計算機科学基礎'11 ＃4 – ネットワークの原理と構造

久野　靖*

2010.5.11

1　ネットワークの基礎概念

1.1　イントロ: ネットワークの知識とは…

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す。そのことは多くの人が経験ずみであり、今更取り上げるまでもないでしょう。しかし、その裏側にどのような技術的背景があるのかについては、あまり知られていません。今回はこれらの側面にも焦点を当てながら、計算機ネットワークの原理や仕組みについて体系的に整理してみます。

なぜ「ネットワークの原理」を学ぶ必要があるのでしょうか? 別に原理なんか知らなくても使えればよい? 技術的なことは技術屋に相談すれば済む? しかし、技術屋と相談するには、ある程度「技術屋のことば」が分かっていないととんでもない勘違いが起きる恐れがあります。あと、すぐそこに相談できる人がいない時に次のような疑問が湧いたらどうしますか?

> 地球の裏側 (例:ブラジル) の事業所の××管理を日本のシステムで行ってしまうことで、現地でのシステム運用をカットしてはどうかという提案があった (図 1)。となると、ブラジルとの間でデータ往復させることになるが、その所要時間はどれくらいだろう?

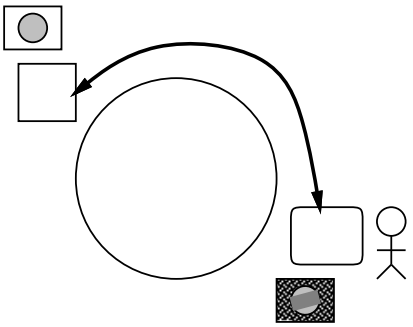

図 1: ブラジルの画面の処理を日本で?

実はこの程度のことは簡単に自分でも調べられます。具体的にはこうすればよいのです。

- Google などで「ぜったいにブラジル現地にありそうな施設」の Web ページを検索して探す。ここでは「ブラジル 国立博物館」の検索で「ブラジル国立歴史博物館」を見つけました。
- そのサーバ名「`www.museuhistoriconacional.com.br` を指定して `ping` コマンドを実行し、パケット往復時間を調べる。

```
% /sbin/ping www.museuhistoriconacional.com.br
PING museuhistoriconacional.com.br (200.198.87.5): 56 data bytes
64 bytes from 200.198.87.5: icmp_seq=0 ttl=228 time=334.865 ms
```

*経営システム科学専攻

```
64 bytes from 200.198.87.5: icmp_seq=1 ttl=228 time=328.980 ms
64 bytes from 200.198.87.5: icmp_seq=2 ttl=228 time=324.031 ms
64 bytes from 200.198.87.5: icmp_seq=3 ttl=228 time=327.908 ms
^C  ←止めるには Ctrl-C
%
```

- 往復時間が 320msec 程度だから、片道は 160msec 程度と分かる。

種明かしをすれば簡単ですよね。そして、片道 160msec くらいなら、たとえば現地で管理画面のボタンを押すと、日本にそれが伝わって処理を行い、結果を現地に返送する、といった処理をしてもさほど問題はないことが分かるわけです。

1.2 ネットワークとその目的

まずそもそも、計算機ネットワークとは何でしょうか? 物理的には、複数の計算機システムが、互いに通信できるように接続されたものがネットワークである、と言えます。しかし例によって、計算機が「何をするか」はソフトウェアによってすべて変わってきますから、そのようなハードウェア (物理的な接続) をどのように使うかが重要です。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挙げられます:

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、複数のマシン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ります。たとえば、信頼性や経済性のために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でしょう (**LAN** — Local Area Network、局所ネットワーク)。一方、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はずです (**WAN** — Wide Area Network、広域ネットワーク)。

1.3 通信媒体とトポロジ option

計算機どうしが通信するためには、互いに信号を伝達する「もの」が必要です。これを**通信媒体**と呼びます。代表的な通信媒体としては次のものがあります。

- 銅線 — もっとも広く使われており、用途や通信速度に応じて**同軸ケーブル** (中心線とそれを囲む網状の線から成り、電気的干渉に強い)、**ツイストペア** (2 本の信号線を一定の率でより合わせてあり、電気的干渉を相殺するようになっている) をはじめ、多くの種類がある。
- **光ファイバー** — ガラスを細長く伸ばしたもので、中にレーザー光線を通すことで遠距離まで減衰なく高速に大容量の通信ができる。
- 赤外線 — ごく近くにある機器どうしで配線をつなげずに通信できる。

- 電波 — 遠距離では、線を引くのが難しいか、コスト的に見合わない場合に使われる。**通信衛星**を経由した通信もこれに含まれる。また、携帯電話の電波を利用した**モバイル通信**も行われている。さらにごく近距離 (数 m〜十数 m) では、高速な**無線 LAN** として普及してきている。

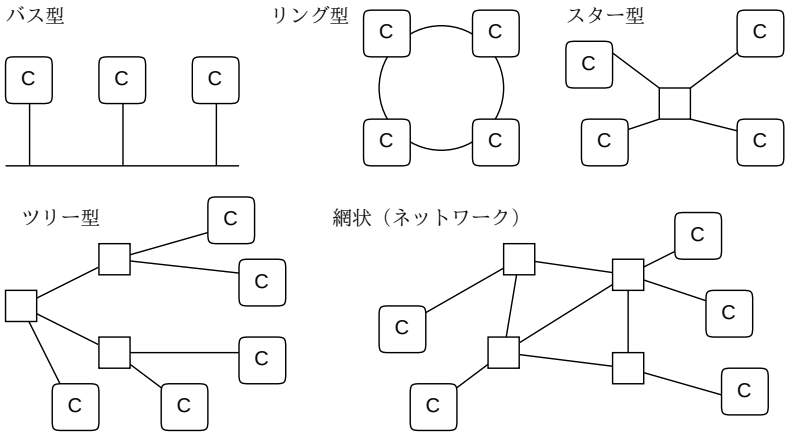

図 2: ネットワークトポロジの基本形

次に、これらの媒体を組み合わせてネットワークを構成するとき、その「つながり方」(**トポロジ**) にもさまざまな基本形があります (図 2)。ネットワークの話をする場合は、そこにつながる各マシンのことを**ホスト**と呼びます (図では C と記してあります)。ネットワークを構成する場合は、ホスト以外に中継機器 (図では無印の箱で示してあります) も使うのが普通です (以前は中継機器も高価だったため、ホストに中継機能を兼ねさせたりして費用を節約することがよくありました。ネットワークが普及した結果、量産効果で中継機器も安くなったため、今ではそのような手間を掛けることは減っています)。

- バス型 — 1 つのケーブルなり媒体に多数のホストをつなぐ。無線 LAN なども 1 つの電波帯域を多数のホストが共有するためこれに分類できる。
- リング型 — 環状につながったホスト間を信号が「一方向周り」に順次伝わることで通信を行う。現在ではあまり見られない。
- スター型、ツリー型 — 中継機器を中心に放射状にホストや中継機器をつなぐ。1 段の場合はスター、多段の場合はツリーだが実際には混在した形になることが多い。
- 網状 — ツリーと異なり中継機器どうしが多数の経路で相互に結ばれているもの。

実際のネットワークはこれらの形がさまざまに入り混じって構成されています。

ではインターネットはどうでしょう。インターネットは多くのネットワークが草の根的につながることで、今日のような全世界にまたがるネットワークを形づくってきました。その一番小さい単位は個々のサイトにある LAN です。そして、各組織は自組織のサイトを相互に接続して組織内ネットワークを構成しています。これが WAN になります。今日では、ネットワーク接続業者 (**インターネットプロバイダ**) が商売のために WAN を構成している部分の比重も大きくなってきています。そして、そのような WAN どうしが 1-1 に接続したり相互接続点 (**NSPIXP** — Network Service Provider Internet eXchange Point) と呼ばれる箇所で多数の相互接続を行ったりして、インターネットができあがっています (図 3)。とりわけ米国はネットワーク発祥の国だけあって、ネットワークが密にあって相互接続が多く、インターネットの中核部分と言えます。たとえば日本から欧州への接続も距離は遠回りになりますが、米国を経由します。

1.4 回線交換とパケット交換

計算機ネットワーク以前から存在しているネットワークの例として、電話網やテレビ放送網などが挙げられますが、これらと計算機ネットワークとでは大きく違う点があります。それは、電話やテレ

