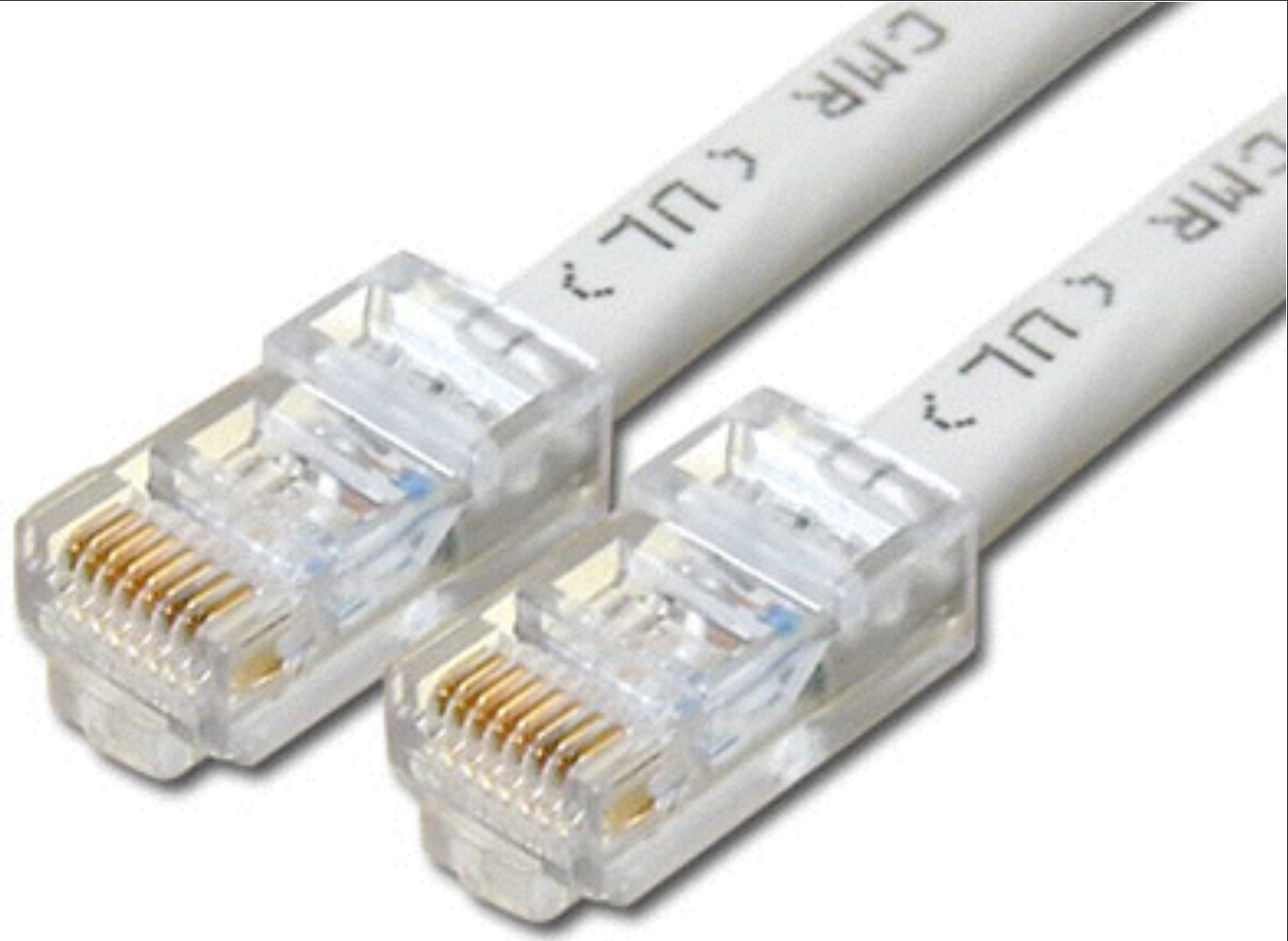
Programmation Réseau

Multicast
Broadcast



Jean-Baptiste.Yunes@univ-paris-diderot.fr

UFR Informatique

2014

université PARIS DIDEROT PARIS 7

La diffusion

- pour l’instant nous n’avons vu que de la communication point à point

- mais il peut être utile ou nécessaire de vouloir atteindre plusieurs destinataires

