UNIXシステムプログラミング

第8回 ネットワークプログラミング(1)

2019年11月15日

情報工学科

寺岡文男

ネットワークプログラミングの前に

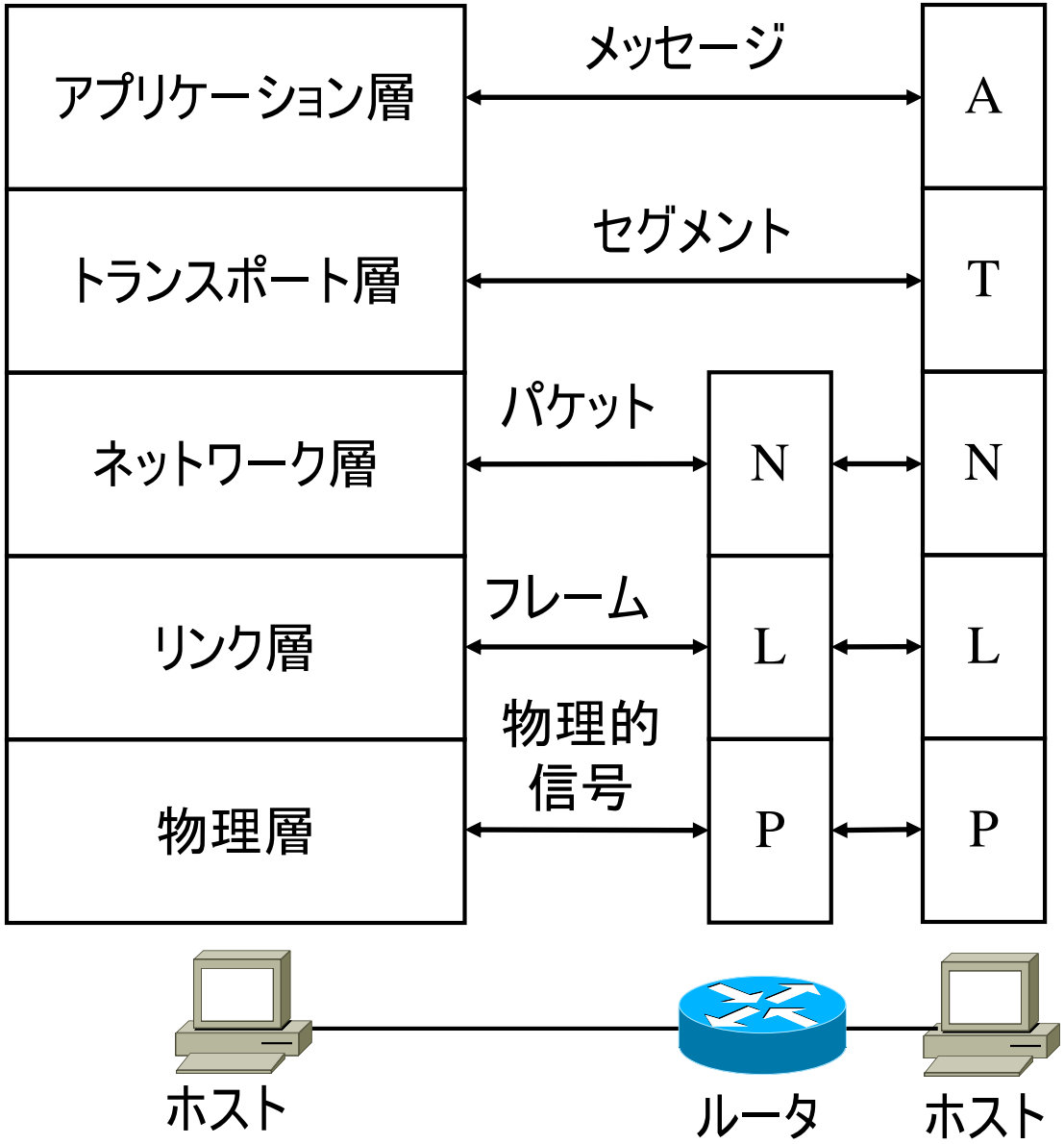
- システムプログラミングでは，関数やシステムコールの使い方を覚えるだけではよいプログラムは書けない.
  - 自分が書いたプログラムにより，コンピュータがどのように動作するかが想像できなければならない.
  - 例：HDDとの入出力におけるブロックサイズ.

- ネットワークプログラミングの前に，ネットワークの基礎を説明.
  - ネットワークプログラミングに関係するシステムコールと，プロトコルの関係を理解.

プロトコル階層

- プロトコル: 通信規約
  - メッセージフォーマットや，メッセージ受信時の動作を規定.
- プロトコル階層
  - 通信するにはさまざまな取り決めや機能が必要.
    - 電圧，周波数，符号，誤り検出・訂正，経路制御，順序制御，流量制御，輻輳制御，セッション管理，データ表現， etc.
  - さまざまな取り決めや機能を整理して階層化
    - 各階層の独立性向上
- ISOのOSI (Open Systems Interconnection)参照モデル
  - 7階層
- インターネット: 5階層

