ネットワークコンピューティング
第9回

中山　雅哉 (m.nakayama@m.cnl.t.u-tokyo.ac.jp)
関谷　勇司 (sekiya@nc.u-tokyo.ac.jp)

授業に関する情報

- 授業スライド、連絡事項、課題等に関する連絡
  - Web
    - https://lecture.sekiya-lab.info/
  - Mail
    - lecture@sekiya-lab.info
- 実験用ホスト
  - Resources
    - login1.sekiya-lab.info
    - login2.sekiya-lab.info

名前解決

DNSを用いたプログラミング

- ドメイン名⇔IPアドレス変換のライブラリを利用
  - gethostbyname(), gethostbyname2()
    - （基本的な）ドメイン名→IPアドレス変換関数
  - getaddrinfo()
    - getaddrbyname を汎用的に拡張した関数
    - マルチプロトコル対応
  - getnameinfo()
    - sockaddr 構造体を引数とするため、よりマルチプロトコル対応可

これらの利用方法については、第6回 (2018/05/24) の資料を参照のこと

getnameinfo() を用いたサンプルコード (再掲)

```
struct addrinfo hints, *res0, *res;

memset (&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
getaddrinfo(hostname, NULL, &hints, &res0);

res = res0;
while (res != NULL) {
  getnameinfo(res->ai_addr, res->ai_addrlen, buff, sizeof(buff),
                          NULL, 0, NI_NUMERICHOST);
  printf("addr: %s¥n", buff);
  res = res->ai_next;
}
freeaddrinfo(res0);
```

DNS を用いたサーバ指定の利点

- 大量のクライアントから同時期にアクセスが集中するケース（特に並行サーバでは）
  - 1サーバで同時に処理できるクライアント数には
    - （OS で扱うことができる）プロセス数
    - （OS で扱うことができる）ソケット数
    - CPU やメモリの使用量
    - ネットワーク I/F の処理能力
  - などの制約で上限がある

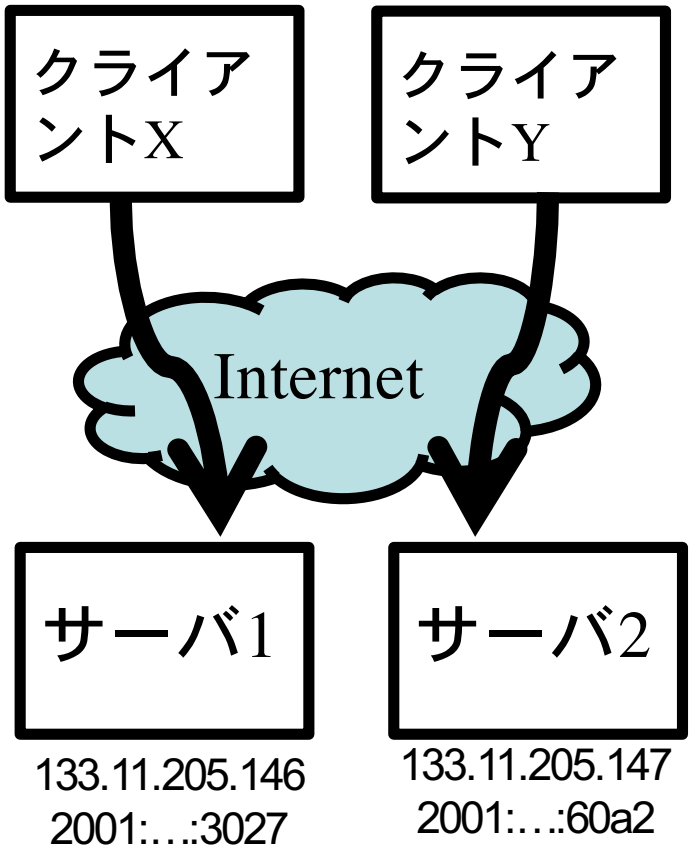
⇒ DNS で複数の同じ機能を提供するサーバを指定することで負荷分散できる（例：www.yahoo.com）

