RFC 6143 : The Remote Framebuffer Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 mars 2011

Date de publication du RFC : Mars 2011

https://www.bortzmeyer.org/6143.html

Il est assez fréquent à l'IETF qu'un protocole ne soit documenté dans un RFC que longtemps après son déploiement. C'est le cas de RFB ("*Remote Framebuffer Protocol*"), le protocole utilisé dans le système VNC, système qui permet de transmettre via un réseau TCP/IP l'interface graphique d'une machine. VNC permet, par exemple, d'avoir sur son ordinateur le bureau d'une machine distante, se s'y loguer, etc, comme si on était physiquement devant. C'est donc un concurrent de X11 avec XDMCP ou bien de NX mais avec l'avantage sur ce dernier d'avoir désormais une spécification publique, et de nombreuses mises en œuvre. Je compte une quarantaine de clients, de serveurs et de bibliothèques VNC sur mon Ubuntu.

RFB se veut un protocole simple, bien plus simple que X11, basé sur l'idée d'un **serveur**, tournant sur la machine distante, qui écrit dans une mémoire graphique (le "*framebuffer*") qui est en fait située sur la machine de bureau (le **client** ; notez que « client » et « serveur » ont des sens différents de celui de X11). Le système VNC, qui utilise ce protocole, est très largement utilisé pour l'accès distant à une machine, exactement dans les mêmes conditions que si on était présent devant cette machine (système de fenêtrage, accès à la souris, etc).

RFB travaille au niveau de la mémoire graphique et est donc compatible avec tous les systèmes de fenêtrage comme X11 ou comme celui de Windows. Le protocole se veut simple, surtout côté client, pour pouvoir être mis en œuvre sur de petites machines. Il existe même un "*widget*" VNC/RFB pour GTK, pour ceux qui veulent mettre un client VNC dans leurs programmes.

Le client est sans état et peut donc redémarrer à tout moment et reprendre la session là où il l'avait laissée. C'est le serveur qui maintient l'état de l'interface utilisateur. Cela permet par exemple de changer de client en cours de session.

RFB, je l'ai dit, a été implémenté avant d'être spécifié et a connu plusieurs versions. Ce RFC essaie donc de documenter l'état actuel du protocole.

Si les détails prennent parfois de la place, les principes sont simples et les premières sections du RFC sont donc courtes. La section 2 décrit les connexions entre le client et le serveur. Le mode normal de fonctionnement de VNC est que le serveur tourne pendant une longue période, gardant en mémoire l'état de la mémoire graphique, le ou les clients se connectant à loisir, pendant parfois seulement de courtes périodes. Cela ressemble donc, adapté au graphique, au principe de screen <https://www.bortzmeyer.org/screen.html>. Le client peut être n'importe où sur l'Internet et se connecte par défaut au port 5900.

Loin de la richesse du protocole de X11, RFB ne fournit qu'une seule primitive : « afficher un rectangle à la position X,Y indiquée » (section 3). Cela peut sembler très inefficace mais divers encodages subtils permettent de rendre ce protocole supportable. Les mises à jour (nouveaux rectangles à afficher) ne sont jamais envoyées par le serveur mais demandées par le client (message `FramebufferUpdateRequest`, section 7.5.3). Le protocole est donc adaptatif : un client lent ne demandera pas souvent des mises à jour et aura donc moins de travail. Si une même région de la mémoire graphique voit plusieurs modifications successives, le client lent peut donc parfaitement sauter les étapes intermédiaires.

Afficher des rectangles, c'est très joli mais un environnement graphique digne de ce nom doit aussi permettre à l'utilisateur de dire quelque chose, par exemple de taper au clavier ou de cliquer avec la souris. La section 4 résume le protocole d'entrée de RFB : ce dernier ne connait que clavier et souris (multi-boutons). Tout autre périphérique d'entrée doit donc être modélisé comme clavier ou comme souris.

Et les pixels, comment sont-ils modélisés ? La section 5 décrit le modèle. Il repose sur une négociation entre client et serveur, à l'ouverture de la connexion, où le client est roi. C'est lui qui choisit la représentation des pixels sur le réseau, et il peut donc choisir ce qui lui fait le moins de travail, tout reposant alors sur le serveur. Les deux formats les plus classiques sont le « couleurs réelles » où chaque pixel est un nombre sur 24 bits, chaque groupe de 8 bits indiquant l'intensité de rouge, de vert ou de bleu, ou bien le « table des couleurs » où chaque pixel est un nombre sur 8 bits indiquant une entrée d'une table où sont stockées les 256 couleurs possibles.

Plusieurs encodages sont ensuite possibles dans RFB, six étant définis par notre RFC 6143 [1], permettant plus ou moins de compression.

En parlant de négociation, comme RFB existe depuis un certain temps et a connu plusieurs versions, la section 6 décrit la notion de version du protocole. Notre RFC 6143 normalise la version 3.8. Les versions précédentes résumées dans l'annexe A. Si une version 4.x apparaîtra peut-être un jour, la version actuelle, la 3.8, est extensible sans avoir besoin de changer de version. On peut ainsi introduire :

— De nouveaux encodages en plus des six indiqués plus haut,
— Des nouvelles fonctions du protocole (appelées « pseudo-encodages » car, dans le processus de négociation initial, elles sont représentées comme des encodages),
— Des nouveaux mécanismes de sécurité.