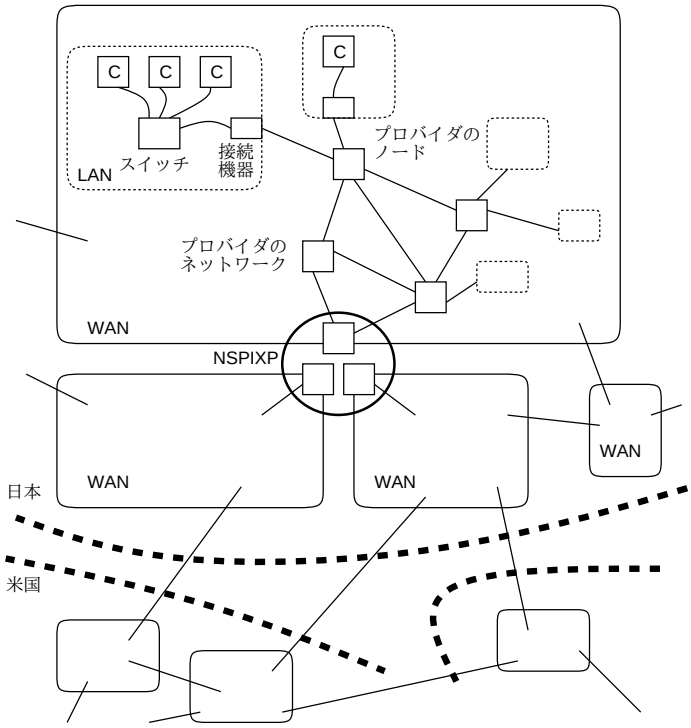

図 3: インターネットの構造

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という点です (もっとも、最近の電話網は計算機ネットワークの技術を利用するようになってきているので、そのような場合は計算機ネットワークと同様になります)。

ですから、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のです。その代わり、いつでも再度話し始めたり、放送を開始することができます。このようなネットワークの接続形態を**回線交換**と呼びます。

これに対し、計算機ネットワーク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います。これを**パケット交換**と呼びます。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でしょう。

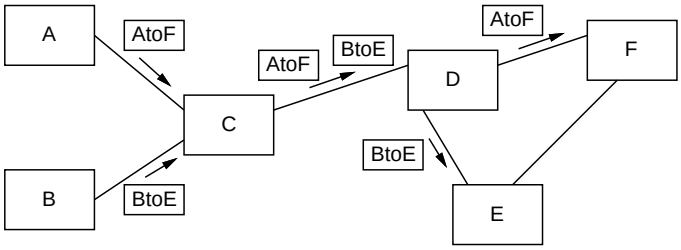

図 4: パケット交換の原理

たとえば、図 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 に着くことになります。各中継点でパケットは一旦受け取られて格納されるので、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 で転送に失敗したことが検知できれば、D に格納されたパケットをもう 1 度送るだけで済みます。[1]

[1]上記は「一時的なエラー」を想定した説明でしたが、たとえば D → F のリンクが長時間不調になっているようなら、

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ります。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すことができます。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ます。

また、回線交換ではリンクを割り当てたり解放したりという管理作業を全中継ノードで行う必要がありますが、ノードはできるだけ簡単にしておきたいのでこのような作業は避けたいということもあります。

さらに、パケット交換では伝送エラーなどにより1つのパケットが失われても、そのパケットだけを再送すれば済みます。回線交換だとエラーが起きたら通信を止めてそこからやり直すことになるでしょうが、これは非常に大変で時間も多く掛かります。

なお、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なので、多数の連続したパケットを用いて転送を行うことになります。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ます。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**と呼びます。

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」を指定する必要があります。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**と呼びます。

現在広く使われているインターネット上の約束 (IPv4) ではアドレス (IP アドレス) は4バイト (32ビット) の値です。IP アドレスを指定するときは、16 進数で指定してもいいはずなのですが、歴史的な慣習から4つのバイトをそれぞれ0〜255の十進数で表したものを「.」でつなげて書きます。たとえば32ビットの値が (16 進数で表して)「0a020205」であれば4バイトはそれぞれ「0a」「02」「02」「05」なので「10.2.2.5」のように書き表すわけです。

普段使っている「google.co.jp」とかそういう「読める名前」と違う、と思いましたか? これは、パケットに入れ、多数の中継点で参照されるアドレスは、計算機が効率よく扱える固定長の数値が望ましいためです。普段我々が使う長い名前は、最後は IP アドレスに変換されますが、その仕組みについては少し後で。

また、IP アドレスだけでは「どのホスト」までしか指定できないので、「どのホストのどのプログラムないしサービス」を指定するために別に 16 ビットの値を指定します。これを**ポート番号**と呼びます。**ポート**というのは、ネットワークでの接続を行う「接続点」のことだと思えばよいでしょう。1つのマシン上で多数のプログラムが同時に動作してネットワーク通信を行いますが、それぞれが自分用のポートを持っていて他のホストのプログラムと通信するので、ごちゃまぜになることはないわけです。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」だと言えます。

1.5 簡単なネットワークプログラム

お話ばかりではつまらないので、ごく簡単なネットワークプログラムを動かしてみましょう。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ます。まず受け取る側から見ましょう。

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

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

main(int argc, char *argv[]) {
  int fd, len, fromlen;
  char buf[100];
  static struct sockaddr_in adr;
  adr.sin_family = AF_INET;
```

A から F へのパケットを E 経由で送るように切り替えることもあります。

```
  adr.sin_addr.s_addr = INADDR_ANY;
  adr.sin_port = htons(atoi(argv[1]));
  if((fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0 ||
      bind(fd, (struct sockaddr*)&adr, sizeof(adr)) < 0) {
    perror("socket: "); 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取り込むための指示です。次に main の冒頭で必要な変数を用意しています。fd はソケット (ポート) のためのファイルディスクリプタ番号を入れる変数、len と fromlen はデータの長さを入れる変数、buf はパケットデータを格納するバッファです。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、そのホスト部は「任意」、ポート番号はコマンド引数で指定した文字列を整数に変換して、なおかつネットワークバイト順に変換したものを入れます。

ここまでで用意ができたので、まずソケットを作成してそのディスクリプタ番号を fd に入れ、次に 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します (いずれかに失敗したらメッセージを出して終わり)。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ください。要は OS に頼んでポート (ソケット) を準備している、ということです。

ポートが準備できたらあとは recvfrom というシステムコールを呼び、パケットの到着を待つよう OS に頼みます。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ます。

このプログラムを動かす前に、動かすマシンの IP アドレスを調べておいてください。それには ifconfig(パスの設定によっては/sbin/ifconfig などのように絶対パスでコマンドのありかを指定する必要があるかも知れません) というコマンドを使います。

- ifconfig — ホストに備わっている**インタフェース** (ネットへの接続口) の一覧とそれぞれの情報を出力

```
% /sbin/ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      options=3<RXCSUM,TXCSUM>
      inet 10.3.25.20 netmask 0xff000000 broadcast 10.255.255.255
      ether 00:0f:ea:10:4c:71
      media: Ethernet autoselect (1000baseTX <full-duplex>)
      status: active
plip0: flags=8810<POINTOPOINT,SIMPLEX,MULTICAST> mtu 1500
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
      inet 127.0.0.1 netmask 0xff000000
%
```

インタフェースが複数表示されると思いますが、その中から inet(IP アドレス) の値が表示されていて、なおかつ 127.0.0.1 ではないものを選んでください。上の例では 10.3.25.20 が IP アドレスということになります。IP アドレスが分かったら、プログラム自体は

```
% gcc -o recv recv.c  ←プログラム名を指定してコンパイル
% recv ポート番号
(受信待ちになる)
```

のようにして起動します。ここで指定するポートは 0〜65535 の数値ですが、1024 より小さい値は指定できない (特別なポート番号として予約されている) ので、適宜大きな値を指定してください。

では次に送り側のプログラムを示しましょう。

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

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

main(int argc, char *argv[]) {
  int fd;
  char buf[20];
  int len;
  struct sockaddr_in adr;
  adr.sin_family = AF_INET;
  adr.sin_addr.s_addr = INADDR_ANY;
  if((fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0 ||
      bind(fd, (struct sockaddr*)&adr, sizeof(adr)) < 0) {
    perror("socket: "); exit(1); }
  adr.sin_port = htons(atoi(argv[1]));
  adr.sin_addr.s_addr = inet_addr(argv[2]);
  while((len = read(0, buf, 20)) > 0) {
    if(sendto(fd, buf, len, 0, (struct sockaddr*)&adr, sizeof(adr)) < 0) {
      perror("sendto: "); exit(1); }
  }
}
```

こちらもポートの準備までは先と同様です。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ます。あとは繰り返し、入力からデータを読んでは sendto でパケットとして送ります。なお、IP アドレス、ポートはプログラムの引数として指定します。ではこれを使ってメッセージを送ってみましょう。

