Programmation réseau 8

Juliusz Chroboczek

12 mars 2019

1 Communication locale au lien

L'intérêt principal de la commutation de paquets est qu'elle permet de construire des internets, des réseaux constitués de plusieurs liens interconnectés par des routeurs, et de communiquer de façon transparente à travers ces derniers. Il existe par ailleurs des technologies de communication locales au lien, telles que USB (une technologie de réseau local centralisé et filaire, dont le principal intérêt est d'être très peu chère), *BlueTooth* (réseau radio à faible portée), *Zigbee* et *LORA* (réseaux à faible débit et haute portée conçus pour les senseurs, par exemple le compteur d'eau de votre appartement) ou encore *BlueTooth Low Energy* (BLE) (qui, contrairement à ce qu'implique son nom, est une technologie complètement différente de *BlueTooth*). Par ailleurs, rien n'empêche d'écrire des applications directement au-dessus d'une couche lien traditionnelle, par exemple Ethernet ou WiFi.

Le problème principal de ces technologies est qu'en l'absence d'une couche de convergence chaque application doit supporter toutes les technologies nécessaires. Par exemple, une application de musique qui supporte un *mixer* USB ne va pas automatiquement être capable de parler à un *mixer BlueTooth*. Il est donc naturel de vouloir utiliser IP pour la communication locale au lien.

1.1 Communication locale au lien en IPv4

IPv4 n'a pas de support particulier pour la communication locale au lien ; il est donc nécessaire d'affecter une adresse à chaque interface pour pouvoir envoyer des paquets IPv4. En présence de connectivité Internet, il est possible d'utiliser l'adresse globale pour la communication locale au lien ; en l'absence d'une adresse globale, on peut tirer au hasard une adresse dans le préfixe `169.254.0.0/16`. En l'absence d'une table de routage, on peut envoyer des paquets sur le lien local en liant une *socket* à une interface particulière à l'aide d'un appel à `setsockopt` avec l'option `SO_BINDTODEVICE` ; pour envoyer des paquets, il faut passer l'option `MSG_DONTROUTE` à `sendmsg`.

Pour les nouveaux protocoles, cependant, il est plus simple et plus robuste d'utiliser le support natif d'IPv6 pour la communication locale au lien.

1.2 Communication locale au lien en IPv6

Chaque interface IPv6 possède une adresse dite « locale au lien » dans le préfixe `fe80::/80`. Ces adresses ne sont pas globalement uniques — à un moment donné, elles sont uniques sur un lien donné.

Les adresses locales au lien IPv6 ont deux propriétés qui les rendent plus robustes et plus faciles à utiliser que les adresses globales :

- elles existent toujours, même en l'absence de connectivité à l'Internet Global ;
- elles sont relativement stables — l'adresse locale au lien d'une interface ne change normalement pas lors d'une renumérotation (par exemple après un événement de mobilité).

Pour envoyer un paquet à une adresse locale au lien, il faut spécifier le lien dont il s'agit ; en pratique, on spécifie l'interface par laquelle le paquet doit sortir. Dans la syntaxe textuelle, on spécifie l'interface en faisant suivre l'adresse d'un signe pourcent « `%` » et du nom de l'interface ; par exemple, je me suis récemment connecté à un routeur qui n'avait pas encore été numéroté[1] à l'aide de la commande :

```
ssh fe80::204:75ff:fe90:9ba9%eth0
```

(J'avais déterminé son adresse locale au lien comme décrit au paragraphe 2.2.1 ci-dessous.)

Au niveau de l'API *sockets*, l'indice de l'interface sortante est spécifié dans le champ `sin6_scope_id` de la structure `sockaddr_in6`. Je connais trois techniques pour obtenir un indice d'interface :

- dans un serveur UDP, il suffit de recopier la structure `sockaddr_in6` retournée par `recvfrom` ou `recvmsg`, elle contient un `sin6_scope_id` valide (et du coup, le serveur UDP simpliste décrit au poly 5 supporte les clients locaux au lien sans modification) ;
- si on connaît le nom de l'interface, on peut obtenir son indice à l'aide de la fonction `if_nametoindex` ;
- on peut énumérer toutes les interfaces à l'aide de la fonction `getifaddrs`.

