RFC 6956 : ForCES Logical Function Block (LFB) Library

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 29 juin 2013

Date de publication du RFC : Juin 2013

https://www.bortzmeyer.org/6956.html

Le protocole ForCES vise à permettre la création d'équipements réseaux (par exemple des routeurs) en assemblant des pièces d'origines diverses, parlant un protocole commun et pouvant donc interagir. Ce RFC marque l'achèvement de la première (et très longue) phase de ForCES, la normalisation complète de ce protocole. Notre RFC 6956 [1] décrit la bibliothèque standard de ForCES, les parties d'un routeur qu'on trouvera partout et qui assurent les fonctions indispensables comme faire suivre un paquet IP.

ForCES peut servir à bien d'autres choses et il y aura aussi des bibliothèques pour d'autres équipements que les routeurs, et aussi des bibliothèques non-standard, développées par tel ou tel projet. Mais pour atteindre l'objectif premier de ForCES, créer un routeur par assemblage de pièces standards, cette bibliothèque « routeur » était le dernier élement manquant.

Avant de lire ce RFC 6956, il faut bien se pénétrer du cadre général (RFC 3746), du protocole (RFC 5810) et du modèle de données (RFC 5812). J'en extrais quelques termes qu'il faut connaître :

— NE ou "*Network Element*", l'engin complet (un routeur, un commutateur, etc).
— CE ou "*Control Element*", la partie du NE qui fait tourner les protocoles de contrôle et de signalisation (pour un routeur, OSPF, BGP, etc).
— FE ou "*Forwarding Element*", la partie de l'engin qui traite les paquets (pour un routeur, qui les fait suivre d'une interface à l'autre). CE et FE se parlent en suivant le protocole ForCES. Un NE comprend au moins un CE et un FE.
— LFB ou "*Logical Function Block*", c'est le concept qui fait l'objet de ce RFC. Un LFB est une fonction d'un FE, contrôlée par le CE. Un FE typique met en œuvre plusieurs fonctions (par exemple : faire suivre les paquets, filtrer les paquets, compter les paquets et chacune de ces trois fonctions est représentée par un LFB). Un CE typique va donc utiliser le protocole ForCES pour configurer les LFB (« arrête de faire suivre les paquets », « coupe l'interface `Ethernet/0/2` », etc). Un LFB est une fonction logique : dans la matériel, on ne retrouve pas forcément ce concept.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6956.txt`

— Classe de LFB et instance de LFB sont des concepts empruntés à la programmation objet. L'idée est qu'on définit des classes de LFB et que le LFB dans un FE donné est une instance d'une de ces classes.
— Un « port » dans ce RFC est une interface physique du routeur.

La **bibliothèque** composée des **classes** définies dans ce RFC 6956 permet de réaliser un routeur complet, mettant en œuvre toutes les fonctions obligatoires du RFC 1812, notamment :

— Encapsuler et décapsuler les paquets pour la couche 2 (Ethernet, par exemple),
— Envoyer et recevoir des paquets de taille allant jusqu'à la MTU du lien, fragmenter les autres (pour IPv4),
— Traduire les adresses IP en adresses de la couche 2 (par exemple avec ARP),
— Gérer ICMP, et notamment créer et envoyer les messages d'erreur ICMP en réponse à des problèmes, ou bien lorsque le TTL tombe à zéro,
— Gérer les tampons d'entrée/sortie, la congestion, un juste traitement des différents paquets,
— Trouver le routeur suivant ("*next hop*") lorsqu'il faut transmettre un paquet,
— Être capable de faire tourner un IGP comme OSPF, dans certains cas un EGP comme BGP, et accepter évidemment de simples routes statiques,
— Avoir toutes les fonctions de gestion du réseau qu'on attend : statistiques, débogage, journalisation...

Place maintenant au cœur de ce RFC, les classes qui y sont définies. Le choix du groupe de travail forces `<http://tools.ietf.org/wg/forces>` (section 3.2) a été de privilégier la souplesse en définissant plutôt trop de classes que pas assez. Chaque classe ne va donc effectuer qu'un minimum de fonctions, être logiquement séparée des autres classes et notre routeur devra donc mettre en œuvre beaucoup de classes. Cela rend donc le RFC très long et je ne vais donc pas parler de toutes les classes, seulement de quelques unes choisies comme exemples. Pour la totalité des classes, vous devez lire le RFC.

