RFC 793 : Transmission Control Protocol (TCP) - Specification

Crédits : Jon Postel / ISI
Traduction : Valéry G. FREMAUX / Ingénieur Professeur / EISTI
Statut : Standard

AVANT-PROPOS

Note du Traducteur :

Le texte suivant est la traduction intégrale de la spécification TCP, telle qu'éditée par les auteurs originaux du protocole, sans ajouts, commentaires, ni
omissions. Ce document a valeur de "standard".

PREFACE

Ce document décrit le protocole DoD Standard Transmission Control Protocol (TCP). Neuf versions précédentes de la spécification ARPA TCP ont été éditées.
Ce texte s'appuie très fortement sur ces précédentes version. Ce texte réunit les contributions de nombreux rédacteurs et de développeurs. Cette édition clarifie
plusieurs détails et supprime le principe d'ajustement de taille de tampon, il décrit de nouveau le principe de courrier et l'entrée de pile.

Jon Postel
Editor

RFC: 793
Remplace: RFC 761
IENs: 129, 124, 112, 81, 55, 44, 40, 27, 21, 5

1. INTRODUCTION

Le protocole TCP est défini dans le but de fournir un service de transfert de données de haute fiabilité entre deux ordinateurs "maîtres" raccordés sur un réseau de type "paquets commutés", et sur tout système résultant de l'interconnexion de ce type de réseaux.

Ce document décrit les fonctions exécutées par TCP, les programmes qui les implémentent, et les interfaces entre ce

dernier et les applications sensées utiliser ce service.

1.1. Motivation

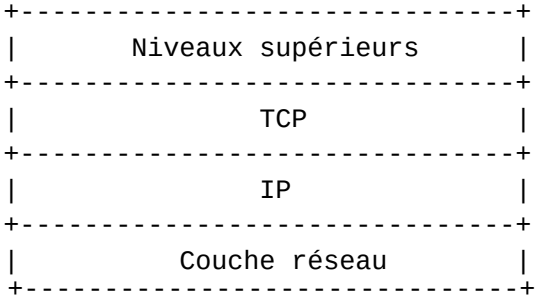
La communication entre systèmes d'information joue un rôle croissant dans les domaines militaires, institutionnels, scientifiques et commerciaux. Ce document prend en compte en tout premier lieu les exigences du secteur militaire, en particulier les exigences de fonctionnement avec des communications peu fiables et dans une situation de congestion du réseau. La plupart de ces problèmes sont rencontrés aussi dans les domaines non militaires.

Au fur et à mesure que les réseaux de communication informatiques à caractère stratégiques ou tactiques sont déployés, il devient essentiel de trouver un moyen d'interconnexion de ces réseaux, et des standards de transmission de données permettant de supporter une vaste gamme d'applications. Anticipant le besoin de tels standards, le député et sous-secrétaire d'état à la recherche de la Défense Américaine a officialisé le protocole décrit ici en tant que base pour la standardisation des processus d'intercommunication de données du Département de la Défense Américaine (DoD).

TCP est un protocole sécurisé orienté connexion conçu pour s'implanter dans un ensemble de protocoles multicouches, supportant le fonctionnement de réseaux hétérogènes. TCP fournit un moyen d'établir une communication fiable entre deux tâches exécutées sur deux ordinateurs autonomes raccordés à un réseau de données. Le protocole TCP s'affranchit le plus possible de la fiabilité intrinsèques des couches inférieures de communication sur lesquelles il s'appuie. TCP suppose donc uniquement que les couches de communication qui lui sont inférieures lui procurent un service de transmission de paquet simple, dont la qualité n'est pas garantie. En principe, TCP doit pouvoir supporter la transmission de données sur une large gamme d'implémentations de réseaux, depuis les liaisons filaires câblées, jusqu'aux réseaux commutés, ou asynchrones.

TCP s'intègre dans une architecture multicouche des protocoles, juste au-dessus du protocole Internet IP. Ce dernier permet à TCP l'envoi et la réception de segments de longueur variable, encapsulés dans un paquet Internet appelé aussi "datagramme". Le datagramme Internet dispose des mécanismes permettant l'adressage d'un service TCP source et un destinataire, quelles que soient leur position dans le réseau. Le protocole IP s'occupe aussi de la fragmentation et du réassemblage des paquets TCP lors de la traversée de réseaux de plus faibles caractéristiques. Le protocole IP transporte aussi les informations de priorité, compartimentation et classification en termes de sécurité relatives aux segments TCP. Ces informations se retrouvent alors transmises de bout en bout de la communication.

Couches de protocoles



De grandes parties de ce document sont écrites dans un contexte où les implémentations TCP sont concomitantes à d'autres protocoles de haut niveau dans la même machine. Certains systèmes informatiques seront raccordés au réseau via un frontal qui accueillera les fonctions TCP et IP, ainsi que les protocoles réseau de bas niveau. La spécification TCP décrit une interface à destination des applications de niveau supérieur, y compris dans le cas d'une architecture avec un frontal, pour autant que les protocoles "poste vers frontal" soient implémentés.

1.2. Portée

TCP prétend fournir un service de communication de processus à processus, dans un environnement réseau complexe. TCP est défini comme un protocole de communication "host to host", c'est à dire de maître à maître (par opposition à "central à terminal").

1.3. A propos de ce document

Ce document spécifie en détail le comportement de toute implémentation TCP, tant dans ses transactions avec les couches applicatives supérieures, qu'avec d'autres TCPs. Le reste de cette section offre une vue d'ensemble des fonctions réalisées et des interfaces proposées. La Section 2 résume le concept "philosophique" ayant aboutit au design TCP. La Section 3 décrit en détail les réactions de TCP face à divers événements (arrivée d'un nouveau segment, appel d'utilisateur, erreurs, etc.) ainsi que le format détaillé des segments TCP.

1.4. Interfaces

TCP s'interface avec un processus utilisateur ou applicatif et un protocole de niveau inférieur du type Internet Protocol.

L'interface avec les applicatifs consiste en un ensemble de commandes comme le ferait une application à un système d'exploitation pour la manipulation de fichiers. Par exemple, on trouvera des commandes pour établir et rompre une communication, pour envoyer ou recevoir des données sur une connexion ouverte. Il est aussi prévu que TCP puisse communiquer avec les applications sur un mode asynchrone. Bien qu'une grande liberté soit laissé aux développeurs pour la constructions d'interfaces TCP pour un environnement donné, des fonctionnalités minimales sont requises pour reconnaître la validité TCP de l'implémentation.

L'interface entre TCP et les protocoles de couche base restent largement non spécifiés excepté le fait qu'il doit y exister un mécanisme de transfert asynchrone de données. En général, c'est le protocole inférieur qui est sensé fournir la définition de cette interface. TCP assume un fonctionnement avec un large ensemble de protocoles réseau. Dans ce document, nous nous limiterons au fonctionnement avec IP.

1.5. Fonctionnement

Comme notifié ci-avant, TCP est conçu pour fournir un service de transmission de données sécurisé entre deux machines raccordés sur un réseau de paquets. Pour pouvoir assurer ce service même au dessus d'une couche de protocole moins fiable, les fonctionnalités suivantes sont nécessaires:

- Transfert de données de base
- Correction d'erreur
- Contrôle de flux
- Multiplexage
- Gestion de connexions
- Priorité et Sécurité

Ces fonctionnalités sont décrites en grandes lignes dans les paragraphes qui suivent.

Transfert de données de base:

TCP est capable de transférer un flux continu de données entre deux ordinateurs, en découpant ce flux en paquets ou datagrammes. En général, TCP décide de lui-même là où le flux de données doit être coupé.

Parfois les utilisateurs ont besoin de savoir que toutes les données soumises à TCP ont bien été émises. La fonction "push" a été prévue a cet effet. Pour s'assurer de la transmission complète de données jusqu'à un point spécifié, l'utilisateur activera la fonction "push" de TCP. Cette fonction oblige TCP à transmettre rapidement les données situées avant le point spécifié vers le destinataire. Il n'est nul besoin de fournir un marqueur spécifique pour ce point, dans la mesure ou le destinataire accepte ces données comme un transmission normale.

Contrôle d'erreur:

TCP doit considérer et traiter les cas de données perdues, erronées, dupliquées, ou arrivées dans le désordre à l'autre bout de la liaison Internet. Ceci est réalisé par l'insertion d'un numéro de séquence, et par l'obligation d'émission d'un "accusé de réception" (ACK) par le TCP destinataire. Si l'accusé de réception n'est pas reçu au bout d'un temps prédéfini, le paquet sera réémis. Côté récepteur, les numéros de séquence sont utilisés pour reconstituer dans le bon ordre le flux original, et éliminer les paquets dupliqués. L'élimination des erreurs physiques de transmission se fait par

encodage d’un Checksum à l’émission, recalcul de ce Checksum par le destinataire, et élimination des paquets pour les quels les deux valeurs ne correspondent pas.

Tant que TCP fonctionne correctement, et que le réseau Internet n’est pas saturé, aucune faute de transmission ne devrait transparaître dans la communication. TCP est donc sensé récupérer les erreurs de la transmission Internet.

Contrôle de flux:

TCP fournit un moyen au destinataire pour contrôler le débit de données envoyé par l’émetteur. Ceci est obtenu en retournant une information de "fenêtre" avec chaque accusé de réception indiquant la capacité de réception instantanée en termes de numéros de séquence. Ce paramètre noté "window" indique le nombre d’octets que l’émetteur peut envoyer avant une autorisation d’émettre ultérieure.

Multiplexage:

Pour permettre à plusieurs tâches d’une même machine de communiquer simultanément via TCP, le protocole définit un ensemble d’adresses et de ports pour la machine. Un "socket" est défini par l’association des adresses Internet source, destinataire, ainsi que les deux adresses de port à chaque bout. Une connexion nécessite la mise en place de deux sockets. Une socket peut être utilisée par plusieurs connexions distinctes.

L’affectation des ports aux processus est établi par chaque ordinateur. Cependant, il semble judicieux de réserver certains numéros de ports pour des services caractérisés et souvent utilisés. Ces services standard pourront alors être atteints via ces ports "réservés". L’affectation, l’utilisation et l’apprentissage des ports associés à d’autres services moins courants ou propriétaires nécessitera l’utilisation de mécanismes plus dynamiques.

Connexions:

Les mécanismes de fiabilisation et de contrôle de flux décrits ci-dessus imposent à TCP l’initialisation et la maintenance de certaines informations pour chaque communication. La combinaison de ces informations, dont les sockets, les fenêtres, et les numéros de séquence formeront ce que nous appelons une connexion. Chaque connexion est identifiée de manière unique par sa paire de sockets, définissant chacun des deux sens de la communication.

Lorsque deux processus désirent communiquer, leur TCP respectifs doivent tout d’abord négocier et établir une connexion (initialiser ces informations d’état de part et d’autre). Lorsque la communication s’achève, elle sera fermée, en libérant ses ressources à d’autres usages.

Dans la mesure où l’on considère que les ordinateurs, ainsi que le réseau Internet n’est pas d’une fiabilité absolue, on utilise une méthode d’initialisation par négociation bilatérale basée sur une horloge pour les numéros de séquence.

Priorité et Sécurité:

Les exploitants de TCP peuvent indiquer le degré de sécurité et la priorité de la communication établie. TCP permet cependant de ne pas traiter ce besoin.

2. PHILOSOPHIE

2.1. Eléments constitutifs du réseau

L’environnement réseau est constitué de machines raccordées sur des réseaux, eux-mêmes interconnectés par l’intermédiaire de routeurs. Ces réseaux peuvent être des réseaux locaux (ex., Ethernet) ou réseaux étendus (ex., ARPAnet), mais sont dans tous les cas des réseaux de type "à commutation de paquets" ou "asynchrones". Les éléments actifs qui consomme et produisent des paquets sont appelés des processus. Un certain nombre de niveaux de protocoles réseau, au niveau des machines ou des routeurs, permettent d’établir une chaîne complète de communication entre les processus actifs de n’importe quelle machine.

Le terme paquet, générique, désigne les données d'une transaction unitaire entre un processus et le réseau. Le format physique des données à l'intérieur du réseau est en dehors du champ d'application de ce document.

Les "hosts" sont des ordinateurs raccordés au réseau, et, du point de vue de ce dernier, sont les sources et destinations des paquets. Les processus sont identifiés comme les éléments actifs dans ces machines (conformément à la définition courante de "programme en cours d'exécution"). Les terminaux, fichiers, et autres périphériques d'entrée/sortie peuvent être identifiés comme "éléments communicants" par l'intermédiaire d'un processus. De ce fait, toute communication reste identifiée comme un échange inter-processus.

Comme un processus particulier doit pouvoir discerner des communications distinctes avec d'autres processus, nous avons imaginé que chaque processus puisse disposer d'un certain nombre de "ports" d'entrée sortie, chacun attribué à une communication unique avec un autre processus.

2.2. Modèle de fonctionnement

Les processus transmettent les données en faisant appel à TCP et en passant des tampons de données comme arguments. TCP met en forme les données de ces tampons, les segmente afin de les transférer au protocole Internet qui a son tour les acheminera vers le TCP distant. Celui-ci reçoit les segments, les copie dans un tampon temporaire, et en avise l'émetteur. Le protocole TCP incluse les information nécessaires à la "reconstruction" en bon ordre des données originales.

Le modèle d'une communication Internet fait qu'il existe pour chaque TCP actif un module de protocole Internet chargé de l'acheminement de données. Ce module Internet "encapsule" à son tour les paquets TCP sous la forme de paquets Internet, transmis à un module Internet distant via des "routeurs". Pour transmettre le paquet ainsi constitué à travers un réseau local, une troisième couche de protocole, spécifique au réseau, est ajoutée.

Les paquets peuvent alors subir un grand nombre de transformations, fragmentations, ou autres opérations pendant leur acheminement au module Internet distant.

A l'arrivée dans un routeur, le paquet Internet est "désossé", puis ses informations sont examinées pour savoir vers quel réseau le paquet doit être acheminé. Un nouveau paquet Internet est constitué, selon les spécifications du segment de réseau désigné, puis transmis sur ce réseau.

Un routeur peut procéder à une segmentation plus fine des paquets, si le réseau en sortie n'a pas les performances suffisantes pour véhiculer le paquet d'origine. Pour réaliser ceci, le routeur exécute une nouvelle segmentation en "fragments". Ces mêmes fragments peuvent à leur tour être redécoupés en chemin par un autre routeur. Le format de paquets fragmentés est standardisé de sorte que le réassemblage soit toujours possible, étape par étape, jusqu'à restitution des données originales.

Le module Internet d'arrivée extrait le datagramme de niveau supérieur, pour nous, TCP (après défragmentation du paquet si nécessaire) et le passe à la couche TCP.

Cette description rapide du protocole ignore certains détails. Le type de service en est un d'importance. Celui-ci indique au routeur (ou au module Internet) quels paramètres de service doivent être utilisés pour traverser la section suivante. L'un de ces paramètres est la priorité du paquet. Un autre est le codage de sécurité du paquet, permettant aux réseaux traitant des aspects de sécurité de discriminer les paquets conformément aux exigences établies.

2.3. Les Hosts

TCP est conçu sous la forme d'un module du système d'exploitation. L'utilisateur exploitera ses fonctions comme une autre ressource système (ex. le système de fichiers). TCP pourra lui-même invoquer d'autres ressources système, par exemple pour utiliser les structures de données locales. Actuellement, l'interfaçage vers le réseau est implémentée sous la forme d'un pilote de périphérique. TCP n'adresse pas le pilote directement, mais transfère le paquet à IP qui prendra en charge à son tour les transactions avec le pilote.