2 Communication 1-*n*

Les programmes que nous avons écrits jusqu'à maintenant implémentaient de la communication *unicast* : la destination d'un paquet était l'adresse d'une interface, et un paquet émis était reçu au plus une fois. A contrario de ce paradigme de communication 1-1, il existe des paradigmes de communication 1-*n* : le *broadcast*, où un paquet est envoyé à toutes les interfaces à portée (typiquement le lien local), et le *multicast*, où un paquet est envoyé à un *groupe multicast*, et reçu par toutes les interfaces qui ont choisi de s'*abonner* au groupe. (Il existe en principe aussi le *multi-unicast*, où un paquet est envoyé à un ensemble d'adresses unicast choisies par l'émetteur, mais je ne connais pas de technologies de multi-unicast déployées en production.)

Comme TCP maintient de l'état 1-1, il n'est pas possible d'utiliser les techniques de communication 1-*n* avec TCP. Par contre, comme UDP est sans état, il supporte sans aucun problème la communication 1-*n* — un datagramme est simplement émis une fois et reçu plusieurs fois.

1. En fait, j'avais oublié son adresse IP. Il faut que je rachète des étiquettes adhésives.

Les techniques 1-*n* sont utiles pour deux types d'applications :

- la *découverte*, où un nœud recherche des pairs dont il ne connaît pas encore l'adresse IP ; par exemple, les protocoles de configuration d'adresses (DHCPv4, RA, DHCPv6), de découverte de voisins (ND) ou de découverte de services (mdns et DNS-SD, commercialisés par Apple sous le nom *Bonjour*, et utilisés notamment pour localiser les imprimantes[2]) ;
- le transfert de données 1-*n*, où les données sont émises une fois et dupliquées par le réseau, par exemple pour la vidéo ou l'audio en temps réel.

En pratique, les protocoles de découverte sont vastement déployés et utilisés, notamment dans le cas local au lien. Par contre, comme l'infrastructure de *multicast* n'est pas déployée à l'échelle globale, le transfert de données 1-*n* est très rarement utilisé, sauf dans le cas particulier de la télévision IP dans les réseaux privés (les 300 chaînes de télévision, la plupart dans des langues que vous ne comprenez pas, qui vous sont fournies avec votre abonnement Internet).

2.1 *Broadcast* IPv4

IPv4 supporte un *broadcast* local au lien. Le *broadcast* IPv4 est inefficace — un paquet *broadcast* est reçu par toutes les interfaces du lien (même celles qui ne parlent pas IPv4), ce qui a notamment pour effet de réveiller les nœuds mobiles qui étaient en veille.

Pour émettre des paquets *broadcast*, il suffit de les envoyer à l'adresse `255.255.255.255`. Les paquets avec une destination *broadcast* sont émis sur une interface qui est choisie par le système. Pour spécifier explicitement l'interface sortante, on peut utiliser l'option de *socket* `SO_BINDTODEVICE`, déjà mentionnée ci-dessus.

Pour recevoir les *broadcasts*, il n'y a rien de spécial à faire, il suffit d'écouter sur une *socket* liée au bon port. (Il n'est pas possible de distinguer un paquet *broadcast* d'un paquet *unicast*, sauf en consultant l'adresse de destination — voyez le cours précédent —, attention cependant au *broadcast dirigé*, une technique obsolète de *broadcast* qui utilise la dernière adresse d'un préfixe comme adresse de destination.)

2.2 *Multicast*

IPv4 et IPv6 supportent une communication *multicast*, c'est à dire où un paquet est envoyé à une *adresse de groupe*, et reçu par toutes les interfaces qui ont exprimé leur désir de recevoir les paquets avec cette destination (on dit que ces interfaces sont *abonnées* au groupe). Dans le cas local au lien, le *multicast* est simplement une alternative plus efficace au *broadcast*. Dans le cas global, le *multicast* requiert que le réseau maintienne de l'état pour chaque groupe et duplique les paquets envoyés au groupe ; en pratique, aucun fournisseur ne déploie l'infrastructure nécessaire au *multicast* global, et le *multicast* ne marche donc pas à travers les routeurs (sauf dans certains réseaux conçus spécialement pour supporter le *multicast*, par exemple les réseaux dédiés à la télévision IP).