Les définitions des classes s'appuient sur un certain nombre de types définis en section 4. Le RFC 5812 avait normalisé les types de base comme `uint32`, `char` ou `boolean`. Notre RFC y ajoute quelques types atomiques comme `IPv4Addr`, `IPv6Addr`, `IEEEMAC` (une adresse MAC), `PortStatusType` (permettant d'indiquer si le port est activé ou non et s'il fonctionne), `VlanIDType` pour le numéro de VLAN, etc. Il définit également des types composés comme `MACInStatsType` (une entrée dans le tableau de statistiques pour les paquets entrants, analysés à la couche 2), `IPv6PrefixInfoType` (un préfixe IPv6), `IPv4UcastLPMStatsType` (une entrée dans le tableau de statistiques pour les paquets IPv4), etc. Logiquement, les tableaux qui utilisent ces types sont également définis comme `IPv6PrefixTableType` (un tableau de préfixes IPv6) ou `VlanInputTableType` (un tableau de VLAN).

Un NE manipule des paquets donc il y a évidemment des types de paquets parmi lesquels `EthernetII`, `IPv4`, `IPv6`, et même `Arbitrary` pour des paquets quelconques.

Le tout est formellement décrit en XML dans la section 4.4. Ainsi, une adresse IPv4 est :

```
<dataTypeDef>
         <name>IPv4Addr</name>
         <synopsis>IPv4 address</synopsis>
         <typeRef>byte[4]</typeRef>
</dataTypeDef>
```

et une IPv6 est :

```
<dataTypeDef>
         <name>IPv6Addr</name>
         <synopsis>IPv6 address</synopsis>
         <typeRef>byte[16]</typeRef>
</dataTypeDef>
```

Un type atomique comme la capacité du lien est définie par :

```
<dataTypeDef>
        <name>LANSpeedType</name>
        <synopsis>LAN speed type</synopsis>
        <atomic>
         <baseType>uint32</baseType>
         <specialValues>
           <specialValue value="0x00000001">
            <name>LAN_SPEED_10M</name>
            <synopsis>10M Ethernet</synopsis>
           </specialValue>
           <specialValue value="0x00000002">
            <name>LAN_SPEED_100M</name>
            <synopsis>100M Ethernet</synopsis>
           </specialValue>
...
```

Même chose pour le `PortStatusType` mentionné plus haut :

```
<dataTypeDef>
        <name>PortStatusType</name>
        <synopsis>
          Type for port status, used for both administrative and
          operative status.
        </synopsis>
        <atomic>
         <baseType>uchar</baseType>
         <specialValues>
           <specialValue value="0">
            <name>Disabled</name>
            <synopsis>Port disabled</synopsis>
           </specialValue>
           <specialValue value="1">
            <name>Up</name>
            <synopsis>Port up</synopsis>
           </specialValue>
           <specialValue value="2">
            <name>Down</name>
            <synopsis>Port down</synopsis>
           </specialValue>
         </specialValues>
        </atomic>
      </dataTypeDef>
<dataTypeDef>
```

Et les types structurés ? Un exemple est celui des statistiques sur les paquets sortants, qui a deux composants (les champs des objets), un pour les paquets transmis et un pour les paquets jetés :

```
<dataTypeDef>
         <name>MACOutStatsType</name>
         <synopsis>
           Data type defined for statistics in EtherMACOut LFB.
         </synopsis>
         <struct>
            <component componentID="1">
               <name>NumPacketsTransmitted</name>
               <synopsis>Number of packets transmitted</synopsis>
               <typeRef>uint64</typeRef>
            </component>
            <component componentID="2">
               <name>NumPacketsDropped</name>
               <synopsis>Number of packets dropped</synopsis>
               <typeRef>uint64</typeRef>
            </component>
         </struct>
</dataTypeDef>
```

Un autre exemple est le préfixe IPv6, qui réutilise le type atomique `IPv6Addr` :

```
<dataTypeDef>
         <name>IPv6PrefixInfoType</name>
         <synopsis>Data type for entry of IPv6 longest prefix match
          table in IPv6UcastLPM LFB. The destination IPv6 address
          of every input packet is used as a search key to look up
          the table to find out a next hop selector.</synopsis>
         <struct>
            <component componentID="1">
               <name>IPv6Address</name>
               <synopsis>The destination IPv6 address</synopsis>
               <typeRef>IPv6Addr</typeRef>
            </component>
            <component componentID="2">
               <name>Prefixlen</name>
               <synopsis>The prefix length</synopsis>
               <atomic>
                  <baseType>uchar</baseType>
                  <rangeRestriction>
                     <allowedRange min="0" max="32"/>
                  </rangeRestriction>
               </atomic>
            </component>
...
```

Une fois tous ces types définis (et cela prend un bon bout du RFC), on peut les utiliser pour bâtir les classes (section 5). Chaque classe va être un gabarit dont on tirera les instances. Premier exemple : notre routeur ForCES va évidemment avoir besoin de connaître Ethernet et de traiter des paquets Ethernet, en envoi et en réception. Ethernet étant très varié, commençons par un LFB (”*Logical Function Block*”) qui représente le niveau physique, `EtherPHYCop`. Ce LFB (cette partie logique d'un routeur) a des composants pour indiquer l'état du port (`AdminStatus` et `OperStatus`), composants qui peuvent être lus et modifiés via le protocole ForCES. Ce LFB peut aussi générer des événements comme par exemple le changement d'état d'un port (`PHYPortStatusChanged`).

