じて接続する、という利用形態が多く存在した。ので、ネットワーク時代になっても同様に「手元の計算機から向うの計算機に接続する」ことがまず求められた。この場合、手元の計算機はそれ以前の遠隔端末の「まねをする」わけなので、この機能を「ネットワーク仮想端末 (Network Vritural Terminal、NVT)」などと呼ぶこともある。

我々の Unix では仮想端末ソフトとして Internet 文明共通の `telnet` と Unix 固有に特化した `rlogin` の 2 つが存在する。これらの使い方は

```
telnet  ホスト名
telnet  IP アドレス
rlogin  ホスト名
```

である。`telnet` の場合は、手元のホスト名データベースにアドレスが載っていないホストにも接続できるようにするため、IP アドレスでも相手が指定できる (例えば `telnet 192.50.17.2` のように)。いずれの方法でも接続ができると普通に login プロンプトが出て、ユーザ名とパスワードを入れると接続できる。接続は向うのホストから exit すると切れるが、それ以外にも `^]` を押すと `telnet` の指令モードに入り、そこで「`q[ret]`」とやってこちらから切ることもできる。

一方、rlogin は Unix システム間専用であるが、まずこちらのユーザ名を自動的に向うに転送するので、自分のユーザ名を打ち込まずに済む。また、管理者の設定によって一群のホストどうしの間ではパスワード問い合わせを省略するようにもできる (うちの専攻内ではそうなっている)。その他、端末の種別も自動的に設定される (`telnet` では接続した後に端末タイプを設定してやらないと画面制御を使うプログラムが動かせないのでやや不便である)。従って、Unix システム間で行き来する場合には `rlogin` を使うのが便利である。

6.3.2 ファイル転送 — ftp/rcp

遠隔ログインとならんで多くの需要があるサービスは、やはりファイル転送であろう。これも遠隔ログインと同様、Internet 文明共通の `ftp` と Unix 固有に特化した `rcp` の 2 つが存在する。これらの使い方は

```
ftp  ホスト名
ftp  IP アドレス
rcp  ホスト名:パス名  ホスト名:パス名
```

である。`ftp` では相手ホストに接続した後、ユーザ名とパスワードを聞いてくる。これらを正しく入れると `ftp>`とプロンプトが出てくるので、次のような指令を使ってファイルを転送する。

```
cd  パス名       --- 「向こう側で」現在位置を移動
lcd パス名       --- 「こちら側で」   〃
type text        --- テキスト転送モードにする
type binary      --- バイナリ転送モードにする
get ファイル名   --- 向うからこちらにファイルを転送
put ファイル名   --- こちらから向うにファイルを転送
bye              --- ftp を終る
```

一方、rcp はこのような「対話」は不要で、いきなり両ホストのパス名を指定してコピーを行なう。その代わり、パスワードを打ち込む機能はないので、前項で述べた rlogin でのパスワード不要の設定にしてあるホスト間でのみ使うことができる。ちなみに現在入っているホストのホスト名は省略してよい。

一般には FTP は相手計算機のアカウントを持っていなければ利用できないが、インターネットではソフトウェアや文書の配布などを目的として、だれでもアクセスしてよい FTP サイトを運用しているホストもいくつかある。これを「無名 FTP サービス」と呼ぶ。それらのサイトの FTP に接続した場合はユーザ名として「ftp」、パスワードとしてあなたのメールアドレス (ここでは「ユーザ名@gssm.otsuka.tsukuba.ac.jp」ですね) を指定すればそのサイトが公共用に準備してあるファイル群を取って来ることができる。次に例を示す。

```
% ftp ftp.iij.ad.jp
Connected to ftp.iij.ad.jp.
220 ftp.iij.ad.jp FTP server ... ready.
Name (ftp.iij.ad.jp:kuno): ftp
331 Guest login ok, send your complete e-mail address as password.
Password: kuno@gssm.otsuka.tukuba.ac.jp
....       ↑本当は打ち返されないので見えない
230 Guest login ok, access restrictions apply.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
...
pub
etc
...
226 Transfer complete.
136 bytes received in 0.11 seconds (1.2 Kbytes/s)
ftp> cd pub
250-Please read the file README
...
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
GNU
inet
386bsd
...
```

```
README
...
226 Transfer complete.
471 bytes received in 0.36 seconds (1.3 Kbytes/s)
ftp> get README
200 PORT command successful.
150 Opening ASCII mode data connection for README (2396 bytes).
226 Transfer complete.
local: README remote: README
2446 bytes received in 2.2 seconds (1.1 Kbytes/s)
ftp> bye
221 Goodbye.
% cat README
                        IIJ 情報提供サービス
                    Anonymous UUCP のご案内
                    インターネットイニシアティブ
...
```