負荷分散

DNS を用いた負荷分散

- ラウンドロビン方式
  - 同一ドメイン名に複数の IP アドレスの RR が指定されていると、DNSサーバへの問合せに応じて応答順序が変更される。

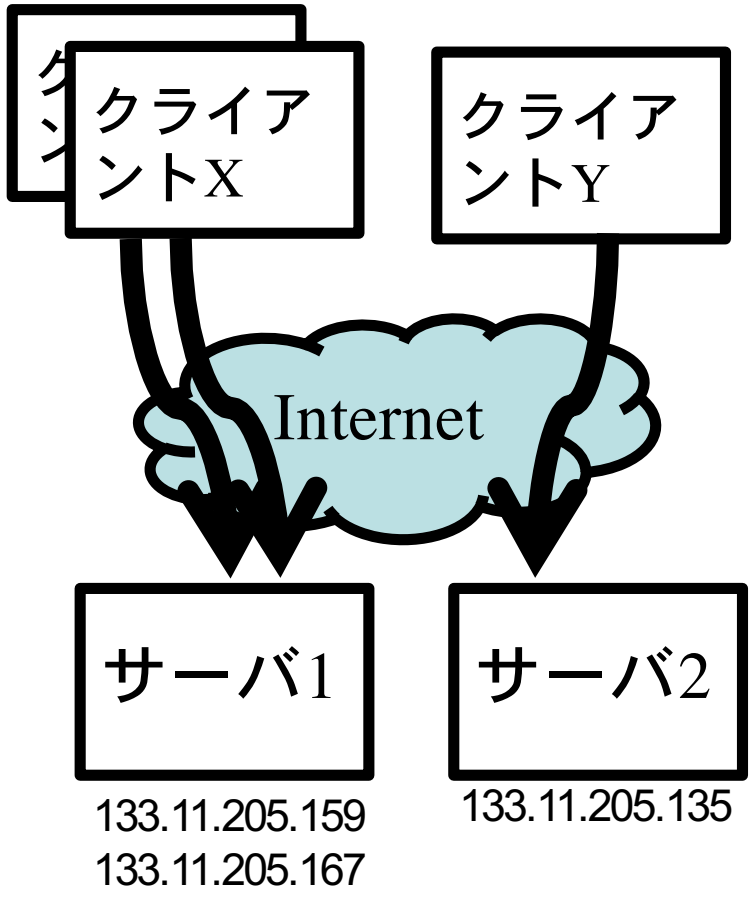
```
$ ./pr2 kiku.itc.u-tokyo.ac.jp
addr: 2001:200:180:452:213:72ff:fefc:3027
addr: 2001:200:180:452:215:c5ff:fee1:60a2
addr: 133.11.205.147
addr: 133.11.205.146

$ ./pr2 kiku.itc.u-tokyo.ac.jp
addr: 2001:200:180:452:215:c5ff:fee1:60a2
addr: 2001:200:180:452:213:72ff:fefc:3027
addr: 133.11.205.146
addr: 133.11.205.147
```



DNS を用いた負荷分散

- 重み付きラウンドロビン
  - DNS サーバの応答順序変更の仕組みを利用し、特定のサーバに複数の IP アドレスを割当ててサーバへのアクセス数を制御する方式

```
$ ./pr2 load3.nc.u-tokyo.ac.jp
addr: 133.11.205.167
addr: 133.11.205.135
addr: 133.11.205.159
$ ./pr2 load3.nc.u-tokyo.ac.jp
addr: 133.11.205.159
addr: 133.11.205.167
addr: 133.11.205.135
$ ./pr2 load3.nc.u-tokyo.ac.jp
addr: 133.11.205.135
addr: 133.11.205.159
addr: 133.11.205.167
```



DNS (getaddrinfo) 利用時の通信状況

```
$ ./pr2 load3.nc.u-tokyo.ac.jp
addr: 133.11.205.167
addr: 133.11.205.135
addr: 133.11.205.159
```

```
$ ./pr2 kiku.itc.u-tokyo.ac.jp
addr: 2001:200:180:452:215:c5ff:fee1:60a2
addr: 2001:200:180:452:213:72ff:fefc:3027
addr: 133.11.205.147
addr: 133.11.205.146
```

上記コマンド実行時のネットワーク上の通信状況

```
# tcpdump -n -i eth0 port53
11:13:04.635009 IP 133.11.206.165.35810 > 133.11.124.164.53: 10905+ A? load3.nc.u-tokyo.ac.jp. (40)
11:13:04.635427 IP 133.11.206.165.35810 > 133.11.124.164.53: 22099+ AAAA? load3.nc.u-tokyo.ac.jp. (40)
11:13:04.636004 IP 133.11.124.164.53 > 133.11.206.165.35810: 10905 3/3/4 A 133.11.205.159, A 133.11.205.167, A 133.11.205.135 (247)
11:13:04.636310 IP 133.11.124.164.53 > 133.11.206.165.35810: 22099 0/1/0 (90)

11:15:34.824602 IP 133.11.206.165.51779 > 133.11.124.164.53: 42872+ A? kiku.itc.u-tokyo.ac.jp. (40)
11:15:34.825098 IP 133.11.206.165.51779 > 133.11.124.164.53: 44050+ AAAA? kiku.itc.u-tokyo.ac.jp. (40)
11:15:34.825668 IP 133.11.124.164.53 > 133.11.206.165.51779: 42872 2/5/8 A 133.11.205.147, A 133.11.205.146 (339)
11:15:34.825982 IP 133.11.124.164.53 > 133.11.206.165.51779: 44050 2/5/8 AAAA 2001:200:180:452:215:c5ff:fee1:60a2, AAAA 2001:200:180:452:213:72ff:fefc:3027 (363)
```

DNS を用いた負荷分散

- 各種パラメータを用いた負荷分散方式
  - サーバの CPU 利用率や応答時間
  - （クライアントの）問合せIPアドレス

  に応じて、応答する IP アドレスを制御する方式

```
$ ./pr2 www.ring.gr.jp
addr: 2001:240:3:2::1
addr: 2001:2f8:2c:44::4
addr: 210.146.64.7
addr: 133.37.44.6
addr: 160.26.2.184
addr: 202.18.64.24
```

```
$ ./pr2 www.dnsbalance.ring.gr.jp
addr: 202.18.64.24
addr: 210.146.64.7
addr: 133.37.44.6
```

