XI - La diffusion intégrale (Broadcast)

Diffusion intégrale

Broadcast

- communication sur le protocole `UDP`
- $\Rightarrow$ `communication par paquets`, ce qui signifie

  1 `sendto` $\leftrightarrow$ 1 paquet envoyé $\leftrightarrow$ 1 paquet reçu $\leftrightarrow$ 1 `recv`
- la diffusion s'effectue en direction de toutes les machines d'un réseau donné
- possible uniquement en `IPv4`

Rappel : procotocole TCP : communication par flux, ce qui signifie

pas de frontière aux messages

$\Rightarrow$ 1 `send` peut-être fragmenté en plusieurs envois et nécessiter plusieurs `recv`
Inversement plusieurs `send` peuvent être réceptionnés avec un seul `recv`

$\Rightarrow$ boucle pour la lecture avec `recv`

Diffusion intégrale

adresse

La diffusion intégrale s'effectue en envoyant un paquet sur la « dernière »adresse possible du réseau

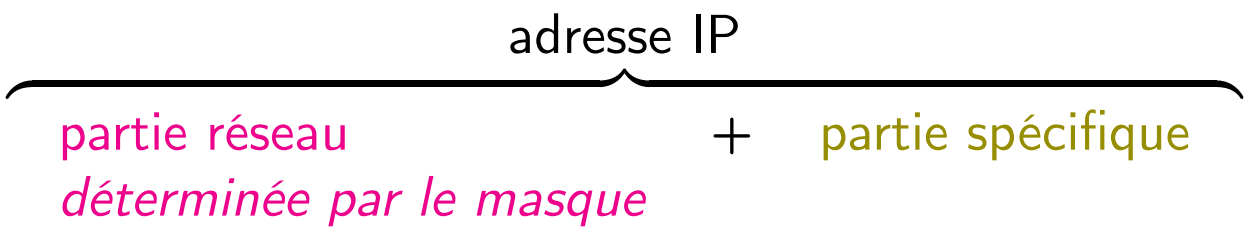
- l'adresse de broadcast du réseau `0.0.0.0` est donc `255.255.255.255`
- `0.0.0.0` est l'adresse générique du réseau local
- diffusion intégrale sur l'adresse `255.255.255.255` = diffusion intégrale sur le réseau local
  $\rightarrow$ pas de routage des paquets

Comment diffuser intégralement en dehors du réseau local ?

Diffusion intégrale

adresse

On a besoin de connaître l'adresse du réseau sur lequel on veut diffuser.




```
lulu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 ...
2: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 4a:49:43:49:79:bf brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.70.236/24 brd 192.168.70.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fdc7:9dd5:2c66:be86:4849:43ff:fe49:79bf/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::4849:43ff:fe49:79bf/64 scope link
       valid_lft forever preferred_lft forever
```

L'adresse IPV4 de lulu est 192.168.70.236 et le masque est de 24

i.e. 24 bits (3 octets) de poids forts = adresse du réseau de lulu

$\Rightarrow$ adresse de broadcast de ce réseau : 192.168.70.255

Remarque : `brd` indique l'adresse de broadcast dans l'affichage de la commande `ip a`

Diffusion intégrale

envoi

Pour envoyer des messages en diffusion intégrale, les étapes sont

1. créer une socket UDP IPv4
2. activer l'option de diffusion intégrale de la socket

   → option `SO_BROADCAST` au niveau `SOL_SOCKET`
3. initialiser l'adresse de diffusion : IP + port
4. envoyer les messages à l'adresse de diffusion

   → `sendto`

Diffusion intégrale

envoi

```
int main() {
  int sock=socket(PF_INET,SOCK_DGRAM,0);

  int ok=1;
  int r=setsockopt(sock,SOL_SOCKET,SO_BROADCAST,&ok,sizeof(ok));
  if(r == -1){
    perror("setsockopt SO_BROADCAST");
    exit(1);
  }

  struct sockaddr_in adrdiff;
  memset(&adrdiff, 0, sizeof(adrdiff));
  adrdiff.sin_family = AF_INET;
  adrdiff.sin_port = htons(8888);
  r = inet_pton(AF_INET, "255.255.255.255", &adrdiff.sin_addr);
  if( r <= 0){
    perror("pb adresse");
    exit(1);
  }

  char buf[100];
  sprintf(buf,"bonjour la terre\n");
  r = sendto(sock, buf, strlen(buf), 0, (struct sockaddr *) &adrdiff, (socklen_t)sizeof(struct sockaddr_in));
  if(r < 0){
    perror("sendto");
    exit(1);
  }

  return 0;
}
```

Diffusion intégrale

réception

Pour recevoir des messages diffusés, les étapes sont

1. créer une socket UDP IPv4
2. initialiser l'adresse de réception avec le port d'écoute
3. lier la socket à l'adresse de réception
4. recevoir les messages

C'est un récepteur UDP standard.

