# RFC 9006 : TCP Usage Guidance in the Internet of Things (IoT)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 mars 2021

Date de publication du RFC : Mars 2021

https://www.bortzmeyer.org/9006.html

---

À côté de machines disposant de ressources matérielles suffisantes (électricité, processeur, etc), qui peuvent faire tourner des protocoles comme TCP sans ajustements particuliers, il existe des machines dites contraintes, et des réseaux de machines contraintes, notamment dans le cadre de l'Internet des Objets. Ces machines, pauvres en énergie ou en capacités de calcul, doivent, si elles veulent participer à des communications sur l'Internet, adapter leur usage de TCP. Ce RFC documente les façons de faire du TCP « léger ».

Ces CNN ("*Constrained-Node Networks*", réseaux contenant beaucoup d'objets contraints) sont décrits dans le RFC 7228 [1]. On parle bien d'objets contraints, soit en processeur, soit en énergie. Un Raspberry Pi ou une télévision connectée ne sont pas des objets contraints, ils peuvent utiliser les systèmes habituels, avec un TCP normal, au contraire des objets contraints, qui nécessitent des technologies adaptées. Le RFC 8352 explique ainsi les problèmes liés à la consommation électrique de certains protocoles. Des protocoles spéciaux ont été développés pour ces objets contraints, comme 6LoWPAN (RFC 4944, RFC 6282 et RFC 6775) ou comme le RPL du RFC 6550 ou, au niveau applicatif, le CoAP du RFC 7252.

Côté transport, on sait que les principaux protocoles de transport actuels sur l'Internet sont UDP et TCP. TCP a été parfois critiqué comme inadapté à l'Internet des Objets. Ces critiques n'étaient pas forcément justifiées mais il est sûr que le fait que TCP ait des en-têtes plutôt longs, pas de "*multicast*" et qu'il impose d'accuser réception de toutes les données peut ne pas être optimal pour ces réseaux d'objets contraints. D'autres reproches pouvaient être traités, comme expliqué dans l'article « " *TCP in the Internet of Things : from ostracism to prominence*" <https://upcommons.upc.edu/handle/2117/116511> ». Notez que CoAP, à l'origine, tournait uniquement sur UDP mais que depuis il existe aussi

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7228.txt

sur TCP (RFC 8323). Les CNN ("*Constrained-Node Networks*") utilisent parfois d'autres protocoles applicatifs tournant sur TCP comme HTTP/2 (RFC 9113) ou MQTT.

TCP est certes complexe si on veut utiliser toutes les optimisations qui ont été développées au fil du temps. Mais elles ne sont pas nécessaires pour l'interopérabilité. Un TCP minimum peut parfaitement communiquer avec des TCP optimisés, et notre RFC explique comment réduire l'empreinte de TCP, tout en restant évidemment parfaitement compatible avec les TCP existants. (Notez qu'il y avait déjà eu des travaux sur l'adaptation de TCP à certains environnements, voir par exemple le RFC 3481.)

Bon, maintenant, au travail. Quelles sont les propriétés des CNN (RFC 7228) qui posent problème avec TCP (section 2 du RFC) ? Ils manquent d'énergie et il ne peuvent donc pas émettre et recevoir en permanence (RFC 8352), ils manquent de processeur, ce qui limite la complexité des protocoles, et ils utilisent souvent des réseaux physiques qui ont beaucoup de pertes (voire qui corrompent souvent les paquets), pas vraiment les réseaux avec lesquels TCP est le plus à l'aise (RFC 3819).

La communication d'un objet contraint se fait parfois à l'intérieur du CNN, avec un autre objet contraint et parfois avec une machine « normale » sur l'Internet. Les types d'interaction peuvent aller de l'unidirectionnel (un capteur transmet une mesure qu'il a faite), à la requête/réponse en passant par des transferts de fichiers (mise à jour du logiciel de l'objet contraint, par exemple). Voyons maintenant comment TCP peut s'adapter (section 3 du RFC).

D'abord, la MTU. En IPv6, faire des paquets de plus de 1 280 octets, c'est prendre le risque de la fragmentation, qui n'est pas une bonne chose pour des objets contraints en mémoire (RFC 8900), qui n'ont en plus pas très envie de faire de la "*Path MTU discovery*" (RFC 8201). Donc, notre RFC conseille d'utiliser la MSS ("*Maximum Segment Size*") de TCP pour limiter la taille des paquets. Attention, les CNN tournent parfois sur des réseaux physiques assez spéciaux, où la MTU est bien inférieure aux 1 280 octets dont IPv6 a besoin (RFC 8200, section 5). Par exemple, IEEE 802.15.4 a une MTU de 127 octets seulement. Dans ce cas, il faut prévoir une couche d'adaptation entre IPv6 et le réseau physique (ce que fait le RFC 4944 pour IEE 802.15.4, le RFC 7668 pour Bluetooth LE, le RFC 8105 pour DECT LE, etc). Heureusement, d'autres technologies de réseau physique utilisées dans le monde des CNN n'ont pas ces limites de MTU, c'est le cas par exemple de "*Master-Slave/Token-Passing*" (cf. RFC 8163), IEEE 802.11ah, etc.