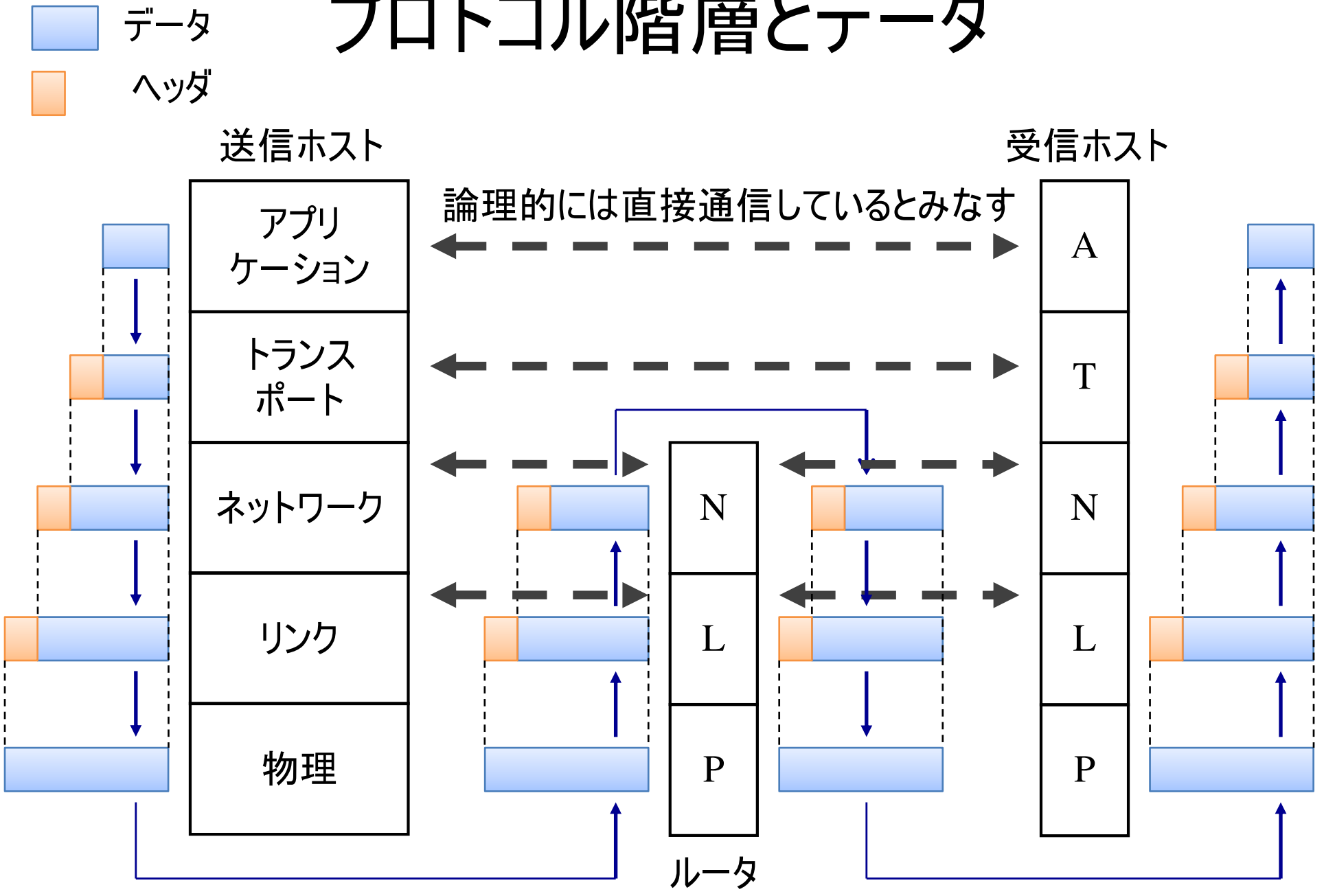
インターネットのプロトコル階層モデル



- アプリケーションプログラム間でのデータ交換
- アプリ間(end-to-end)の通信サービスを提供
- host-to-hostの通信サービスを提供．経路制御．
- 対向するノード間での通信サービスを提供
- 物理信号の伝達

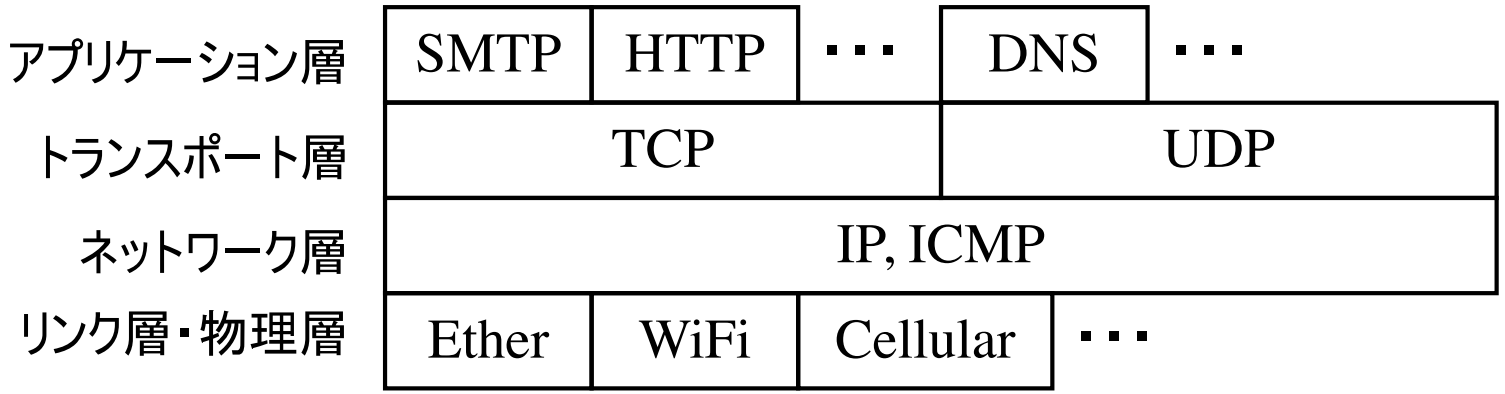
仮想的には対向する階層同士(peer)が直接通信していると考える

プロトコル階層とデータ



インターネットのプロトコル

- TCP (Transmission Control Protocol)
  - 信頼性のあるバイトストリームを提供
  - データの誤りなし，順序どおり
- UDP (User Datagram Protocol)
  - 信頼性を保証しないデータグラム通信を提供
  - データ誤り，セグメントロス，不整順序などの可能性あり
- IP (Internet Protocol)
  - 信頼性を保証しないデータグラム通信を提供
  - Best Effort