Les mécanismes TCP n'interdisent pas l'implémentation de TCP dans un frontal. Cependant le frontal devra disposer d'un protocole vis à vis du central permettant la prise en compte de l'interface application vers TCP, telle que définie

dans ce document.

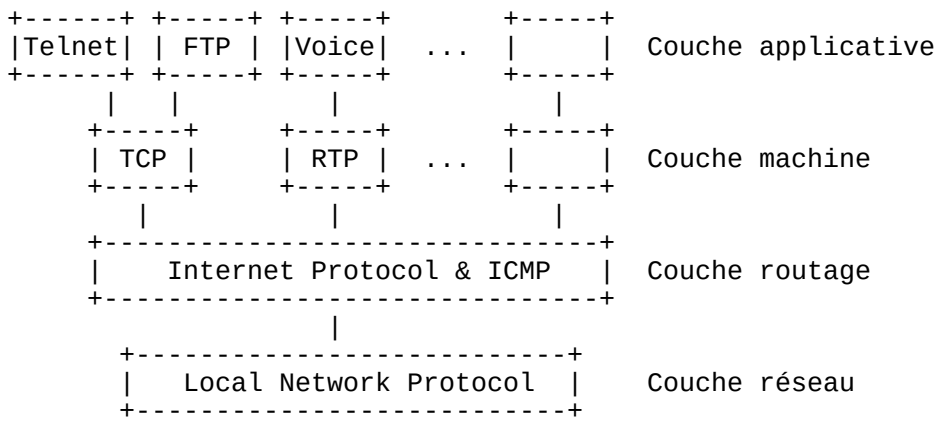
2.4. Interfaces

L’interface TCP/application fournit l’accès à des commandes OPEN ou CLOSE pour l’ouverture et la fermeture d’une communication, SEND ou RECEIVE pour l’émission ou la réception de données, ou STATUS pour connaître l’état d’une communication. Ces commandes seront appelées comme toute autre primitive système, par exemple celle qui permet l’ouverture ou la fermeture de fichiers.

L’interface TCP/Internet dispose de commandes pour recevoir et émettre des paquets vers des modules TCP où qu’ils soient sur le réseau. Ces appels ont des paramètres tels qu’adresses, type de service, priorité, sécurité, et autres informations de contrôle.

2.5. Relations avec d’autres protocoles

Le diagramme suivant montre la place de TCP dans la hiérarchie de protocoles



Relations interprotocoles - Figure 2.

TCP doit nécessairement supporter les niveaux de couches supérieures avec toute l’efficacité requise. L’interfaçage de protocoles tels que Telnet ARPAnet ou THP AUTODIN II doit demeurer aisée.

2.6. Fiabilité de communication

Un flux de donnée s’appuyant sur une connexion TCP doit être pouvoir considéré comme "fiable".

La fiabilité de cette transmission s’appuie sur l’utilisation de numéros de séquence et sur un mécanisme d’accusés de réception. Dans le concept, à chaque octet de données est attribué un numéro de séquence. Le numéro de séquence du premier octet transmis dans un segment est appelé le numéro de séquence du segment. Celui-ci est explicitement transmis avec le segment. En retour, l’accusé de réception est constitué du numéro de séquence du prochain octet à transmettre. Lorsque TCP transmet un segment contenant des données, celui-ci est copié dans une pile de réémission et une temporisation est lancée; lorsque l’accusé de réception est reçu, la copie dans la pile de retransmission est supprimée. Dans le cas contraire, le paquet est réémis une nouvelle fois.

L’accusé de réception ne garantit pas que les données sont effectivement arrivées à l’utilisateur final. Il indique simplement que le FTP destinataire à bien pris en charge ces données, en vue de les transmettre à cet utilisateur.

Pour contrôler le débit de données entre les deux TCP, un mécanisme dit de "contrôle de flux" est employé. Le TCP récepteur aménage une "fenêtre de réception" pour le TCP émetteur. Cette "fenêtre" indique le nombre d’octets, à partir du numéro de séquence inscrit dans l’accusé de réception, que le TCP distant est capable de recevoir.

2.7. Etablissement et rupture des connexions

TCP indique un identificateur de port. Comme ces identificateurs sont choisis indépendamment par chaque extrémité, ils peuvent se révéler identiques. L'adresse unique d'une communication TCP est obtenue par la concaténation de l'adresse Internet avec l'identificateur du port sélectionné, constituant ainsi ce que l'on nome une "socket". Cette socket est alors unique dans l'ensemble du réseau.

Une connexion de base est définie par un couple de sockets, l'un définissant l'émetteur, l'autre le récepteur. Un socket peut devenir le destinataire ou la source pour plusieurs sockets distinctes. La connexion est résolument bidirectionnelle, et prend la dénomination de "full-duplex".

TCP est libre d'associer ses ports avec les processus exécutés sur sa machine. Cependant, quelques règles ont été établies pour l'implémentation. Ont été définis un certain nombre de sockets "réservés" que TCP ne doit associer qu'avec certains processus bien identifiés. Ceci revient à dire que certains processus peuvent s'attribuer la propriété de certains ports, et ne pourront initier de communication que sur ceux-ci. (Actuellement, cette "propriété" est issue d'une implémentation locale, mais nous envisageons une commande utilisateur Request Port, ou une autre méthode pour assigner automatiquement un ensemble de ports à une application, par exemple en utilisant quelques bits de poids fort du numéro de port pour coder l'application).

Une connexion est demandée par activation de la commande OPEN indiquant le port local et les paramètres du socket distant. En retour, TCP répond par un nom local (court) symbolique que l'application utilisera dans ses prochains appels. Plusieurs choses doivent être retenues à propos des connexions. Pour garder la trace de cette connexion, nous supposons l'existence d'une structure de données appelée Transmission Control Block (TCB). Une des stratégies d'implémentation est de dire que le nom local donné est un pointeur vers le TCB associé à cette connexion. La commande OPEN spécifie en outre si le processus de connexion doit être effectué jusqu'à son terme, ou s'il s'agit d'une ouverture en mode passif.

Une ouverture passive signifie que le processus de connexion se met en attente d'une demande de connexion plutôt que de l'initier lui-même. Dans la plupart des cas, ce mode est utilisé lorsque l'application est prête à répondre à tout appel. Dans ce cas, le socket distant spécifié n'est composé que de zéros (socket indéfini). Le socket indéfini ne peut être passé à TCP que dans le cas d'une connexion passive.

Un utilitaire désireux de fournir un service à un processus non identifié pourra initier une connexion passive. Tout appelant effectuant une requête de connexion sur le socket local sera reconnu. Il sera bon de garder en mémoire que ce socket est associé à ce service.

Les sockets "réservés" sont un bon moyen d'associer à priori des ports à des applications standard. Par exemple, le serveur "Telnet" est en permanence associé à un socket particulier, d'autres étant réservés pour les transferts de fichiers, sessions de terminal distant, générateur de texte, écho (ces deux pour des besoins de test), etc. Un socket peut être réservé à la fonction de serveur de domaines, transcodant les "noms explicites" de services en sockets Internet. Si le concept même de l'assignation à priori de sockets fait partie de TCP, l'assignation concrète des sockets "réservés" est définie dans un autre document.

Les processus peuvent ouvrir une connexion passive et attendre qu'une connexion active les impliquant provienne d'une autre machine. TCP aura la charge d'avertir l'application qu'une communication est établie. Deux processus émettant au même moment une requête de connexion l'un vers l'autre se retrouveront normalement connectés. Cette souplesse est indispensable pour assurer un bon fonctionnement du réseau composé d'éléments totalement asynchrones.

Les deux cas de conclusion d'une communication impliquant une connexion passive et une active sont les suivants. Soit le socket distant a été précisé lors de la requête de connexion passive, auquel cas seule une requête de connexion du distant attendu vers le local peut aboutir à l'établissement d'une communication. Soit le socket distant a été laissé indéfini, et toute requête de connexion sur le socket local, d'où qu'elle vienne aboutit à une communication valide. D'autres fonctionnalités permettront une acceptation sur correspondance partielle entre sockets.

Si plusieurs requêtes de connexion passive sont en attente (enregistrées dans la table de TCBs) pour le même socket local, et qu'une demande de connexion active provient de l'extérieur, le protocole prévoit de d'abord chercher s'il l'une des requêtes dont le socket distant a été clairement exprimé correspond à celui de la demande. Si tel est le cas, ce socket sera activé. Sinon, c'est une requête "indéfinie" qui sera activée.

La procédure de connexion utilise le bit de contrôle de synchronisation (SYN) et suppose la transmission de trois messages. Cet échange est appelé "négociation ternaire".

La connexion suppose le rendez-vous d'un segment marqué du bit SYN et d'une requête locale (TCB), chacun des deux étant créé par l'exécution d'une commande de connexion. La correspondance entre le socket arrivé et le socket attendu détermine l'opportunité de la connexion. Celle-ci ne devient réellement établie que lorsque les deux numéros de séquence ont été synchronisés dans les deux directions.

La rupture d'une connexion suppose l'émission de segments, marqués du bit FIN.

2.8. Communication de données

Les données circulant dans la connexion ouverte doivent être vues comme un flux d'octets. L'application indique dans la commande SEND si les données soumises lors de cet appel (et toutes celles en attente) doivent être immédiatement émises par l'activation du flag PUSH.

Par défaut, TCP reste libre de stocker les données soumises par l'application pour les émettre à sa convenance, jusqu'à ce que le signal PUSH soit activé. Dans ce dernier cas, toutes les données non émises doivent être envoyées. Symétriquement, lorsque le TCP récepteur voit le flag PUSH marqué, il devra passer immédiatement toutes les données collectées à l'application destinataire.

Il n'y a à priori aucune corrélation entre la fonction PUSH et les limites des segments. Les données d'un segment peuvent être le résultat d'une seule commande SEND, en tout ou partie, ou celui de plusieurs appels SEND.

La fonction de la fonction push et du flag PUSH est de forcer la transmission immédiate de toutes les données latentes entre les deux TCP. Il ne s'agit aucunement d'une fonction d'enregistrement (Cf. langage Perl).

Il y a par contre une relation entre la fonction push et l'usage des tampons dans l'interface TCP/application. Chaque fois qu'un flag PUSH est associé à des données stockées dans le tampon de réception, celui-ci est intégralement transmis à l'application même s'il n'est pas plein. Si le tampon est rempli avant qu'un flag PUSH soit vu, les données sont transmises à l'application par éléments de la taille du tampon.

TCP dispose d'un moyen d'avertir l'application que, dans le flux de données qu'il est en train de lire, au delà de la position de lecture courante, des données de caractère urgent sont apparues. TCP ne définit pas ce que l'application est sensée faire lorsqu'elle est avisée de la présence de ces données. En général, c'est l'implémentation de l'application qui traitera ces données urgentes selon ses besoins propres.

2.9. Priorité et Sécurité

TCP utilise le champ "type de service" et les options de sécurité du protocole Internet pour fournir les fonctions relatives à la priorité et la sécurité des communications TCP, sur un principe de "détection". Tous les modules TCP ne fonctionneront pas nécessairement dans un environnement sécurisé à plusieurs niveaux; certains pourront être limités à un fonctionnement sans sécurité, d'autres ne pourront prendre en compte qu'un seul niveau à la fois. Par conséquent, les implémentations TCP ne pourront répondre en termes de sécurité qu'à un sous ensembles de cas du modèle sécurisé multi-niveaux.

Les modules TCP opérant dans un environnement sécurisé à plusieurs niveaux devront correctement renseigner les segments sortants en termes de sécurité, niveau de sécurité, et priorité. De tels modules TCP doivent fournir aux applications supérieures telles que Telnet ou THP une interface leur permettant de spécifier ces paramètres.

2.10. Principe de robustesse

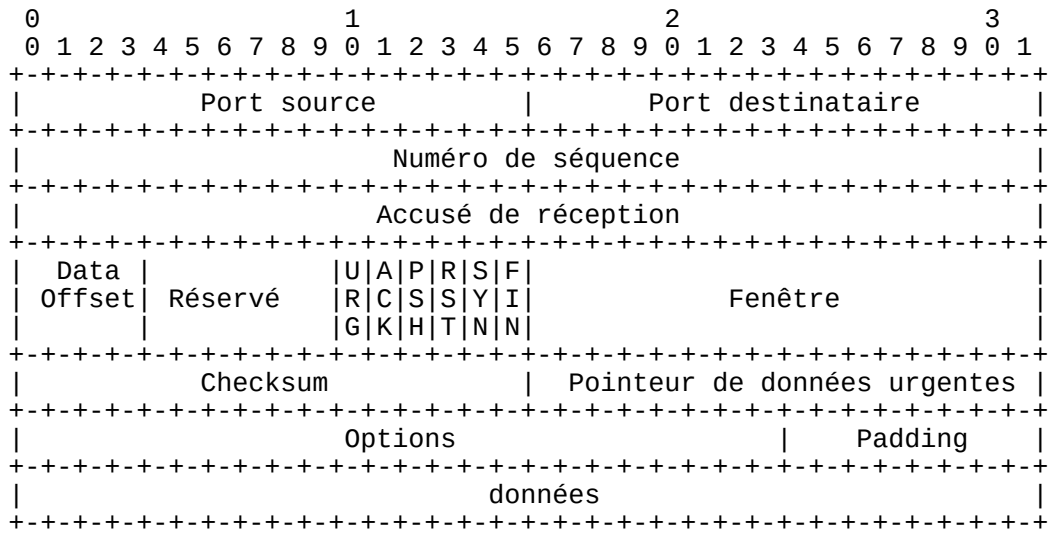
Les implémentations TCP devront suivre un principe de base: soyez rigoureux dans ce que vous émettez, soyez tolérants dans ce que vous recevez de l'extérieur.

3. SPECIFICATION FONCTIONNELLE

3.1. Format de l’en-tête

Les paquets TCP sont envoyés sous forme de datagrammes Internet. L’en-tête IP transmet un certain nombre de paramètres, tels que les adresses Internet source et destinataires. L’en-tête TCP est placée à la suite, contenant les informations spécifiques au protocole TCP. Cette division permet l’utilisation de protocoles autres que TCP, au dessus de la couche IP.

En-tête TCP



Notez qu’une case représente une position bit.

Port source: 16 bits **1000**

Le numéro de port de la source.

Port Destinataire: 16 bits

Le numéro de port du destinataire.

Numéro de séquence: 32 bits

Le numéro du premier octet de données par rapport au début de la transmission (sauf si SYN est marqué). Si SYN est marqué, le numéro de séquence est le numéro de séquence initial (ISN) et le premier octet à pour numéro ISN+1.

Accusé de réception: 32 bits

Si ACK est marqué ce champ contient le numéro de séquence du prochain octet que le récepteur s’attend à recevoir. Une fois la connexion établie, ce champ est toujours renseigné.

Data Offset: 4 bits

La taille de l’en-tête TCP en nombre de mots de 32 bits. Il indique là ou commence les données. L’en-tête TCP, dans tous les cas à une taille correspondant à un nombre entier de mots de 32 bits.

Réservé: 6 bits

Réservés pour usage futur. Doivent nécessairement être à 0.

Bits de contrôle: 6 bits (de gauche à droite):

- URG: Pointeur de données urgentes significatif
- ACK: Accusé de réception significatif
- PSH: Fonction Push
- RST: Réinitialisation de la connexion
- SYN: Synchronisation des numéros de séquence
- FIN: Fin de transmission

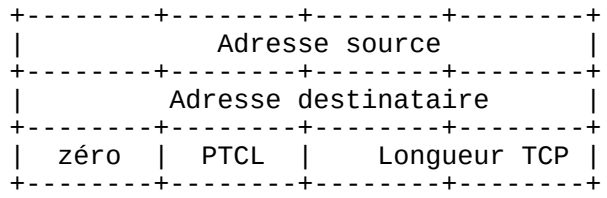
Fenêtre: 16 bits

Le nombre d'octets à partir de la position marquée dans l'accusé de réception que le récepteur est capable de recevoir.