Deuxième endroit où on peut optimiser, ECN (RFC 3168). ECN permet aux routeurs intermédiaires de marquer dans un paquet que la congestion est proche ; le destinataire peut alors prévenir l'émetteur de ralentir. Le RFC 8087 décrit les avantages de l'ECN. Permettant de détecter l'approche de la congestion avant qu'on ait perdu un seul paquet (et donc sans qu'on ait à dépenser des watts pour retransmettre), l'ECN est particulièrement intéressant pour les CNN. Le RFC 7567 donne des conseils pratiques pour son déploiement.

Un problème classique de TCP sur les liens radio est que TCP interprète une perte de paquet comme signal de congestion, le poussant à ralentir, alors que cette perte peut en fait être due à la corruption d'un paquet (suite à une perturbation radio-électrique, par exemple). Il serait donc intéressant de pouvoir signaler explicitement ce genre de perte (la question était déjà discutée dans le RFC 2757 mais aussi dans l'article « "*Explicit Transport Error Notification (ETEN) for Error-Prone Wireless and Satellite Networks*" <http://www.icsi.berkeley.edu/icsi/node/1794> »). Pour l'instant, il n'existe aucun mécanisme standard pour cela, ce qui est bien dommage.

Pour faire tourner TCP sur une machine contrainte, une technique parfois utilisée est de n'envoyer qu'un segment à la fois (et donc d'annoncer une fenêtre dont la taille est d'un MSS - "*Maximum Segment Size*"). Dans ce cas, pas besoin de mémoriser plus qu'un segment de données envoyé mais dont l'accusé

de réception n'a pas encore été reçu. C'est très économique, mais ça se paie cher en performances puisqu'il faut attendre l'accusé de réception de chaque segment avant d'en envoyer un autre. La capacité effective du lien va chuter, d'autant plus que certaines optimisations de TCP comme le "*fast recovery*" dépendent d'une fenêtre plus grande qu'un seul segment. Au niveau applicatif, on voit la même technique avec CoAP, qui est par défaut purement requête/réponse.

Si on veut faire du TCP « un seul segment », le code peut être simplifié, ce qui permet de gagner encore en octets, mais notre RFC rappelle quand même que des options comme MSS (évidemment), `NoOp` et `EndOfOptions` restent nécessaires. En revanche, on peut réduire le code en ne gérant pas les autres options comme `WindowScaling` (RFC 7323), `Timestamps` (RFC 7323) ou SACK (RFC 2018). TCP a le droit d'ignorer ces options, qui, en « un seul segment » sont parfois inutiles (`WindowScaling`, SACK) et parfois moins importantes (`Timestamps`). En tout cas, si la machine a assez de mémoire, il est sûr que transmettre plusieurs segments avant d'avoir eu l'accusé de réception du premier, et utiliser des algorithmes comme le "*fast recovery*" améliore certainement les performances. Même chose pour les accusés de réception sélectifs, les SACK du RFC 2018.

La détermination du RTO ("*Retransmission TimeOut*") est un des points cruciaux de TCP (RFC 6298). S'il est trop long, on attendra longtemps la retransmission, quand un paquet est perdu, s'il est trop court, on ré-émettra parfois pour rien, gâchant des ressources alors que le paquet était juste en retard. Bref, une mise en œuvre de TCP pour les CNN doit soigner ses algorithmes de choix du RTO (cf. RFC 8961).

Continuons avec des conseils sur TCP dans les réseaux d'objets contraints. Notre RFC rappelle que les accusés de réception retardés, utiles pour accuser réception d'une plus grande quantité de données et ainsi diminuer le nombre de ces accusés, peuvent améliorer les performances…ou pas. Cela dépend du type de trafic. Si, par exemple, le trafic est surtout dans une direction, avec des messages courts (ce sera typiquement le cas avec CoAP), retarder les accusés de réception n'est sans doute pas une bonne idée.

Les paramètres par défaut de TCP sont parfois inadaptés aux CNN. Ainsi, le RFC 5681 recommande une taille de fenêtre initiale d'environ quatre kilo-octets. Le RFC 6298 fait des recommandations qui peuvent aboutir à des tailles de fenêtre initiale encore plus grandes. C'est très bien pour un PC connecté via la fibre mais pas pour la plupart des objets contraints, qui demandent des paramètres adaptés. Bref, il ne faut pas lire le RFC 6298 trop littéralement, car il faut en général une taille de fenêtre initiale plus petite.

Il n'y a pas que TCP lui-même, il y a aussi les applications qui l'utilisent. C'est l'objet de la section 4 du RFC. En général, si un objet contraint communique avec un non-contraint, c'est le premier qui initie la connexion (cela lui permet de dormir, et donc d'économiser l'énergie, s'il n'a rien à dire). L'objet contraint a tout intérêt à minimiser le nombre de connexions TCP, pour économiser la mémoire. Certes, cela crée des problèmes de "*head-of-line blocking*" (une opération un peu lente bloque les opérations ultérieures qui passent sur la même connexion TCP) mais cela vaut souvent la peine.