```
% gcc -o send send.c ←コンパイル指定は recv と同じ
% send ポート番号 IP アドレス
hello.
this is a pen.
...
```

すると、recv を動かしている側に打ち込んだものが現れるはずです (ちなみに止める機能はないので、Ctrl-C で中止させてください)。

ところで、send でリダイレクションを使えばファイルの内容を送ることができます。ちょっとやってみましょう。

```
% less t
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
...
% cat t t t t | send アドレス ポート ←沢山送る
```

すると、データを待っていた recv が再開されます。

```
(先の続き)
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
...
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbb  ←あれ?
...
```

大体よさそうですが…一部データが抜け落ちている??? これは、送り側のマシンが受け側のマシンより速いため、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ためです (これを体験するためには速度差のあるマシンから送るなどして、しかもやや大きいファイル、たとえば 4096 文字くらいあるファイルで実験しないと再現しないのでそのつもりで)。

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ります。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、ということはおわかり頂けると思います。次節以降では、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ましょう。

2 ネットワークとプロトコル

2.1 システムと階層構造

計算機システムもそうですが、多くのシステムは「階層構造」(layered architecuture) を持ちます。その基本的な考え方は、「全体を層状に構成し」「下の層の詳細は上の層に影響しないようにする」ことです。

たとえば、あなたが遠方の相手方と文書を検討するとして、その文書を秘書に送らせるとします。秘書が文書を送るのには「郵便」「クロネコメール」「FAX」など複数の手段がありますが、それはあなたは関知しなくてもよい。秘書はどの手段を使うかで作業内容が違いますが、郵便を使うとして、郵便ポストに入れたら、郵便局が集配に自動車を使うか、バイクかといったことは関知しなくてもよい (図 5)。逆に秘書はあなたの検討の内容には関知しないし、郵便局は秘書が送ったものの内容には関知しないわけです。

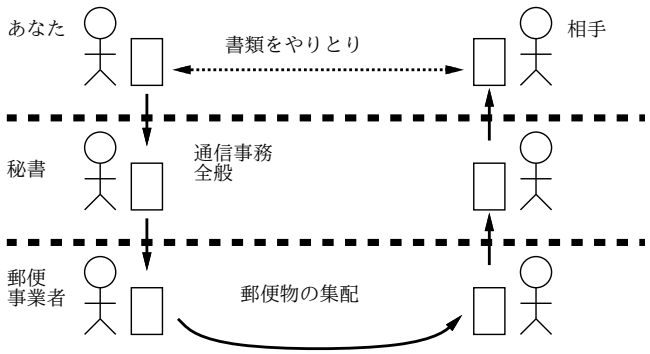

図 5: 階層構造

このように、階層構造を用いると「それぞれの層の中をうまく構成する」ことと「隣接する層とのやりとりをする」ことだけに限定して考えれば済むので、問題の複雑さが小さくなります。ネットワークシステム全体もそういう風に構成されているわけです (そうしないと作れないくらい込み入っている)。

たとえば、「パケットの抜けや追い越しがある伝送」という階層の上に「パケットが順番に抜けなく送られる伝送」を行う階層を作るとしたら、どうすればいいでしょうか? それには次のようにすればよいのです (図 6)。

1. 送り側は送り出すパケットに一連番号をつける。
2. 受け側は相手からパケットが来たら確認パケットを送る。
3. 送り側は一定時間待っても確認が来なかったら再送する。
4. 受け側は一連番号の順にパケットを渡す

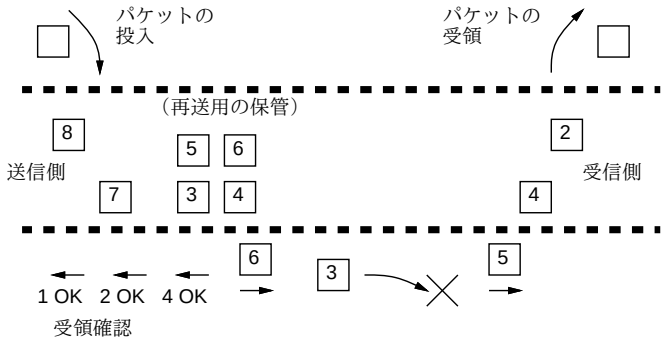

図 6: エラー制御のある伝送

このようにすると、下の階層でエラー (抜け等) があっても、上の階層にはエラーのないサービスが提供できます。ただし、下の階層でのエラーに対処するためには「再送」「確認待ち」などが必要であり、エラーが多くなればなるほど伝送は (上の階層から見て)「遅く」なるように見えます。

2.2 ネットワークソフトの階層構造

話を戻して、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のようなものがあると思いますか? たとえば、あなたが遠隔地のホストから手元のホストにデータを取り寄せたいとします。そのとき、具体的にどのような機能が必要になるでしょうか? (以下にある箇条書きには同じ記号が何回も現れていますが、その理由は後で分かります。)

a. まず、あなたは相手ホストとデータの所在を指定して取り寄せるためのコマンドを起動します。

a. そのコマンドは何らかの形で相手ホストの対応する (データの取り寄せのような機能を提供する) サービスに連絡をとり、指定したファイルを送ってくれるように依頼します。

a. 相手ホストのサービスは依頼に応じて、データをパケットに入れて順次送り出してくれます。それを手元側で順次受け取ってファイルに格納します。

b. 既に学んだように、相手側がパケットを順次送ってくれたとしても、途中でデータが欠落したり順番が入れ替わったりすることもあります。ですから、送られた通りのものが受け取れているかチェックし、並べ直したり欠落部分を再送してもらったりすることが必要です。これを**エラー制御**といいます。

c. さらに、パケットは多くの中継機器を経由して来るので、それぞれの中継機器において正しい行き先に向けてデータを中継してもらう必要があります。これを**経路制御**といいます。

d. ホストや中継機器相互でのやりとりに当たっては、パケットを正しく伝送するための機能が必要です。とくにバス型の媒体では複数の機器が同時に送信しようとして信号が干渉するのを防ぐ制御が必要です。

e. 一番下のハードウェアレベルでは、媒体を構成する物質の上を信号が伝わって行くことで情報を運びます。

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 (データを取り寄せる等) から一番下のハードウェアまで、多数の**階層** (**レイヤ**) が積み重なって作られていることが分かります。

なぜ階層が重要なのかを整理しておきましょう。それは、階層に分けて考えることで機能を適切な (複雑すぎない) 大きさに分解して考えることができ、階層と階層の間の約束を決めておけばそれぞれを独立に用意し組み合わせられるからです。たとえば「パケットを送る」という機能さえ提供されれば、具体的な媒体は何であっても (光でも銅線でも赤外線でも) 同じようにネットワークを使うことができる、というのはこのような階層構造の利点です。

2.3 OSI 参照モデルとプロトコル

ISO(国際標準化機構) 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、上に述べたような階層化の標準モデルを提案しました。これは 7 つの階層から成り、**OSI 参照モデル**ないし **7 層モデル**などと呼ばれます。その構成を図 7 に示します。

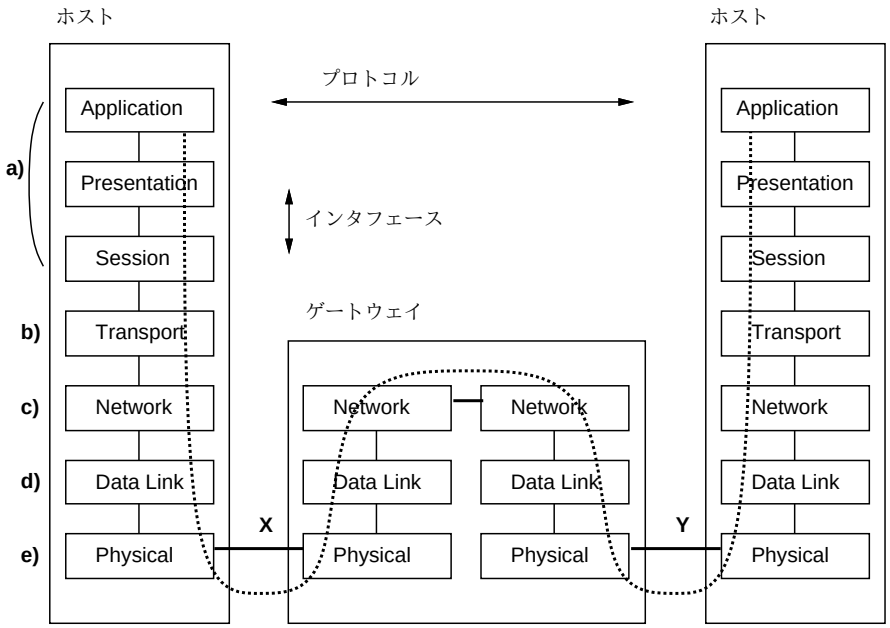

図 7: OSI 参照モデル

7 つの層は具体的には下から順に、次のような機能に対応しています。

1. 物理層 — 信号を運ぶ媒体に対応 (前節の e)
2. データリンク層 — 媒体を使った信号の送受を制御 (前節の d)
3. ネットワーク層 — 経路制御によりパケットを目的地まで中継していく (前節の c)
4. トランスポート層 — エラー制御により誤りのないデータ送受を行う (前節の b)

ここから先は前の節では区別せず a と記しましたが、OSI モデルでは次のように別れています。

5. セッション層 — 通信の開始/終了/状態記憶などの機能を提供する
6. プレゼンテーション層 — 通信に適した形にデータを変換したり、それを復元する
7. アプリケーション層 — 具体的な個々のサービス (ファイル転送、メール送信、等) に対応

データを実際に送信するときは、送り側のホストのアプリケーション層の機能がそれを下へ下へと渡して、物理層を通じて送り出します。中継点ではパケットを受け取り、経路制御を行い、次の行き先に向かって送り出しますから、ネットワーク層までが稼働していることになります。受け側のホス

トでは、受け取ったパケットのデータは上へ上へと渡され、最後はアプリケーション層でそれを用途にあった形で処理します。

このとき、同じ層のソフトウェアどうしが共通に従う約束を**プロトコル** (通信規約) と呼びます。たとえば、物理層では使用する電圧や信号の周波数などの約束が必要です。データリンク層では、パケットの開始や終了を表す信号、干渉を避けるための信号の取り決めなどが必要です。ネットワーク層では、パケットのどの場所に送り先のアドレスを格納し、アドレスの形式はどのようなものか、といった取り決めが必要です。トランスポート層では、順番をチェックするためにどのような形でパケットに一連番号を割り振るか、パケットの正しい到着や欠落をどうやって送り元に通知するか、といった取り決めが必要です。これらはすべてプロトコルの例です。そして、各層のプロトコルを合わせた全体を**プロトコル群**と呼びます。その代表的なものに、インターネットで使われている**インターネットプロトコル群** (その中の代表的なプロトコルの名前を取って **TCP/IP** とも呼ばれます) があります。

2.4 TCP/IP

