RFC 6256 : Using Self-Delimiting Numeric Values in Protocols

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 26 mai 2011

Date de publication du RFC : Mai 2011

https://www.bortzmeyer.org/6256.html

Comment encoder des valeurs dans un protocole ? Il existe des tas de méthodes mais celle de ce RFC 6256 [1], les « valeurs numériques auto-terminées » (SDNV pour "*Self-Delimiting Numeric Values*") sont encore très peu connues et guère utilisées.

Soit un concepteur de protocoles réseau. Il doit indiquer dans les données qui circulent des valeurs, par exemple un gâteau qui identifie une session ou bien une durée ou une longueur ou bien une clé cryptographique ou des tas de choses encore. Comment transmettre ces valeurs (section 1.1) ? Une technique courante, très utilisée dans les protocoles « binaires » comme IP (RFC 791) ou bien le DNS (RFC 1035) est le **champ de taille fixe**. On décide que la longueur sera représentée sur 16 bits et les producteurs et les lecteurs de ce champ connaissent cette taille, qu'ils ont lu dans le RFC correspondant. Plus la taille est grande, plus on consomme de ressources réseaux pour la transmettre mais, plus elle est petite, et moins on pourra représenter de valeurs. Dans mon exemple, que fera t-on si les changements dans les techniques ou dans les usages nécessitent de stocker des longueurs supérieurs à 65 536 ? Le problème est fréquent car des protocoles sont souvent utilisés pour des dizaines d'années consécutives, années pendant lesquelles les hypothèses de base ont changé. Le RFC cite ainsi les exemples de la fenêtre TCP (RFC 793, la fenêtre est limitée à 16 bits, soit 65 kilo-octets, ce qui est bien trop peu pour les réseaux à 10 Gb/s d'aujourd'hui, et a nécessité le "*hack*" de l'échelle de fenêtre du RFC 1323, désormais en RFC 7323) et du bien connu champ Adresse d'IPv4, qui s'est montré trop petit <https://www.bortzmeyer.org/epuisement-adresses-ipv4.html>. Le RFC note bien que les deux décisions avaient semblé raisonnables à l'époque : sur une ligne spécialisé à 64 kb/s, la fenêtre maximale était de huit secondes, ce qui donnait largement aux accusés de réception le temps de revenir. Aujourd'hui, la même taille ne laisse que 50 microsecondes... Même problème pour les adresses IPv4, épuisées par le changement du modèle de l'ordinateur (de « un par entreprise » à « plusieurs par

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6256.txt

personnes ») et par le succès inespéré de l'Internet. Les SDNV permettent donc d'éviter de prendre des décisions (« 16 bits ? 32 bits ? Allons-y pour 16 bits, ça suffira bien. ») qu'on regrettera ensuite.

Une autre approche, commune à l'IETF, est d'utiliser TLV. C'est le mécanisme le plus souple, mais, pour des valeurs assez petites, il prend plus de place que SDNV.

Le groupe de recherche IRTF DTNRG <http://www.dtnrg.org/>, qui est occupé notamment à développer des protocoles réseau pour les vaisseaux spatiaux (cf. RFC 4838) a son idée sur la question (la section 4 présente une discussion des alternatives), et cela a donné ce RFC. Par exemple, dans l'espace, vu le RTT, les protocoles qui négocient des paramètres au début de la connexion (comme l'échelle de fenêtre TCP), et qui permettent ainsi d'avoir des champs de taille fixe et de choisir la taille au début de la connexion, ne sont pas envisageables. Mais le problème n'est pas spécifique aux réseaux « DTN » ("*Delay-Tolerant Networking*", les réseaux où l'acheminement d'un bit prend un temps fou, ce qui est le cas dans l'espace où la vitesse de la lumière est une limite gênante, cf. RFC 5325) et on peut retrouver des ancêtres de SDNV dans des encodages ASN.1 ou bien dans certains protocoles UIT. Par contre, en dehors du projet DTN, il ne semble pas que les SDNV aient été utilisés par des protocoles IETF, peut-être parce que leur encodage et décodage est plus lent qu'avec des champs de taille fixe (le protocole WebSocket utilise les SDNV). À noter que notre RFC n'est pas la premier à normaliser SDNV, ceux-ci avaient déjà été décrits dans les RFC 5050 et RFC 5326 mais de manière sommaire. Un description plus complète n'était donc pas du luxe.