TCPとUDPの使い分け

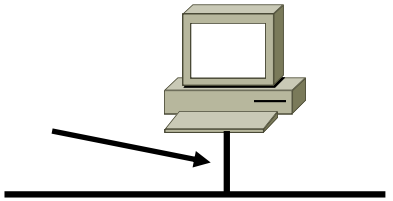
- それぞれの特徴を知り，目的に合う方を使う.

- TCP
  - 通信前にコネクション確立，終了後は解放 → オーバヘッド
  - 信頼性を保証 → パケットロスの場合は再送
  - 用途の例：ファイル転送，メール転送，etc.

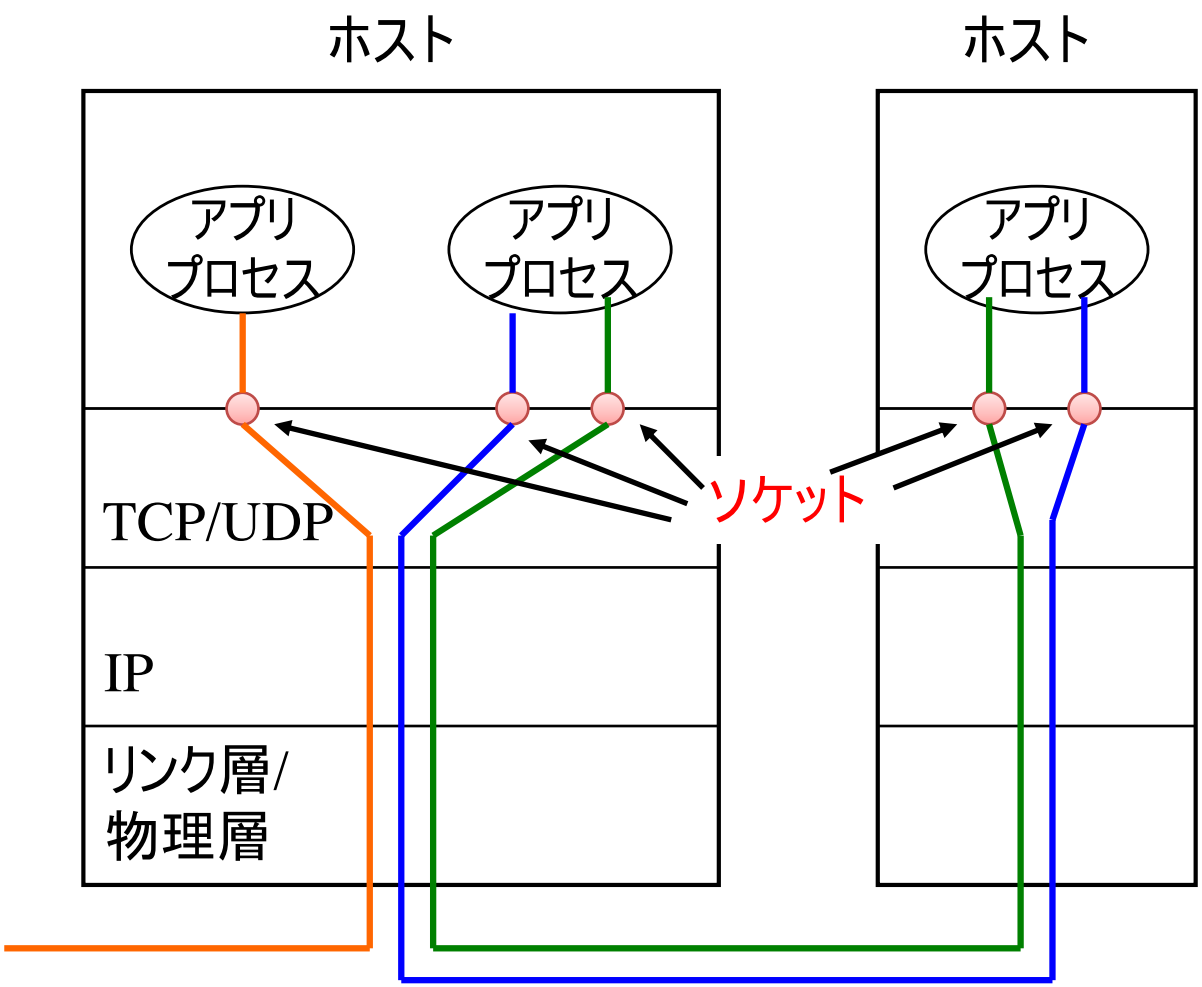
- UDP
  - コネクションの確立・解放は不要
  - 信頼性の保証なし
  - 用途の例：音声，動画，単純な問合せと応答，etc.