さて、抽象的なお話はこれくらいにして、ここからは TCP/IP を題材としてもっと具体的に見ていくことにします。実は TCP/IP にも現在広く使われている **IPv4** と、次世代の規格として作られ実用化され始めている IPv6 とがありますが、以下では IPv4 について説明します。本書で述べる範囲では、アドレス表記やプロトコルやプログラムの名前が違う程度でして、両者に原理的な違いはありません。

プロトコル群とはネットワークの各層を構成するプロトコルの集まりだと先に書きましたが、TCP/IP の場合は図 8 のような構成になっています。

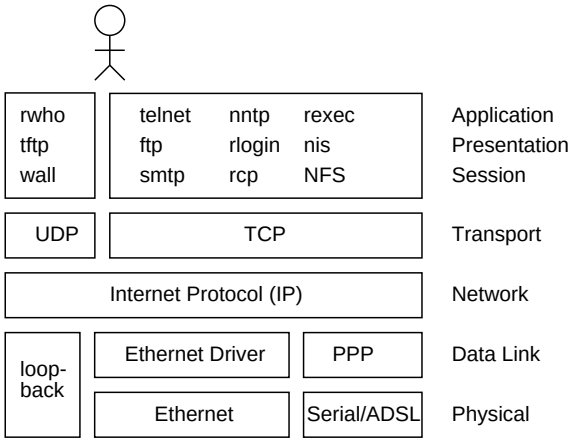

図 8: TCP/IP のプロトコル構成

ここで目につくのは、中間のネットワーク層に相当する **IP**(Internet Protocol) はすべてに共通していて、その上や下では層ごとにプロトコルが複数存在している点です。このうち、下側 (データリンク層と物理層) については、通信のための媒体が何であるかによって使い分けられます。しかし、どの媒体であっても最終的には IP のパケットをやりとりさせてくれるので、その上側では媒体が何であるかに関わらず同じように使えるわけです。ノート PC から携帯電話経由でネットに接続しているときと学校の無線 LAN 経由でネットに接続してるときとで使えるコマンドがまったく違っていたら嫌でしょう?

一方、IP より上側のプロトコルは、**UDP**(User Datagram Protocol) に基づくものと **TCP**(Transmission Control Protocol) に基づくものとに大別されます。UDP は単独のパケットのやりとりに基づくもので、少量のデータを迅速にやりとりしたい場合に向いています。これに対し、TCP は先に述べた仮想回線の機能を提供し、遠距離でも安定してデータをやりとりできる特徴があります。

次節以降で、TCP/IP のプロトコル階層を下から順にもう少し詳しく見ていきます。

3　物理層とデータリンク層

3.1　イーサネットと無線 LAN

前節で説明したように、物理層とデータリンク層はおもに媒体の種類によって決まってきます。まず最初に、LAN の媒体としてもっとも広く使われている**イーサネット** (Ethernet) について説明しましょう。

イーサネットは LAN のための媒体と伝送制御の方式として 1970 年代後半に Xerox 社によって開発され、その後 IEEE によって **IEEE 802.3** という番号の規格として標準化されました。イサーネットは通信媒体として当初は干渉の起きにくい同軸ケーブルを使用していましたが、技術の進歩から今日ではツイストペアケーブル (より線対 — 2 本の線をより合わせて外部からの電波干渉が相殺されるようにしたもの) を使い、ハブまたはスイッチと呼ばれる箱から各機器までの間を結びます (図 9)。

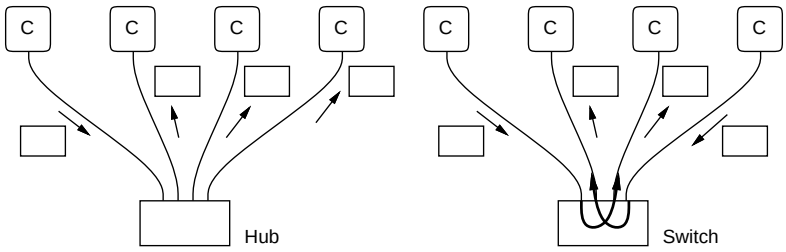

図 9: Ethernet の原理

ハブは単なる増幅器で、1 つのマシンがパケットを送出すると全部のマシンがそれを受信できます。イーサネットの機器にはすべて 48 ビットの **MAC アドレス**と呼ばれる固有のアドレスが割り振ってあり、特定マシン宛のパケットはその宛先のマシン以外は受け取っても無視します。どのマシンがどの MAC アドレスかを調べるなどの場合は**ブロードキャスト**と呼ばれる「全員あて」のパケットを利用します。これらの制御がデータリンク層の機能に相当します。

各パケットが全マシンに中継されるため、パケットは同時には 1 つのマシンからしか送信できません。このため、各マシンは信号を観測していて、他のマシンが送信中は送信を開始せず、ケーブルが「空いて」いるときだけ送信を試みます。しかし、たまたま複数のマシンが同時に送信を始め、信号が衝突して読みとれなくなることも起こります。このため各マシンは送信中もケーブルの信号を観測していて、衝突が起きたら送信を中止し、しばらく (乱数によって決めた時間) 待ってから再度送信を試みるようになっています (これもデータリンク層の機能)。この方式を **CSMA/CD**(Carrier Sense Multiple Access/Collision Detection) 方式と呼びます。CSMA/CD では伝送量が多くなってくると衝突と再送が増え、ネットワークの有効通信量 (スループット) はそれ以上増えなくなります。[2]

この問題を解消したのがスイッチで、スイッチでは各パケットの Mac アドレスを認識し、「正しい」宛先だけにパケットを中継します。このため、関係ないマシンどうしであれば (スイッチ内部の多重度の範囲内で) 同時に通信することができ、高いスループットが得られます。

より線対を使ったイーサネットでは通信速度は 10Mbps〜1Gbps で距離は 100m 程度ですが、光ケーブルを使ってより高速/長距離の通信を行える機器もあります。また、中継機器の中にはさらに上の IP 層の処理まで取り込んだものもあります (レイヤ 3 スイッチ、ルータなどと呼ばれます) が、これについては次の節で説明します。

無線 LAN はイーサネットの通信線の代わりに電波を使うもので、いわば「空間」がハブに相当することになります。当然、衝突によるスループットの低下もありますが、他の機器等電波との干渉などの問題も起きやすいという弱点もあります。また、電波だと盗聴しやすいので、WPA(Wi-Fi Protected Access)、WPA2(Wi-Fi Protected Access 2)、WPS(Wi-Fi Protected Setup) などの暗号機能を併用することが多く行われます (無線 LAN 機器に組み込まれている)。[3]

[2]たとえば 10Mbit/秒の伝送能力があっても実際には 3〜4Mbit/ 秒くらいで伝送能力が飽和してしまうわけです。

[3]無線 LAN 普及初期の暗号化機能である WEP(Wireless Encription Privacy) は、解読方法が確立されてしまっている

3.2 対向接続、仮想ネットワーク option

イーサネットや無線 LAN は局所的に複数のマシンを接続するのに適していますが、これとは違う接続方法として、たとえば自分のマシンを電話線経由などでプロバイダのアクセスポイントにつなげてインターネットに接続する場合もあります。この場合、「自分のマシン」と「アクセスポイント」という 2 つの地点の間だけでネットワークを構成するので (対向接続)、イーサネットのような制御は必要としませんが、その代わり接続した 2 点間でアドレスを調整したり、接続開始/切断などの処理が必要になります。このような機能を持ったデータリンク層のプロトコルとして **PPP** (Point-to-Point Protocol) が広く使われています。PPP は、物理的な媒体からは独立していて、電話線のほかに ADSL や光ファイバや赤外線による接続などでも使われます。

図 8 の下の方には、**ループバック**と名付けられたデータリンク層が存在しています。実はこれは「本物の」ネットワーク媒体を扱うものではなく、あるマシンから自分のマシンに対する通信を扱う仮想的な (実質はないがあたかもネットワークのように使える) 媒体を表しています。

なぜこういうものが必要なのだでしょう? たとえば、マシン A とマシン B で 2 つのプログラムが通信し合いながら動くようなシステムがあったとして、マシン B が不調なので一時的に両方のプログラムをマシン A で動かすように変更したいとします。このとき、プログラムを変更しなくても単に「マシン B あて」の通信を「マシン A あて」に変更するだけで済めば助かります。このような場合に「マシン A からマシン A あて」の通信を扱うデータリンクが役に立つわけです。

これと似た例として、既存のネットワークを経由して、あるマシンと他のマシン (ないしマシン群) を結ぶデータリンクを作ることもあります。なぜ、既にネットワークプロトコル一式が動いているのに、その上でまた「ケーブルの真似」をするのでしょうか? それは、上に載せたデータリンクによって、既にあるネットワークをそのままでは通過できない種類のパケットを通したり、暗号機能を使って盗み見られても通信内容が知られないようにするなどの機能を追加できるからです。このようなデータリンクを、2 地点間の場合は**トンネル**、多地点間の場合は**仮想ネットワーク**、**仮想プライベートネットワーク** (VPN) などと呼びます。

3.3 ネットワークインタフェースの観察

Unix ではデータリンク層の機能は**ネットワークインタフェース**に付随した形で提供されています。先に出てきた **ifconfig** コマンドを使うと、自分が使っているマシンにどのようなインタフェースが備わっているかが分かりますが、併せてそれぞれのインタフェースがどのようなデータリンク機能を提供しているか、その状態がどうなっているかも見ることができます。物理的な (ケーブルのつながった) インタフェース以外に、仮想ネットワークに対応するインタフェースも、`ifconfig` コマンドでそのようすを調べることができます。

また、ネットワークに関する各種情報を表示させるコマンド **netstat** にはさまざまなオプションが用意されていますが、その中の`-I` というオプションを指定することで、あるインタフェースを通過したパケット数を調べることができます。

- `netstat -I` インタフェース名 — システム起動時から現時点までにそのインタフェースを通過したパケット数累計を表示する。
- `netstat -I` インタフェース名 秒数 — 指定した秒間隔ごとに、前回表示時点以降にそのインタフェースを通過したパケット数を表示する。

たとえば 2 番目の指定を使って通過パケット数を表示させながらネットワークを使う操作 (たとえばブラウザによるページ表示など) を行うと、データの取り寄せ時に多数のパケットがインタフェースを通過することが分かります。