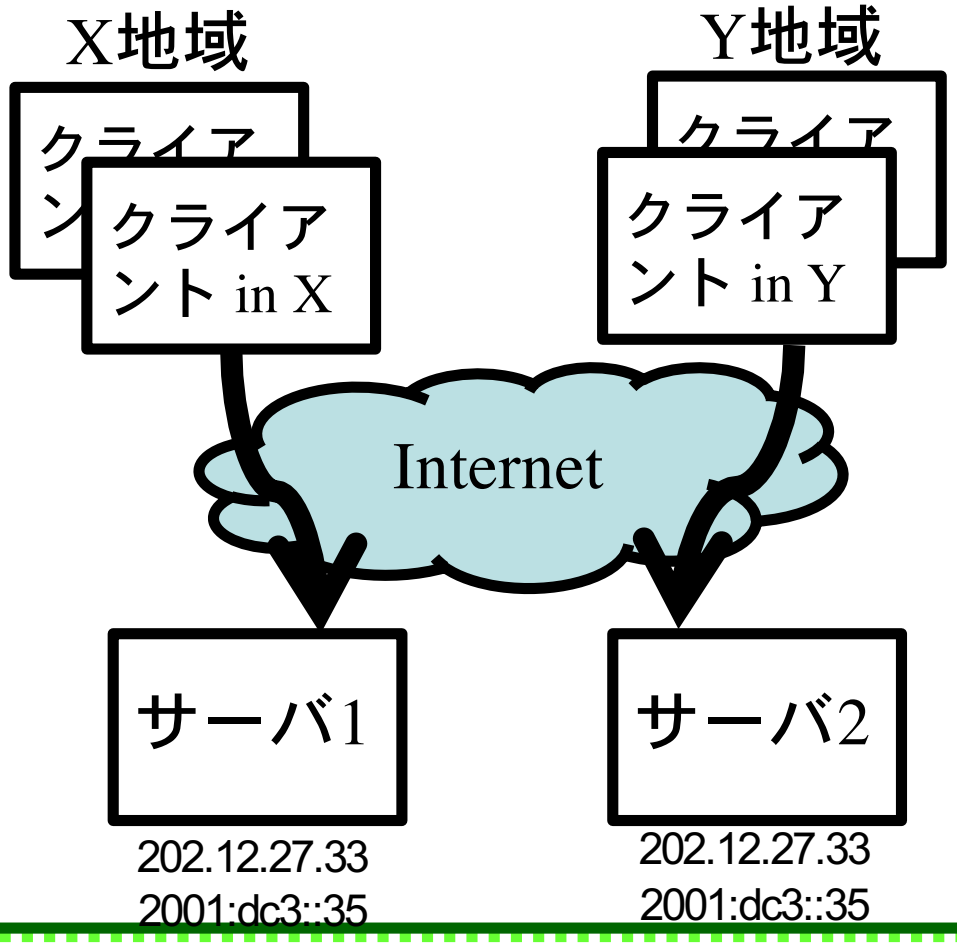
```
$ ./pr2 www.t.ring.gr.jp
addr: 202.18.64.24
addr: 210.146.64.7
addr: 133.37.44.6
addr: 160.26.2.184
```

経路制御を用いた負荷分散

- Anycast 方式
  - 特定のIPアドレス(Anycast address) を複数の経路で広告することで、クライアントに応じてアクセスする先を変更させる方式

