ネットワークプログラミング

Jacques Garrigue, 2004 年 12 月 14 日

2　OSI 参照モデルとネットワーク通信

異なるシステムを接続するために開発されて階層化.

	層	機能	例
7	アプリケーション層	特定のアプリケーションに特化したプロトコル	電子メールの smtp やウェブの http
6	プレゼンテーション層	機器とネットワークのデータフォーマットの変換	文字コードなど
5	セッション層	通信の管理. トランスポート層以下の管理	コネクションの確立・切断
4	トランスポート層	両端ノード間のデータ転送の管理	データの抜けがないか？
3	ネットワーク層	アドレスの管理と経路の選択	次のステップはどこ？
2	データリンク層	直接接続された機器間のデータフレームの識別と転送	フレームとビット列の変換など
1	物理層	0 と 1 を電圧の高低や電波に変換する	どうつなぐか？

この表が逆向きに書かれる理由は，通信を行うときに上の層からデータを送り，層から層へと下まで行き，他の機器に伝えるからである．受け取ったデータは逆に段々上の層に伝えられて行く.

直接につながっていない場合，他の機器を伝って目的まで行かなければならない．途中の機器はどの層まで理解しているかによって分類される.

リピータ 物理層レベルで信号を拡大し，伝える．単純なハブ.

ブリッジ データリンク層で，必要に応じてデータを伝える．スイッチング・ハブ．

ルータ ネットワーク層で，IP アドレスなどを使い，正しい経路にデータを送る．

ゲートウェイ トランスポート層以上で通信を解釈し，依頼を転送したり，中身を変換したりするもの．ウェブ・プロキシなど．

2.1 物理層

様々な規格が使われる．

- ケーブルでつなぐ (10・100BaseT の Ethernet など)
- 光ファイバー (信号は光の変換)
- 無線 LAN (802.11a/b/g など)

2.2 データリンク層

通信機器間でデータをやりとりするプロトコル．多くの場合ではイーサネット (Ethernet) が使われる．

イーサネットのフレームはデータを細切れにし，イーサネットレベルのアドレスを付けて送るパケットである．内部構造は以下のとおり．

宛先 MAC	送信元 MAC	タイプ	データ	チェックサム
6 バイト	6 バイト	2 バイト	46〜1500 バイト	4 バイト

MAC アドレスは各機器の物理的インターフェース (カード) に生産時に割り振られる 48 ビットのシリアル番号．世界中では全て異っているはずなので，曖昧性がない．

2.3 ネットワーク層・インターネットプロトコル (IP)

データリンク層では直接接続される必要があるので，間接的につながっている機器を通信させるには，もう一つ上位の抽象化が必要になる．ネットワーク層では，データが複数の経由機器を伝って，宛先までとどけられる．宛先を指定するために，論理的な IP アドレスが割当てられる．

経路の計算を簡単にするために，32 ビットの IP アドレスは一定のビット数のサブネット部と機器部に分けらる．例えば，サブネット部の長さが 24 ビットの場合

サブネット部	機器部
24 ビット	8 ビット

サブネット部の長さと機器の IP アドレスが分かると，いくつかの値が決まる．

- サブネットマスクは，サブネット部は全部 1 で，機器部が全部 0 にした 32 ビットの値
- ネットワークアドレスは，サブネット部を IP アドレスから取り，機器部を 0 にしたもの
- ブロードキャストアドレスは，サブネット部を IP アドレスから取り，機器部を全部 1 にしたもの

それらを調べるには，`ifconfig` というコマンドを使う．

```
$ ifconfig -a
ath0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      inet 192.168.0.9 netmask 0xffffff00 broadcast 192.168.0.255
      ether 00:05:4e:46:a4:e9
```

以上の出力で分かるのは，MAC アドレスは (16 進で)00:05:4e:46:a4:e9，IP アドレスは 192.168.0.9，サブネット部の長さは 24 である．

一行目の mtu は IP パケットの最大の大きさを与えているが，それがちょうどイーサネットの最長のデータに当る．IP では必要に応じてパケットを分割できるが，効率のためにそれを避けた方がいいので，最もよく使われているイーサネットに合わせるのが普通．

パケットをどこに送るべきかを決めるとき，まず宛先が同じネットワーク内からどうかを調べる．サブネット部が一致していれば，直接に宛先まで送れる．そのために宛先の MAC アドレスを調べて，データリンク層に任せればいい．サブネット部が自分のアドレスと異っている場合，どこかを経由しないといけない．経由するべきところはルート表に記されている．各ネットワークに対して，自分のネットワーク内の誰に渡せばいいかが分かる．その機器に送って，先を任せる．