```
% netstat -I fxp0 1
            input        (fxp0)         output
   packets  errs   bytes    packets  errs   bytes colls
```

ため、今日では使用すべきではない。

```
          0     0       0          0     0       0     0
          1     0     160          0     0     226     0
          2     0     675          3     0    2056     0
          3     0      60          2     0      66     0
          2     0       0          3     0       0     0
          1     0     941          0     0     785     0 ←データ
          3     0   41961          4     0   27492     0  取り寄せ
         36     0   19145         32     0   17968     0
         56     0   67193         50     0   60948     0
         33     0   38244         30     0   57064     0 ←完了
        118     0       0        111     0       0     0
          2     0       0          2     0       0     0
          0     0       0          0     0       0     0
^C  ←Control-Cにより中止
%
```

4　ネットワーク層

4.1　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す。TCP/IP のネットワーク層である IP では、「相手」を指定するのためには、既に学んだ 32 ビットの IP アドレスを使います。

実際には、IP アドレスはマシンに備わったインタフェースごとに割り振られます (このことは `ifconfig` コマンドで各インタフェースごとに違う IP アドレスが表示されることを見ればよく分かります)。通常ユーザが使うマシンは物理的なインタフェースは 1 つだけ持つことが多いので、その IP アドレスがマシンのアドレスだと思って構わないのですが、サーバなど多数のネットワークに接続されたホストでは「複数あるうちのどのインタフェースに接続するか」を意識する場合もあります。

IP アドレスの話題に戻りますが、32 ビットのアドレスのうち、上位の何ビットかが「ネットワーク番号」、残りのビットが「ホスト番号」となります。[4][5]たとえば 1 つのイーサネットセグメント (スイッチやハブなどで結合され通信し合える範囲) が 1 つのネットワークに対応し、そこに接続されている各ホストはネットワーク番号部分はどれも同じで、ホスト番号部分だけがそれぞれ異なる IP アドレスを持つ、というふうになります。

正式な IP アドレス (**グローバル IP アドレス**) は、世界中 (インターネット中) で重複がないように管理されていて、ある特定の IP アドレスを持つホストは世界中でただ 1 つしか存在しないようになっています (実際にはネットワーク番号を重複しないように各組織に割り当て、ホスト番号はその組織のネットワーク管理者が割り当てます)。

これらの割り当てについては、国際非営利組織 **ICANN**(Internet Corporation for Assigned Names and Numbers) が管理を行っています。

```
http://www.icann.org/
```

また、各国には下請けとしてその国の範囲での割り当て調整組織がありますが、日本の場合は **JPNIC** がこれに相当します。