- C’est la diffusion

La diffusion

établir une connexion point à point fiable est déjà relativement coûteux, alors une diffusion fiable…

- C’est pourquoi n’est disponible que de la diffusion non fiable

- on ne sait pas si même **un** destinataire a pu lire le message…

- la diffusion est disponible de deux façons différentes :
  - diffusion intégrale (broadcasting) dans laquelle la diffusion s'effectue en direction de toutes les machines d'un réseau donné
  - multidiffusion ou diffusion de groupe (multicasting) dans laquelle la diffusion s'effectue à destination d'applications abonnées à un groupe

- diffusion intégrale
  - dans ce cas, la communication s’apparente à celle de la diffusion d’un message à tout le monde
    - sirène incendie…
- multidiffusion
  - dans ce cas, la communication s’apparente plutôt à celle de la radiodiffusion ou de la télévision
    - on s’abonne (en choisissant un canal)

Les adresses et la diffusion

- Rappelons qu'il existe (en IPv4 et à l'origine) cinq classes d'adresses :
  - la classe A `0.0.0.0` — `127.255.255.255`, soit 128 réseaux de 16M machines
  - la classe B `128.0.0.0` — `191.255.255.255`, soit 16k réseaux de 64k machines
  - la classe C `192.0.0.0` — `223.255.255.255`, soit 2M réseaux de 256 machines
  - la classe D `224.0.0.0` — `239.255.255.255`, réservée à la multidiffusion
  - la classe E `240.0.0.0` — `255.255.255.255`

- la **diffusion intégrale** s'effectue en envoyant un paquet à l'adresse de la « dernière » adresse possible du réseau

  - il existe un alias standard pour désigner l'adresse de diffusion intégrale pour n'importe quel réseau :

    `255.255.255.255`

  - qui signifie donc (en théorie) « à toutes les machines du réseau Internet »...

est normalement limitée au réseau local (pas de routage)

- la **multidiffusion** s'effectue par utilisation d'un groupe (identifié par une adresse de classe D)
  - un groupe identifie un **service** de diffusion
  - lorsqu'un message est émis à destination d'un groupe, tous les membres peuvent recevoir un exemplaire du message
  - pour recevoir il faut être membre du groupe
  - pour envoyer cela n'est pas nécessaire...

est encore rarement (correctement) routée

- pour l'envoi il est donc suffisant d'utiliser une adresse de multidiffusion (classe D)

  - en pratique on oubliera les adresses `224.x.x.x`, `232.x.x.x`, `233.x.x.x` et `239.x.x.x` (qui sont en partie réservées)

La diffusion intégrale aka broadcast

- le broadcast s'effectue **normalement** en complémentant le masque de réseau puis en l'additionnant à l'adresse du réseau

  - ex : si je suis dans le réseau `173.50.24.0` que le masque de réseau est `0xFFFFFC00` alors l'adresse de diffusion est `173.50.25.255`

La diffusion intégrale aka broadcast

- la diffusion intégrale s'effectue en pratique sur l'adresse `255.255.255.255` (`FF.FF.FF.FF`)
  - en fait un broadcast sur le réseau nul `0.0.0.0` qui correspond au réseau sur lequel on est connecté (`this` du réseau)
- Tous les processus en écoute sur le port concerné pourront recevoir le message transmis

Récepteur

```
import java.net.*;

public class Receiver {
  public static void main(String args[]) {
    try {
      byte [] data = new byte[256];
      DatagramSocket ds = new DatagramSocket(60123);
      DatagramPacket dp =
                new DatagramPacket(data,data.length);
      while (true) {
        ds.receive(dp);
        String s = new String(dp.getData(),
                              0,dp.getLength());
        System.out.println("Received "+s);
      }
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Rien de particulier

Envoyeur

```
import java.net.*;

public class Sender {
  public static void main(String args[]) {
    try {
      String s = "Ca va Paris ?";
      DatagramSocket ds = new DatagramSocket();
      DatagramPacket dp =
        new DatagramPacket(s.getBytes(),
         s.getBytes().length,
         new InetSocketAddress("255.255.255.255",60123));
      ds.send(dp);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Rien de particulier

API C

- en C il faut positionner la socket dans un mode autorisant la diffusion intégrale à l'envoi