Checksum: 16 bits

Le Checksum est constitué en calculant le complément à 1 sur 16 bits de la somme des compléments à 1 des octets de l'en-tête et des données pris deux par deux (mots de 16 bits). Si le message entier contient un nombre impair d'octets, un 0 est ajouté à la fin du message pour terminer le calcul du Checksum. Cet octet supplémentaire n'est pas transmis. Lors du calcul du Checksum, les positions des bits attribués à celui-ci sont marqués à 0.

Le Checksum couvre de plus une pseudo en-tête de 96 bits préfixée à l'en-tête TCP. Cette pseudo en-tête comporte les adresses Internet source et destinataires, le type de protocole et la longueur du message TCP. Ceci protège TCP contre les erreurs de routage. Cette information sera véhiculée par IP, et est donnée comme argument par l'interface TCP/Réseau lors des appels d'IP par TCP.



La longueur TCP compte le nombre d'octets de l'en-tête TCP et des données du message, en excluant les 12 octets de la pseudo en-tête.

Pointeur de données urgentes: 16 bits

Communique la position d'une donnée urgente en donnant son décalage par rapport au numéro de séquence. Le pointeur doit pointer sur l'octet suivant la donnée urgente. Ce champs n'est interprété que lorsque URG est marqué.

Options: variable

Les champs d'option peuvent occuper un espace de taille variable à la fin de l'en-tête TCP. Ils formeront toujours un multiple de 8 bits. Toutes les options sont prises en compte par le Checksum. Un paramètre d'option commence toujours sur un nouvel octet. Il est défini deux formats types pour les options:

Cas 1: Option mono-octet.

Cas 2: Octet de type d'option, octet de longueur d'option, octets de valeurs d'option.

La longueur d’option prend en compte l’octet de type, l’octet de longueur lui-même et tous les octets de valeur et est exprimée en octets.

Notez que la liste d’option peut être plus courte que ce que l’offset de données pourrait le faire supposer. Un octet de remplissage (padding) devra être dans ce cas rajouté après le code de fin d’options. Ce octet est nécessairement à 0.

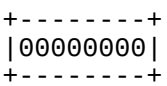
TCP doit implémenter toutes les options.

Actuellement, les options définies sont (type indiqué en octal):

Type	Longueur	Description
0	-	Fin de liste d’option
1	-	Nop
2	4	Taille de segment maximal

Définition des options spécifiques

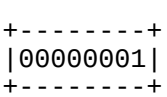
Fin de liste d’options



Type=0

Ce code indique la fin du champ d’options. Sa position peut ne pas coïncider avec l’indication du début du champ de données marqué dans l’Offset de données. Il doit être placé après toutes les options, et non après chaque option. Il ne doit être utilisé que dans le cas ou la fin des options ne coïncide pas avec le début du champ de données.

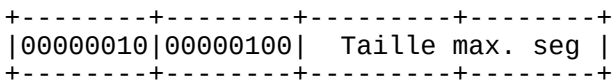
No-Operation



Type=1

Cette option peut être utilisée entre deux options, par exemple pour aligner le début d’une option sur un début de mot de 16 bits. L’utilisation de ce séparateur n’est pas une obligation. L’implémentation doit donc prévoir de pouvoir prendre en compte un option même au milieu d’un mot.

Taille maximale de segment



Type=2 Longueur=4

Donnée d’option : Taille maximale de segment: 16 bits

Si cette option est présente, elle communique à l’émetteur la taille maximale des segments qu’il pourra envoyer. Ce champ doit être envoyé dans la requête de connexion initiale (avec SYN marqué). Si cette option est absente, le

segment pourra être pris de n'importe quelle taille.

Bourrage (padding): variable

Les octets de bourrage terminent l'en-tête TCP:

- de sorte que le nombre d'octet de celle-ci soit toujours multiple de 4 (32 bits)
- de sorte que l'offset de données marqué dans l'en-tête corresponde bien au début des données applicatives.

3.2. Terminologie

Avant de préciser en profondeur le fonctionnement de TCP, nous devons tout d'abord prendre quelques conventions sur la terminologie. La maintenance d'une connexion TCP nécessite la mémorisation d'un certain nombre de variables. Nous avons prévu que ces données soient enregistrées dans une structure nommée "Transmission Control Block" ou TCB. Parmi les données enregistrées dans ce TCB, on trouvera les sockets local et distant, les informations de sécurité et de priorité de la connexion, les pointeurs vers les tampons de réception et d'émission, les pointeurs vers la pile de retransmission et vers le segment courant. On y trouvera de plus quelques données relatives à la gestion des numéro de séquence :

Variables de séquence d'émission

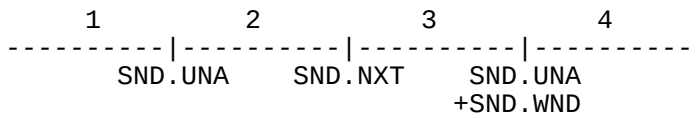
- SND.UNA - "unacknowledge" - envoi sans accusé de réception
- SND.NXT - "next" - envoi du suivant
- SND.WND - "window" - envoi de la fenêtre
- SND.UP - "urgent pointer" - pointeur de données urgentes
- SND.WL1 - "window last 1" - dernier numéro de séquence de segment envoyé
- SND.WL2 - "window last 2" - dernier accusé de réception
- ISS - "initial send sequence" - premier numéro de séquence du message

Variables de séquence de réception

- RCV.NXT - "next" - réception du suivant
- RCV.WND - "window" - réception de fenêtre
- RCV.UP - "urgent pointer" - pointeur de données urgentes
- IRS - "initial receive sequence" - premier numéro de séquence en réception

Le diagramme suivant montre une vue des tampons d'émission et de réception et l'implication des données de contrôle de séquence ainsi définis.

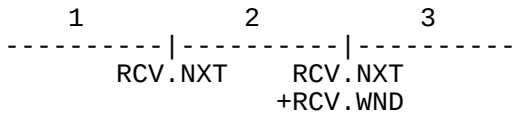
Visualisation de la séquence d'émission (vue à plat du tampon d'émission)



1. - anciens numéros de séquence ayant été acquittés
2. - numéros de séquences non acquittés
3. - numéros de séquence autorisés pour une nouvelle émission
4. - futurs numéros de séquence non encore autorisés

On notera que la fenêtre d'émission donnée par le récepteur représente la portion de l'espace de séquence noté 3. Les segments symbolisés en gras ont déjà été émis sur le réseau, les autres segments n'ont pas encore été émis et sont toujours dans le tampon.

Visualisation de la séquence de réception (vue à plat du tampon de réception)



1. - numéros de séquence reçus et acquittés
2. - numéros de séquence autorisés en réception (fenêtre)
3. - numéros de séquences futurs non encore autorisés en réception

La fenêtre de réception est la portion de séquence notée 2. Les segments symbolisés en gras ont déjà été reçus via le réseau. Les autres restent encore "à recevoir".

On trouvera enfin des variables dont les valeurs sont déduites des données inscrites dans le segment courant.

Variables du segment courant

- SEG.SEQ - "sequence" - numéro de séquence du segment courant
- SEG.ACK - "acknowledge" - numéro de séquence inscrit dans l'accusé de réception
- SEG.LEN - "length" - longueur du segment courant
- SEG.WND - "window" - fenêtre inscrite dans le segment courant
- SEG.UP - "urgent pointer" - pointeur de données urgentes du segment courant
- SEG.PRC - "precedence" - valeur de priorité courante

Une connexion connaît plusieurs états durant sa durée de vie. Les états définis sont: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, et l'état fictif CLOSED. CLOSED est dit fictif car il correspond à une situation où la connexion elle-même n'existe plus (son TCB non plus). Nous donnons ci-dessous un descriptif rapide des états cités:

LISTEN - La connexion reste en attente d'une requête de connexion externe par un TCP distant. Cet état est atteint après une demande de connexion passive.

SYN-SENT - La connexion se met en attente d'une requête de connexion, après avoir envoyé elle-même une requête à un destinataire.

SYN-RECEIVED - Les deux requêtes de connexion se sont croisées. La connexion attend confirmation de son établissement.

ESTABLISHED - La connexion a été confirmé de part et d'autre et les données peuvent transiter sur la voie de communication. C'est l'état stable actif de la connexion.

FIN-WAIT-1 - Sur requête de déconnexion émise par l'application, la connexion demande la confirmation d'une requête de déconnexion qu'elle a elle-même émise vers le distant.

FIN-WAIT-2 - La connexion se met en attente d'une requête de déconnexion par le distant, une fois reçue la confirmation de sa propre requête.

CLOSE-WAIT - La connexion se met en attente d'une requête de déconnexion émise par l'application.

CLOSING - La connexion attend la confirmation de sa requête de déconnexion par le TCP distant, lequel avait auparavant émis sa propre requête de déconnexion.

LAST-ACK - La connexion attend la confirmation de sa requête de déconnexion, émise suite à une requête similaire à

l’initiative du distant.

TIME-WAIT - Un temps de latence avant fermeture complète du canal, pour s’assurer que toutes les confirmations ont bien été reçues.

CLOSED - La connexion n’existe plus. C’est un pseudo-état.

Le changement d’état d’une connexion TCP intervient sur réception d’un certain nombre de signaux, appelée "événement". Les événements peuvent être des commandes émises par l’application, OPEN, SEND, RECEIVE, CLOSE, ABORT, et STATUS; la réception d’un flag dans un segment en provenance du distant SYN, ACK, RST et FIN; la fin d’une temporisation.

Le diagramme suivant montre l’enchaînement des états, en fonction des événements reçus, ainsi que les événements produits. Il occulte par contre le traitement des fautes, ainsi que tous les autres événements qui ne sont pas en relation avec les changements d’état. Un descriptif plus détaillé des événements et de leur fonctionnement sera exposé plus avant.

NOTA BENE: ce diagramme est un résumé et ne doit pas être compris comme une spécification complète.

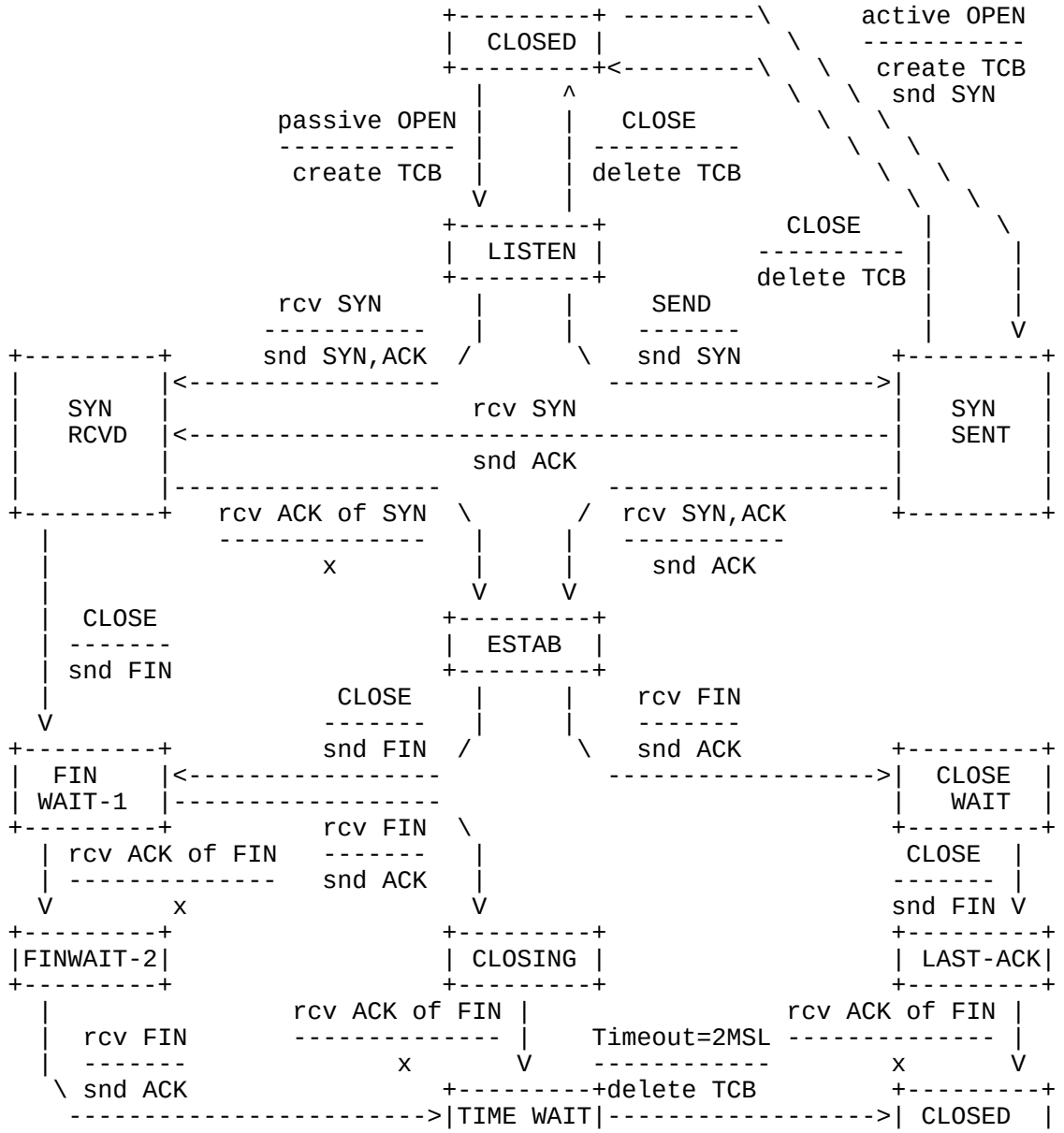


Diagramme d’état d’une connexion TCP

3.3. Numéros de séquence

Une notion fondamentale dans le design du protocole TCP est l’attribution à chaque octet de données d’un numéro de séquence. Cette technique de "marquage" permet de confirmer chaque octet individuellement. Le mécanisme d’acquittement est cumulatif, en ce sens que la confirmation de l’octet de numéro de séquence X indique que tous les octets précédents ont bel et bien été reçus. Ce mécanisme permet en outre l’élimination de toute donnée reçue en double par le principe de retransmission de séquences en faute. La technique de numération commence dès le premier octet de donnée, qui reçoit le numéro de séquence le plus faible. Les autres octets sont numérotés en séquence par ordre croissant.

Un des points essentiels à se souvenir est que l’espace de numérotation de séquence est fini, bien que très grand. Cet espace, codé sur 32 bits permet le comptage de 0 à 2**32 - 1 octets. Comme ce champ est de taille finie, toute opération arithmétique sur les numéros de séquence doit se faire modulo 2**32. Cette arithmétique non signée permet de préserver la continuité de numérotation, celle-ci repartant à 0 après la valeur 2**32 - 1. Toute opération arithmétique opérée sur les numéros de séquence devra être programmée avec beaucoup de précautions du fait de l’aspect cyclique de la numérotation, en particulier les comparaisons de numéros de séquence aux alentours de la limite supérieure.

Les principales comparaisons de numéro de séquence par TCP ont pour fonction:

(a) de déterminer qu’un accusé de réception reçu concerne bien des données émises et non encore confirmées.