```
http://www.nic.ad.jp/
```

[4]かっては、32 ビットのうち上位ビットのパターンによってネットワーク部の長さが 8 ビット、16 ビット、24 ビットのどれかに決まるようになっていました (それぞれ「クラス A」「クラス B」「クラス C」と呼ばれていました)。しかしそれではアドレスを柔軟に割り当てるのが難しいため、現在ではネットワークアドレスごとにネットワーク部の長さを指定しています。

[5]例えば単にネットワーク番号が `192.168.0.0` と言っただけでは、何ビットがネットワーク部の長さか分からないので、その長さを併せて示す必要があります。ネットワーク部の長さを指定する方法として、「`192.168.0.0/24`」のように、ビット数をアドレスの後ろに「`/`」で区切って指定する方法と、そのビット数だけ上位に「`1`」が並んだ 32 ビットの値 (**ネットマスク**値) を書いて「`192.168.0.0 netmask 0xffffff00`」のように指定する方法とがあります。

実際にアドレスを必要とするプロバイダ (接続業者) 等は、これらの管理組織から自社のネットワークおよび自社の顧客のためのネットワーク番号を割り当ててもらって使用する、という仕組みになっています。

逆にいえば、自分で勝手に IP アドレスを設定してインターネットに接続することは厳禁なわけです。しかし、外部に接続しない孤立したネットワークを構成する場合にもいちいちアドレスの割り当てを受けるのは合理的でありませんから、そのような孤立したネットワークでは自由に使っていいネットワーク番号がいくつか用意されています。これらのネットワーク番号に属する IP アドレスのことを**ローカルアドレス**と呼びます。

ところで、インターネットの急激な成長のため、当初 32 ビットあれば十分だろうと考えられていた IP アドレスの数が足りなくなってきました。この教訓から、新しい世代の IP(IPv6) では IP アドレスを一挙に 128 ビットに増やし、アドレス不足がまず起こり得ないようにしています。

一方、IPv4 を使っている多くの組織は、グローバル IP アドレスを少しだけ割り当ててもらい、大多数のマシンはローカルアドレスを用いて運用するという方針を採っています。というのは、組織内のマシンが直接インターネット全体と通信することはセキュリティ上の理由から避けた方がよいので、多くのマシンは外部には直接接続されないネットワークにつなぎ、外部とのやりとりするには特別な中継ノード (ないし中継マシン) を経由してのみ許すようにしているからです。その場合は、外部とやりとりしないマシンはローカルアドレスでも済みます (中継ノードは内部のマシンのパケットを外部に中継したり、その逆を行わないように設定する)。

しかしそれでも用途によっては内部のマシンと外部との接続を行いたい場合があります。その場合にはパケットに埋め込まれているアドレスを系統的に書き換えることで、インターネット側の接続相手にとってはグローバルアドレスを持った中継ノードと接続しているように見せかけながら、実際には内部のマシンと接続を行うという手法が使われます。これを **NAT**(Network Address Translation) と呼びます。

ただし、NAT は「見せかけ」を行って外部の相手をだましているので、グローバル IP アドレスを持つマシン相互のような自由な通信は難しいという問題があります。将来的には、IPv6 への移行によって内部のマシンまですべてが (必要なら) グローバル IP アドレスを持ち、中継ノードはアドレスの書き換えは行わず、安全が確認されている接続とそうでない接続を区別して前者だけを通過させることに専念するのが筋でしょう。

さて、実際にインターネットにつながっているホストからは、制御パケットを特定ホストあてに送って相手が「生きている」かどうか、パケット往復時間はどれくらいかを計測できます。それには **ping** というコマンドを使います。これは冒頭のイントロで例を出しましたね。なお、一時期クラッカーの攻撃対象探索用にこの `ping` が悪用されたため、`ping` が使うパケットへの応答を止めているホストも多数あります。

宛先は IP アドレスでも指定できますが、通常使っている名前をした場合も後で述べる方法で IP アドレスに変換してくれます。混雑した/品質の悪いネットワークを経由している場合は、パケットが落ちて番号が「とびとび」になるのでそれと分かります。

4.2 IP と経路制御

ネットワーク層の最大の「魔法」は、行き先を指定すると与えられたパケットをその行き先にちゃんと送り届けてくれるところにあります。この機能を (パケットが通過する経路を適切に制御してくれるところから) **経路制御**と呼びます。経路制御の機能を持つ中継機器を一般に**ルータ**と呼びます。また、データリンク層の機器に属するスイッチも高度化してルータのような機能を持つようになったため、このような機器をとくに**レイヤ 3 スイッチ**と呼ぶこともあります。

指定された行き先にパケットを届けることがなぜそんなに偉いのでしょう? たとえば、郵便物の場合を考えて見ましょう。ある郵便局に集まってきた葉書に「東京都目黒区駒場 1-1-1」という宛先が書いてあったとすると、そこが東京都以外の郵便局であれば、東京に「目黒区」という区があるかどうか知らなかったとしても、とにかく東京中央郵便に送れば済みます。東京中央郵便局では、東京都

のどの区や市はどの局の受け持ちか知っていますから、「駒場」という地名を知っていてもいなくてもとにかく目黒郵便局に送れば済みます。

これが可能なのは、住所が「都道府県→区市町村→地名→番地」という**階層構造**になっているからです。アドレスが階層構造になっていると、それぞれの中継地点では「自分の受け持ち範囲でないアドレスはとにかく上位の中継地点に送る」「受け持ち範囲のアドレスはより小さい受け持ち範囲の中継地点や個々の宛先に送る」という方法で経路制御が行えます。

しかし IP ではアドレスは 32 ビットの数値であり、このような階層構造になっていません。いわば、「駒場」とか「八雲」とか「鷹番」とかいう名前だけが与えられて、それだけで送り先を決めなければならないのに相当します。そうなると、すべての主要な郵便局には全国のあらゆる地名の一覧表があり、その一覧表に「この地名だったらこちらの方に送れば付く」と書かれているのでそれに従って郵便物を送る、といったことが必要になるわけです。

言うのは簡単ですが、これは非常に大変です。どう大変かというと、まず巨大な一覧表 (**経路表**) をすべての中継点で保持し、パケットが来るごとにこれを検索して行き先を決める必要があります。次に、その一覧表を正しく維持し続けることも大変です。そこで、これらの手間をできるだけ少なくするために、次のような工夫をしています。

- ネットワークの末端部分 (たとえば図 10 のような部分) では、インターネットの主要部分と行き来する経路は 1 通りであることが普通なので、その入口に当たるルータ (図の A、B、C など) では末端側のネットワークのみ正確な経路を表に記載しておき、「その他はすべてこちら」という「標準 (default) の経路」を設定することで、大きな表を持つ必要がなくなります。

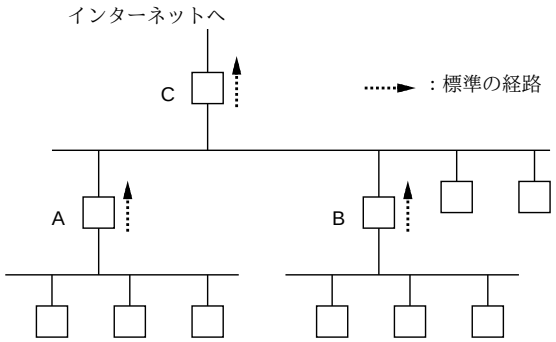

図 10: 標準経路

- アドレス範囲が隣接したネットワークが多数まとまっている場合、それらのアドレスをネットワーク部分の長さがより短い 1 つのネットワークアドレスとして**集約** (aggregate) することで、外部からは 1 つのネットワークとして経路制御できます。たとえば `10.1.0.0/24`〜`10.1.255.0/24` という連続した範囲の 256 個のネットワークがあった場合、これらはまとめて `10.1.0.0/16` という 1 つのアドレスに集約できます (図 11)。[6]

アドレスの集約については、そのような都合がよいことがたまたま起きるのかと思われるかも知れませんが、実際には大学やプロバイダなど多数のネットワークを抱える組織ではまとまった範囲のアドレスの割り当てを受け、各組織や客先にはこの範囲から実際にこのような集約が可能なように小分けにしたアドレスを配るわけです。

このような工夫をしてもやはり、インターネットの主要な中継点では非常に大きな経路表が必要となります。また、経路表を正しく維持するためには、ルータどうしで経路情報を交換して、常に経路表を正しい状態に保つためのプロトコル (**経路制御プロトコル**) とそれを取り扱うソフトウェアが運用されます。

手元のホストから特定ホストまでの経路を調べるためのコマンド **traceroute** を使うと、どの中継点を通り、そこまでどれだけ往復時間が掛かっているかを調べることができます。これも使えなくし

[6] もちろん、これらのネットワークの近辺では別々のネットワークとして経路制御を行う必要がありますが、外部からは 1 つのネットワークに見えます。

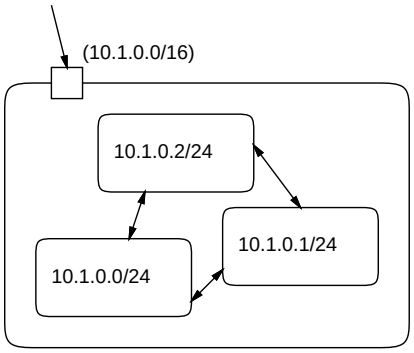

図 11: アドレスの集約

てあるところがあります (我々のサイトでも普段は止めてあり、実習の時だけ使えるように設定を変更しています)。

