RFC 8272 : TinyIPFIX for Smart Meters in Constrained Networks

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 15 novembre 2017

Date de publication du RFC : Novembre 2017

https://www.bortzmeyer.org/8272.html

Le format IPFIX, normalisé dans le RFC 7011 [1], permet à un équipement réseau d'envoyer des données résumées à un collecteur, à des fins d'études ou de supervision. À l'origine, l'idée était que l'équipement réseau soit un routeur, une machine relativement grosse, munie de suffisamment de ressources pour pouvoir s'accomoder d'un protocole un peu compliqué, et qui nécessite l'envoi de pas mal d'octets. L'IPFIX original était donc peu adapté aux engins contraints, par exemple aux capteurs connectés. D'où cette variante d'IPFIX, TinyIPFIX, qui vise à être utilisable par des objets connectés comme ceux utilisant le 6LoWPAN du RFC 4944 (un compteur Linky ?)

Mais prenons plutôt comme exemple un capteur non connecté au réseau électrique (donc dépendant d'une batterie, qu'on ne peut pas recharger tout le temps, par exemple parce que le capteur est dans un lieu difficile d'accès) et n'ayant comme connexion au réseau que le WiFi. L'émission radio coûte cher en terme d'énergie et notre capteur va donc souvent éteindre sa liaison WiFi, sauf quand il a explicitement quelque chose à transmettre. Un protocole de type "*pull*" ne peut donc pas convenir, il faut du "*push*", que le capteur envoie ses données quand il le décide. Ces contraintes sont détaillées dans « "*Applications for Wireless Sensor Networks*" », par l'auteure du RFC (pas trouvé en ligne, c'est publié dans le livre « "*Handbook of Research on P2P and Grid Systems for Service-Oriented Computing : Models, Methodologies and Applications*" », édité par Antonopoulos N. ; Exarchakos G. ; Li M. ; Liotta A. chez "*Information Science Publishing*").

Le RFC donne (section 3) l'exemple de l'IRIS de Crossbow <http://www.xbow.com> : sa taille n'est que de 58 x 32 x 7 mm, et il a 128 ko de flash pour les programmes (512 ko pour les données mesurées), 8 ko de RAM et 4 d'EEPROM pour sa configuration. On ne peut pas demander des miracles à un objet aussi contraint. (Si c'est vraiment trop peu, le RFC cite aussi l'engin d'Advantic <https:

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7011.txt

//www.advanticsys.com/> avec ses 48 ko de flash « programme », 1024 ko de flash « données », 10 ko de RAM et 16 d'EEPROM.) Question énergie, ce n'est pas mieux, deux piles AA de 2 800 mAh chacune peuvent donner en tout 30 240 J.

Autre contrainte vécue par ces pauvres objets connectés, les limites du protocole réseau (section 3.3 de notre RFC). 6LoWPAN (RFC 4944) utilise IEEE 802.15.4. Ce protocole ne porte que 102 octets par trame. Ce n'est pas assez pour IPv6, qui veut une MTU minimum de 1 280 octets. Il faut donc utiliser la fragmentation, un mécanisme problématique, notamment parce que, si un seul fragment est perdu (et ces pertes sont des réalités, sur les liaisons radio), il faut retransmettre tout le paquet. Il est donc prudent de s'en tenir à des paquets assez petits pour tenir dans une trame. C'est un des apports de TinyIPFIX par rapport au IPFIX classique : des messages plus petits.

Enfin, dernière contrainte, le protocole de transport. IPFIX impose (RFC 7011, section 10.1) que SCTP soit disponible, même s'il permet aussi UDP et TCP. Mais SCTP (et TCP) ne permettent pas d'utiliser les mécanismes de compression des en-têtes de 6LoWPAN. Et SCTP n'est pas toujours présent dans les systèmes d'exploitation des objets, par exemple TinyOS. TinyIPFIX utilise donc UDP. À noter que, comme le demande la section 6.2 du du RFC 5153, TinyIPFIX sur UDP n'est pas prévu pour l'Internet ouvert, mais uniquement pour des réseaux fermés.

TinyIPFIX est dérivé de IPFIX (RFC 7011) et en hérite donc de la plupart des concepts, comme la séparation des données ("*Data Sets*") et de la description des données (dans des gabarits, transmis en "*Template Sets*").