Au niveau au dessus (couche 2), il y a le LFB `EtherMACIn`. C'est par exemple là que se trouve le composant `LocalMACAddresses` qui indique l'adresse MAC. ou bien le composant `PromiscuousMode` qui dit si on veut recevoir (ou pas) tous les paquets, même adressés à une autre machine.

Rappelez-vous : cette bibliothèque standard de ForCES privilégie la souplesse, et crée donc **beaucoup** de LFB, pour pouvoir les arranger comme souhaité. Rien que pour Ethernet, il y en a encore plusieurs comme `EtherEncap` qui modélise les fonctions d'encapsulation et de décapsulation des trames (ajout ou retrait des en-têtes Ethernet).

Mais on ne peut pas passer toute la journée sur les couches basses. Pour « construire » un routeur par assemblage de LFB, il nous faut maintenant des fonctions de manipulation d'IP. C'est le cas du LFB `IPv6Validator` qui représente les fonctions de validation d'un paquet (avant qu'on décide de le traiter, ou de le jeter s'il est invalide). On suit le RFC 2460 et on considère comme invalide (ne devant pas être transmis) les paquets ayant une taile inférieure à la taille minimale, les paquets ayant une adresse source ou destination inacceptable, etc. Il y a aussi des paquets nommés « exceptionnels » qui, sans être à proprement parler invalides, ne pourront pas être transmis normalement (par exemple ceux avec un "*hop limit*" descendu à zéro).

Et, ensuite, le travail sera fait par les LFB de transmission. Oui, ils sont plusieurs. Pour IPv6, au moins `IPv6UcastLPM` (trouver le LPM, "*Longest Prefix Match*", dans la table) et `IPv6NextHop` (sélection du routeur suivant), sans compter des futurs LFB optionnels, par exemple pour mettre en œuvre RPF.

Les LFB sont eux aussi formalisés en XML (section 6). Par exemple, le dernier que nous avons vu, `IPv6NextHop`, sera :

```
<LFBClassDef LFBClassID="13">
   <name>IPv6NextHop</name>
   <synopsis>
     The LFB abstracts the process of next hop information
     application to IPv6 packets. It receives an IPv6 packet
     with an associated next hop identifier (HopSelector),
     uses the identifier as a table index to look up a next hop
     table to find an appropriate output port.
   </synopsis>
   <inputPorts>
      <inputPort group="false">
         <name>PktsIn</name>
         <synopsis>
           A port for input of unicast IPv6 packets, along with
           a HopSelector metadata.
          </synopsis>
        ...
      </inputPort>
   </inputPorts>
   <outputPorts>
      ...
   </outputPorts>
   <components>
      <component componentID="1">
         <name>IPv6NextHopTable</name>
         <synopsis>
           The IPv6NextHopTable component. A HopSelector is used
           to match the table index to find out a row which
           contains the next hop information result.
         </synopsis>
         <typeRef>IPv6NextHopTableType</typeRef>
      </component>
   </components>
</LFBClassDef>
```

Si la tête commence à vous tourner sérieusement, c'est sans doute normal et prévu. Les auteurs du RFC ont pensé à vous et inclus une section 7 qui décrit comment on crée une fonction de haut niveau à partir des LFB de cette bibiothèque standard. Leur premier exemple est la transmission de paquets IPv4 et le dessin ne compte pas moins de treize LFB... (Dont, il est vrai, neuf juste pour Ethernet.) Pour ajouter ARP, cela prend sept LFB (certains sont communs par exemple `EtherEncap` est utilisé à la fois par la transmission de paquets IPv4 et par ARP).

Les classes des LFB sont désormais dans un registre IANA <`https://www.iana.org/assignments/forces/forces.xml#logical-types`>, ce qui permettra d'en ajouter plus facilement d'autres.

Voilà, arrivé à ce stade, on a de quoi « fabriquer » un routeur ou plus exactement de quoi le modéliser. En pratique, les vrais routeurs seront bien différents mais ils exporteront aux clients ForCES des fonctions représentées par ces LFB standards, fonctions qu'on pourra configurer avec Forces.