例：世界各地から頻繁に大量のアクセスを受ける root DNS サーバの負荷分散方式として利用されている

http://www.root-servers.org/



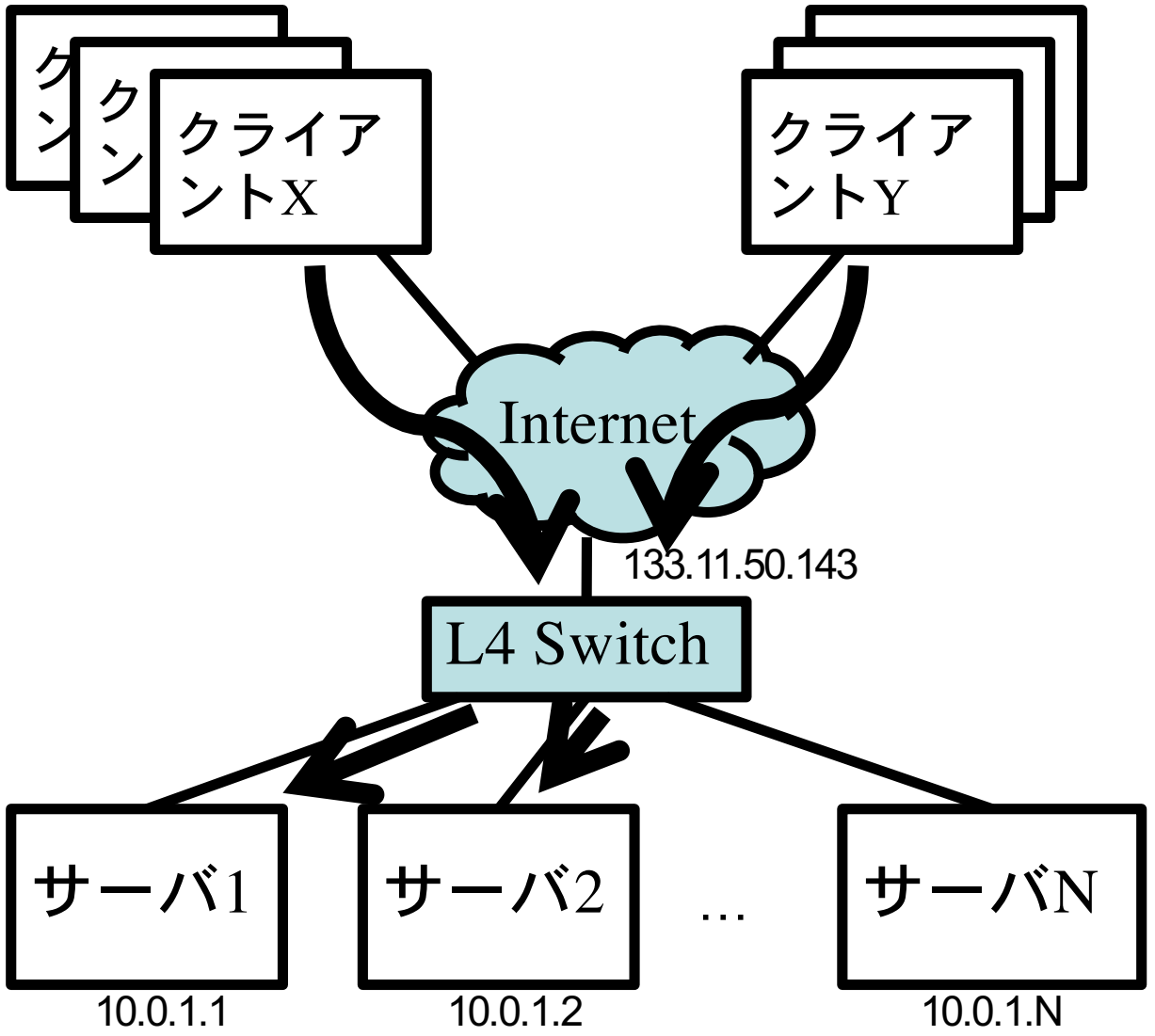
m.root-servers.net への経路

$ traceroute m.root-servers.net（東京大学内のホストからの例）
traceroute to m.root-servers.net (202.12.27.33), 64 hops max, 40 byte packets
1 ra35 (133.11.205.189) 0.429 ms 0.372 ms 0.373 ms
2 ra36-vlan2 (133.11.127.43) 0.339 ms 0.521 ms 1.017 ms
3 ra37-vlan3 (133.11.127.78) 0.436 ms 0.402 ms 0.459 ms
4 foundry4.nezu.wide.ad.jp (133.11.125.238) 0.459 ms 0.304 ms 0.324 ms
5 ve-42.foundry6.otemachi.wide.ad.jp (203.178.136.65) 0.460 ms 0.463 ms 0.443 ms
6 ve-5.alala1.otemachi.wide.ad.jp (203.178.140.215) 2.297 ms 2.239 ms 2.283 ms
7 m-gw.nspixp2.wide.ad.jp (202.249.2.86) 1.226 ms 1.158 ms 1.160 ms
8 M.ROOT-SERVERS.NET (202.12.27.33) 1.328 ms 1.347 ms 1.545 ms

$ traceroute m.root-servers.net (京都大学内のホストからの例)
traceroute to m.root-servers.net (202.12.27.33), 64 hops max, 40 byte packets
1 130.54.34.254 (130.54.34.254) 0.602 ms 0.575 ms 0.493 ms
2 CR3-V1889.gw.kuins.kyoto-u.ac.jp (133.3.1.81) 0.630 ms 0.649 ms 0.707 ms
3 FR3-eth3-0-3.gw.kuins.kyoto-u.ac.jp (133.3.1.5) 1.165 ms 0.941 ms 0.799 ms
4 150.99.186.25 (150.99.186.25) 5.023 ms 5.035 ms 5.042 ms
5 tokyo-dc-rm-ae12-vlan10.s4.sinet.ad.jp (150.99.2.97) 13.134 ms 13.160 ms 13.154 ms
6 tokyo-dc-gm2-ae0-vlan10.s4.sinet.ad.jp (150.99.2.54) 13.233 ms 13.127 ms 13.180 ms
7 as7500.ix.jpix.ad.jp (210.171.224.5) 13.448 ms 13.602 ms 13.423 ms
8 M.ROOT-SERVERS.NET (202.12.27.33) 13.573 ms 13.593 ms 13.844 ms