SDNV se limite à représenter des entiers positifs ou des chaînes de bits. Pour des types plus complexes, on peut utiliser les encodages d'ASN.1 ou bien MSDTP (décrit dans le RFC 713).

Après ces préliminaires, attaquons-nous à la définition d'un SDNV (section 2). Un SDNV est encodé sous forme d'une série d'octets en n'utilisant que sept bits de chaque octet. Le huitième indique si l'octet est le dernier du SDNV. Ce mécanisme impose un décodage strictement séquentiel, en examinant un octet après l'autre. Quelques exemples :

— Le nombre 3 est représenté par 00000011. Le premier bit, ici à zéro, indique que ce premier octet est aussi le dernier.
— Le nombre 130 est représenté par 1000001 00000010. Le premier bit du premier octet indique que cet octet n'est pas le dernier. Le dernier bit du premier octet est celui qui code 128. (On ne peut pas utiliser le premier bit du deuxième octet, réservé pour indiquer la fin de la série des octets.)

L'annexe A contient d'autres exemples et j'en montre certains à la fin de cet article. Il y a quelques détails pratiques comme les zéros initiaux : le premier nombre est-il 3 où 03 ? Le RFC explique que les zéros initiaux ne sont pas significatifs. Pour les nombres, cela ne change évidemment rien (3 = 03) mais si on voulait représenter des chaînes de bits, c'est plus compliqué et SDNV ne fournit pas de solution satisfaisante (le RFC suggère par exemple d'indiquer explicitement la longueur dans un autre champ, ce qui annule l'intérêt de SDNV ; une autre suggestion est de toujours mettre un bit à 1 au début de toute chaîne).

Les SDNV ont parfois été décrits sous le nom de "*Variable Byte Encoding*" (cf. le livre de Manning, Raghavan, et Schuetze, « "*Introduction to Information Retrieval*" <http://informationretrieval.org/> », qui s'intéressait toutefois à la compression plutôt qu'à la souplesse). On peut remarquer qu'aucune valeur encodée en SDNV n'est un préfixe d'une valeur encodée en SDNV (SDNV est "*prefix-free*", autre façon de dire que les données sont auto-terminées).

Comment encode et décode-t-on ces SDNV ? La section 3 décrit quelques algorithmes qui peuvent être utilisés à cette fin. La section 3.1 contient un algorithme d'encodage très simple en O(log(N)) où N est le nombre à encoder. Le décodage est plus sioux : on ne connait en effet pas la valeur du nombre avant de l'avoir décodé et on ne sait donc pas quel type on doit utiliser pour stocker le résultat. Même un type de grande taille, genre `uint64_t` peut être trop petit. La section 3.2 propose un algorithme

de décodage récursif, qui évalue à chaque étape si le résultat est de taille suffisante et alloue une autre variable si nécessaire.

Cette histoire de taille de la variable qui stocke le résultat est un des problèmes pénibles de SDNV. On ne peut pas exiger de chaque implémentation qu'elle mette en œuvre un système d'entiers infinis (le RFC cite GNU MP comme exemple). La section 3.3 propose que chaque protocole qui utilise des SDNV fixe une taille maximale aux entiers stockés (retombant ainsi partiellement dans les problèmes des champs de taille fixe). Ainsi, le protocole Bundle (RFC 5050) fixe cette limite à la valeur maximale pouvant être représentée avec 64 bits. Le décodeur peut donc utiliser un `uint64_t` sans crainte.