```
% /usr/sbin/traceroute www.museuhistoriconacional.com.br
traceroute to museuhistoriconacional.com.br (200.198.87.5),
  64 hops max, 40 byte packets
 1  plalagw (210.154.96.161)  1.306 ms  0.429 ms  0.331 ms
 2  218.44.77.194 (218.44.77.194)  1.840 ms  1.985 ms  2.325 ms
 3  218.44.77.193 (218.44.77.193)  2.499 ms  2.662 ms  2.363 ms
 4  218.47.158.169 (218.47.158.169)  2.911 ms  2.537 ms  2.886 ms
 5  218.43.251.249 (218.43.251.249)  3.086 ms  3.245 ms  2.493 ms
 6  221.184.4.5 (221.184.4.5)  3.155 ms  2.970 ms  2.902 ms
 7  210.145.252.153 (210.145.252.153)  3.039 ms  3.567 ms  2.773 ms
 8  61.207.0.150 (61.207.0.150)  3.438 ms  2.737 ms  3.447 ms
 9  xe-1-3-0.a21.tokyjp01.jp.ra.verio.net (61.120.145.189)
    3.708 ms  3.432 ms  2.964 ms
10  xe-1-0-0.r20.tokyjp01.jp.bb.verio.net (61.213.162.229)
    11.963 ms  3.381 ms  2.596 ms
11  129.250.4.41 (129.250.4.41)  120.197 ms  117.723 ms  121.253 ms
12  p64-1-3-0.r21.snjsca04.us.bb.verio.net (129.250.5.3)
    119.417 ms  118.924 ms  119.250 ms
13  xe-0-2-0.r20.snjsca04.us.bb.verio.net (129.250.2.72)  118.523 ms
    118.969 ms  118.863 ms
14  interconnect-eng.SanJose1.Level3.net (209.245.146.241)  113.397 ms
    117.990 ms  113.287 ms
15  so-3-2-0.bbr2.SanJose1.Level3.net (4.68.121.197)  118.736 ms
    114.859 ms  120.566 ms
16  as-1-0.mp1.Miami1.Level3.net (64.159.0.1)  199.557 ms
    as-0-0.mp2.Miami1.Level3.net (64.159.3.249)  199.404 ms
    as-1-0.mp1.Miami1.Level3.net (64.159.0.1)  205.298 ms
17  ge-9-1.hsa2.Miami1.Level3.net (64.159.0.22)  195.430 ms
    unknown.Level3.net (64.159.1.178)  200.684 ms
    ge-9-0.hsa2.Miami1.Level3.net (64.159.0.14)  199.673 ms
18  unknown.Level3.net (63.209.150.98)  199.561 ms  199.290 ms  194.934 ms
19  diveo-tb-mia-stm1.ipb.diveo.net.br (200.215.189.98)  313.151 ms
    308.485 ms  313.937 ms
20  h200-215-179-9.ipb.diveo.net.br (200.215.179.9)  307.939 ms
    375.406 ms  307.562 ms
21  200.202.113.49 (200.202.113.49)  313.544 ms  308.473 ms  312.274 ms
22  cl-S0-5521040038-cpe.rt.rjo.ipaccess.diveo.net.br (200.198.80.254)
    329.319 ms  330.050 ms  323.844 ms
23  anita.visualnet.com.br (200.198.87.5)  328.080 ms  322.645 ms  323.022 ms
```

4.3 ドメインアドレスと DNS

ここまでに述べて来たように、IP での通信はあくまでも IP アドレスを用いて行われますが、人間が見て理解するにはもっと「普通の」(意味のある) 名前を使うことが望まれます。たとえば Unix システムには`/etc/hosts`というファイルが存在し、そこに次のような形でホスト名と IP アドレスの対応が書かれています (他の OS の多くもこの種の対応表ファイルを持っています)。

```
127.0.0.0     localhost
192.168.0.1   piyo01
...
```

ここに書かれている名前については、ネットワーク接続を行うコマンドでその名前を宛先として指定すると、ファイルを調べて対応する IP アドレスを見つけ、その IP アドレスを使用してくれるわけです。LAN などで多数のホストがあるサイトでは、このような表を各ホストに配る代わりに後述するディレクトリサービスで名前から IP アドレスを検索できるように設定することもあります。

ではインターネット全体ではどうでしょう? インターネットには何百万ものホストが接続されていますし、絶えず新しいホストが追加されたり古いホストが削除されたりしますから、それをすべて網羅したファイルをそのつど配ったり、またはどこかにデータベースを用意して集中管理するのはどう考えても現実的ではありません。

このため、インターネット上では**ドメインアドレス**と呼ばれる階層構造を持った名前を使用し、ドメインアドレスから IP アドレスを検索するためのシステムとして **DNS**(Domain Name System) と呼ばれるサービスが運用されています (DNS そのものは後述の UDP を使って実装されていて、プロトコル的にはアプリケーション層のサービスですが、IP アドレスの話題なのでこの節で説明します)。

ドメインアドレスは簡単にいえば、複数の名前を「.」でつなげたもの、です。そしてその複数の名前は、右側ほど広い範囲に対応するような階層構造になっています (日本の住所の表記方法とは反対ですが、英語では住所、街、州、国の順で書くのでそれに合わせたわけです)。たとえば筆者の所属する組織のサーバのドメインアドレス「`www.gssm.otsuka.tsukuba.ac.jp`」は次のような階層に対応しています。

```
www.gssm.otsuka.tsukuba.ac.jp
                                  ↑日本
                               ↑教育組織
                          ↑筑波大学
                     ↑大塚地区
                ↑専攻名
           ↑ホスト名
```

そして、DNS はこの階層構造を利用して、たとえば次のような形で検索を実現しています。

- インターネット全体について、「ルート」と呼ばれる数台の DNS の「元締め」マシンがある。
- そこに最右側の名前 (**TLD**、Top-Level Domain) を渡して問い合わせると、その TLD ならどこに聞いたらいいかを教えてくれる。
- `.jp` を管轄するサーバは「`.ac.jp`」「`.co.jp`」などそれぞれについて、どこに聞いたらいいかを教えてくれる。
- `.ac.jp` を管轄するサーバに`.tsukuba.ac.jp` を聞くと、筑波大の DNS サーバを教えてくれる。
- 筑波大のサーバに`.otsuka.tsukuba.ac.jp` を聞くと、大塚の DNS サーバを教えてくれる。

このように階層を降りていくと、いつかはそのドメインアドレスに対応するホストの IP アドレスを直接知っている (またはそのようなドメインアドレスのホストはないことを知っている) サーバに到達しますから、そこから結果を返してもらえます。

TLD には.jp(日本)、.uk(英国) など各国に対応した国別 TLD と、.com(企業等)、.edu(教育機関) など国を特定せず組織種別等で分類した **gTLD**(Generic TLD、汎用 TLD) があります。IP アドレスと同様、ドメインの割り当ては IANA が元締めですが、gTLD については個別に IANA の委託を受けた組織 (たとえば.com であれば米国 VeriSign 社等)、国別 TLD については各国の組織 (日本の場合は JPRS) が割り当てを行っています。インターネットの発達の経緯から、gTLD の割り当てを受けている組織は米国のものが多くなっています。

DNS サーバに対する検索は多くの場合、ドメインアドレスを受け取る各プログラムが (対応する IP アドレスを調べるために) 発行しますが、コマンド **nslookup** を使ってユーザが直接 DNS サーバに検索を依頼することもできます。

- `nslookup` ドメインアドレス — ドメインアドレスに対応する IP アドレスを検索表示させる

これを使ったようすを次に示しておきます。

```
% /usr/sbin/nslookup www.yahoo.co.jp.
Server:  utogw.gssm.otsuka.tsukuba.ac.jp  ←検索の入口となるサーバ
Address:  192.50.17.2