Application ベースの負荷分散

- 負荷分散装置による方式



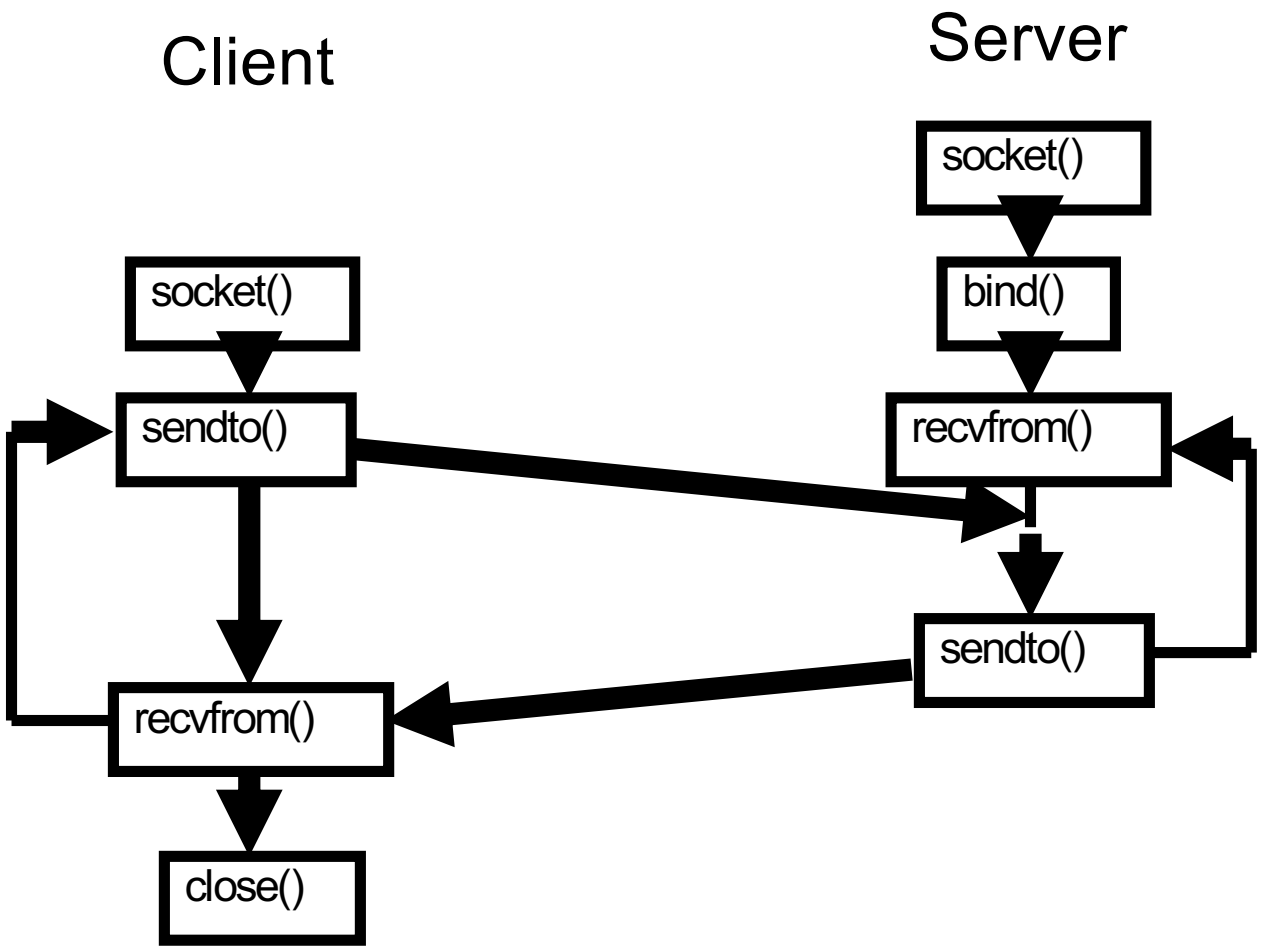
UDP を用いたプログラム

TCP/IP v.s. UDP/IP

- TCP/IP
  - コネクション型サービス
  - socket(), bind() に加え connect(), listen()/accept() が使われる
  - OS が信頼性通信を実現する

- UDP/IP
  - コネクションレス型サービス
  - （通常）socket(), bind() のみが使われる
  - ユーザプログラムで通信のエラー処理等を行う必要あり

Basic Scenario (UDP)



socket() 関数 (再掲)

#include <sys/socket.h>

int socket(int family, int type, int protocol);

- family: Protocol Family
  - AF_INET, AF_INET6, (AF_LOCAL, AF_ROUTE, AF_KEY)
- type: Type of socket
  - SOCK_STREAM, SOCK_DGRAM, (SOCK_RAW)
- protocol: raw socketの場合を除き “0” を用いる

- 成功すると 0以上の値を返す。失敗すると -1 を返す。

socket 関数のパラメータ

	AF_INET	AF_INET6	AF_LOCAL	AF_ROUTE	AF_KEY
SOCK_STREAM	TCP	TCP	Yes		
SOCK_DGRAM	UDP	UDP	Yes		
SOCK_RAW	IPv4	IPv6		Yes	Yes

AF_INET: IPv4, AF_INET6: IPv6, AF_LOCAL: UNIX domain
AF_ROUTE: routing control socket, AF_KEY: key socket

SOCK_STREAM: stream socket
SOCK_DGRAM: datagram socket
SOCK_RAW: raw socket

sendto()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *msg, size_t len,
        const struct sockaddr *toaddr, socklen_t tolen);
```

- sockfd: socket() が返した socket descriptor 値
- msg: 送信メッセージのポインタ
- len: メッセージ長
- toaddr: 送信先アドレスの sockaddr 構造体へのポインタ
- tolen: sockaddr 構造体のサイズ

- 成功すると(送信された)メッセージサイズを返す。失敗すると -1 を返す。

recvfrom()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void * restrict buf, size_t len,
        int flags, struct sockaddr *restrict fromaddr,
        socklen_t *restrict fromlen);
```

- sockfd: socket() が返した socket descriptor 値
- buf: 受信メッセージのポインタ
- len: 受信メッセージサイズ
- flag: パラメータ
- fromaddr: 宛先アドレスの sockaddr 構造体へのポインタ
- fromlen: sockaddr 構造体のサイズ