これを含め、著名な無名 FTP サービスサイトを示しておく。[5]

ftp.iij.ad.jp	IIJ(インタネットイニシアティブ)
ftp.ascii.co.jp	アスキー
srawgw.sra.co.jp	SRA
etlport.etl.go.jp	ETL(電総研)
ftp.foretune.co.jp	フォアチューン
scslwide.sony.co.jp	ソニー CSL
miki.cs.titech.ac.jp	東工大
utsun.s.u-tokyo.ac.jp	東大
ftp.math.metro-u.ac.jp	都立大

6.3.3 遠隔指令実行 — rsh

これは全く Unix 固有の機能で、

```
rsh ホスト名 '指令 ...'
```

とやると、指定したホストで指令が実行できる、というものである。これも rcp と同様、パスワード不要の設定になっている場合のみ利用できる。これは何の役に立つか分かりますか?

[5] ただし! 無用のファイルをやたら取るのはみんなの迷惑だから、探検するのはよいが片端から取ろうとしないこと。皆様のファイル容量制限ではどのみちほとんど持っていられない。共通に入れて欲しいものは久野まで相談のこと。

6.3.4　遠隔ファイルアクセス — NFS

ところで、これまで皆様は smb、smf、smd、smg などいくつかのホストに気ままに窓を開いて作業してきたわけだが、どれにおいても自分のホームディレクトリ以下にあるファイルは同じようにアクセスできた。これは不思議だと思いませんか? 実は、皆様の個人ファイルはすべてサーバである `smb` に入っていて、`ls` とか `cat` とか言うたびにファイルの内容が NFS(Network File System) 機能により **遠隔アクセス** されて持って来られるように設定されていたのでした。NFS の設定状況は

```
/etc/mount
```

により表示できる。例えばホスト `smd` で実行すると次のように表示される。

```
smd% /etc/mount
/dev/sd0a on / type 4.2 (rw)
/dev/sd0d on /usr type 4.2 (rw)
/dev/sd0e on /root type 4.2 (rw)
/dev/sd0f on /swap type 4.2 (rw)
smb:/dc1 on /tmp_mnt/dc1 type nfs (rw,intr)
smb:/u1 on /tmp_mnt/u1 type nfs (rw,intr)
smb:/u2 on /tmp_mnt/u1 type nfs (rw,intr)
smd%
```

これらのうち、最初が`/dev/...` で始まるのは自分のディスクに入っているファイルシステムを意味する。一方、ホスト名`:...` で始まるのは NFS アクセスが指定されていることを意味する。

6.3.5　遠隔ホスト情報 — rwho、ruptime、rmap

ネットワークでつながっている他のホストがどのくらい混雑しているか、またそこでどんなユーザが login しているかを知りたいことがある。これらの情報は次のような指令で見ることができる。

`ruptime`	---	動いているホストと負荷の一覧。
`rwho`	---	login しているユーザの一覧。
`rmap`	---	上 2 つの機能を併せ持つ。画面端末版。