La section 4 du RFC décrit des scénarios d'usage. Comme TinyIPFIX (comme IPFIX) est unidirectionnel (de l'exporteur vers le collecteur), et qu'il tourne sur UDP (où les messages peuvent se perdre), le développeur doit être conscient des limites de ce service. Si on perd un paquet de données, on perd des données. Pire, si on perd un paquet de gabarit (RFC 7011, sections 3.4.1 et 8), on ne pourra plus décoder les paquets de données suivants. On ne doit donc pas utiliser TinyIPFIX pour des systèmes où la perte de données serait critique. Un système d'accusés de réception et de retransmission (refaire une partie de TCP, quoi…) serait trop lourd pour ces engins contraints (il faudrait stocker les messages en attendant l'accusé de réception).

Le RFC recommande de renvoyer les paquets de gabarit de temps en temps. C'est normalement inutile (on n'imagine pas un capteur contraint en ressources changer de gabarit), mais cela permet de compenser le risque de perte. Le collecteur qui, lui, n'a pas de contraintes, a tout intérêt à enregistrer tous les messages, même quand il n'y a pas de gabarit, de manière à pouvoir les décoder quand le gabarit arrivera. (Normalement, on ne fait pas ça avec IPFIX, le gabarit ne peut s'appliquer qu'aux messages reçus après, mais, avec TinyIPFIX, il y a peu de chances que les gabarits changent.)

Le RFC donne un exemple animalier qui conviendrait au déploiement de TinyIPFIX, afin de surveiller des oiseaux (Szewczyk, R., Mainwaring, A., Polastre, J., et D. Culler, « " *An analysis of a large scale habitat monitoring application*" <https://people.eecs.berkeley.edu/~culler/papers/sensys04.pdf> ».) Les capteurs notent ce que font ces charmants animaux et le transmettent.

Cet exemple sert à illustrer un cas où TinyIPFIX serait bien adapté : collecte de type "*push*", efficacité en terme de nombre de paquets, perte de paquets non critique, pas de nécessité d'un estampillage temporel des messages (qui existe dans IPFIX mais que TinyIPFIX supprime pour alléger le travail).

La section 5 décrit l'architecture de TinyIPFIX, très similaire à celle d'IPFIX (RFC 5470).

Enfin, la section 6 décrit les aspects concrets de TinyIPFIX, notamment le format des messages. Il ressemble beaucoup à celui d'IPFIX, avec quelques optimisations pour réduire la taille des messages. Ainsi, l'en-tête de message IPFIX fait toujours 16 octets, alors que dans TinyIPFIX, il est de taille variable, avec seulement 3 octets dans le meilleur des cas. C'est ainsi que des champs comme le numéro de version (qui valait 11 pour IPFIX) ont été retirés. De même, l'estampille temporelle (« "*Export Time*" » dans IPFIX) est partie (de toute façon, les objets contraints ont rarement une horloge correcte).

Les objets contraints déployés sur le terrain n'ont souvent pas à un accès direct à Internet, à la fois pour des raisons de sécurité, et parce qu'un TCP/IP complet serait trop lourd pour eux. Il est donc fréquent qu'ils doivent passer par des relais qui ont, eux, un vrai TCP/IP, voire un accès Internet. (Cette particularité des déploiements d'objets connectés est une des raisons pour lesquelles le terme d'« Internet des Objets » n'a pas de valeur autre que marketing.)

TinyIPFIX va donc fonctionner dans ce type d'environnement et la section 7 de notre RFC décrit donc le mécanisme d'« intermédiation ». L'intermédiaire peut, par exemple, transformer du TinyIPFIX/UDP en TinyIPFIX/SCTP ou, carrément, du TinyIPFIX en IPFIX. Dans ce dernier cas, il devra ajouter les informations manquantes, comme l'estampille temporelle ou bien le numéro de version.

Côté mise en œuvre, Tiny IPFIX a été mis dans TinyOS et Contiki. Voir <http://www.net.in.tum.de/pub/cs/TinyIPFIX_GUI_Licenced.zip> et <http://www.net.in.tum.de/pub/cs/TinyIPFIX_and_Extentions_Licenced.zip>.