(b) de déterminer que tous les numéros de séquence (donc, toutes les données) d’un segment ont bien été reçues (par exemple, pour purger la pile de retransmission).

(c) de déterminer qu’un segment reçu contient bien des numéros de séquence (et donc des données) attendues (c’est à dire que le principe de fenêtre de réception a bien été respecté par l’émetteur).

En réponse à l’émission de données, TCP reçoit des "accusés de réception". La confirmation de transmission doit s’appuyer sur les comparaisons suivantes:

```
SND.UNA = dernier numéro de séquence non acquitté
SND.NXT = prochain numéro de séquence à émettre
SEG.ACK = valeur d’accusé de réception (prochain numéro de séquence attendu par
le distant)
SEG.SEQ = premier numéro de séquence du segment
SEG.LEN = la taille des données en octets dans le segment (y compris pour des
segments SYN et FIN)
SEG.SEQ+SEG.LEN-1 = dernier numéro de séquence du segment
```

Un nouvel accusé de réception (appelé "ack acceptable"), est un accusé de réception pour lequel l’inégalité ci-dessous est vérifiée:

```
SND.UNA < SEG.ACK =< SND.NXT
```

Un segment ne peut être effacé de la pile de retransmission que si la somme de son numéro de séquence (premier octet de donnée) et sa longueur est inférieure au numéro de séquence du dernier accusé de réception reçu (ceci revient à dire en clair que l’acquittement doit pointer une donnée au delà de la fin du segment).

Lorsque des données sont reçues, les conditions ou affectations suivantes doivent être remplies:

```
RCV.NXT = numéro de séquence détecté sur le segment entrant. Doit être la limite
```

```
gauche (ou inférieure) de la fenêtre de réception.
RCV.NXT+RCV.WND-1 = plus grand numéro de séquence admis sur le segment entrant.
Est la limite droite (ou supérieure) de la fenêtre de réception.
SEG.SEQ = premier numéro de séquence du segment entrant
SEG.SEQ+SEG.LEN-1 = dernier numéro de séquence du segment entrant.
```

Un segment est considéré comme à l'intérieur de l'espace de réception si:

```
RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
```

et

```
RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND
```

Le premier test détermine si le premier octet des données du segment est dans la fenêtre. Le deuxième test vérifie que le dernier octet de données du segment reçu est aussi à l'intérieur de la fenêtre.

En pratique, cette vérification est un peu plus compliquée. Quatre cas d'acceptabilité sont discernable, à cause de la possibilité d'une largeur 0 pour la fenêtre et d'une longueur nulle de segment:

Longueur du segment	Fenêtre de réception	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	non acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND ou RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.W

Notez que lorsque la largeur de fenêtre est nulle, aucun segment excepté un acquittement (sans données) ne peut être reçu. Il est donc possible à un TCP de maintenir une largeur de fenêtre à zéro, tout en transmettant des données et recevant des accusés de réception. Cependant, même lorsque la fenêtre à une largeur nulle, TCP doit examiner les champs RST et URG pour tous les segments entrants.

Le mécanisme de numérotation nous permet en outre de protéger quelques informations de contrôle. Ceci est réalisé en introduisant implicitement des flags de contrôle dans la séquence (ou plus explicitement, attribuer un numéro de séquence à un contrôle).Ces contrôles pourront alors être retransmis et acquittés sans confusion (ex., une instance unique du contrôle sera traitée). Mais les informations de contrôle ne sont pas physiquement transmises dans le champ de données, le seul à être numéroté. Il faudra donc mettre en place une technique pour assigner ce numéro de séquence au contrôle. Les bits SYN et FIN sont les seuls contrôles pouvant nécessiter cette protection. Ceux-ci ne sont émis que pendant une phase d'établissement ou de rupture d'une connexion. Pour des raisons de numérotation, SYN est toujours considéré arriver avant le premier octet de données du segment dans lequel il est marqué. A l'inverse FIN est toujours considéré arriver après le dernier octet du segment. La longueur du segment (SEG.LEN) prend en compte l'espace de données ainsi que celui des contrôle. Lorsque SYN est marqué SEG.SEQ est alors le numéro de séquence du associé au contrôle SYN.

Sélection du premier numéro de séquence

Le protocole n'impose pas de restrictions à l'établissement simultané de plusieurs connexions identiques. Un connexion est définie par une paire de sockets. De nouvelles instances d'une connexion peuvent être ouvertes. Le problème qui en découle est -- "comment TCP identifie les segments dupliqués lorsqu'ils arrivent d'instances différentes d'une connexion?" Ce problème devient apparent lorsqu'une connexion s'ouvre et se ferme à une cadence rapide, ou si une connexion se coupe par manque de mémoire et se rétablit par la suite.

Pour éviter toute confusion, il faut éviter que les numéros de séquence utilisés par une instance de connexion ne soient utilisés lorsque les mêmes numéros de séquence sont émis sur le réseau par une autre instance. Ceci doit être assuré y

compris si TCP se bloque et perd toute connaissance des numéros de séquence qu'il utilisait. Lorsqu'une connexion est établie, un générateur de numéro de séquence initial (ISN) est utilisé, qui génère un nouvel ISN sur 32 bits. Ce générateur est basé sur une horloge (qui peut être virtuelle) 32 bit dont le bit de poids faible est incrémenté environ toutes les 4 microsecondes. De ce fait, le cycle des ISN dure environ 4,55 heures. Comme nous supposons qu'un segment ne peut être présent dans le réseau plus longtemps que sa durée de vie maximale (Maximum Segment Lifetime = MSL) et que MSL est inférieur à 4,55 heures, il est raisonnable de penser que l'ISN sera unique.

Pour chaque connexion, on met en place un numéro de séquence d'émission et un numéro de séquence de réception. Le numéro initial d'émission (ISS) est choisi par le TCP émetteur, le numéro de séquence initial de réception (IRS) est "appris" par le récepteur durant la phase d'établissement.

Pour qu'une connexion puisse être considérée comme établie, les deux TCPs doivent auparavant avoir pu synchroniser leurs numéros de séquence respectifs. Ceci est réalisé grâce à l'émission de segments particuliers de synchronisation avec le bit SYN marqué et les numéros de séquence initiaux. Par extension, les segments marqués du bit SYN seront nommés aussi SYN. La solution demande donc un mécanisme pour "prendre" un numéro de séquence initial, ainsi qu'un dialogue pour se communiquer les ISNs.

La solution consiste à faire envoyer par chaque extrémité son propre numéro de séquence initiale, à charge de l'autre bout d'acquitter cette information.

1) A --> B SYN "mon numéro de séquence est X"
2) A <-- B ACK "ton numéro de séquence est X"
3) A <-- B SYN "mon numéro de séquence est Y"
4) A --> B ACK "ton numéro de séquence est Y"

Comme les étapes 2 et 3 peuvent être rassemblées dans le même segment, ce dialogue s'appelle "ternaire".

Cet échange en trois étapes est nécessaire dans la mesure où le choix des numéros de séquence initiaux ne dépend pas d'une horloge commune à l'ensemble du réseau, et les diverses implémentations de TCP peuvent recourir à des méthodes diverses pour choisir un ISN. Le récepteur d'un premier segment SYN n'a aucun moyen de déterminer qu'il ne s'agit pas d'un ancien segment "perdu" sur le réseau, sauf s'il se "souvient" du dernier numéro de séquence utilisé par la dernière connexion (ce qui n'est pas toujours possible). Il doit donc demander à l'émetteur de vérifier ce segment SYN.

Savoir ne rien émettre

Pour être certain que TCP n'utilise pas un numéro de séquence déjà contenu dans un segment "perdu" en cours de transmission sur le réseau, TCP doit attendre au moins la durée d'un MSL (durée de vie maximale des segments) avant toute assignation d'un nouvel ISN pour une nouvelle connexion ou sur récupération d'une connexion dont l'historique de la numérotation de séquence a été perdue. Dans le cadre de cette spécification, MSL vaut 2 minutes. Cette valeur résulte de l'expérimentation, et est susceptible de changer si le besoin s'en fait sentir. Notez que si un TCP est réinitialisé en ayant pu conserver l'historique de séquence, alors cette attente ne sera pas nécessaire; il lui suffira de produire des numéros de séquence supérieurs aux dernier émis.

Le concept de "silence" TCP

Cette spécification prévoit que des ordinateurs en faute ayant perdu toute trace des numéros de séquence sur les connexions non fermées ne puisse émettre sur le réseau Internet des segments TCP pendant au minimum la durée MSL. Dan ce qui suit, une explication détaillée de cette spécification est fournie. Les développeurs implémentant TCP pourront toujours ne pas tenir compte de ce principe, mais au risque de voir des anciens segments acceptés comme nouveaux, ou des nouveaux rejetés car apparaissant comme des doubles d'anciens segments.

TCP consomme l'espace de numérotation des séquences chaque fois qu'un nouveau segment est constitué et placé dans la pile d'émission du pilote de réseau. La détection de doublons, et l'algorithme de séquence du protocole TCP s'appuie sur le champ de numérotation fini en supposant que la séquence est suffisamment longue pour que tous les segments, y compris toutes ses réémissions ou doublons aient été retirés du réseau avant que la séquence ne reboucle. Sans ce principe, deux segments TCP pourraient se voir attribuer des plages de séquence superposées, provoquant une extrême

difficulté au récepteur pour savoir quelle donnée est ancienne et quelle donnée est récente. Souvenez vous ici que dans une transmission normale, les segments successifs d'une transmission utilisent des plages de séquence disjointes et contiguës.

En temps normal, TCP garde la trace du numéro de séquence à émettre, et du plus ancien acquittement attendu. Ceci lui permet de ne pas utiliser un numéro de séquence attribué à un segment en attente d'acquittement. Ceci ne garantit pas que d'anciens doublons de paquets aient été totalement retirés du réseau. Le champ de numérotation a donc été choisi très grand pour éviter qu'un paquet "fantôme" ne vienne perturber une réception. A 2 megabits/seconde, l'utilisation des 2**32 valeurs de l'espace de numérotation prend environ 4,5 heures. Comme la durée de vie d'un paquet dans le réseau ne peut excéder quelques dizaines de secondes, on considère que cette protection est largement suffisante même lorsque la vitesse de transmission atteint 10 megabits/seconde. A 100 megabits/seconde, le cycle dure 5,4 minutes, ce qui peut paraître assez court, mais laisse encore une marge raisonnable. L'algorithme de détection de doublons et de séquencement de TCP peut cependant être mis en défaut, dans le cas où TCP perd la mémoire du numéro de séquence qu'il utilisait pour les dernières transmissions. Un exemple de ceci serait un TCP initiant toutes ses connexion au numéro de séquence 0, puis sur faute et réinitialisation, rétablit une connexion (ex. en exécutant l'analyse d'une demi connexion restée ouverte) et émet des paquets avec des plages de séquences recoupant celles de paquets toujours en transit dans le réseau, et provenant de la connexion avant la rupture. Lorsque cette synchronisation de séquence est perdue, la spécification TCP recommande de respecter un délai de MSL secondes avant de réemettre des segments sur la connexion, afin de permettre à tous les anciens segments encore en transit d'être éliminés du réseau.

Même les ordinateurs avec horloge interne absolue, et utilisant cette horloge pour le choix d'un ISN ne sont pas immunisés contre ce problème.

Supposons, pat exemple, qu'une connexion soit ouverte, disons, en partant du numéro de séquence S. Supposons de plus que cette connexion est très peu chargée et qu'en plus, la fonction de génération des numéros de séquence initiaux (ISN(t)) prenne comme base la valeur du numéro de séquence, disons S1, du dernier segment émis par ce TCP sur une connexion particulière, qui se trouve être cette celle-ci. Supposons enfin que l'ordinateur redémarre et qu'une connexion soit restaurée. Le numéro de séquence initial repris sera S1 = ISN(t) -- le même numéro que celui du dernier paquet envoyé par l'instance précédente de la connexion ! Pour peu que la récupération soit suffisamment rapide, tous les paquets sur le réseau portant un numéro de séquence proche de S1 risquent d'être compris comme des nouveau paquets, même s'ils proviennent de l'ancienne instance de la connexion.

Le problème est que l'ordinateur ayant eu la défaillance ne sait absolument pas combine de temps celle-ci a duré, et par conséquent si des paquets de l'ancienne connexion ne se trouvent pas encore en transit dans le réseau.

L'une des façons de se sortir de ce problème est d'imposer un silence d'au moins un MSL après récupération sur faute - C'est la spécification dite de "silence TCP". Les implémentation qui ne tiennent pas compte de ce temps de silence et l'ignorent risquent de provoquer des confusions entre nouveaux et anciens segments dans les TCP récepteurs. Au minimum, il est conseillé que les développeurs prévoient de laisser à l'utilisateur la possibilité d'ignorer le temps de silence connexion par connexion, ou, encore mieux, de systématiser de manière informelle ce fonctionnement. Bien évidemment, même lorsque l'attente TCP est implémentée, elle n'est pas indispensable lorsque la défaillance intervient quelques MSL après que la connexion ait cessé d'émettre.

Pour résumer: chaque segment occupe une plage de séquence à l'intérieur de l'espace de numérotation, tous les numéros à l'intérieur de ces plages sont "occupés" pendant MSL secondes. Après une défaillance, un bloc temporel est pris par la plage de séquence du dernier segment émis. Si une connexion reprend trop tôt, réémettant un segment utilisant un numéro de séquence compris dans la plage de séquence précédente, il y aura risque pour le récepteur de mal interpréter la séquence de segments et produire une erreur de réassemblage de données.

3.4. Etablissement d'une connexion

L'échange "ternaire" est la technique de négociation utilisée pour la connexion. Cette procédure est habituellement initiée par un TCP vers un autre TCP. La procédure fonctionne même si les deux TCP essaient de se connecter simultanément l'un à l'autre. Lorsqu'une telle tentative simultanée survient, chaque TCP reçoit un segment "SYN" ne transportant aucun accusé de réception après l'émission de son propre "SYN". Bien sûr, il restera toujours le cas d'un segment "SYN" d'une ancienne connexion qui peut faire croire à TCP qu'un établissement est en cours. Une bonne utilisation du segment de réinitialisation peut résoudre cette ambiguïté.

Plusieurs exemples d'initialisation d'une connexion suivent. Aucun de ces exemples ne montrent des segments d'initialisation transportant des données, ce qui est parfaitement légitime, dans la mesure où TCP n'envoie pas de donnée tant qu'il n'est pas sûr de la stabilité du canal (c'est-à-dire que TCP est sensé tamponner les données qui lui sont transmises par l'application tant que l'état ESTABLISHED n'a pas été atteint). La négociation "ternaire" réduit le risque de fausses connexions. L'implémentation des échanges entre la mémoire et les messages apportera les informations nécessaires à cette vérification.

La transaction ternaire la plus simple est décrite en figure 7. Ces figures doivent être interprétées de la façon suivante. Chaque ligne est numérotée pour référence. La flèche vers la droite (-->) indique le départ d'un paquet TCP du TCP A vers le TCP B, ou l'arrivée en B d'un segment issu de A. La flèche inverse (<--), la transmission dans le sens opposé. L'ellipse (...) indique un segment toujours sur le réseau (retardé). Un "XXX" indique un segment perdu, ou rejeté. Les commentaires apparaissent entre parenthèse. L'état TCP représente l'état obtenu APRES le départ ou l'arrivée du segment (dont le contenu est mentionné au centre de la ligne). Le contenu des segments est montré sous forme abrégée, avec son numéro de séquence, les flags marqués, et le champ ACK. Les autres champs tels qu'adresses, fenêtres, offset, etc. ne sont pas montrés par souci de clarté.

```
    TCP A                                                          TCP B
1.  CLOSED                                                         LISTEN
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>                  -->        SYN-RECEIVED
3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>     <--        SYN-RECEIVED
4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>         -->        ESTABLISHED
5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA>   -->        ESTABLISHED
```

Dialogue "ternaire" basique de connexion
Figure 7.