Les adresses de groupe *multicast* IPv4 sont dans le préfixe `224.0.0.0/4`. Je n'en parle pas davantage dans ce cours.

2. Tapez `avahi-browse -a` et moquez-vous des gens qui utilisent un Mac.

Les adresses de groupe *multicast* IPv6 sont dans le préfixe `ff00::/8`. Le premier octet vaut `0xff` (255) pour indiquer une adresse *multicast*. Le deuxième octet consiste de deux champs de 4 bits chacun. Le champ d'ordre haut contient des *flags* ; le seul qui nous intéresse est le *flag* d'ordre bas, qui vaut 0 pour une adresse officiellement attribuée[3], et 1 pour une adresse choisie localement (par exemple en la tirant au hasard). Le dernier demi octet indique la portée de l'adresse : 2 pour une adresse locale au lien, 5 pour une adresse locale au site[4], et `0xE` (14) pour une adresse globale. Par exemple, l'adresse

```
ff02::1:6
```

est une adresse *multicast* locale au lien affectée globalement.

2.2.1 Adresses *multicast* importantes

Lors du lancement, chaque nœud IPv6 abonne chacune de ses interfaces au groupe local au lien `ff02::1`, qui contient donc tous les nœuds IPv6 du lien (« `ip6-allnodes` ») ; ce groupe remplace donc l'adresse *broadcast* en IPv6. Il est notamment utile pour découvrir tous les nœuds d'un lien, par exemple quand on a oublié l'adresse d'un routeur :

```
ping ff02::1%eth0
```

Un routeur abonne en outre toutes ses interfaces aux groupes `ff02::2` (tous les routeurs du lien) et `ff05::2` (tous les routeurs du site).

2.2.2 Émission de paquets *multicast*

Pour envoyer un paquet *multicast*, il n'y a rien de particulier à faire — il suffit de stocker une adresse *multicast* dans le champ `sin6_addr` de la destination lors d'un appel à `sendto` ou `sendsg`. Si l'adresse destination est locale au lien (de la forme `ff`*x*`2:`...), il faut prendre soin de stocker un indice d'interface dans le champ `sin6_scope_id`. Remarquez qu'il n'est pas nécessaire d'être abonné à un groupe pour y envoyer des données.

Il existe plusieurs options qui permettent de paramétrer l'émission. La plus importante est l'option `IPV6_MULTICAST_LOOP`, qui spécifie si les paquets émis sont reçus par l'émetteur si celui-ci est abonné au groupe ; il est souvent utile de la mettre à 0 pour éviter de recevoir ses propres paquets.

L'option `IPV6_MULTICAST_HOPS` permet de limiter la portée des paquets émis. Son paramètre spécifie le nombre de routeurs qu'un paquet va traverser avant d'être éliminé ; en particulier, s'il vaut 1, le communication est restreinte au lien local. Cette option n'est évidemment utile que pour un groupe de portée strictement supérieure à 2.

3. Ce qui est complètement idiot — il n'y a aucune raison de coordonner l'affectation des adresses *multicast* dans un espace de 120 bits, mais les gens adorent la bureaucracie. Voyez `https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml`.

4. Même si personne ne sait exactement ce qu'est un site.

2.2.3 Réception de paquets *multicast*

Pour recevoir un paquet destiné à un groupe *multicast*, il faut non seulement écouter sur une *socket* liée au bon port, mais aussi être abonné. On s'abonne une interface à un groupe à l'aide de l'option `IPV6_JOIN_GROUP`, qui prend en paramètre une structure `ipv6_mreq`.

```
struct ipv6_mreq mreq;
unsigned char addr[16];
int ifindex;

memset(&mreq, 0, sizeof(mreq));
memcpy(&mreq.ipv6mr_multiaddr, addr, 16);
mreq.ipv6mr_interface = ifindex;
rc = setsockopt(s, IPPROTO_IPV6, IPV6_JOIN_GROUP,
                &mreq, sizeof(mreq));
if(rc < 0) ...
```

Le champ `ipv6mr_multiaddr` contient l'adresse du groupe *multicast* auquel on veut s'abonner. Le champ `ipv6mr_interface` contient l'indice de l'interface qu'on veut abonner (il est possible d'abonner plusieurs interfaces au même groupe).