Lire des paquets capturés sur le réseau en C

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 décembre 2008

https://www.bortzmeyer.org/libpcap-c.html

Il y a des tas de raisons de vouloir lire soi-même les paquets capturés sur le réseau. Cela peut être par curiosité, par souci de sécurité (ou pour pirater !) ou bien pour faire des statistiques. Pour un rapide coup d'œil, on utilise en général tcpdump, pour un examen interactif détaillé, l'excellent Wireshark mais, pour des études à long terme, il vaut mieux programmer et développer un programme d'analyse spécifique, ce que je fais ici en langage C.

Souvent les articles sur la question parlent ensemble de la capture des paquets et de leur analyse. Ici, je ne mentionne que l'analyse, supposant que la capture a été faite par d'autres moyens <https://www.bortzmeyer.org/capture-paquets.html>, par exemple `tcpdump -w FICHIER` ou bien `pcapdump`.

L'outil d'analyse le plus fréquent pour les programmeurs C est libpcap. Il existe un excellent tutoriel d'introduction <http://www.tcpdump.org/pcap.htm>.

Je commence par un programme trivial, qui lit un fichier au format pcap, capturé par un des outils cités dans « Capturer les paquets qui passent sur le réseau <https://www.bortzmeyer.org/capture-paquets.html> », et qui décode suffisamment d'IPv4 pour afficher le protocole de transport utilisé. L'essentiel du programme est :

```
/* Ouvrir le fichier pcap */
handle = pcap_open_offline(filename, errbuf);
/* Parcourir le fichier */
while (1) {
    packet = pcap_next(handle, &header);
    /* Décoder l'en-tête Ethernet en convertissant le packet en une
    struct */
    ethernet = (struct sniff_ethernet *) (packet);
    /* Si c'est de l'IPv4 */
    if (ntohs(ethernet->ether_type) == IPv4_ETHERTYPE) {
          /* Décoder IPv4 : convertir le paquet en une struct. Ne pas
  oublier de décaler de SIZE_ETHERNET octets */
          ip = (struct sniff_ip *) (packet + SIZE_ETHERNET);
          /* Afficher */
          printf ("Paquet IPv4 avec le protocole %d\n", ip->ip_p);
```

Notons bien qu'il faut utiliser `ntohs` pour convertir le paquet dans la bonne boutianité. Le code complet du programme est disponible en (en ligne sur `https://www.bortzmeyer.org/files/readfile-ipv4.c`). On peut le compiler sur Unix avec :

```
cc -o readfile -lpcap readfile.c
```

(Si libpcap est bien présente. Sur une Debian, il aura fallu installer le paquetage `libpcap-dev`.)

On note qu'il a fallu faire le décodage soi-même, en travaillant au niveaux des bits de l'en-tête du paquet IP. Cela nécessite donc de connaitre le protocole décodé. Pour IPv4, il faut donc lire le RFC 791 [1], section 3.1.

Et pour IPv6 ? Le format est décrit dans le RFC 2460, section 3. Traduisons-le en C :

```
/* IPv6 header. RFC 2460, section3.
   Reading /usr/include/netinet/ip6.h is interesting */
struct sniff_ip {
uint32_t         ip_vtcfl;   /* version then traffic class
                                and flow label */
uint16_t         ip_len;   /* payload length */
uint8_t          ip_nxt;   /* next header (protocol) */
uint8_t          ip_hopl;   /* hop limit (ttl) */
struct in6_addr  ip_src, ip_dst;  /* source and dest address */
};
```

Il suffit ensuite de « plaquer » cette définition sur le paquet :

```
if (ntohs(ethernet->ether_type) == IPv6_ETHERTYPE) {
ip = (struct sniff_ip *) (packet + SIZE_ETHERNET);
```

et hop, on a un paquet IPv6 à analyser. On peut par exemple lire `ip->ip_nxt` qui contient le type du prochain en-tête (en général, la couche transport, mais attention, avec IPv6, d'autres en-têtes intermédiaires peuvent être présents <`https://www.bortzmeyer.org/analyse-pcap-ipv6.html`>). Le programme complet est en (en ligne sur `https://www.bortzmeyer.org/files/readfile-ipv6.c`).

Si on veut décoder des protocoles de plus haut niveau, c'est souvent plus complexe. Prenons le DNS comme exemple. Le format est décrit dans le RFC 1035, section 4.1. Il peut se traduire en :

```
struct sniff_dns {
/* RFC 1035, section 4.1.1 */
uint16_t         query_id;
uint16_t         codes;
uint16_t         qdcount, ancount, nscount, arcount;
};
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc791.txt`

(Cela ne contient que l'en-tête DNS, il existe également des champs ultérieurs pour stocker la requête, la réponse, etc.)

Les codes sont des bits ou des groupes de bits et, pour y accéder, il faut faire de l'arithmétique de bits. Je définis des macros pour cela :

```
#define DNS_QR(dns) ((ntohs(dns->codes) & 0x8000) >> 15)
#define DNS_OPCODE(dns) ((ntohs(dns->codes) >> 11) & 0x000F)
#define DNS_RCODE(dns) (ntohs(dns->codes) & 0x000F)
```

Ainsi, `DNS_QR` va extraire le bit 15, qui indique si le paquet est une requête ou bien une réponse (on masque avec une valeur où seul ce bit est à 1, 0x8000, puis on décale vers la droite). Une fois le paquet décodé :

```
if (source_port == DNS_PORT || dest_port == DNS_PORT) {
     dns = (struct sniff_dns *) (packet + SIZE_ETHERNET +
                                         size_ip + SIZE_UDP);
```

on peut utiliser ces macros pour afficher le contenu du paquet, par exemple :

```
DNS_QR(dns) == 0 ? "Query" : "Response"
```

Le code complet figure en (en ligne sur `https://www.bortzmeyer.org/files/readfile-ipv6-dns.c`).

Les programmeurs Python peuvent regarder mon article équivalent pour Python <`https://www.bortzmeyer.org/libpcap-python.html`>.