En ligne 2 de la figure 7, TCP A émet une requête envoyant un segment SYN indiquant qu'il utilisera des numéros de séquence à partir de 100. En ligne 3, TCP B renvoie sa requête SYN tout en acquittant le SYN reçu de TCP A. Notez que le champ d'accusé de réception indique que TCP B attend maintenant un numéro de séquence égal à 101, le SYN de la première requête ayant consommé la valeur 100 suivant le principe du numérotage implicite des contrôles.

En ligne 4, TCP A répond par un segment vide contenant l'accusé de réception au SYN de TCP B; En ligne 5 enfin, TCP A envoie des données. Notez alors que le numéro de séquence de la ligne 5 est identique à celui de la ligne 4, car ACK n'occupe pas d'espace dans la séquence (si c'était le cas, il nous faudrait acquitter les acquittements et on n'en sortirait plus!).

L'établissement d'une double requête simultanée est légèrement plus complexe, comme cela apparaît en figure 8. Chaque TCP bascule de l'état CLOSED vers SYN-SENT puis vers SYN-RECEIVED et enfin vers ESTABLISHED.

```
    TCP A                                                  TCP B
1.  CLOSED                                                 CLOSED
2.  SYN-SENT     --> <SEQ=100><CTL=SYN>                    ...
3.  SYN-RECEIVED <-- <SEQ=300><CTL=SYN>                    <-- SYN-SENT
4.               ... <SEQ=100><CTL=SYN>                    --> SYN-RECEIVED
5.  SYN-RECEIVED --> <SEQ=100><ACK=301><CTL=SYN,ACK>       ...
6.  ESTABLISHED  <-- <SEQ=300><ACK=101><CTL=SYN,ACK>       <-- SYN-RECEIVED
7.               ... <SEQ=101><ACK=301><CTL=ACK>           --> ESTABLISHED
```

Synchronisation d'une requête de connexion simultanée
Figure 8.

La principale raison du dialogue "ternaire" est d'éviter que d'anciens paquets issus d'une précédente tentative de connexion ne vienne perturber la nouvelle. Pour maîtriser ce cas de figure, un message de contrôle spécial pour la réinitialisation, "reset", a été institué. Si le TCP récepteur est dans un état non synchronisé (c'est à dire, SYN-SENT ou SYN-RECEIVED), il peut retourner à l'état d'attente LISTEN sur réception d'une commande de réinitialisation valable. Si TCP est dans l'un des états synchronisés (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), la commande de réinitialisation casse la connexion et l'application en est alors informée. Nous verrons ce cas plus en détail lorsque nous analyserons les connexion semi-ouvertes ou

"demi-connexions".

```
    TCP A                                                    TCP B
1.  CLOSED                                                   LISTEN
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>              ...
3.  (duplicate) ... <SEQ=90><CTL=SYN>               --> SYN-RECEIVED
4.  SYN-SENT    <-- <SEQ=300><ACK=91><CTL=SYN,ACK>  <-- SYN-RECEIVED
5.  SYN-SENT    --> <SEQ=91><CTL=RST>               --> LISTEN
6.              ... <SEQ=100><CTL=SYN>              --> SYN-RECEIVED
7.  SYN-SENT    <-- <SEQ=400><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED
8.  ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>     --> ESTABLISHED
```

Récupération sur doublon d'un ancien SYN
Figure 9.

La figure 9 présente un cas simple de récupération sur réception d'un SYN échappé d'une ancienne tentative. A la ligne 3, le doublon du SYN arrive au TCP B. TCP B n'a aucun moyen de savoir s'il s'agit d'un doublon ou d'un segment valide, Il répond donc normalement (ligne4). TCP A détecte que le champ d'accusé de réception est incorrect et renvoie une commande "reset" en marquant le bit RST en renseignant le champ SEQ pour maintenir la conformité du processus. TCP B, en recevant RST, revient à l'état LISTEN. Lorsque le SYN valide arrive enfin à la ligne 6, la synchronisation se déroulez normalement. Si le segment SYN de la ligne 6 était arrivé avant RST, un échange plus complexe aurait eu lieu, impliquant des RST envoyés dans les deux directions.

Connexions semi-ouvertes et autres anomalies

On parle de connexion "demi-ouverte" lorsque l'un des TCP a fermé ou coupé brutalement la connexion sans que l'autre extrémité n'en ait eu connaissance, ou si les deux extrémités se retrouvent désynchronisées suite à une défaillance avec perte de mémoire. De telles connexions vont déclencher l'opération de réinitialisation à la moindre tentative d'envoi de données dans l'un ou l'autre sens. Cependant, une telle situation est considéré comme inhabituelle ou du moins anormale, et la procédure de récupération ne sera pas toujours invoquée.

Si la connexion n'existe plus au site A, alors une tentative d'émission de données du site B va résulter en la réception d'un message de réinitialisation par le TCP du site B. Un tel message indique au site B que quelque chose ne fonctionne pas, et qu'il serait plus sûr d'arrêter la communication.

Supposez maintenant que deux applications A et B communiquent ensemble, et qu'une défaillance quelconque provoque une perte de mémoire au niveau du TCP A. Suivant le système d'exploitation qui supporte A, il se peut qu'il existe un mécanisme de récupération d'erreur. Dans ce cas, et lorsque le TCP est rétabli, A va certainement vouloir renvoyer les données à partir du début, ou à partir du point de récupération si cela lui est possible. En conséquence de quoi A va certainement essayer de rouvrir (OPEN) la connexion de nouveau ou essayer d'envoyer (SEND) des données sur la connexion qu'il croit encore ouverte. Dans ce dernier cas, il recevra un message du type "connexion inexistante" du TCP local (A) TCP. Dans l'intention de rouvrir la connexion, le TCP A va émettre un segment SYN. Ce scénario conduit à l'exemple de la figure 10. Après le crash du TCP A, l'application tente de rouvrir la connexion. TCP B, dans le même temps, pense toujours que la connexion est ouverte.

```
    TCP A                                                TCP B
1.  (CRASH)                                          (send 300,receive 100)
2.  CLOSED                                               ESTABLISHED
3.  SYN-SENT --> <SEQ=400><CTL=SYN>                  --> (??)
4.  (!!)     <-- <SEQ=300><ACK=100><CTL=ACK>         <-- ESTABLISHED
5.  SYN-SENT --> <SEQ=100><CTL=RST>                  --> (Abort!!)
6.  SYN-SENT                                             CLOSED
7.  SYN-SENT --> <SEQ=400><CTL=SYN>                  -->
```

Découverte d'une connexion semi-ouverte
Figure 10.

Lorsque le segment SYN arrive en ligne 3, TCP B, toujours synchronisé, et le segment reçu se trouvant en dehors de la fenêtre de réception, répond par un acquittement indiquant quelle séquence il s'attend à recevoir (ACK 100). TCP A s'aperçoit que cet accusé de réception n'accuse rien du tout (puisqu'il est totalement désynchronisé). Il envoie donc un

reset (RST), ayant détecté une situation de connexion semi-ouverte. TCP B abandonne en ligne 5. TCP A va continuer à tenter d'établir la connexion; le problème est alors résolu dans la mesure où l'on revient à la procédure de base "ternaire" de la figure 7.

Une alternative intéressante apparaît dans le cas d'un crash de TCP A alors que TCP B essaie d'envoyer des données sur ce qu'il pense être une connexion synchronisée. Ceci est illustré en figure 11. Dans ce cas, les données arrivant au TCP A à partir de TCP B (ligne 2) ne peuvent être accepté puisqu'aucune connexion n'existe, TCP A envoie donc un RST. RST est acceptable par TCP B qui abandonne la connexion.

```
    TCP A                                                TCP B
1.  (CRASH)                                          (send 300,receive 100)
2.  (??)    <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK> <-- ESTABLISHED
3.          --> <SEQ=100><CTL=RST>                   --> (ABORT!!)
```

Découverte de connexion semi-ouverte par le côté actif
Figure 11.

En figure 12, nous trouvons les deux TCPs A et B en état de connexion passive attendant un SYN. Un doublon ancien arrivant au TCP B (ligne 2) réveille celui-ci. Un SYN-ACK est renvoyé (ligne 3) et force TCP A à générer un RST (l'acquittement de la ligne 3 n'est pas recevable). TCP B accepte la réinitialisation et retourne sagement dans l'état d'attente LISTEN.

```
    TCP A                                                TCP B
1.  LISTEN                                           LISTEN
2.       ... <SEQ=Z><CTL=SYN>                  -->  SYN-RECEIVED
3.  (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>     <--  SYN-RECEIVED
4.       --> <SEQ=Z+1><CTL=RST>                -->  (return to LISTEN!)
5.  LISTEN                                           LISTEN
```

Réinitialisation suite à réception d'un doublon sur deux connexions passives
Figure 12.

Une grande variété de cas est possible, chacune résultant dans une procédure de réinitialisation selon les règles qui suivent.

Génération d'un signal de réinitialisation

En règle générale, un signal de réinitialisation (RST) doit être émis sur toute réception d'un segment qui ne répond visiblement pas aux exigences de la connexion ouverte. Ce message ne **doit pas être émis** si il n'est **pas certain** que c'est le cas. .

On dénotera trois types de situations:

1. Si la connexion n'existe pas (CLOSED) alors un "reset" est envoyé sur toute réception d'un segment sur cette connexion, excepté s'il s'agit lui-même d'un "reset". En particulier, des segments SYN adressés à une connexion inexistante sont rejetés par ce moyen.

Si le segment entrant présente un accusé de réception, le segment RST émis récupère le numéro de séquence du champ ACK de ce segment. Autrement le numéro de séquence du "reset" est zéro et l'accusé de réception est fixé à la somme du numéro de séquence et de la longueur de segment du segment entrant. La connexion demeure fermée.

2. Si la connexion est dans un état non synchronisé (LISTEN, SYN-SENT, SYN-RECEIVED), et le segment entrant acquitte quelque chose qui n'a pas été encore envoyé (ACK non recevable), ou le segment entrant est sur un niveau de sécurité ou un compartiment de sécurité ne correspondant pas exactement à celui attendu pour la connexion, un segment RST est émis.

Si notre propre segment SYN n'a pas été acquitté et le niveau de priorité du segment entrant est supérieur à celui attendu, on pourra relever le niveau de priorité de la connexion (si l'application et le système d'exploitation

l’autorisent) ou émettre un "reset"; ou si le niveau de priorité du segment entrant est inférieur à celui requis, on continuera à le traiter sans changer sa propre priorité (si le TCP distant ne peut augmenter le niveau de priorité pour répondre à nos exigences locales, cela sera signifié dans les prochains segments reçus, et la connexion se fermera). Dans le cas ou notre segment SYN a été acquitté (peut être dans ce segment entrant) le niveau de priorité doit correspondre exactement. Dans le cas contraire un "reset" doit être émis.

Si le segment entrant présente un accusé de réception, le segment RST émis récupère le numéro de séquence du champ ACK de ce segment. Autrement le numéro de séquence du "reset" est zéro et l’accusé de réception est fixé à la somme du numéro de séquence et de la longueur de segment du segment entrant. La connexion demeure dans le même état.

3. Si la connexion est dans un état synchronisé (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), tout segment non recevable (hors fenêtre de réception, ou accusé de réception invalide) provoque l’émission d’un segment vide contenant le numéro de séquence courant en émission et l’accusé de réception indiquant le prochain numéro de séquence attendu en réception, et la connexion reste dans le même état.

Si un segment entrant a un niveau de sécurité, ou compartiment, ou une priorité qui ne correspond pas exactement à la sécurité, et compartiment, et priorité requise pour la connexion, une réinitialisation est envoyée et la connexion se referme. L’accusé de réception du segment RST est renseigné avec la valeur de l’accusé de réception du segment entrant.

Traitement sur réinitialisation

Dans tous les cas, excepté SYN-SENT, tous les segments de réinitialisation (RST) sont validés en inspectant le champ SEQ. Une réinitialisation n’est reconnue que si elle intervient dans la fenêtre de réception. Dans l’état SYN-SENT (correspondant à un RST reçu en réponse à un segment SYN initial), le segment RST est considéré comme valide si son champ ACK acquitte le message SYN auquel il répond.

Le récepteur d’un segment RST commence par le valider, puis change d’état. Si le récepteur était dans l’état LISTEN, il l’ignore. Si le récepteur était dans l’état SYN-RECEIVED après être sorti de l’état LISTEN, alors il revient dans l’état LISTEN. Dans tous les autres cas, la connexion est abandonnée. L’application est informée de la rupture de la connexion.

3.5. Fermeture d’une connexion

CLOSE est une opération signifiant "Je n’ai plus d’autre données à envoyer". La notion de fermeture d’une communication bidirectionnelle peut être sujette à une interprétation ambiguë. L’analyse à faire du côté réception n’est pas de la plus grande simplicité. Nous avons choisi de l’expliquer de la façon la plus simple. L’application demandant la fermeture peut continuer à recevoir jusqu’à être averti que l’autre extrémité à fermé à son tour la connexion. Ainsi, un programme pourrait émettre plusieurs SEND suivi d’un CLOSE, et ensuite continuer à recevoir des données jusqu’à ce que TCP lui signale que le distant a fermé sa connexion. Du moins supposons nous que TCP le fera, même si aucune commande RECEIVE ne reste pendante, et ce dès que l’autre extrémité ferme la connexion, et permette à l’extrémité locale de terminer ses opérations proprement. TCP devra émettre toutes les données qui lui auront été passées par l’application avant l’activation de la commande CLOSE. Ainsi, une application peut sans problème fermer une connexion qui continuera à émettre des données et lancer une autre tâche. Par contre il faudra toujours lire les tampons de réception, jusqu’à ce que le TCP distant indique qu’il n’a plus de données à envoyer.

On distingue trois cas de figure principaux:

1. 1) L’application demande la fermeture au TCP en activant la commande CLOSE.
2. 2) La connexion est rompue par le distant qui marque son flag FIN
3. 3) Les deux applications coupent simultanément la connexion

Cas 1: L’application locale déclenche la déconnexion

Dans ce cas, un segment FIN est constitué et inscrit au bas de la pile d’émission. Aucune commande SEND ne sera plus acceptée par TCP, celui-ci passant à l’état FIN-WAIT-1. Les commandes RECEIVE restent acceptées dans cet état, la demi-connexion inverse fonctionnant toujours. Tous les segments présents dans la pile, y compris le dernier segment

FIN seront retransmis jusqu'à acquittement. Lorsque le TCP a acquitté le FIN, et envoyé son FIN propre, le local acquitte le FIN distant à son tour. Notez qu'un TCP recevant un segment FIN peut l'acquitter, mais n'enverra son propre segment FIN que lorsque son application aura exécuté la commande CLOSE.

Cas 2: TCP reçoit un segment FIN du distant

Si un segment FIN arrive inopinément par le réseau, TCP peut l'acquitter et avertit l'application de la notification de fermeture. L'application émettra une commande CLOSE après avoir pris ses dispositions, suite à laquelle TCP peut alors envoyer son propre FIN au TCP distant, une fois toutes les données restantes émises. TCP attend alors l'acquittement de ce FIN pour effacer le TCB de cette connexion. Si aucun acquittement ne survient après un certain laps de temps, la connexion est coupée et l'application en est avisée.