IPアドレス

- ネットワーク層(IP)におけるホストの位置情報
  - ネットワーク部とホスト部からなる32ビットの数字
  - ネットワーク部とホスト部の境界は任意に決められる
  - 4つの10進数をドット (“.”) で区切って表記
  - “/” のあとにネットワーク部のビット長を表記
- 例 : 131.113.71.3/24
  - ネットワーク部は24ビット
  - “131.113.71” がネットワーク部, “3” がホスト部
- IPアドレスはネットワークインタフェースに割当てられる
  - ルータは複数のIPアドレスを持つ

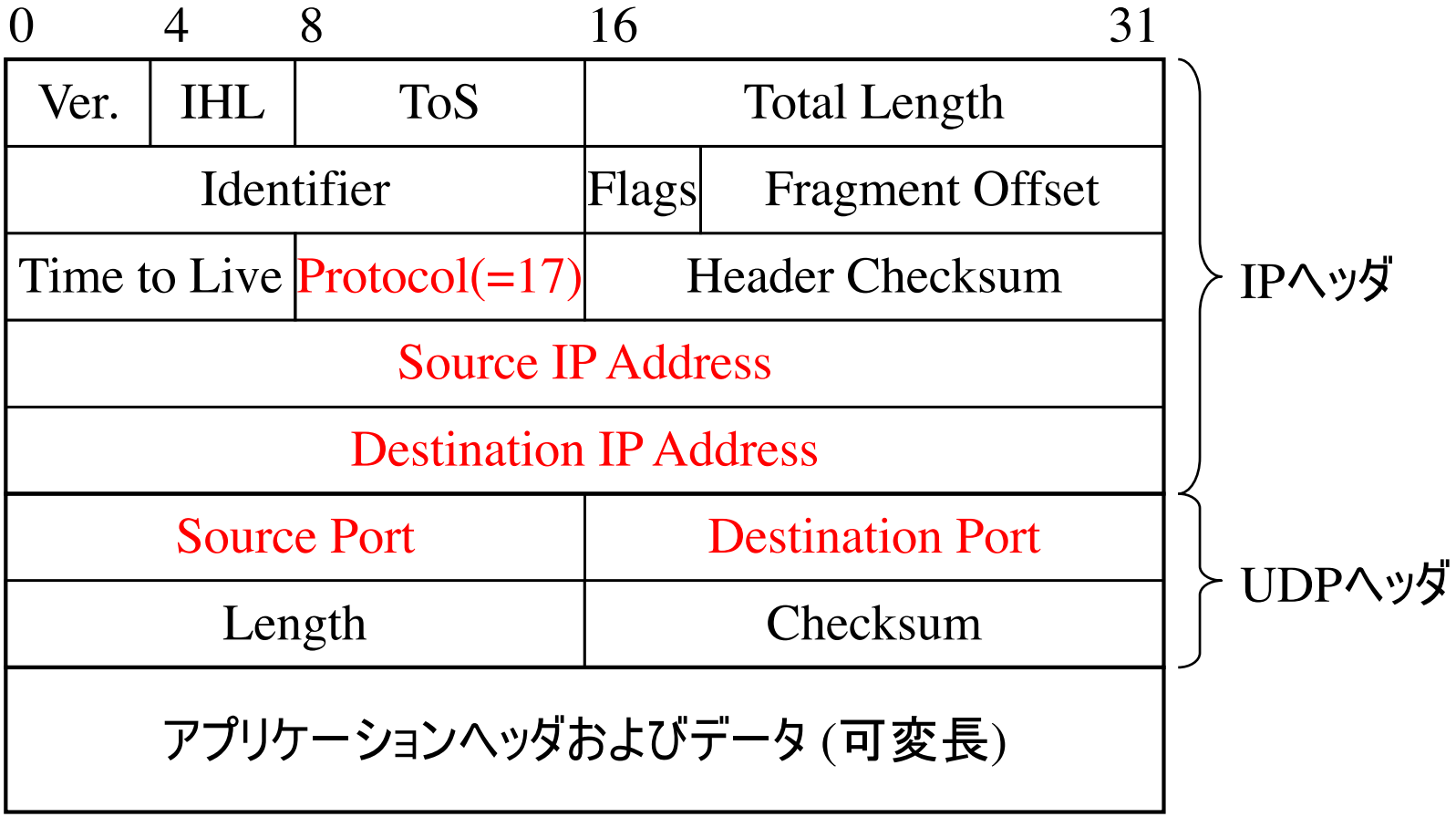


ソケットとポート

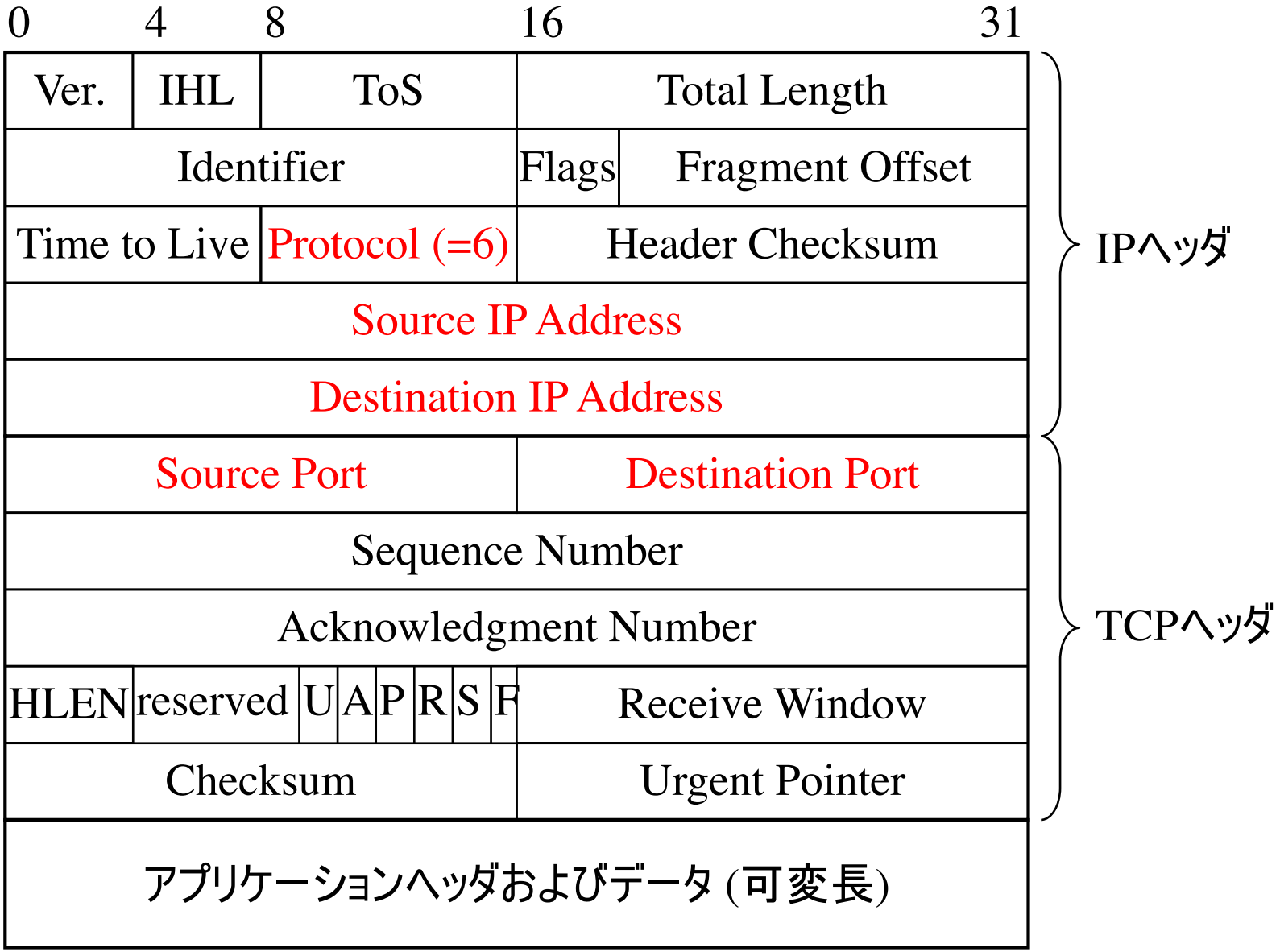


- アプリプロセスはソケットを介して通信
- 通信相手をソケットで指定
- ソケット= IPアドレス ＋ ポート番号
- ポート番号 : 16ビットの整数
- ポート番号 0～1023 : 特定のアプリ用に予約されている

UDP / IP パケット


0
4
8
16
31
Ver.
IHL
ToS
Total Length
Identifier
Flags
Fragment Offset
Time to Live
Protocol(=17)
Header Checksum
Source IP Address
Destination IP Address
Source Port
Destination Port
Length
Checksum
アプリケーションヘッダおよびデータ (可変長)
IPヘッダ
UDPヘッダ

TCP / IP パケット


0
4
8
16
31
Ver.
IHL
ToS
Total Length
Identifier
Flags
Fragment Offset
Time to Live
Protocol (=6)
Header Checksum
Source IP Address
Destination IP Address
Source Port
Destination Port