Voilà, les SDNV sont donc très simples dans leur principe. Mais quels sont leurs avantages et leurs inconvénients ? La section 4 discute des alternatives : SDNV actuels, TLV, et l'ancien système de SDNV qui avait été utilisé dans des versions antérieures de certains protocoles DTN. En gros, le système le plus souple est TLV mais c'est aussi le moins efficace, en place et en temps. Un avantage de TLV est que, via le champ Type, la sémantique est connue, alors que les SDNV ne sont que des entiers non typés. Autre avantage : la taille du résultat est connue tout de suite et l'allocation mémoire pour le résultat tombe donc toujours juste (l'ancien SDNV avait un système analogue). Mais l'un des problèmes des TLV est que la représentation des champs Type et Longueur. S'ils sont de taille fixe, les TLV héritent certains des problèmes des champs de taille fixe. Notons qu'on peut combiner les techniques, par exemple utiliser SDNV pour le champ Longueur d'un TLV.

Bien que le RFC n'en discute guère, je me permets d'ajouter ici une comparaison essentiellement personnelle entre champs de taille fixe et SDNV. Les champs de taille fixe :

— Sont bien plus faciles à encoder/décoder, dans un langage comme C, c'est même trivial. Ce n'est pas seulement un avantage pour le programmeur, c'est aussi une garantie de rapidité.
— Ont pour principal inconvénient le fait qu'une mauvaise estimation de la taille nécessaire n'est pas corrigeable a posteriori (cas des fenêtres TCP et des adresses IPv4 cités plus haut). Si la capacité du réseau est grande (ce qui n'est typiquement pas le cas dans l'espace), une solution possible est de gaspiller, en utilisant systématiquement des champs très grands, genre 128 bits.
— Ont une taille connue à l'avance (par définition), ce qui permet au programmeur de réserver la bonne taille (cf. section 3.3)
— Sont très sensibles aux erreurs de transmission car les modifications sont indécelables. Au contraire, la modification d'un bit dans un SDNV peut entraîner une erreur de format, qui sera détecté au décodage.

Bref, je trouve personnellement que l'utilisation fréquente de champs de taille fixe dans les protocoles Internet n'est pas due au hasard.

Reparlons de décodage avec la question de la sécurité (section 5). Un décodeur SDNV doit faire attention aux valeurs qui dépassent ses limites, non seulement pour ne pas planter, mais aussi parce que cela peut avoir des conséquences de sécurité (débordement de tampon, par exemple). Autre piège, les entiers gérés par SDNV sont **toujours** positifs. Une implémentation ne doit donc pas les décoder vers un type signé, ce qui risquerait d'introduire des valeurs négatives dans un processus qui ne s'y attend pas. (TLV a le même problème si on place un champ Longueur dans une variable de type signé.)

Place aux implémentations. Le programme Python sdnv.py (en ligne sur `https://www.bortzmeyer.org/files/sdnv.py`), qui utilise le code présent dans l'annexe A du RFC, permet de tester encodage et décodage. L'affichage (pas le calcul) nécessite l'excellent module bitstring <`http://code.google.com/p/python-bitstring/`> :

```
# Encodage

% python sdnv.py -e 3
3 -> 0b00000011

% python sdnv.py -e 130
130 -> 0b1000000100000010

% python sdnv.py -e 4598
4598 -> 0b1010001101110110

% python sdnv.py -e 46598
46598 -> 0b10000010111010000000110

# Décodage

% python sdnv.py -d 00000000
00000000 -> 0

% python sdnv.py -d 00000011
00000011 -> 3

% python sdnv.py -d 1000000100000010
1000000100000010 -> 130

[Ajout d'un troisième octet, fait de bits nuls, _après_ le SDNV.]
% python sdnv.py -d 100000010000001000000000
100000010000001000000000 (only 2 bytes read) -> 130
```

À noter que les SDNV disposent d'une page Web officielle <http://www.dtnrg.org/wiki/SDNV>. En dehors du code Python du RFC, je n'ai pas vu d'autres implémentations mais on peut probablement en extraire du code DTN <http://www.dtnrg.org/wiki/Code>.