Diffusion intégrale

réception

```
int main() {
  int sock=socket(PF_INET,SOCK_DGRAM,0);

  struct sockaddr_in address_sock;
  address_sock.sin_family=AF_INET;
  address_sock.sin_port=htons(8888);
  address_sock.sin_addr.s_addr=htonl(INADDR_ANY);

  int r = bind(sock,(struct sockaddr *)&address_sock,sizeof(struct sockaddr_in));
  if(r){
    perror("bind");
    exit(1);
  }

  struct sockaddr_in emet;
  socklen_t taille=sizeof(emet);
  char buf[100], adr[100];
  while(1){
    int rec=recvfrom(sock,buf,100,0,(struct sockaddr *)&emet,&taille);
    buf[rec]='\0';
    printf("Message recu : %s\n",buf);
    printf("Port de l'emetteur: %d\n",ntohs(emet.sin_port));
    inet_ntop(AF_INET, &(emet.sin_addr), adr, sizeof(adr));
    printf("Adresse de l'emetteur: %s\n\n", adr);
  }

  return 0;
}
```

Complément : un client TCP générique

Complément

un client TCP générique

- On a vu au cours 5 comment écrire un client TCP qui peut se connecter en IPv6 ou IPv4 à un serveur.

  Si la connexion au serveur se fait en IPv4, alors ce client transforme localement l'adresse IPv4 du serveur en adresse « IPv4 mapped ».

- On peut également écrire un client générique qui utilise des adresses IPv6 ou IPv4 selon la connexion au serveur toujours en faisant appel à `getaddrinfo`.
  - il faut initialiser le champs `ai_family` de `hints` à `AF_UNSPEC`
  - faire l'appel à `getaddrinfo(hostname, port, &hints, &res)`
  - parcourir les éléments `p` de `res` et tenter une connexion jusqu'à ce que cela réussisse
  - tester alors le champs `ai_family` de `p`, lorsqu'on a besoin de savoir si on a établi une connexion IPv6 ou IPv4

Complément

un client TCP générique

```
int main(int argc, char** argv) {
  int sock=socket(PF_INET,SOCK_STREAM,0);

  struct addrinfo *res, *p, hints;
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype=SOCK_STREAM;

  int ret;
  if ((ret = getaddrinfo(argv[1], argv[2], &hints, &res)) != 0 || res == NULL){
    fprintf(stderr, "erreur getaddrinfo : %s\n", gai_strerror(ret));
    return -1;
  }

  p = res;
  while( p != NULL ){
    if((sock = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) > 0){
      if(connect(sock, p->ai_addr, p->ai_addrlen) == 0)
        break;
      close(sock);
    }
    p = p->ai_next;
  }
  if (p == NULL) return 1;

  affiche_adresse(p->ai_addr);
  freeaddrinfo(res);

  char buf[100];
  sprintf(buf,"bonjour la terre");
  send(sock,buf,strlen(buf),0);
  close(sock);
  return 0;
}
```

Complément

un client TCP générique

Une fois la connexion réalisée, on peut par exemple afficher l'adresse. Pour cela, il faut agir en fonction du type de connexion.

```
void affiche_adresse(struct sockaddr *adr){
  if(adr->sa_family == AF_INET6){
    char adr_buf[INET6_ADDRSTRLEN];
    struct sockaddr_in6 *adr6 = (struct sockaddr_in6 *) adr;
    memset(adr_buf, 0, sizeof(adr_buf));

    inet_ntop(AF_INET6, &(adr6->sin6_addr), adr_buf, sizeof(adr_buf));
    printf("adresse serveur : IP: %s port: %d\n", adr_buf, ntohs(adr6->sin6_port));
  }
  else{
    char adr_buf[INET_ADDRSTRLEN];
    struct sockaddr_in *adr4 = (struct sockaddr_in *) adr;
    memset(adr_buf, 0, sizeof(adr_buf));

    inet_ntop(AF_INET, &(adr4->sin_addr), adr_buf, sizeof(adr_buf));
    printf("adresse serveur : IP: %s port: %d\n", adr_buf, ntohs(adr4->sin_port));
  }
}
```

Complément

un client TCP générique

Attention, il ne faut pas libérer la liste d'adresses `res` tant qu'on souhaite y avoir accès.

On peut copier les informations que l'on veut garder avant de libérer `res`.

Par exemple, dans notre client générique, on peut remplacer les deux lignes affichant l'adresse du serveur auquel notre client s'est connecté et libérant `res` par :

```
struct sockaddr *sadr;
socklen_t adrlen;
adrlen = p->ai_addrlen;
sadr = malloc(adrlen);
memcpy(sadr, p->ai_addr, adrlen);

freeaddrinfo(res);

affiche_adresse(sadr);
```