Sequence Number
Acknowledgment Number
HLEN
reserved
U
A
P
R
S
F
Receive Window
Checksum
Urgent Pointer
アプリケーションヘッダおよびデータ (可変長)
IPヘッダ
TCPヘッダ

ネットワークプログラミングの注意点(1)

- 通信相手との相互作用をきちんと考える
  - e.g., 双方が相手からのパケット待ちをするとデッドロックする
- ソケットインタフェースは汎用的に設計されている
  - ソケットインタフェース: ネットワークプログラミングのための システムコールのインタフェース
- インターネットはプロトコルファミリーの1つ (PF_INET)
  - 他にもPF_UNIX, PF_PUP, PF_OSI, PF_SNAなどがある
- ソケットの名前やアドレスを表す構造体も汎用的
  - struct sockaddr: 汎用的なソケットアドレス構造体
  - struct sockaddr_in: インターネット用のソケットアドレス構造体

ネットワークプログラミングの注意点(2)

- ビット長を明示した型を使う
- short, int, long などのビット長はコンピュータのアーキテクチャや処理系により異なる
  - 16ビットマシン: short = 16bits, int = 16bits, long = 32bits
  - 32ビットマシン: short = 16bits, int = 32bits, long = 32bits
- Q: IPアドレスのフィールドを int で宣言するとどのような不都合が起こるか？

型	説明
int8_t	8ビットの整数
uint8_t	8ビットの符号なし整数
int16_t	16ビットの整数
uint16_t	16ビットの符号なし整数
int32_t	32ビットの整数
uint32_t	32ビットの符号なし整数
in_addr_t	uint32_t
sa_family_t	uint8_t
in_port_t	uint16_t
socklen_t	uint32_t