Non-authoritative answer:          ←結果は「ヒント」であることを示す
Name:    www.yahoo.co.jp
Addresses:  202.229.198.216, 203.141.35.113, 210.81.150.5
%
```

上の例では IP アドレスが 3 つ返されていますが、これは大量のアクセスをさばく必要がある WWW サーバなどでは複数のマシンにアクセスを分散させるようにしているため、IP アドレスも複数持たせているためです。

では、特定のドメインアドレスを DNS に登録するのにはどうすればよいのでしょうか? それは、そのアドレスを収録するべき DNS サーバの管理者に頼んで登録情報を追加してもらいます。または、自分がまとまったアドレス群の管理者になるために「ドメインを」追加したのであれば、そのドメインを管轄する DNS サーバを用意し、それを親に相当するドメインに登録してもらいます。こちらの場合は、以後の個々のアドレスについては手元の DNS サーバに追加すればよいわけです。

では個人の場合は? それは、ドメインアドレスを取得したらプロバイダに依頼してそこの DNS サーバにデータを登録してもらうわけでしょうね。プロバイダによっては、ドメインの取得と DNS の運用を一括で引き受けてくれるところもあります。

5 伝達層

5.1 UDP と TCP

伝達層のプロトコルには TCP と UDP の 2 つがあることは既に述べました。**UDP** は IP の機能をほぼそのまま利用者に提供するものであり、パケット単位で送り先を指定してデータを送受信する機能を持ちます。このようなサービスの種別を**データグラム**と呼びます。本章冒頭の例題も UDP を使っていましたが、そこでも分かったように UDP では (ということは IP では) パケットを送っても状況 (回線の混雑、受信バッファの不足など) によっては途中で捨てられてしまい、到着しないことがあります。途中の各ノードでは、できる範囲でパケットを送り届けるように努力する (best effort) ことだけが求められています。その代わりに UDP ではオーバヘッドの小さな通信が行えます。

パケットが到着しないかも知れないのでは役に立たないと思われるかも知れませんが、UDP を使うアプリケーション側の用途によってはそれでも問題ないこともありますし (たとえば音声通話などの場合は時々パケットロスがあっても雑音が入る程度で実用上問題ないかも知れません)、アプリケーション側をパケットロスに対処するように作ることも考えられます。

一方、**TCP** は接続元と相手先の間に仮想回線を用意し、その上で信頼のある (reliable) 通信をサポートします。その代わり、TCP には接続の開始/切断処理が必要だったり、データの送受信に係わるオーバヘッド (余分な処理の負荷) が大きいという性質があります。TCP の仮想回線が提供している機能を次に挙げておきましょう。

- 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。
- 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ます。

- 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す。

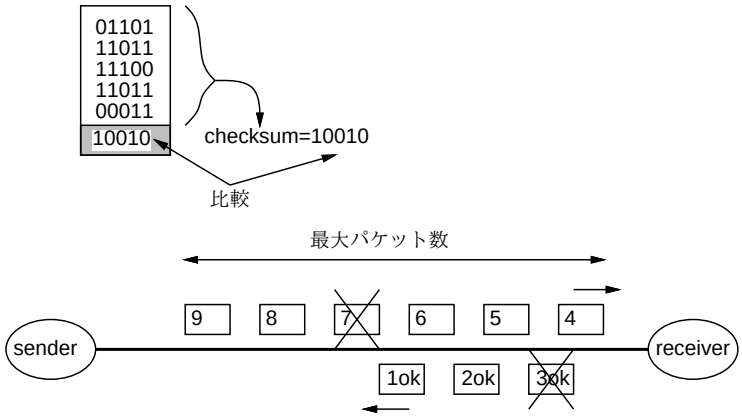

図 12: TCP の各種制御

5.2 ポート番号とサービスの同定

IP アドレスはあくまでもホスト (正確にはホストについている各ネットワークインタフェース) を識別するものです。実際には各ホストには多数のプロセスが動いていますから、ネットワーク接続に際しては「どこと接続するか」を指定する必要があります。たとえば同じホストに対してでも、「メールを送信したい」というのと「ファイルを転送したい」というのでは使用する (つまり接続相手となる) プログラムはまったく違うわけですから。

TCP と UDP では、この「どの相手」を指定するのに**ポート番号**と呼ばれる 16 ビットの数値を使用します。本章の冒頭で出てきた例題プログラムでは、実験用に適当なポート番号を選んでそのポート番号で接続を行っていました。

普段実用に使っているネットワークソフトウェアでも、何らかの方法でポート番号を決める必要があります。このためにポート番号 0〜1023 の範囲は**公知ポート番号** (well-konwn ports) として予約されており、TCP と UDP それぞれ個別に、どのサービスはポート何番を使うかが決っています (この割り当てもやはり IANA が管理しています)。Unix システムではポート番号とサービス名称の対応表が
`/etc/services` というファイルに記述されていて、これに基づいてポート番号と名前の変換を行っています。

プロセスが使用している TCP および UDP ポートの一覧は `netstat` コマンドに「`-f inet`」というオプションを指定することで表示させられます。

- `netstat -f inet` — 現在活きている TCP ポートおよび UDP ポートの一覧を表示

とあるマシンでこれを実行してみた様子を示します。

```
% netstat -f inet | less
Active Internet connections
Proto Recv-Q Send-Q  Local Address    Foreign Address  (state)
tcp4       0      0  sma.nfsd         smp.798          ESTABLISHED
tcp4       0      0  sma.nfsd         smri06.1019      ESTABLISHED
tcp4       0      0  sma.49244        smr04.x11        ESTABLISHED
tcp4       0      0  sma.canna        smm.36645        ESTABLISHED
tcp4       0      0  sma.nfsd         smr04.1017       ESTABLISHED
tcp4       0      0  sma.canna        smri21.50085     ESTABLISHED
tcp4       0      0  sma.nfsd         smri21.999       ESTABLISHED
tcp4       0      0  sma.nfsd         smri17.1011      ESTABLISHED
tcp4       0      0  sma.nfsd         smr05.987        ESTABLISHED
tcp4       0      0  sma.nfsd         utogw.788        ESTABLISHED
tcp4       0      0  localhost.smtp   *.*              LISTEN
udp4       0      0  localhost.ntp    *.*
udp4       0      0  sma.ntp          *.*
udp4       0      0  localhost.1019   localhost.1022
%
```

`tcp4`、`udp4` はそれぞれ IPv4 の TCP と UDP を意味します。アドレスの「`.`」より前はホスト名、後はポート番号ですが、`/etc/services` にサービス名記述されているポートについてはサービス名で表示されています。TCP については接続状態 (接続中、接続待ち) が表示されています。

6　セッション層以降の上位層

OSI の 7 層モデルでは、伝達層より上にセッション層、プレゼンテーション層、アプリケーション層の 3 つを置いていますが、TCP/IP ではこれらの層を明確に区別せず、ネットワークを利用する (ひいてはネットワークサービスの機能を提供する) 各種アプリケーションがこれらの機能を必要に応じて組み合わせて提供しています。これは次のような理由によります。

- TCP/IP の設計時点ではネットワークの各種機能についてあまりよく知られていなかったので、これらの機能を分けなかった。
- セッション層やプレゼンテーション層の機能が必要かどうかは、アプリケーションの種類によって違って来るので、各アプリケーションに任せてしまうのが簡単だった。

このため、以下では各アプリケーションに対応するプロトコルを (セッション層、プレゼンテーション層、アプリケーション層を併せて) 単に「アプリケーションプロトコル」と呼ぶことにします。整

理すると、TCP/IP の伝達層より上では、各種のネットワークサービスごとにそれを取り扱うためのアプリケーションプロトコルが用意されている、と考えればよいでしょう。

ネットワークを通じてデータを送受するという機能は伝達層以下がきちんと提供してくれていますから、アプリケーションプロトコルのレベルでは、「ネットワークサービスのために必要な情報をこの順序でやり取りする」という形での約束が中心になります。具体的なネットワークサービスとそのためのプロトコルについては、次章で取り上げていくことにします。

7 まとめと演習問題

本章では、コンピュータネットワークの基本的な概念や原理について学びました。普段何気なく利用しているネットワーク機能は膨大な約束ごとやソフトウェア群によって支えられていることがお分かり頂けたかと思います。

4-1. 世界の主要な地域のサイトを WWW で調べ、そこまでのパケット応答時間を「`/sbin/ping` アドレス」で調べてみなさい。地域によってどのくらいの違いがあるか整理して報告すること。(`ping` に応答しないサイトもあるのでそのつもりで。) Windows でやる場合もコマンドは「`ping`」です。

4-2. 上と同様だが、今度は「`/usr/sbin/traceroute` アドレス」で経路を調べ、世界各地までの「経路地図」を作ってみなさい。Windows でやる場合はコマンドは「`tracert`」です。

4-3. `nslookup` コマンドでさまざまなサイトの IP アドレスを調べてみなさい。たとえば大学、企業などで複数の WWW サーバをもっている場合、それぞれのアドレスを調べてどのくらい「近い」かもチェックしてみなさい。

4-4. 例題プログラム `send.c/recv.c` を打ち込んで動かしなさい。まず `ifconfig -a` で自分の使っているマシンの IP アドレスを調べ、自分のマシン上で窓を 2 つ開いてそれぞれで動かしてみなさい。これを実行している状態で上と同様に `netstat -f inet` や `netstat 1` を調べてみなさい。

4-5. 例題プログラム `send.c/recv.c` を使って隣の人とチャットをやってみなさい。さらにやる気なら、3 人でチャットするにはどうしたらいいか考え、プログラムを改良して動かしなさい。[7]

4-6. 手元のマシンで `netstat -f inet` で現在の TCP/IP ポート接続状況を調べてみなさい。次に、そのマシンでポートの状況が変わるような操作を行ってからから再度 `netstat` を使い、どのような変化が起きているかを検討してみなさい。Windows でやる場合はコマンド名は同じですが、パラメタが違って「`netstat -a`」になると思います。

4-7. `netstat 1` で 1 秒間隔でパケット数を表示させ、どのような場合にパケットが飛ぶか、パケットの大きさは何バイトくらいか、探求してみなさい。Windows でやる場合はコマンド名は同じですが、パラメタが違って「`netstat -e 1`」になると思います。

注意: 外部に接続されたマシンからしかできない演習 (4-1、4-2、4-3) は「rlogin utogw」でマシン utogw に接続してそこで行ってください。また、調査に使うコマンド類は Windows マシンにもあるので (「コマンドプロンプト」を出してコマンドを打てばよい)、自宅でやってみて比較していただけるとなおよいです。

[7] **ヒント:** `recv.c` は何人からでも受け取れるから変更の必要はありません。`send.c` は同時に 2 人に送る必要があるので、ホストとポートを 2 組指定するようにして、それぞれの宛先に同じものを送るようにすればよいでしょう。