Cas 3: les deux applications ferment simultanément

Une commande CLOSE simultanée aux deux extrémités provoque un échange de segments FIN. Tous les segments de données restants, y compris le dernier FIN seront émis et acquittés, chaque TCP pourra acquitter le segment FIN reçu. Les deux côtés, sur acquittement, effaceront chacun le TCB de cette connexion.

```
    TCP A                                                TCP B
1.  ESTABLISHED                                          ESTABLISHED
2.  (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  --> CLOSE-WAIT
3.  FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>      <-- CLOSE-WAIT
4.                                                       (Close)
    TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>  <-- LAST-ACK
5.  TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>      --> CLOSED
6.  (2 MSL)       CLOSED
```

Séquence de fermeture normale
Figure 13.

```
    TCP A                                                TCP B
1.  ESTABLISHED                                          ESTABLISHED
2.  (Close)                                              (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  ... FIN-WAIT-1
                <-- <SEQ=300><ACK=100><CTL=FIN,ACK>  <--
                ... <SEQ=100><ACK=300><CTL=FIN,ACK>  -->
3.  CLOSING     --> <SEQ=101><ACK=301><CTL=ACK>      ... CLOSING
                <-- <SEQ=301><ACK=101><CTL=ACK>      <--
                ... <SEQ=101><ACK=301><CTL=ACK>      -->
4.  TIME-WAIT                                            TIME-WAIT
    (2 MSL)                                             (2 MSL)
    CLOSED                                               CLOSED
```

Séquence de fermeture bilatérale Figure 14.

3.6. Priorité et Sécurité

Le propos de ce chapitre est de ne permettre l'établissement d'une connexion qu'entre deux ports fonctionnant exclusivement avec les mêmes valeurs de sécurité et compartiment, et la priorité la plus élevée des deux côtés.

Les paramètres de priorité et de sécurité utilisés dans TCP sont exactement ceux définis par le protocole Internet (IP). Dans la présente spécification, le couple "sécurité/compartiment" désigne l'ensemble des paramètres de sécurité d'IP dont le niveau de sécurité, le compartiment, le groupe d'utilisateur, et les restrictions d'usage.

Une tentative de connexion avec des valeurs de sécurité/compartiment non concordantes, ou un niveau de priorité plus faible devra être rejetée via un segment de réinitialisation "reset". Le rejet d'une connexion uniquement du à une priorité plus faible ne pourra être exécuté qu'après acquittement en bonne et due forme du segment SYN..

Notez qu'un module TCP travaillant à la priorité par défaut devra néanmoins surveiller la priorité des segments entrants, et augmenter son niveau de priorité si requis.

Les paramètres de sécurité peuvent être exploités y compris en dehors d'un environnement sécurisé (les valeurs indiquent des données non classifiées), Ainsi, les ordinateurs doivent toujours être en mesure, dans un environnement sécurisé, de lire les informations de sécurité, même s'ils n'ont pas l'obligation de les transmettre.

3.7. Transfert de données

Une fois la connexion établie, les données peuvent commencer à être échangées dans des segments. Ceux-ci pouvant être rejetés (erreur de Checksum), ou perdus dans le réseau (congestion du réseau), TCP utilise un principe de retransmission (au bout d'une temporisation) pour garantir l'acheminement de tous les segments de données. Ce principe de retransmission peut conduire à la réception de "doublons". Comme il a été précisé dans les paragraphes consacrés aux numéros de séquence, TCP effectue un certain nombre de tests sur les séquences et les acquittements afin de déduire l'acceptabilité des données reçues.

L'émetteur des données garde toujours en mémoire le prochain numéro de séquence à utiliser dans la variable SND.NXT. Le récepteur garde en mémoire le prochain numéro de séquence à recevoir dans la variable RCV.NXT. L'émetteur garde en mémoire la valeur du plus ancien numéro de séquence non acquitté dans la variable SND.UNA. Si le flux de données s'interrompt pendant un temps suffisant pour que tous les segments sur le réseau soient parvenus à destination, ces trois valeurs doivent être égales.

Lorsque l'émetteur constitue un segment et le transmet, il incrémente la valeur de SND.NXT (souvenez-vous: modulo 2**32 !). Lorsque le récepteur reçoit un segment, il incrémente la valeur de RCV.NXT et envoie un acquittement. Lorsque l'émetteur reçoit cet acquittement, il incrémente SND.UNA. L'écart moyen entre ces variables est une mesure du "temps de propagation" à l'intérieur du réseau. A chaque fois, c'est la longueur du paquet qui est ajouté à ces variables. Notez que dans un état ESTABLISHED tous les segments transporteront l'information d'acquittement courant.

Le déclenchement d'une commande CLOSE impose l'utilisation de la fonction push, et l'ajout final d'un segment FIN à la pile de transmission.

Temporisation de retransmission

Du fait de la grande variété de réseaux formant Internet, et l'emploi massif de connexions TCP à travers ce réseau la valeur de temporisation de retransmission doit être déterminée dynamiquement. Nous donnons ici un exemple de procédure permettant de définir cette temporisation.

Exemple de procédure de calcul de temporisation de retransmission

Mesurer tout d'abord le temps entre l'envoi d'un octet de numéro de séquence particulier (appelons le "marqueur") et la réception d'un acquittement englobant ce numéro de séquence (les plages de séquence dans les deux segments n'ont pas à être identique). Ceci mesure ce que nous appellerons le "temps de boucle" (RTT = Round Trip Time). Puis calculez un "temps de boucle" corrigé (SRTT = Smoothed Round Trip Time), par la formule:

$$SRTT = (ALPHA * SRTT) + ((1-ALPHA) * RTT)$$

et enfin, calculer la temporisation de retransmission (RTO = Retransmission Time Out) comme suit:

$$RTO = min[UBOUND,max[LBOUND,(BETA*SRTT)]]$$

où UBOUND est une valeur limite de la temporisation (ex. 1 minute), LBOUND une valeur limite inférieure (ex.1 seconde), ALPHA le facteur de correction (ex. 0,8 à 0,9), et BETA est un facteur de variance du temps de propagation (ex., 1,3 à 2,0).

Transmission d'informations urgentes

L’objectif du mécanisme d’urgence de TCP est de fournir un moyen à l’application émettrice d’inciter le récepteur à prendre en compte des données urgentes, et un moyen pour signaler en retour que ces données urgentes ont bien été prise en compte par le récepteur.

Ce mécanisme permet de marquer un numéro de séquence particulier comme étant la fin d’un bloc de données urgentes. Lorsque ce marqueur pointe sur un numéro de séquence non encore reçu (RCV.NXT) par le TCP récepteur, ce dernier doit prévenir l’application de passer en "mode d’urgence"; et lorsque le numéro de séquence reçu rattrape ce marqueur, le TCP doit signifier à l’application de repasser en "mode normal". Si le pointeur d’urgence est réactualisé en "mode d’urgence", l’application utilisatrice ne pourra en avoir connaissance.

Ce mécanisme utilise un champ dédié dans chaque segment émis, appelé "pointeur urgent". Le bit de contrôle URG indique que ce champ contient une information valide. Le TCP récepteur ajoutera la valeur de ce champ au numéro de séquence du segment concerné pour obtenir le "marqueur". Lorsque le bit URG n’est pas marqué, ce champs n’est pas marqué, aucune mention d’urgence n’est contenue dans le segment.

Pour signifier l’état d’urgence, le segment émis doit au moins contenir un octet de données. Si l’émetteur utilise de plus la fonction push, le temps de transmission des données urgentes sera optimisé.

Gestion de la fenêtre de transmission

La fenêtre transmise dans chaque segment indique la plage de numéros de séquence que l’émetteur de la fenêtre (celui qui reçoit les données) est prête à accepter. La taille de cette fenêtre est en relation avec la taille disponible des tampons de données associés à cette connexion.

Une grande taille de fenêtre encourage l’émission. Si le nombre de données reçues est supérieur à ce que la fenêtre indique, les données hors fenêtre seront rejetées. Cette déperdition entraîne un grand nombre de retransmissions et surcharge inutilement le réseau et les TCP. L’usage d’une petite taille de fenêtre morcelle le débit en ajoutant un certain retard supplémentaire au "temps de boucle", mais en limitant la surcharge du réseau due aux retransmissions.

Le mécanisme instauré n’interdit pas à un TCP de signifier une largeur de fenêtre importante, puis beaucoup plus mince dans le segment suivant sans avoir reçu pour autant la quantité de données qui le justifie. Cette technique dit "d’étranglement de fenêtre," est fortement déconseillée. Le principe de robustesse nous dicte qu’un TCP ne doit pas de lui même étrangler une fenêtre, mais devra être préparé à accepter un tel comportement de la part d’autres TCP.

Le TCP émetteur doit être prêt à accepter de l’application et à envoyer au moins un octet de nouvelles données même lorsque la fenêtre de transmission est de largeur nulle. L’émetteur devra retransmettre régulièrement le segment au TCP récepteur. L’intervalle recommandé entre deux retransmissions, lorsque la largeur de fenêtre est nulle est d’environ 2 minutes. Cette retransmission est essentielle pour s’assurer que la réouverture de la fenêtre par le récepteur sera bien signifiée au TCP émetteur.

A l’autre bout, lorsque le TCP récepteur affiche une largeur de fenêtre nulle et qu’un segment arrive, il doit nécessairement répondre an acquittant le segment, un bon moyen de confirmer régulièrement le nouveau numéro de séquence attendu et la largeur de fenêtre courante (zéro).

Le TCP émetteur met en forme les données afin de constituer des segments à émette tenant dans l’espace de séquence autorisé par la fenêtre, et répète la même opération pour inscrire les copies de segments dans la pile de retransmission. Cette dernière opération n’est pas obligatoire, mais cependant très utile.

Pour une connexion n’utilisant qu’un seul flux de données unidirectionnel (mais nécessairement "ouverte" en bidirectionnel), la largeur de fenêtre est envoyée dans les acquittements sous un numéro de séquence unique (un segment ACK ne "consomme" pas d’espace de séquence). Il n’est donc pas possible de remettre dans l’ordre l’historique de la largeur de fenêtre. Cela ne représente pas un problème critique, bien qu’il signifie que le TCP émetteur puisse se baser sur d’anciennes valeurs de largeur de fenêtre lors d’une émission. Une technique pour s’affranchir de ce problème est de toujours récupérer l’information de fenêtre dans l’accusé de réception contenant le plus grand numéro de séquence (une méthode indirecte de numérotation des segments ACK !).

La gestion de la largeur de fenêtre a une influence importante sur les performances de la communication. Les commentaires qui suivent sont à destination des développeurs.

Suggestion pour gestion de la largeur de fenêtre

L'ouverture d'une toute petite fenêtre réduit les performances en augmentant le poids des en-têtes par rapport aux données.

Côté récepteur: Une suggestion pour éviter des fenêtres trop petites est de ne différer l'ouverture de la fenêtre jusqu'à ce qu'un certain pourcentage minimum X de l'espace total des tampons alloués à la communication n'ait été libéré (X peut être pris entre 20 et 40%).

Côté émetteur: L'émetteur peut attendre que la fenêtre atteigne une certaine largeur avant de commencer à envoyer des données. La fonction push permettra d'envoyer un reliquat de données même si la largeur de fenêtre est inférieure à la limite choisie.

Notez qu'il n'est pas une bonne stratégie de retarder l'émission des acquittements, au risque de provoquer des retransmissions inutiles et coûteuses en termes de performances. Une stratégie est d'émettre des acquittements dès qu'un segment court arrive (sans modifier la largeur de fenêtre), la largeur de fenêtre étant transmise dans les acquittements donnés pour des segments de taille supérieure.

Le segment envoyé pour tester une largeur de fenêtre nulle peut conduire à une "atomisation" de la transmission. Lorsque le segment testant cette information et contenant un seul octet de données est accepté (à la réouverture de la fenêtre), il consomme un octet du nouvel espace ouvert. Si l'émetteur envoie systématiquement autant de données qu'il peut lorsque la fenêtre est ouverte, la transmission va se stabiliser en une succession de petits et longs segments. Au fur et à mesure du temps, les variations de débit interne entre les tampons et la connexion conduit à une séquence de segments courts et "pas si longs que ça". Après un certain temps, la connexion est principalement constituée de segments courts.

La suggestion faite ici est que les implémentations de TCP doivent gérer attentivement la variation de largeur de fenêtre, les versions les plus simplistes conduisant toujours à une fragmentation excessive de la transmission.

3.8. Interfaces

Deux interfaces sont concernées dans ce chapitre: l'interface application/TCP et l'application TCP/réseau (la dernière via le protocole de niveau inférieur). Nous définirons assez précisément le protocole entre l'application et l'interface, et passerons sous silence l'interface entre TCP et la couche inférieure, dans la mesure où celle-ci est parfaitement définie dans la spécification de ce protocole. Dans le cas où ce protocole est IP, on remarquera quelques paramètres communs à ces deux protocoles.

Interface application/TCP

La description à suivre des commandes à envoyer à TCP sera tout au plus une spécification d'intention, dans la mesure où les ressources système et leur forme diffèrent considérablement d'un système à l'autre. Par conséquent, nous devons prévenir le lecteur que différentes implémentations de TCP peuvent présenter des interfaces distinctes. Cependant, il existera toujours un noyau de fonctions que TCP devra fournir à toute application, afin d'assurer la compatibilité globale du système multicouches. Ce chapitre décrit les commandes essentielles qu'un TCP se doit d'accepter.

Commandes applicatives vers TCP

Les paragraphes suivants décrivent les aspects fonctionnels de l'interface Application/TCP. La notation utilisée est très proche de la syntaxe habituellement acceptée pour les appels de fonctions par les langages de haut niveau, mais cet usage n'interdit pas l'utilisation d'appels sous forme condensée (ex., SVC, UUO, EMT).

Les commandes de base décrites ici sont essentielles pour qu'un TCP puisse supporter une communication inter-processus. Chaque implémentation concrète pourra adopter sa propre syntaxe, et pourra y adjoindre des

combinaisons ou fonctions partielles issues de ces fonctions de base. Par exemple, certains utilisateurs souhaiteront qu'une premier appel à la fonction SEND puisse automatiquement ouvrir la connexion (OPEN).

Dans son rôle d'intermédiaire de communication, TCP doit non seulement accepter des commandes, mais doit retourner un certain nombre d'informations à l'application, soit en réponse à une commande, soit de façon unilatérale. Ces dernières consistent en:

(a) une information générale sur la connexion (ex., interruptions, fermeture distante, connexion sur tel socket passif en attente).

(b) des réponses à des commandes applicatives, indiquant le succès ou l'échec pour telle ou telle raison.

Convention : Dans la formulation des commandes qui suivent, nous adoptons la convention de notation suivante pour les paramètres :

- Les paramètres de type bit (sémaphores) sont notés en MAJUSCULE.
- Les paramètres d'un autre type (adresse, entier, long, etc. sont notés en minuscule.

OPEN
Format:

OPEN (port_local, socket_distant, ACTIF/PASSIF [, tempo] [, priorité] [, sécurité/compartiment] [, options]) -> nom_local

Nous supposons ici que le TCP local connaît l'identité du processus qu'il sert, et vérifiera que ce processus est bien autorisé à ouvrir une connexion sur le socket spécifié. Suivant l'implémentation de TCP, les identificateurs TCP et réseau d'adresse source seront soit fournis par TCP soit par le protocole de routage (ex. IP). Ces considérations découlent d'une réflexion sur la sécurité, dont le but est d'interdire à un TCP de se faire passer pour un autre. De même, aucun processus ne peut se faire passer pour un autre sans la complicité des couches inférieures.