これらのサービスは実際にはコマンドを起動するたびに情報を探しに行くわけではなく、各マシンで定期的に負荷やユーザ状況を報告するプログラム (デーモン) がずっと走っていて、これが決まったホスト (我々の所では `utogw`) のデーモンに状況を UDP パケットで送りつけてくるので、`utogw` のデーモンがその状況をファイルに保管する。各コマンドはこのファイルを読んで必要な情報を抽出して表示しているだけのことである。

6.3.6 情報交換サービス — mail、news

電子メールと電子ニュースについてはもはやそれが何であるかの説明は不要と思うが、その中身がどうなっているかは簡単に説明しておこう。まず、我々のサイト内ではすべてのメール送出やニュース投稿はネットワーク経由でホスト utogw に送られる。utogw ではメールエージェント (MA)、ニュースエージェント (NA) と呼ばれるプロセスが動いて、これらの情報を各ユーザのメール格納ディレクトリ (メールスプール) やニュース格納共通ディレクトリ (ニューススプール) に格納する。

メールスプール、ニューススプールは NFS により各ホストからアクセスできるので、皆様がメールやニュースを読み書きする時にはこれらのファイルをアクセスする好きなプログラムを使ってよい。ユーザがメールやニュースを読むのに使うプログラム (elm、mh、rmail、nn、bb、…) のことを総称してユーザエージェント (UA) とよぶ。

では、外部とのやりとりはどうなっているのだろう? これは、utogw のメールエージェントやニュースエージェントが外部の接続相手のホストと直接ネットワーク経由で接続し、データを交換することで行なわれる。これらの様子を図 6.11 に示す。

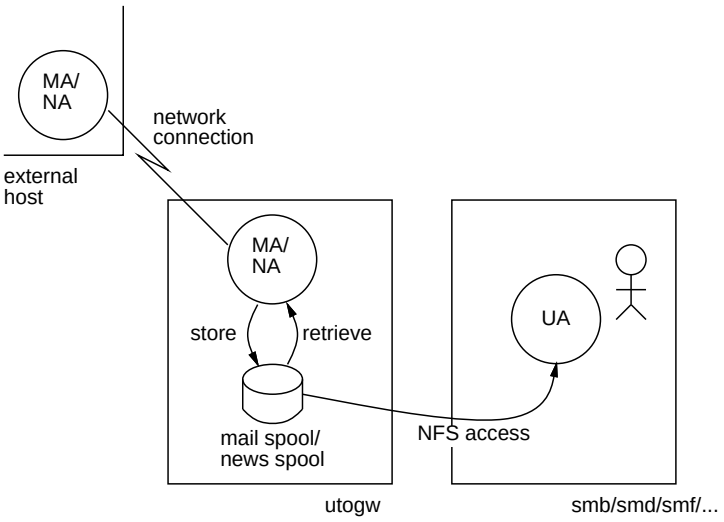

図 6.11: メールとニュースの仕組み

6.4 演習

6-1. `netstat -f inet` で現在の TCP/IP ポート接続状況を調べてみよ。次に、そのホストから telnet/rlogin/X の窓/ftp/rcp などを使って外部と

の接続を行なった上で再度 netstat を使い、どのポートがその接続によるものかを探してみよ。接続を終わるとそれがなくなることも確認せよ。

6-2. send.c/recv.c 例題プログラムを打ち込んで動かせ。ホストアドレスとポートは適宜修正すること。動かしている最中に netstat を使ってどんなポートが増えているか確認すること。

6-3. 送り手が速すぎるという問題を解消する方法を考案し、例題プログラムに組み込んでみよ。

6-4. utogw で traceroute というプログラムを「traceroute 相手先」のようにして動かすと、その相手先までの中継ホストを順次表示してくれる。これをもとに、専攻近辺のネットワーク地図を描いてみよ。(相手先としては匿名 FTP サイトなどを試してみるとよい。)

6-5. 匿名 FTP で適当なサイトに入り、読めそうなファイル (だいたい README などという名前) を持って来て見てみよ。それから解ったことをまとめよ。

本日の課題は上のうち任意の 2 課題とします。