- 成功すると受信メッセージサイズを返す。失敗すると -1 を返す。

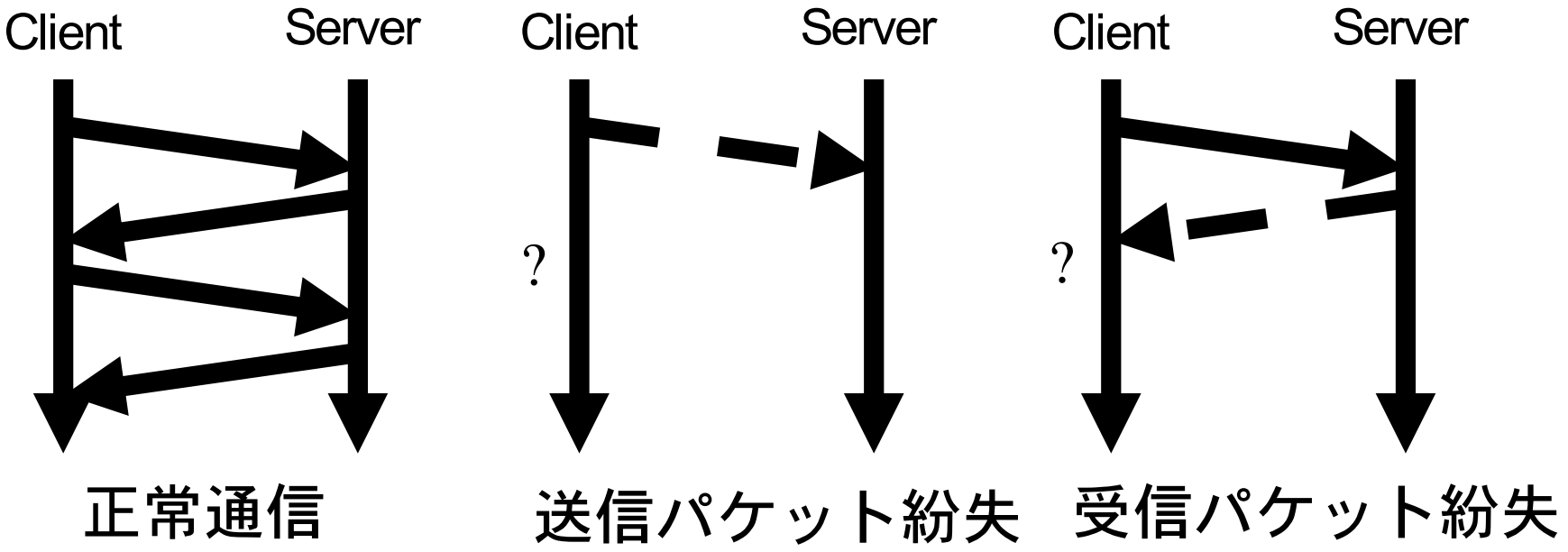
UDP echo client program（抜粋）

```
int sockfd, n;
struct sockaddr_in servaddr;
char sendline[MAXLINE], recvline[MAXLINE+1];

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(7);
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

while (fgets(sendline, MAXLINE, stdin) != NULL) {
  sendto(sockfd, sendline, strlen(sendline), 0,
         (struct sockaddr *)&servaddr, sizeof(servaddr));
  n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
  recvline[n] = '¥0';
  fputs(recvline, stdout);
}
```

UDP を用いたプログラムの注意事項

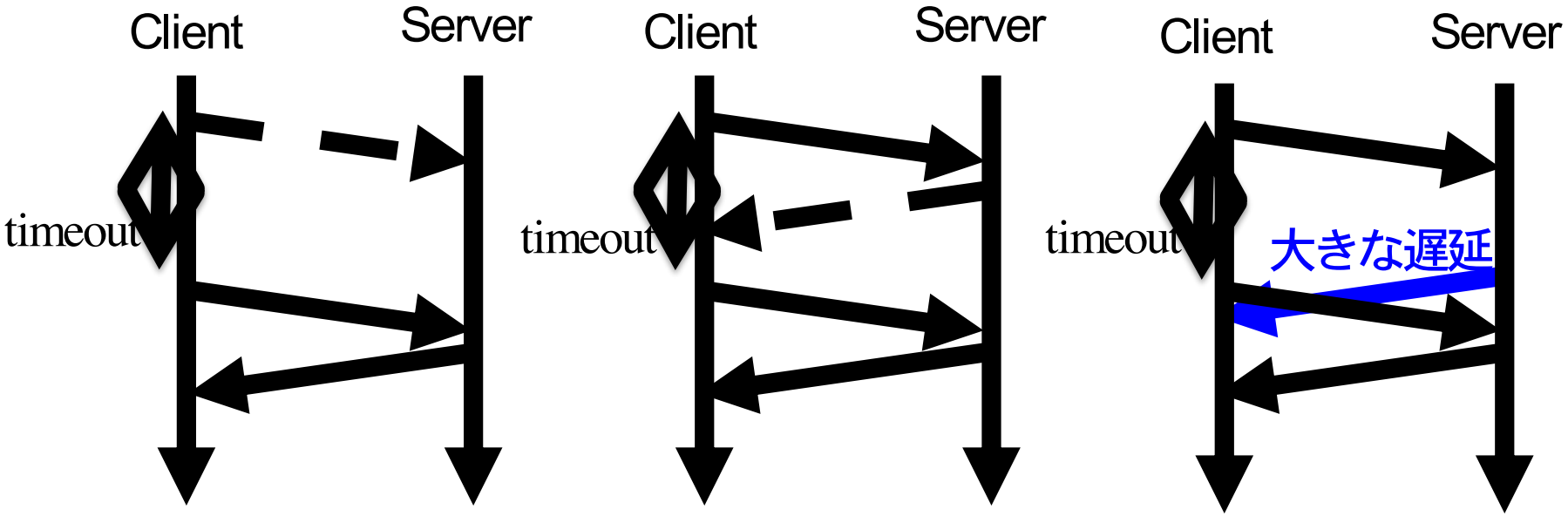
- TCP を用いたプログラムと異なり、ユーザプログラム中で通信のエラー処理などを行う必要がある。