Si le paramètre ACTIF/PASSIF est marqué "passif", TCP se mettra dans l'état LISTEN et "écoutera" le réseau. Dans ce cas, le socket distant pourra soit être spécifié soit laissé "indéfini". Si le socket est spécifié, seule une demande de connexion provenant de ce socket pourra réveiller la connexion. Si le socket est laissé "indéfini", toute tentative de connexion distante sur cette connexion sera prise en compte. Une connexion passive peut être rendue active par l'envoi d'une commande SEND ultérieure.

Un bloc de contrôle de transmission (TCB) est créé, partiellement renseigné avec les paramètres passés par la commande OPEN.

Sur commande OPEN active, TCP démarrera immédiatement la procédure de synchronisation (c'est-à-dire, l'établissement).

Le paramètre de temporisation tempo, si précisé, permet de régler la limite maximale admise pour la transmission de données par TCP. Si les données n'ont pu être transmises au destinataire avant ce temps, TCP aura la charge de couper la communication. La valeur par défaut actuelle pour cette temporisation est de 5 minutes.

TCP ou un autre composant du système d'exploitation aura la charge de vérifier les droits de l'application à ouvrir une communication sur la priorité, la sécurité et le compartiment demandés. L'absence de paramètres à ce niveau implique l'utilisation des valeurs "par défaut".

TCP ne pourra accepter de requête entrante que dans la mesure où les informations de sécurité/compartiment correspondent exactement à celles précisées dans la commande OPEN, et la priorité y est au moins égale ou supérieure.

La priorité choisie pour la connexion sera la valeur maximum entre celle pr&eacut 1000 e;cisée dans la commande OPEN et celle arrivée par le réseau. Elle sera fixée pour toute la durée de vie de la connexion. Certains développeurs voudront pouvoir donner à l'application le pouvoir de négocier cette priorité. Par exemple, l'application pourra souhaiter de n'accepter la connexion que sur le même niveau de priorité, ou souhaitera que toute augmentation du niveau de priorité soit soumis à son approbation.

Un nom local sera retourné par TCP pour la connexion. Ce nom symbolique pourra être par la suite utilisé comme raccourci dans les appels faits à cette connexion définie par la paire <socket local, socket distant>.

SEND
Format:

SEND (nom_local, adresse_tampon, compteur, PUSH, URGENT [, tempo])

Cet appel permet l'envoi des données contenues dans le tampon de taille *compte* débutant à l'*adresse_tampon* sur la connexion définie par *nom_local*. Dans le cas général, si la connexion n'a pas été ouverte auparavant, cette commande génère une erreur. Certaines implémentations permettront l'usage direct de la commande SEND; celles-ci demanderont les paramètres supplémentaires utiles à l'établissement préalable de la connexion. Si l'application demanderesse n'est pas autorisée à ouvrir la connexion demandée, une erreur sera retournée.

Si l'indicateur PUSH est marqué, les données doivent être émises sans attendre vers le destinataire, et le flag PUSH du dernier segment TCP créé à partir de ce tampon sera marqué. Dans le cas contraire, les données pourront être momentanément stockées par TCP pour être assemblées à celles provenant des SEND suivants, si l'efficacité de la transmission le demande.

Si l'indicateur URGENT est marqué, les segments envoyés au TCP distant auront le flag URG marqué, indiquant la validité du "pointeur urgent". Le TCP récepteur signalera à l'application distante l'état d'urgence tant que toutes les données urgentes n'ont pas été récupérées par l'application destinataire. Le but de ce mécanisme et d'inciter l'application à traiter ces données en priorité, et d'aviser le récepteur que toutes les données de ce type ont été prises en compte. Le nombre de fois que le TCP émetteur signale le caractère d'urgence n'est pas nécessairement le même que celui que l'application reçoit du TCP distant.

Si aucun socket distant n'a été spécifié dans la commande OPEN, et la connexion établie malgré tout (ex. à la suite d'une requête de connexion quelconque arrivée sur ce socket), alors le tampon désigné est envoyé vers ce socket "implicite". Les applications utilisant la commande SEND sur une telle connexion n'ont pas le besoin de connaître la valeur du socket distant.

Par contre, si une commande SEND est exécutée **avant** que le socket distant ait été explicité, une erreur est retournée. L'application peut exécuter la commande STATUS pour déterminer l'état de la connexion. Dans certaines implémentations TCP indiquera à l'application lorsqu'une connexion passive "indéfinie" est explicitée.

Lorsqu'une tempo est présente, la valeur courante de la temporisation définie à la commande OPEN est remplacée par la nouvelle valeur.

Dans les implémentations les plus simples, la commande SEND ne rendra pas la main au processus appelant tant que la transmission n'est pas achevée ou, à défaut, la temporisation n'est pas écoulée. Cependant, cette technique simpliste provoque de nombreux temps morts, voire blocages (par exemple, si les deux côtés d'une transmission essaient d'envoyer des données avant de demander la première réception). Elle est donc déconseillée. Une technique plus sophistiquée rendra la main immédiatement à l'application, de sorte qu'elle continue son traitement concurrentiellement avec les routines d'entrées/sorties réseau, et, de plus permettre le lancement de plusieurs émissions successives (une sorte de "batch"). Des envois successifs seront traités dans ce cas sur le mode FIFO (premier arrivé premier servi), et TCP devra aménager une pile pour ceux qu'il ne pourra traiter immédiatement.

Nous venons implicitement de définir une interface de type "asynchrone" dans laquelle une commande SEND induit un SIGNAL ultérieur ou une pseudo interruption provenant du TCP. Une autre alternative est de répondre

immédiatement à l'application. Par exemple, TCP peut donner un acquittement local immédiat en réponse à la commande SEND de l'application, même si lui-même n'a pas encore reçu l'acquittement de réception par le distant, ni même envoyé les données. Nous pouvons avec optimisme penser que le succès est quasi garanti. Si nous avons été vraiment trop optimiste, la temporisation nous indiquera vite que nous nous sommes trop avancé, et coupera la connexion. Une implémentation de ce type (synchrone) ne pourra s'affranchir de quelques signaux asynchrones, mais ceci concernent l'état global de la connexion, plutôt que des informations particulières sur les tampons.

Afin que l'application puisse distinguer les retours d'erreur ou de succès de multiples SENDs, il paraît approprié d'ajouter à la réponse l'adresse du tampon concerné (celui qui aura été transmis par la commande SEND correspondante).

La signalisation TCP vers application est décrite ci-après, indiquant le type d'information qui doit être retourné à l'application appelante.

RECEIVE
Format:

RECEIVE (nom_local, adresse_tampon, compte) -> compte, URGENCE, PUSH

Cette commande alloue un tampon de réception à la connexion spécifiée. Si la connexion n'a pas été ouverte au préalable par une commande OPEN, ou l'application n'a pas l'autorisation d'utiliser une telle connexion, une erreur sera renvoyée.

Dans une implémentation simpliste, la main ne sera pas redonnée à l'application tant que le tampon n'aura pas été rempli, ou une erreur survenue. Ce schéma est malheureusement souvent bloquant. Une implémentation plus sophistiquée consisterait à pouvoir mettre en attente plusieurs commandes de réception. Les tampons associés se remplissent alors dès que les segments concernés arrivent par le réseau. Cette stratégie permet un débit d'information plus rapide, au prix d'un mécanisme plus élaboré (éventuellement asynchrone) destiné à avertir l'application qu'une fonction push a été activée, ou que le tampon est plein.

Si une quantité de données suffisante est reçue avant qu'un flag PUSH n'apparaisse, l'indicateur PUSH ne sera pas marqué dans la réponse à la commande RECEIVE. Le tampon contiendra autant de données qu'il peut en contenir. Si un flag PUSH arrive du réseau, le tampon sera retourné partiellement rempli, et l'indicateur PUSH marqué dans la réponse.

Si des données urgentes sont présentées par le réseau, l'application en sera avertie immédiatement par un signal asynchrone de l'interface TCP vers application. Cette dernière doit passer en "mode urgent". Tant que l'indicateur URGENCE est marqué dans la réponse à une commande RECEIVE, des données urgentes restent latentes dans les tampons de TCP. Lorsque l'indicateur URGENCE n'est plus marqué, les dernières données urgentes ont été transférées dans le tampon associé et l'application peut repasser en "mode normal". Notez que les données "normales" suivantes ne peuvent être transmises dans le même tampon (mais seront disponibles via la commande RECEIVE suivante), à moins que la limite ne soit clairement marquée.

Pour discriminer la réponse à plusieurs commandes RECEIVE en instance et pour s'assurer que les tampons à récupérer ont été complètement remplis, la réponse émise sous forme de signal, après réception, est accompagnée d'un pointeur sur le tampon et de la taille occupée dans celui-ci.

D'autres implémentations de la commande RECEIVE peuvent directement adresser l'espace mémoire alloué en interne par TCP au tampon de réception, ou encore mettre en place le partage d'un tampon tournant par les deux partenaires.

CLOSE
Format:

CLOSE (nom_local)

Cette commande demande la fermeture de la connexion spécifiée. Si la connexion n'existe pas, ou l'application n'a pas l'autorisation nécessaire pour utiliser cette connexion, une erreur est renvoyée. La fermeture de connexions est prévue pour se dérouler de façon "propre" dans la mesure ou toutes les émissions en instance seront menées à leur terme (et retransmises si nécessaire),selon les impératifs du contrôle de flux. Ainsi, il est légitime de lancer plusieurs commandes SEND successives suivies d'une commande CLOSE, et de considérer que le message a été correctement et complètement transmis. Il est aussi clair que la communication doit être toujours "écoutée", afin de récupérer tout ce que le distant a à transmettre. De ce fait, la commande CLOSE signifie "je n'ai plus rien à transmettre", et non pas "je ne veux plus rien recevoir". Il peut arriver (notamment si le protocole au niveau application n'est pas suffisamment complet) que le côté fermant la connexion ne puisse se débarrasser de toutes ses données avant intervention de la temporisation. Dans ce cas, la commande CLOSE revient à une commande ABORT, et TCP finit le travail de fermeture.

L'application doit pouvoir fermer la connexion à tout moment de sa propre initiative, ou en réponse à des signaux venant de TCP (ex. signal de déconnexion distante, temporisation de transmission écoulée, destination inaccessible).

Comme la fermeture d'une connexion demande un échange avec le TCP distant, celle-ci peut rester dans l'état "encours de fermeture" (CLOSING) pendant un petit laps de temps. Toute tentative de réouverture de cette connexion avant que TCP n'ait pu répondre à la commande CLOSE générera une erreur.

L'exécution d'une commande CLOSE déclenche implicitement une fonction push.

STATUS
Format:

STATUS (nom_local) -> état

Cette commande dépend de l'implémentation et peut être "oubliée" sans conséquence majeure. L'information renvoyée est essentiellement issue des données du TCB associé à la connexion.

Cette commande renvoie habituellement une structure de données comprenant les information suivantes:

```
socket local,
socket distant,
nom local de la connexion,
fenêtre en réception,
fenêtre en émission,
état de la connexion,
nombre de tampons attendant un acquittement,
nombre de tampons en attente de réception,
état d'urgence,
priorité,
sécurité/compartiment,
temporisation de transmission courante.
```

Suivant l'état courant de la connexion ou l'implémentation, certaines de ces données ne peuvent être obtenues, ou significatives. Si l'application n'a pas l'autorisation pour adresse cette connexion, une erreur est renvoyée. Ceci permet d'éviter de diffuser des informations sur une connexion à une application non autorisée.

ABORT
Format:

ABORT (nom_local)

Cette commande abandonne toutes les commandes SEND et RECEIVE en cours. Le TCB est effacé, et un message d'abandon est envoyé au TCP distant. Suivant l'implémentation, l'application recevra des indications en retour sur chacune des émissions et réceptions abandonnées, ou simplement un acquittement global de l'abandon.

Messages TCP vers utilisateur

Il est prévu que le système d'exploitation fournisse un moyen pour que TCP puisse signaler des événements de façon asynchrone à l'application qui l'utilise. Lorsque TCP envoie un tel signal, des informations y prennent place. Il arrivera souvent qu'on y trouve un statut d'erreur. Dans les autres cas, on y trouvera un rapport concernant le succès de telle ou telle commande SEND, RECEIVE, ou autre.

Les information suivantes y apparaissent:

```
nom local de connexion              Toujours
Libellé de la réponse               Toujours
Adresse du tampon                   SEND & RECEIVE
compteur (taille reçue)             RECEIVE
Indicateur PUSH                     RECEIVE
Indicateur URGENCE                  RECEIVE
```

Interface TCP vers couche inférieure

TCP effectue des appels vers une couche inférieure du protocole de communication, chargée de "router" les segments. Les réseaux ARPAnet et Internet s'appuient sur le protocole Internet (IP).

Lorsque le protocole de niveau inférieur est I, TCP ajoute aux segments deux informations: le type de service et la durée de vie. TCP donne pour ces informations les valeurs suivantes:

Type de service = Priorité: routine, Delay: normal, Débit: normal, Fiabilité: normal; soit 00000000.

Durée de vie = une minute, soit 00111100.

Notez que la durée de vie maximale encodable pour un segment est de deux minutes. Par cette information, nous indiquons de manière explicite que tout segment doit être détruit s'il n'a pas été acheminé au bout d'une minute.

Si le protocole inférieur est IP (ou tout autre protocole disposant de cette fonction) et le "routage retour" (vers la source) est exploité, L'interface doit permettre de communiquer le chemin pris. Ceci est fondamental pour que les adresses source et destinataire utilisées pour le décodage du Checksum TCP soient bien les adresses originales de l'émetteur et celle de l'expéditeur (et non l'adresse du dernier routeur par lequel a transité le message). Il est aussi important de garder la trace du chemin pour permettre d'établir la connexion inverse.

Tout protocole sur lequel s'appuiera TCP doit pouvoir fournir l'adresse source, destinataire, les champs de protocole, et une façon de déterminer la longueur "TCP length", lesquels garantissent ainsi une compatibilité avec les services offerts par IP, et nécessaires au calcul du Checksum.

3.9. Traitement des événements

Les procédures décrites dans ce chapitre ne sont qu'un exemple d'implémentation possible. D'autres implémentations pourront s'appuyer sur des séquences différentes, mais seulement en détail, et non dans l'intention.

Le principal de l'activité de TCP peut être assimilé comme la réponse à des événements. Les événements pouvant survenir peuvent être répartis en trois catégories: commandes de l'application, arrivée de segments, et écoulement de temporisations. Ce chapitre décrit dans le détail tout ce que TCP exécute sur réception de chacun de ces événements. Dans la plupart des cas, la réaction de TCP dépendra de l'état de la connexion.

Evénements susceptibles de survenir:

Appels applicatifs	**Arrivée d'un segment**	**Temporisations**
OPEN	SEGMENT ARRIVES	USER TIMEOUT
SEND		RETRANSMISSION TIMEOUT

RECEIVE TIME-WAIT TIMEOUT
STATUS
CLOSE
ABORT

Le modèle choisi pour l'interface TCP/application est celui où TCP répond immédiatement à la commande, quitte à fournir un complément de réponse différé par le biais d'un événement ou d'une pseudo-interruption. Dans tout ce qui suit, on nommera "réponse" la réponse immédiate, et "signal" la réponse différée.

Les messages d'erreur sont indiqués sous leur forme explicite. Par exemple, une application émettant une commande pour une connexion inexistante recevra le message d'erreur "erreur: connexion inexistante".