  `setsockopt()`

- qui est une fonction permettant de modifier les caractéristiques associées à une socket, et ce à différentes niveaux de protocoles

  - au moins au niveau socket `SOL_SOCKET`

  - c'est assez similaire à `fcntl()` / `ioctl()`

Broadcast - API C

Terminal — less — 80×24

```
GETSOCKOPT(2)                BSD System Calls Manual                GETSOCKOPT(2)

NAME
     getsockopt, setsockopt -- get and set options on sockets

SYNOPSIS
     #include <sys/socket.h>

     int
     getsockopt(int socket, int level, int option_name,
         void *restrict option_value, socklen_t *restrict option_len);

     int
     setsockopt(int socket, int level, int option_name,
         const void *option_value, socklen_t option_len);

DESCRIPTION
     Getsockopt() and setsockopt() manipulate the options associated with a
     socket.  Options may exist at multiple protocol levels; they are always
     present at the uppermost ``socket'' level.

     When manipulating socket options the level at which the option resides
:
```

- le niveau qui nous intéresse est SOL_SOCKET
- l'option d'activation du broadcast à l'envoi est SO_BROADCAST

- Pour l'adresse d'envoi on peut utiliser le symbole prédéfini INADDR_BROADCAST

Envoyeur

```
s = socket(AF_INET,SOCK_DGRAM,0);
bzero(&sin,sizeof(sin));
sin.sin_family = PF_INET;
sin.sin_port = htons(60123);
sin.sin_addr.s_addr =
                    htonl(INADDR_BROADCAST);
int e = 1;
r = setsockopt(s,SOL_SOCKET,
                SO_BROADCAST,&e,sizeof(e));
r = sendto(s,msg,strlen(msg),0,
    (struct sockaddr *)&sin,sizeof(sin));
```

peut être remplacé par
`inet_addr("255.255.255.255")`

Récepteur

```
s = socket(AF_INET,SOCK_DGRAM,0);
bzero(&sin,sizeof(sin));
sin.sin_family = PF_INET;
sin.sin_port = htons(60123);
sin.sin_addr.s_addr = htonl(INADDR_ANY);
r = bind(s,(struct sockaddr *)&sin,sizeof(sin));
while (1) {
  r = recv(s,msg,100,0);
  if (r==-1) {
    perror("recv");
    close(s);
    exit(EXIT_FAILURE);
  }
  msg[r] = '\0';
  printf("Recu: %s\n",msg);
}
```

La multidiffusion aka multicast

- le multicast s'effectue **normalement** en choisissant une adresse de classe D :
  - `224.0.0.0` — `239.255.255.255`

- pour la réception il faut utiliser une `MulticastSocket`

  ```
  MulticastSocket(int port);
  ```

- et s'enregistrer comme membre d'un groupe par appel à (par exemple)

  ```
  void joinGroup(InetAddress groupe);
  ```

- la diffusion est limitée au TTL (Time To Live - le nombre de routeurs à franchir)...

- On peut sortir d'un groupe par appel à

  `void leaveGroupe(…);`

- le reste de la communication est identique au cas habituel

```
try {
  byte [] data = new byte[256];
  InetAddress ia =
    InetAddress.getByName("225.1.2.4");
  MulticastSocket ms = new MulticastSocket(28888);
  ms.joinGroup(ia);
  DatagramPacket dp =
    new DatagramPacket(data,data.length);
  while (true) {
    ms.receive(dp);
    String s = new String(dp.getData()
                          0,dp.getLength());
    System.out.println("Received "+s);
  }
} catch(Exception e) {
  e.printStackTrace();
}
```

```
try {
  String s = "Bonjour la compagnie";
  DatagramSocket ms =
    new DatagramSocket();
  InetAddress ia =
    InetAddress.getByName("225.1.2.4");
  DatagramPacket dp =
    new DatagramPacket(s.getBytes(),
            s.getBytes().length,ia,28888);
  ms.send(dp);
  System.out.println("Sent");
} catch(Exception e) {
  e.printStackTrace();
}
```

- **Attention** selon les piles réseau, les appels peuvent être légèrement différent, mais le principe est le même. Certains noyaux Linux nécessitent l'activation du multicast. Consultez la doc et le web.