例：先の UDP echo client program で、sendto したメッセージや recvfrom しようとするメッセージ（パケット）が紛失した場合、プログラムはどう動作するか？



UDP を用いたプログラムの注意事項

- サーバからメッセージを受け取れないため、(次に) stdin から入力されたメッセージをサーバに送れない。
  = recvfrom で動作が停止する
  - ⇦ recvfrom が timout する様にプログラムする



setsockopt()

```
#include <sys/types.h>
#include <sys/socket.h>
int setsockopt(int sockfd, int level, int optname,
                    const void *optval, socklen_t optlen);
```

- sockfd: socket() が返した socket descriptor 値
- level: オプションのレベルを指定する
- optname: オプション名を指定する
- optval: (必要に応じて)オプション値へのポインタを示す
- optlen: オプション値のサイズを示す

- 成功した場合は 0 を返す。失敗した場合は -1 を返す。

OPTION の一例

- SO_SNDBUF /* send buffer size */
- SO_RCVBUF /* receive buffer size */
- SO_SNDLOWAT /* send low-water mark */
- SO_RCVLOWAT /* receive low-water mark */
- SO_SNDTIMEO /* send timeout */
- SO_RCVTIMEO /* receive timeout */

timeout を用いたプログラミング（抜粋）

```
struct timeval tv;


tv.tv_sec = 2; tv.tc_usec = 0;
setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

while(fgets(sendline, MAXLINE, stdin) != NULL) {
  sendto(sockfd, sendline, strlen(sendline), 0, servaddr, sizeof(servaddr));
  if( (n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL)) < 0 ) {
    if( errno = EWOULDBLOCK )
      { fprintf(stderr, "socket timeout¥n"); continue; }
  } else {
    recvline[n] = '¥0' ;
    fputs(recvline, stdout);
  }
}
```

UDP echo server program（抜粋）

```
int sockfd, n;
socklen_t len;
struct sockaddr_in servaddr, cliaddr;
char buffer[MAXLINE];

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(7);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servadd));

for(;;) {
  len = sizeof(cliaddr);
  n = recvfrom(sockfd, buffer, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len);
  sendto(sockfd, buffer, n, 0, (struct sockaddr *)&cliaddr, len);
}
```

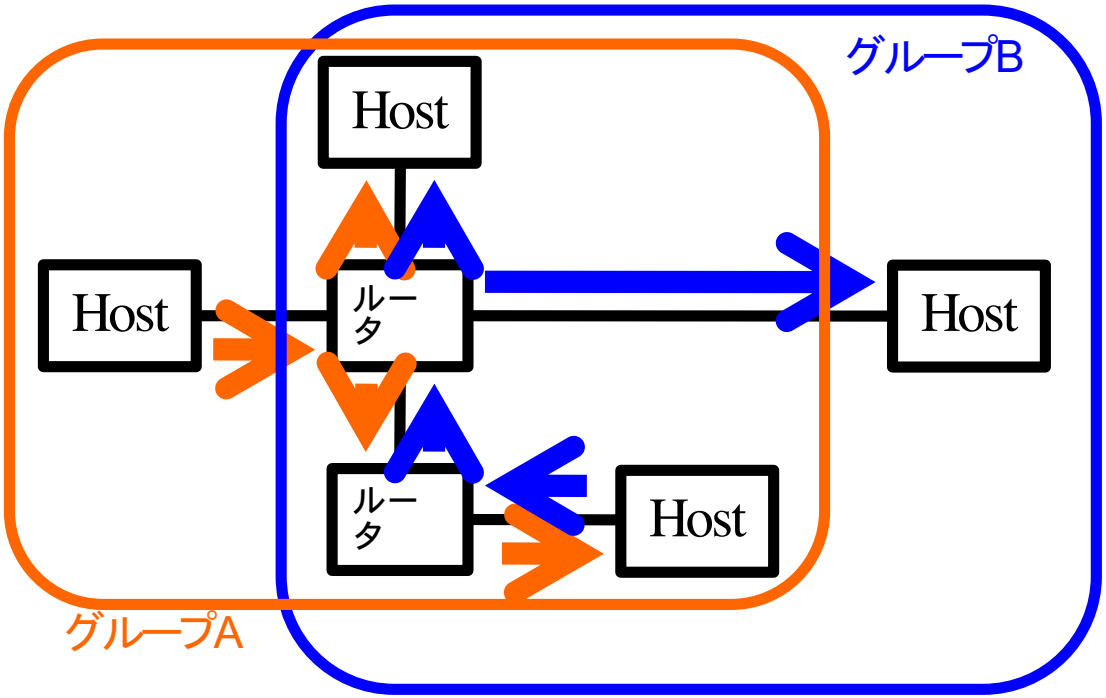
各アプリケーションで使用されるプロトコル

アプリケーション	IP	ICMP	UDP	TCP
ping		√		
traceroute		√	√	
OSPF	√			
RIP			√	
BGP				√
BOOTP			√	
DHCP			√	
NTP			√	
TFTP			√	
SNMP			√	
Multicast			√	