ルート表を調べるには，`netstat -nr` を使う．

```
$  netstat -nr
Destination        Gateway            Flags    Refs      Use  Netif
default            192.168.0.5        UGS         0   659141   ath0
127.0.0.1          127.0.0.1          UH          1     2840    lo0
192.168.0          link#2             UC          0        0   ath0
192.168.0.5        link#2             UHLW        1      124   ath0
192.168.0.9        127.0.0.1          UGHS        0        0    lo0
```

上記のように，通常は表のなくに default という行があり，他の行が当て嵌らない宛先はそちらに送られる．

IP パケットは上層から受けたデータに加えて，それを宛先までとどけるためのヘッダを含んでいる．オプションなしのヘッダの長さは 5×32 ビット．

バージョン・TOS	パケット長
識別子	フラグ・オフセット
TTL ・ プロトコル	ヘッダ・サム
送信元 IP アドレス	
宛先 IP アドレス	
オプション	

この中の TTL は生存時間と言い，このパケットは転送されてもいい回数を表している．転送される度に一つ減るので，ルートの間違いでパケットがぐるぐる回るのを防いでいる．

2.4 トランスポート層・TCP と UDP

ネットワーク層ではパケットを宛先まで送る機能が提供されているが，それをアプリケーションに使うために，まだいくつかの問題が残っている．

- データが正しく転送されているかはデータリンク層ではチェックしているものの，それだけでは安心できない場合がある
- 何かの問題で TTL を過ぎて，パケットが廃棄されて可能性がある
- 二つのコンピュータの通信だけが提供され，どのプログラムに届けるかは扱われていない
- 経路の変化によって，パケットの順序が入れ替わる可能性がある

前回見た TCP は，継続してコネクションを二つのプログラムの間に作ることで，上記の全ての問題を解決している．データが間違っている場合や届かなかった場合は自動的に再配送を行うし，順序も保たれる．そのためにもう一つのヘッダが必要になる．

送信元ポート番号	宛先ポート番号
シーケンス番号	
確認応答番号	
フラグなど	ウィンドウサイズ
チェックサム	緊急ポインター
オプション	

このなかで，シーケンス番号が順番の入れ替わりを防ぎ，確認応答番号が向こうでのパケットの到着状況を知らせてくれる．

TCP のセッション機能がかなり強いので，実は第 4 層の一部も含んでいるという説がある．

それに比べて，UDP がとても単純．素の IP パケットに加えて，チェックサムとポート番号だけを提供している．パケットが届かない可能性はあるが，届いたときは内容の正しさが保証される．

送信元ポート番号	宛先ポート番号
パケット長	チェックサム

2.5 セッション層以上

OSI 参照モデルでは細かい区別があるが，TCP/IP では扱われないので，アプリケーションに任されている．

実習課題

先週のプログラムを元に，以下の機能を実装する

- ネットじゃんけんを作る．サーバが二つのクライアントの接続を待つ．それぞれから rock, scissor, paper のいずれかをもらい，勝った方に You win over (other)，負けた方に You loose to (other)，引き分けのときは両方に You draw with (other) と送り返す．other は相手の IP アドレスである．
- TCP から UDP に変更する．

UDP にするために以下の変更を行わなければならい．

- `socket()` の呼び出しを `socket(PF_INET, SOCK_DGRAM, 0)` に変更．
- `listen()`，`accept()` や `connect()` は不要，その代わりソケットを必ず自分に `bind()` しておかないといけない．
- `write()` の代わりに `sendto()`，`read` の代わりに `recvfrom` を使う．

```
int sendto(int s, void *msg, int len, int flags,
           struct sockaddr *to, int tolen);
int recvfrom(int s, void *buf, int len, int flags,
             struct sockaddr *from, int *fromlen);
```

参考のために，元のクライアントの UDP 版の一部を与える．

```
#define PORT 6333

int main(int argc, char **argv)
{
```

```
    struct sockaddr_in sin, mysin;
    int sd;
    char line[128];

    /* set address */
    sin.sin_family = AF_INET;
    sin.sin_port = htons(PORT);
    sin.sin_addr.s_addr = inet_addr("127.0.0.1");
    mysin.sin_family = AF_INET;
    mysin.sin_port = 0;
    mysin.sin_addr.s_addr = INADDR_ANY;
    /* allocate socket */
    sd = socket(PF_INET, SOCK_DGRAM, 0);
    if (bind(sd, (struct sockaddr*)&mysin, sizeof(mysin)) < 0) {
        perror("bind error");
        exit(2);
    }
    /* connect to server */
    printf("Send message: ");
    fgets(line, 128, stdin);
    if (sendto(sd, line, strlen(line) - 1, 0,
               (struct sockaddr*)&sin, sizeof(sin)) < 0) {
        perror("send error");
        exit(2);
    }
    exit(0);
}
```