- en C il est nécessaire de **joindre le groupe** avec un appel à :

  `setsockopt()`

- qui est une fonction permettant de modifier les caractéristiques associées à une socket, et ce à différentes niveaux de protocoles

  - au moins au niveau socket `SOL_SOCKET`

  - c'est assez similaire à `fcntl()` / `ioctl()`

Multicast - API C

Terminal — less — 80×24

```
GETSOCKOPT(2)                BSD System Calls Manual                GETSOCKOPT(2)

NAME
     getsockopt, setsockopt -- get and set options on sockets

SYNOPSIS
     #include <sys/socket.h>

     int
     getsockopt(int socket, int level, int option_name,
         void *restrict option_value, socklen_t *restrict option_len);

     int
     setsockopt(int socket, int level, int option_name,
         const void *option_value, socklen_t option_len);

DESCRIPTION
     Getsockopt() and setsockopt() manipulate the options associated with a
     socket.  Options may exist at multiple protocol levels; they are always
     present at the uppermost ``socket'' level.

     When manipulating socket options the level at which the option resides
:
```

- le niveau qui nous intéresse est `IPPROTO_IP` (documentée dans le manuel `ip(4)`)

- la valeur est `IP_ADD_MEMBERSHIP` pour rejoindre un groupe

  - `IP_DROP_MEMBERSHIP` pour sortir du groupe

- il existe d'autres valeurs `IP_MULTICAST_LOOP`, `IP_MULTICAST_IF`, `IP_MULTICAST_TTL`

- la valeur est exprimée à l'aide du type

```
struct ip_mreq {
   /* multicast group to join */
   struct in_addr imr_multiaddr;
   /* interface to join on */
   struct in_addr imr_interface;
};
```

attention il peut être nécessaire (en général ça l'est) d'autoriser plusieurs clients sur une même machine à se joindre au groupe

- par conséquent il faut autoriser plusieurs sockets à s'associer sur le port concerné...

- C'est encore un appel à `setsockopt()` qui le permet au niveau socket (SOL_SOCKET)

  - avec l'option SO_REUSEPORT

Sender

```
int main(int argc, char *argv[]) {
  struct sockaddr_in addr;
  int s; char *msg="Salut les gars!";
  if ((s=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
    perror("socket"); exit(EXIT_FAILURE);
  }
  bzero(&addr,sizeof(addr));
  addr.sin_family      = AF_INET;
  addr.sin_addr.s_addr = inet_addr("225.1.2.4");
  addr.sin_port        = htons(28888);
  if (sendto(s,msg,strlen(msg),0,
          (struct sockaddr *)&addr,sizeof(addr)) < 0) {
    perror("sendto"); close(s); exit(EXIT_FAILURE);
  }
  close(s);
  exit(EXIT_SUCCESS);
}
```

Multicast - API C

Receiver

```
int main(int argc, char *argv[]) {
  struct sockaddr_in addr;
  int s; struct ip_mreq mreq; char msg[256]; int ok=1;
  if ((s=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
    perror("socket");
    exit(EXIT_FAILURE);
  }
  if (setsockopt(s,SOL_SOCKET,SO_REUSEPORT,&ok,sizeof(ok)) < 0) {
    perror("Reusing ADDR failed");
    exit(EXIT_FAILURE);
  }
  bzero(&addr,sizeof(addr));
  addr.sin_family      = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port        = htons(28888);
  if (bind(s,(struct sockaddr *)&addr,sizeof(addr)) < 0) {
    perror("bind");
    close(s);
    exit(EXIT_FAILURE);
  }
                                                          …
```

Receiver

```
…
  mreq.imr_multiaddr.s_addr = inet_addr("225.1.2.4");
  mreq.imr_interface.s_addr = htonl(INADDR_ANY);
  if (setsockopt(s,IPPROTO_IP,IP_ADD_MEMBERSHIP,
                 &mreq,sizeof(mreq)) < 0) {
    perror("Subscribing to group failed");
    close(s); exit(EXIT_FAILURE);
  }
  while (1) {
    bzero(msg,256);
    if (recv(s,msg,256,0) < 0) {
      perror("recv"); close(s); exit(EXIT_FAILURE);
    }
    printf("%s\n",msg);
  }
  return EXIT_SUCCESS;
}
```