ネットワークプログラミングの注意点(3)

- バイトオーダーに注意する
- バイトオーダーには2つのタイプがある
  - 0x12345678 という整数のメモリ上の配置は:

big-endian

12	34	56	78

← ネットワークではこちら

little-endian

78	56	34	12

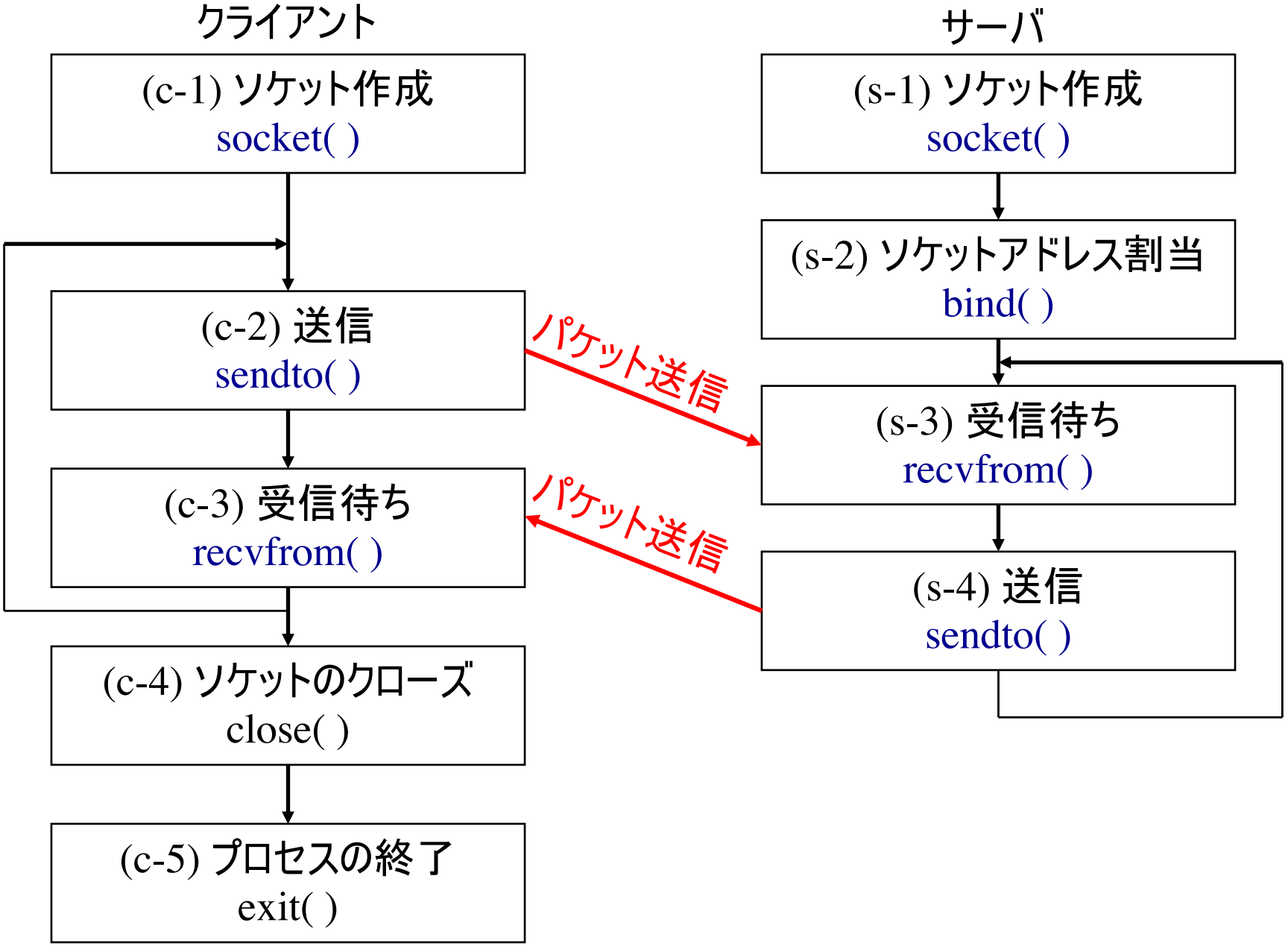
- バイトオーダー変換のための関数を使う

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

uint32_t htonl(uint32_t hostlong);  // host order to
uint16_t htons(uint16_t hostshort); //   network order

uint32_t ntohl(uint32_t netlong);   // network order to
uint16_t ntohs(uint16_t netshort);  //   host order
```

UDPでの通信 (1)


クライアント
(c-1) ソケット作成
socket( )
(c-2) 送信
sendto( )
(c-3) 受信待ち
recvfrom( )
(c-4) ソケットのクローズ
close( )
(c-5) プロセスの終了
exit( )
パケット送信
パケット送信
サーバ
(s-1) ソケット作成
socket( )
(s-2) ソケットアドレス割当
bind( )
(s-3) 受信待ち
recvfrom( )
(s-4) 送信
sendto( )

サーバの動作

1. socket( )で送受信のためのソケットを開く
2. bind( )で自分のソケットにポート番号を割り当てる
   - クライアントはこのポート番号を指定して送信する
   - 自分のソケットのIPアドレスはOSが自動的に設定する
3. recvfrom( )でクライアントからのパケット受信を待つ
   - パケットが受信するまで待つ
   - どのクライアントからのパケットでも受信する
   - クライアントのソケットアドレスは引数で返る
4. sendto( )でクライアントに応答を送信する
   - 送信先のソケットアドレスを指定してパケットを送信する
- 3に戻る

クライアントの動作

1. socket( )で自分のソケットを開く
   - 自分のソケットで要求待ちになることはないので，ポート番号を割り当てる必要はない
   - ポート番号とIPアドレスはOSが自動的に設定する
2. sendto( )でサーバのソケットアドレスを指定して送信する
3. recvfrom( )でサーバからの応答を受信する

- sendto( )とrecvfrom( )を必要なだけ繰り返す

4. close( )でソケットを閉じる
5. exit( )でプロセスを終了する

socket( ) : ソケットの生成

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

int socket(int domain, int type, int protocol);
```