アプリケーション	IP	ICMP	UDP	TCP
SMTP				√
Telnet				√
FTP				√
HTTP				√
NNTP				√
DNS			√	√
NFS			√	√
Sun RPC			√	√
DCE RPC			√	√

マルチキャスト

- マルチキャストとは？
  - 特定のグループ（送受信したいホスト群）間でデータ交換できる様にした仕組み。パケットの複製は必要に応じて（マルチキャスト）ルータで行われる。



マルチキャストアドレス

- IPv4, IPv6 とも特別なアドレスブロックを利用
  - IPv4: 224.0.0.0～239.255.255.255
  - IPv6: ff00::1～fffe:ffff:ffff:ffff:ffff:ffff:ffff:ffff
- 特別なマルチキャストアドレス
  - IPv4:
    - 224.0.0.1　　全ホスト
    - 224.0.02　　全ルータ
  - IPv6:
    - ff02::1　　全ノード
    - ff02::2　　全ルータ

マルチキャストの配布範囲

- どれほど遠くへ到達するかを明示的に指定する指標（スコープ）

スコープ	IPv6 スコープ	IPv4	
		TTL	管理スコープ
ノードローカル	1	0	
リンクローカル	2	1	224.0.0.0～ 224.0.0.255
サイトローカル	5	<32	239.255.0.0～ 239.255.255.255
組織ローカル	8		239.192.0.0～ 239.195.255.255
グローバル	14	<255	224.0.1.0～ 238.255.255

ソケットオプション

- IPPROTO_IP
  - IP_MULTICAST_IF　マルチキャスト出力用I/Fの指定
  - IP_MULTICAST_TTL　出力マルチキャストのTTL指定
  - IP_MULTICAST_LOOP　出力マルチキャストのループバック許可／禁止
  - IP_ADD_MEMBERSHIP　マルチキャストグループへの参加(join)
  - IP_DROP_MEMBERSHIP　マルチキャストグループからの脱退(leave)
- IPPROTO_IPV6
  - IPV6_MULTICAST_IF　マルチキャスト出力用I/Fの指定
  - IPV6_MULTICAST_HOPS　出力マルチキャストのホップ限界指定
  - IPV6_MULTICAST_LOOP　出力マルチキャストのループバック許可／禁止
  - IPV6_ADD_MEMBERSHIP　マルチキャストグループへの参加(join)
  - IPV6_DROP_MEMBERSHIP　マルチキャストグループからの脱退(leave)

Multicast プログラムの例 (1/3)

```
main(int argc, char **argv)
{
  int sockfd;
  struct addrinfo hints, *res, *res0;
  struct sockaddr_storage addr;
  struct ifaddrs *ifp, *ifap;
  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_DGRAM;
  getaddrinfo(argv[1], “ntp”, &hints, &res0);
  res = res0;
  while (res != NULL) {
    sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    memcpy(&addr, res->ai_addr, res->ai_addrlen);
    res = res->ai_next;
  }
  freeaddrinfo(res0);
```

Multicast プログラムの例 (2/3)

```
getifaddrs(&ifp);
for ( ifap = ifp ; ifap != NULL ; ifap = ifap->ifa_next ) {
  if ( ifap->ifa_flags & IFF_MULTICAST) {
    if ( ifap->ifa_addr->sa_family == AF_INET) {
      sd = (struct sockaddr_in *)&addr;
      sa = (struct sockaddr_in *)ifap->ifa_addr;
      memcpy(&mreq.imr_multiaddr, sd->sin_addr, sizeof(struct in_addr));
      memcpy(&mreq.imr_interface, sa->sin_addr, sizeof(struct in_addr));
      setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
    } else if ( ifap->ifa_addr->sa_family == AF_INET6) {
      sd = (struct sockaddr_in6 *)&addr;
      memcpy(&mreq6.ipv6mr_multiaddr, sd->sin6_addr, sizeof(sd));
      mreq6.ipv6mr_interface = if_nametoindex(ifap->ifa_name);
      setsocket(sockfd, IPPROTO_IPV6, IPV6_ADD_MEMBERSHIP, &mreq6, sizeof(mreq6));
    }
  }
}
```

Multicast プログラムの例 (3/3)

```
  from = malloc(salen);
  for (;;) {
    len = salen;
    n = recvfrom(sockfd, buf, sizeof(buf), 0, from, &len);
    gettimeofday(&now, NULL);
    sntp_proc(buf, n, &now);
  }
}

void sntp_proc(char *buf, ssize_t n, struct timeval *nowptr)
{
  if ( n < sizeof(struct ntpdate)) {
    printf("packet size is too small¥n");
    return;
  }
  ...
  printf("clock difference = %d usec¥n", usec);
}
```