Et combien de temps garder la connexion TCP ouverte ? Tant qu'on a des choses à dire, c'est évident, on continue. Mais lorsqu'on n'a plus rien à dire, l'application doit-elle fermer les connexions, qui consomment de la mémoire, sachant que rouvrir la connexion prendra du temps et des ressources (la triple poignée de mains…). C'est un peu le problème de l'automobiliste arrêté qui se demande s'il doit couper son moteur. S'il faut redémarrer tout de suite, il consommera davantage de carburant. D'un autre côté, s'il laisse son moteur tourner, ce sera également un gaspillage. Le problème est soluble si l'application sait exactement quand elle aura à nouveau besoin d'émettre, ou si l'automobiliste sait exactement combien de temps durera l'arrêt mais, en pratique, on ne le sait pas toujours. (Ceci dit, pour l'automobile, le système d'arrêt-démarrage automatique dispense désormais le conducteur du choix.)

Une autre raison pour laquelle il faut être prudent avec les connexions TCP inactives est le NAT. Si un routeur NAT estime que la connexion est finie, il va retirer de ses tables la correspondance entre l'adresse IP interne et l'externe et, lorsqu'on voudra recommencer à transmettre des paquets, ils seront perdus. Le RFC 5382 donne des durées minimales avant ce retrait (deux heures...) mais elles ne sont pas forcément respectées par les routeurs NAT. Ainsi, l'étude « "*An Experimental Study of Home Gateway Characteristics*" <https://eggert.org/talks/2010-imc-hgw-study.pdf> » trouve que la moitié des boitiers testés ne respectent pas la recommandation du RFC 5382, avec des délais parfois aussi courts que quelques minutes ! Une des façons d'empêcher ces coupures est d'utiliser le mécanisme "*keep-alive*" de TCP (RFC 1122, section 4.2.3.6), qui envoie régulièrement des paquets dont le seul but est d'empêcher le routeur NAT d'oublier la connexion. Une autre est d'avoir des « battements de cœur » réguliers dans les applications, comme le permet CoAP (RFC 8323). Et, si on coupe rapidement les connexions TCP inutilisées, avant qu'une stupide "*middlebox*" ne le fasse, comment reprendre rapidement ensuite, si le trafic repart ? "*TCP Fast open*" (RFC 7413) est une solution possible.

Enfin, la sécurité pose des problèmes particuliers dans les CNN, où les ressources de certaines machines peuvent être insuffisantes pour certaines solutions de sécurité. Ainsi, pour TCP, la solution d'authentification AO (RFC 5925) augmente la taille des paquets et nécessite des calculs supplémentaires.

Il existe un certain nombre de mises en œuvre de TCP qui visent les objets contraints mentionnés dans ce RFC. Une machine 32 bits alimentée en courant en permanence, comme un vieux Raspberry Pi, n'est pas concernée, elle fait tourner le TCP habituel de Linux. On parle ici de TCP pour objets vraiment contraints. C'est par exemple (annexe A du RFC) le cas de :

- uIP, qui vise les microcontrôleurs à 8 et 16 bits. Elle est utilisée dans Contiki et sur la carte d'extension Ethernet ("*shield*") pour Arduino. En 5 ko, elle réussit à faire IP (dont IPv6 dans les dernières versions) et TCP. Elle fait partie de celles qui utilisent le « un segment à la fois », ce qui évite les calculs de fenêtres (qui nécessitent des calculs sur 32 bits, qui seraient lents sur ces processeurs). Et c'est à l'application de se souvenir de ce qu'elle a envoyé, TCP ne le fait pas pour elle. L'utiliser est donc difficile pour le programmeur.
- lwIP, qui vise le même genre de processeurs, mais dont l'empreinte mémoire est supérieure (entre 14 et 22 ko). Il faut dire qu'elle n'est pas limitée à envoyer un segment à la fois et que TCP mémorise les données envoyées, déchargeant l'application de ce travail. Et elle dispose de nombreuses optimisations comme SACK.
- RIOT a sa propre mise en œuvre de TCP, nommée GNRC TCP <https://arxiv.org/abs/1801.02833>. Elle vise aussi les engins de classe 1 (cf. RFC 7228 pour cette terminologie). Elle est de type « un segment à la fois » mais c'est TCP, et pas l'application, qui se charge de mémoriser les données envoyées (et qu'il faudra peut-être retransmettre). Par défaut, une application ne peut avoir qu'une seule connexion et il faut recompiler si on veut changer cela. Par contre, RIOT dispose d'une interface "*sockets*", familière à beaucoup de programmeurs.
- freeRTOS a aussi un TCP, pouvant envoyer plusieurs segments (mais une option à un seul segment est possible, pour économiser la mémoire). Il a même les accusés de réception retardés.
- uC/OS peut également faire du TCP avec plusieurs segments en vol.

Un tableau comparatif en annexe A.7 résume les principales propriétés de ces différentes mises en œuvre de TCP sur objets contraints.