- 引数
  - domain: PF_INET : インターネットの場合
  - type:
    - SOCK_STREAM : TCPの場合
    - SOCK_DGRAM : UDPの場合
  - protocol: 0 を指定する
- 返り値 :
  - ソケット記述子(ソケットを識別する整数値)が返される
  - エラーの場合は –1 を返し、errnoに理由が設定される

bind( ) : ソケットアドレスの割当て

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

int bind(int s, struct sockaddr *addr,
                                socklen_t addrlen);
```

- 引数 :
  - s: ソケット記述子 (socket( )の返り値)
  - addr: ソケットアドレス構造体へのポインタ
  - addrlen: ソケットアドレス構造体のサイズ
- 返り値
  - 正常の場合 0 が返る
  - エラーの場合 –1 が返り、errnoに理由が設定される

インターネット用ソケットアドレス構造体

```
/* Internet address */
struct in_addr {
    in_addr_t s_addr;
};

/* socket address, internet style */
struct sockaddr_in {
    uint8_t sin_len;
    sa_family_t   sin_family; // アドレスファミリー(AF_INET)
    in_port_t     sin_port;   // ポート番号
    struct in_addr sin_addr;  // IPアドレス
    char          sin_zero[8];
};
```

32ビットの
符号なし整数.
ネットワーク
バイトオーダ.

16ビットの
符号なし整数.
ネットワーク
バイトオーダ

IPアドレスの表現形式の変換

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

int inet_aton(char *cp, struct in_addr *pin);

char *inet_ntoa(struct in_addr in);
```

- inet_aton( )
  - 文字表記のIPアドレスをバイナリ(ネットワークオーダー)に変換
- inet_ntoa( )
  - バイナリ(ネットワークオーダー)のIPアドレスを文字表記に変換

socket( )とbind( )の使用例

```
int s;                                  // ソケット記述子
in_port_t myport;                       // 自ポート
struct sockaddr_in myskt;               // 自ソケットアドレス構造体

if ((s = socket(AF_INET, SOCK_DGRAM, 0) < 0) {
        perror(“socket”);
        exit(1);
}
// myportに値を代入しておく
memset(&myskt, 0, sizeof myskt);
myskt.sin_family = AF_INET;
myskt.sin_port = htons(myport);
myskt.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(s, (struct sockaddr *)&myskt, sizeof myskt) <0){
        perror(“bind”);
        exit(1);
}
```

OSに自動的に自分のIPアドレスを設定してもらう

recvfrom( ) : パケットの受信 (1)

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

ssize_t recvfrom(int s, void *buf, size_t len,
 int flags, struct sockaddr *from, socklen_t *fromlen);
```

- 引数
  - s: ソケット記述子
  - buf: データを格納する領域のポインタ
  - len: データ格納領域のサイズ
  - flags: 0 を指定しておく
  - (次ページへ)

recvfrom( ) : パケットの受信 (2)

- 引数 (続き)
  - from: 送信側ソケットの情報(i.e., IPアドレス、ポート番号)が格納される領域のポインタ.
  - fromlen: 上記の領域のサイズを格納したsocklen_t領域のポインタ. 受信後には実際のサイズがこの領域に設定される.

- 返り値
  - 正常な場合は受信したデータサイズが返る.
  - エラーの場合は –1 を返し, errnoに理由が設定される.

recvfrom( )の使用例

```
int s, count;
struct sockaddr_in myskt;       // 自ソケット
struct sockaddr_in skt;         // 送信側ソケット
char buf[512];                  // 受信用バッファ
socklen_t sktlen;

// s = socket( ); bind( ); を実行しておく
sktlen = sizeof skt;
if ((count = recvfrom(s, buf, sizeof buf, 0,
              (struct sockaddr *)&skt, &sktlen)) < 0) {
      perror("recvfrom");
      exit(1);
}
// skt.sin_port に送信側ポート番号が設定される
// skt.sin_addr.s_addr に送信側IPアドレスが設定される
//    バイトオーダーに気をつけること
// sktlen に送信側ソケットアドレスの実際のサイズが返される
```

sendto( ) : パケットの送信

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

ssize_t sendto(int s, void *msg, size_t len, int flags,
      struct sockaddr *to, socklen_t tolen);
```

- 引数
  - s: ソケット記述子
  - msg: 送信データが格納されている領域のポインタ
  - len: 送信データのサイズ
  - flags: 0 を設定しておく
  - to: 相手側のソケットアドレス構造体のポインタ
  - tolen: 相手側ソケットのサイズ
- 返り値
  - 正常の場合は送信したデータサイズが返る.
  - エラーの場合は – 1 が返り, errnoに理由が設定される.