Le protocole RFB lui-même figure dans la section 7. Elle est très longue, bien plus que les autres sections, mais pas forcément plus difficile, elle liste juste tous les messages possibles. RFB peut fonctionner sur n'importe quel protocole de transport fiable (comme TCP). La communication commence par une phase de négociation entre le client et le serveur, puis l'initialisation, puis l'envoi des messages de mise à jour de la mémoire graphique. La description des messages utilise quelques types de base comme U8 (entier de huits bits non signé), S16 (entier de seize bits signé, toujours en gros boutien), etc.

La section 7.1 décrit le protocole de négociation. Ainsi, le serveur commence par envoyer un message où il indique la version (message `ProtocolVersion`, ici, le serveur à l'autre bout est x11vnc) :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6143.txt`

```
% nc ec2-xxx.eu-west-1.compute.amazonaws.com 5900
RFB 003.008
```

La sécurité se négocie ensuite (section 7.2). La phase d'initialisation est décrite en section 7.3. Par exemple, le message `ServerInit` contiendra la hauteur et la largeur de la mémoire graphique en nombre de pixels (tous les deux en UA16), ainsi que le format de représentation des pixels (décrit en section 7.4).

La représentation des caractères tapés au clavier du client (message `KeyEvent`, section 7.5.4) est complexe. La valeur utilisée est un `keysym` de X11 (même si ni le client, ni le serveur n'utilisent X11). Si vous avez une machine Unix où X11 est installé, le fichier `/usr/include/X11/keysymdef.h` vous donnera une idée des valeurs possibles. Cela aurait été plus simple si un point de code Unicode était envoyé... Mais il n'y a que pour ASCII que le `keysym` correspond au point de code. Le RFC va même jusqu'à préciser qu'il vaut mieux utiliser le `keysymI` traditionnel, même s'il y en a un pour le caractère Unicode. l faut dire que le protocole RFB ne transmet pas que des caractères mais aussi des états des touches (comme le fait que la touche Majuscule est pressée) et des touches non affectées à un caractère particulier (comme F1, F2, etc). Les règles d'interprétation des `keysyms` sont d'une grande complexité. Le lecteur curieux doit se munir d'aspirine avant de lire lui-même cette section.

Plus facile, le cas du « pointeur » (la souris) en section 7.5.5. Les événements sont simplement « bouton pressé » et « bouton relaché », les attributs étant la position X,Y et le numéro du bouton (huit boutons sont prévus, afin de pouvoir modéliser des dispositifs d'entrée plus complexes comme les roulettes).

Jusqu'à présent, je n'ai parlé que des messages envoyés par le client RFB au serveur. En sens inverse, le principal message est `FramebufferUpdate` (section 7.6.1) qui indique un changement dans la mémoire graphique. Il contient un certain nombre de rectangles et, pour chacun d'eux, sa position X,Y, sa hauteur et sa largeur, et le type d'encodage utilisé.

Les encodages sont définis en section 7.7. Les six standards à l'heure actuelle sont :

- `Raw` : comme son nom l'indique, aucun effort, on envoie juste les pixels l'un après l'autre (largeur, puis hauteur).
- `CopyRect` : le serveur indique les coordonnées d'un rectangle existant dans la mémoire du client. Cela sert, par exemple, lorsqu'une fenêtre est déplacée sur l'écran.
- `RRE encoding` : officiellement abandonné.
- `Hextile encoding` : officiellement abandonné.
- `TRLE encoding` : "*Tiled Run-Length Encoding*" utilise un ensemble de techniques de compression dont le "*run-length encoding*" appliquée à des « tuiles » de 16x16 pixels.
- `ZRLE encoding` : "*Zlib Run-LengthEncoding*" utilise la compression Zlib (RFC 1950 et RFC 1951).

La normalisation du "*Remote Framebuffer Protocol*" a nécessité quelques nouveaux registres IANA, décrits en section 8 : celui des types de sécurité `<https://www.iana.org/assignments/rfb/rfb.xml#rfb-1>`, celui des messages RFB `<https://www.iana.org/assignments/rfb/rfb.xml#rfb-2>`, et celui des encodages `<https://www.iana.org/assignments/rfb/rfb.xml#rfb-4>`, qui permettra d'étendre la liste de six encodages présentée plus haut.

Quelle est la sécurité de RFB ? Par défaut, inexistante : comme le note justement la section 9, l'authentification est faible et la confidentialité et l'intégrité sont nulles. RFB est traditionnellement utilisé avec une protection fournie par, par exemple, un VPN chiffré, seule façon de l'utiliser de manière sûre.

L'annexe A du RFC résume les différences entre cette version du protocole (la 3.8) et les précédentes. Ces différences sont concentrées dans la phase de négociation.

À l'heure d'aujourd'hui, je n'ai pas trouvé de connaissance de ce protocole dans tcpdump mais Wireshark, par contre, le décode `<http://wiki.wireshark.org/VNC>`, sous son ancien nom de VNC (qui est désormais uniquement le nom d'un logiciel). C'est aussi le nom à utiliser sur pcapr `<https://www.bortzmeyer.org/pcapr.html>` : `<http://www.pcapr.net/browse?proto=vnc&order=date>`.