sendto( )の使用例

```
int s, count, datalen;
struct sockaddr_in skt;          // 受信側ソケット
char sbuf[512];                  // 送信用バッファ
in_port_t port;                  // 受信側のポート番号
struct in_addr ipaddr;           // 受信側のIPアドレス

// s = socket( )を実行しておく
// sbuf[ ]に送信データを準備し，datalenにデータ長を設定しておく
// port, ipaddr を設定しておく
skt.sin_family = AF_INET;
skt.sin_port = htons(port);
skt.sin_addr.s_addr = htonl(ipaddr.s_addr);
if ((count = sendto(s, sbuf, datalen, 0,
              (struct sockaddr *)&skt, sizeof skt)) < 0) {
      perror("sendto");
      exit(1);
}
```

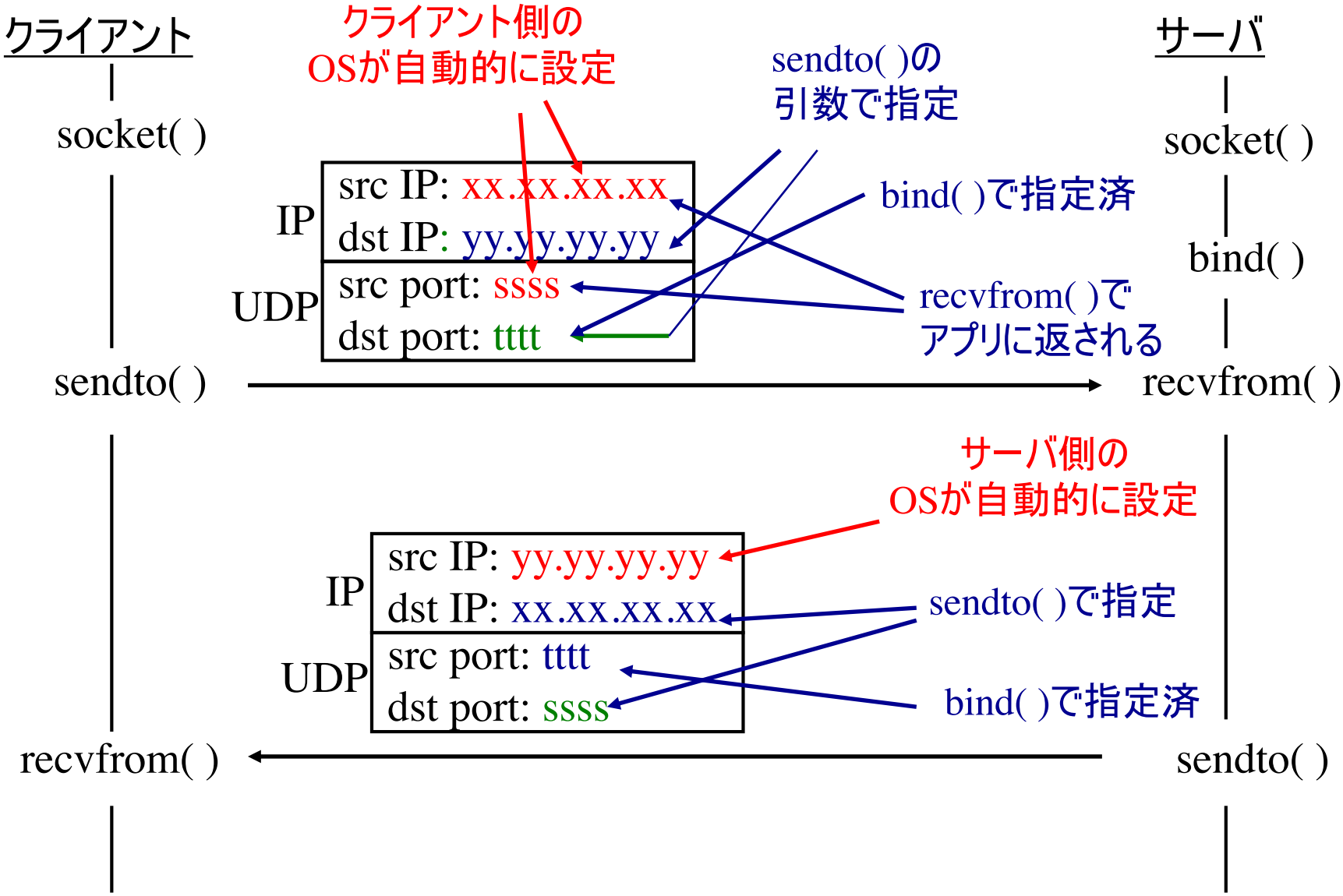
close( ): ソケットのクローズ

```
#include <unistd.h>

int close(int s);
```

- 通信が終了したらソケットをクローズする
- 引数
  - s: ソケット記述子
- 返り値
  - 正常の場合は 0 が返る
  - エラーの場合は – 1 が返り、errnoに理由が設定される

システムコールとパケット送受信の関係(UDP)



演習

- 教卓PCにUDPでメッセージを送信し，次にUDPでメッセージを受信するプログラム udpcli を作成しなさい.
  - 教卓PCのIPアドレスは当日知らせます.
  - 教卓PCのポート番号は 49152
  - 送信メッセージはキーボーから入力
  - 受信したメッセージを表示

- UDPメッセージを受信し，受信したメッセージをそのまま相手に送り返すプログラム udpsrv を作成しなさい.
  - ポート番号 49152 で受信を待つ.
  - メッセージを受信したら，相手のIPアドレスとポート番号，受信メッセージを表示する.
  - 以上を無限ループで繰り返す.