Notez que dans tout ce qui suit, l'arithmétique utilisée pour les calculs sur les numéros de séquence, numéros d'acquittement, fenêtres, etc., est modulo 2**32 (la taille de l'espace de numérotation). Notez que le symbole "=<" signifie inférieur ou égal modulo 2**32.

Le traitement des segments entrants suppose qu'ils ont été au préalable validés en termes de séquence (c'est-à-dire que leur plage de séquence est effectivement incluse dans la fenêtre de transmission courante) et qu'ils sont de ce fait empilés, puis traités par ordre de séquence.

Lorsqu'un segment reçu affiche une plage de séquence qui recoupe celle d'un segment précédemment reçu, nous reconstituons ce segment en ne gardant que les "nouvelles" données, et réajustons les paramètres de son en-tête en conséquence.

Notez enfin qu'en l'absence d'une mention explicite de changement d'état, TCP reste dans l'état où il était avant l'événement.

Commande OPEN

Etat précédent : CLOSED STATE (TCB inexistant)

- Créer un nouveau bloc de contrôle de transmission (TCB) pour mémoriser les informations sur la connexion.
- Remplit l'identificateur du socket local, du socket distant, la priorité, la sécurité/compartiment, et la temporisation "application". Certaines sous adresses du socket distant peuvent ne pas être remplies lorsque la commande OPEN est exécutée en mode "passif". Celles-ci seront explicitées lors de l'arrivée d'une requête de connexion via le réseau.
- Vérifie les autorisations système pour la priorité et la sécurité m
    - si non autorisée
        - Répondre "erreur: priorité non attribuée" ou "erreur: sécurité/compartiment non autorisé".
        - **Retour**
- Si passive,
    - entre dans le mode LISTEN.
    - **Retour**
- Si active,
    - si le socket distant est non spécifié, ou "indéfini",
        - Répondre "erreur: socket distant non spécifié".
        - **Retour**
    - si le socket distant est valide,
        - Composer un segment SYN. Choisir un numéro initial de séquence (ISS).
        - Emettre un segment SYN de forme <SEQ=ISS><CTL=SYN>.
        - Renseigner SND.UNA par l'ISS, SND.NXT par ISS+1,
        - Entrer en mode SYN-SENT.
        - **Retour**

- si l’application n’a pas le droit d’accès au socket spécifié,
  - Répondre "erreur: connexion illégale pour cette application".
  - **Retour**
- si manque de mémoire pour créer la connexion,
  - Répondre "erreur: ressources insuffisantes ".
  - **Retour**

Etat précédent : LISTEN

- Si active
  - Si le socket distant est spécifié,
    - Changer la connexion de passive à active.
    - Choisir un ISS.
    - Envoyer un segment SYN.
    - Renseigner SND.UNA par l’ISS, SND.NXT par ISS+1.
    - Entrer dans l’état SYN-SEN. Des données arrivées par une première commande SEND peuvent être envoyées dans le segment SYN ou empilées pour transmission après passage à l’état ESTABLISHED. Le bit URG doit être marqué si spécifié dans le première commande SEND.
    - **Retour**
  - Si manque de mémoire pour empiler cette commande,
    - Répondre "erreur: ressources insuffisantes".
    - **Retour**
  - Si le socket distant non spécifié ou "indéfini",
    - Répondre "erreur: socket distant non spécifié".
    - **Retour**

Etats précédents : SYN-SENT STATE, SYN-RECEIVED STATE, ESTABLISHED STATE, FIN-WAIT-1 STATE, FIN-WAIT-2 STATE, CLOSE-WAIT STATE, CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

- Répondre "erreur: la connexion existe déjà".
- **Retour**

Commande SEND

Etat précédent : CLOSED STATE (TCB inexistant)

- Si l’application n’a pas l’autorisation d’accès à cette connexion,
  - Répondre "erreur: connexion illégale pour cette application".
  - **Retour**
- Autrement
  - Répondre "erreur: connexion inexistante".
  - **Retour**

Etat précédent : LISTEN

- Si manque de mémoire pour empiler les données,
  - Répondre "erreur: ressources insuffisantes".
  - **Retour**
- Si le socket distant non spécifié ou "indéfini"
  - Répondre "erreur: socket distant non spécifié".
  - **Retour**

- Si le socket distant est spécifié,
  - Passer la connexion de l'état passif à actif.
  - Choisir un ISS.
  - Emettre un segment SYN.
  - Renseigner SND.UNA par l'ISS, SND.NXT par ISS+1.
  - Entrer dans l'état SYN-SENT. Les données jointes peuvent être envoyées dans le segment SYN ou empilées pour transmission après passage à l'état ESTABLISHED. Le bit URG doit être marqué si spécifié dans le première commande SEND.
  - **Retour**

Etats précédents : SYN-SENT STATE, SYN-RECEIVED STATE

- Si manque de mémoire pour empiler les données,
  - Répondre "erreur: ressources insuffisantes".
  - **Retour**
- Sinon
  - Empiler les données pour transmission une fois l'état ESTABLISHED établi.
  - **Retour**

Etats précédents : ESTABLISHED STATE, CLOSE-WAIT STATE

- Si manque de mémoire pour conserver les données,
  - Répondre "erreur: ressources insuffisantes".
  - **Retour**
- Segmenter le tampon et l'envoie avec un acquittement (valeur = RCV.NXT).
- Si l'indicateur URGENT est marqué, alors SND.UP <- SND.NXT-1 et le pointeur d'urgence est activé dans les segments sortants.
  - **Retour**

Etats précédents : FIN-WAIT-1 STATE, FIN-WAIT-2 STATE, CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

  - Répondre "erreur: connexion en fermeture" et ignorer les requêtes ultérieures.

Commande RECEIVE

Etat précédent : CLOSED STATE (TCB inexistant)

- Si l'application n'a pas l'autorisation d'accès à cette connexion,
  - Répondre "erreur: connexion illégale pour cette application".
  - **Retour**
- Autrement
  - Répondre "erreur: connexion inexistante".
  - **Retour**

Etats précédents : LISTEN STATE, SYN-SENT STATE, SYN-RECEIVED STATE

- Si manque de mémoire pour empiler cette requête,
  - Répondre "erreur: ressources insuffisantes".
  - **Retour**
- Empiler la requête pour traitement une fois l'état ESTABLISHED établi.

Etats précédents : ESTABLISHED STATE, FIN-WAIT-1 STATE, FIN-WAIT-2 STATE

- Si un nombre insuffisant de segments sont arrivés,
  - Empiler la requête.
  - **Retour**
- Si manque de mémoire pour empiler cette requête,
  - Répondre "erreur: ressources insuffisantes".
  - **Retour**
- Réassembler les segments empilés dans le tampon de réception et donner la réponse à l'application. Marquer "push seen" (PUSH) le cas échéant.
- Si RCV.UP pointe sur des données non encore reçues, notifier l'application de la présence de données urgentes.

Lorsque TCP prend la responsabilité de passer des données à l'application, il doit envoyer un acquittement à l'émetteur. La construction de cet acquittement est traitée ci-après aux paragraphes concernant l'arrivée d'un segment.

Etat précédent : CLOSE-WAIT STATE

Comme le distant a déjà envoyé son segment FIN, la commande passe à l'application les données encore stockées par TCP.

- Si plus aucune donnée n'est à délivrer,
  - Répondre "erreur: déconnexion en cours".
  - **Retour**
- Autrement toute les données restantes sont utilisables pour satisfaire la requête.

Etats précédents : CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

  - Répondre "erreur: déconnexion en cours".
  - **Retour**

Commande CLOSE

Etat précédent : CLOSED STATE (TCB inexistant)

- Si l'application n'a pas l'autorisation d'accès à cette connexion,
  - Répondre "erreur: connexion illégale pour cette application".
  - **Retour**
- Autrement
  - Répondre "erreur: connexion inexistante".
  - **Retour**

Etat précédent : LISTEN

  - Toute requête RECEIVE empilée provoque une sortie d'erreur
  - Répondre "erreur: déconnexion".
  - Effacer le TCB.
  - Passer la connexion à l'état CLOSED.
  - **Retour**

Etat précédent : SYN-SENT STATE

  - Efface le TCB et retourne "erreur: déconnexion" pour chaque requête SEND ou RECEIVE latente.

Etat précédent : SYN-RECEIVED STATE

- Si aucune commande SEND n’a été lancée et qu’il ne reste plus de données à transmettre
  - Constituer un segment FIN et l’émettre
  - Passer dans l’état FIN-WAIT-1
  - **Retour.**
- Autrement
  - Empiler la requête en attente de passage dans l’état ESTABLISHED.
  - **Retour.**

Etat précédent : ESTABLISHED STATE

  - Place la requête en attente, le temps de segmenter toutes les requêtes SEND en attente
  - Forme un segment FIN et l’envoie.
  - Passe la connexion à l’état FIN-WAIT-1 dans tous les cas.

Etats précédents : FIN-WAIT-1 STATE, FIN-WAIT-2 STATE

- Strictement parlant, il s’agit d’une erreur et devrait provoquer une sortie "erreur: déconnexion en cours". Une réponse "OK" serait aussi bien acceptable, jusqu’à ce que le deuxième FIN soit émis (le premier segment Fin peut encore être retransmis).

Etat précédent : CLOSE-WAIT STATE

  - Placer la requête en attente, le temps de segmenter toutes les requêtes SEND en attente,
  - Former un segment FIN et l’envoie.
  - Passer la connexion à l’état CLOSING.
  - **Retour.**

Etats précédents : CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

  - Retourner "erreur: déconnexion en cours".
  - **Retour.**

Commande : ABORT

Etat précédent : CLOSED STATE (TCB inexistant)

- Si l’application n’a pas d’autorisation d’accès à cette connexion,
  - Répondre "erreur: connexion illégale pour cette application".
  - **Retour.**
- Autrement
  - Répondre "erreur: connexion inexistante".
  - **Retour.**

Etat précédent : LISTEN

  - Toute requête RECEIVE en attente recevra un signal "erreur: abandon" en réponse.
  - Effacer le TCB.
  - Passer la connexion à l’état CLOSED.
  - **Retour.**

Etat précédent : SYN-SENT STATE

- Toute requête SEND ou RECEIVE en attente recevra un signal "erreur: abandon" en réponse.
- Effacer le TCB.
- Passer la connexion à l'état CLOSED.
- **Retour.**

Etats précédents : SYN-RECEIVED STATE, ESTABLISHED STATE, FIN-WAIT-1 STATE, FIN-WAIT-2 STATE, CLOSE-WAIT STATE

- Envoyer un segment de réinitialisation: <SEQ=SND.NXT><CTL=RST>
- Toute requête SEND ou RECEIVE en attente recevra un signal "erreur: abandon" en réponse.
- Libérer tous les tampons liés aux commandes RECEIVE et SEND en attente,
- Supprimer les segments constitués excepté le segment RST ci-dessus.
- Effacer le TCB.
- Passer la connexion dans l'état CLOSED.
- **Retour**

Etat précédent : CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

- Répondre "OK"
- Effacer le TCB,
- Passer la connexion en mode CLOSED.
- **Retour.**

Commande STATUS

Etat précédent : CLOSED STATE (TCB inexistant)

- Si l'application n'a pas d'autorisation d'accès à cette connexion,
  - Répondre "erreur: connexion illégale pour cette application".
  - **Retour.**
- Autrement
  - Répondre "erreur: connexion inexistante".
  - **Retour.**

Etat précédent : LISTEN

- Renvoie "état = LISTEN", et un pointeur sur le TCB. **Retour**

Etat précédent : SYN-SENT STATE

- Renvoie "état = SYN-SENT", et un pointeur sur le TCB. **Retour**

Etat précédent : SYN-RECEIVED STATE

- Renvoie "état = SYN-RECEIVED", et un pointeur sur le TCB. **Retour**

Etat précédent : ESTABLISHED STATE

- Renvoie "état = ESTABLISHED", et un pointeur sur le TCB. **Retour**

Etat précédent : FIN-WAIT-1 STATE

- Renvoie "état = FIN-WAIT-1", et un pointeur sur le TCB. **Retour**

Etat précédent : FIN-WAIT-2 STATE

- Renvoie "état = FIN-WAIT-2", et un pointeur sur le TCB. **Retour**

Etat précédent : CLOSE-WAIT STATE

- Renvoie "état = CLOSE-WAIT", et un pointeur sur le TCB. **Retour**

Etat précédent : CLOSING STATE

- Renvoie "état = CLOSING", et un pointeur sur le TCB. **Retour**

Etat précédent : LAST-ACK STATE

- Renvoie "état = LAST-ACK", et un pointeur sur le TCB. **Retour**

Etat précédent : TIME-WAIT STATE

- Renvoie "état = TIME-WAIT", et un pointeur sur le TCB. **Retour**

Arrivée réseau : SEGMENT ARRIVES

Etat précédent : CLOSED (c'est-à-dire, TCB n'existe pas)

- Toutes les données du segment sont ignorées. Un segment RST est ignoré.. Un segment autre que RST entraîne l'émission d'un segment RST en réponse. Le numéro de séquence et l'accusé de réception sont renseignés de façon à ce que le segment soit acceptable par le TCP qui a envoyé le segment erroné.
  - Si le bit ACK n'est pas marqué, [SEQ=0][ACK=SEG.SEQ+SEG.LEN][CTL=RST,ACK]
  - Si le bit ACK est marqué, [SEQ=SEG.ACK][CTL=RST]
  - **Retour**

Etat précédent : LISTEN

1. Vérifier le marquage du bit RST
   - Un segment RST entrant sera ignoré. **Retour.**
2. Vérifier le marquage du bit ACK
   - Un accusé de réception reçu par une connexion à l'état LISTEN représente une faute. Un segment RST acceptable doit être constitué pour tout acquittement hors contexte. Le RST doit être constitué comme suit: <SEQ=SEG.ACK><CTL=RST>
   - **Retour.**
3. Vérifier le marquage de SYN
   - Si le bit SYN est marqué,
     vérifier la sécurité:
     - Si la donnée de sécurité/compartiment sur le segment entrant ne correspond pas exactement à celle marquée dans le TCB,
       - Envoi d'un segment RST au distant. : [SEQ=SEG.ACK][CTL=RST]
       - **Retour.**
     - Si SEG.PRC est supérieur à TCB.PRC (test de priorité)
       - si non autorisé,
         - envoi d'un segment RST: [SEQ=SEG.ACK][CTL=RST].
         - **Retour.**
       - si autorisé par le système et l'application TCB.PRC<-SEG.PRC.

- Continuer
- Si SEG.PRC est inférieur à TCB.PRC
  - Continuer.

Renseigner RCV.NXT par SEG.SEQ+1, IRS par SEG.SEQ tous les autres contrôles et données doivent être empilés pour traitement ultérieur. Un ISS doit être choisi et un segment SYN émis, sous la forme: [SEQ=ISS][ACK=RCV.NXT][CTL=SYN,ACK] Renseigner SND.NXT par ISS+1 et SND.UNA par ISS. Passer la connexion à l'état SYN-RECEIVED. Tous les autres contrôle ou données seront examinés et traités dans l'état SYN-RECEIVED, sauf SYN et ACK qui auront déjà été traités. Si la connexion était totalement ou partiellement indéfinie, la connexion est alors explicitée à ce moment.

4. Autres données et contrôles

- Tout autre contrôle ou segments portant des données (sans SYN) sont nécessairement marqués en tant que ACK et seront donc écartés par le traitement des acquittements. Un segment RST entrant ne peut pas être valide, car il ne peut correspondre à aucune émission faite par cette instance de la connexion. Ce cas ne devrait jamais arriver. Si par hasard il se produit, rejetez le segment. **Retour.**

Etat précédent : SYN-SENT

1. vérifier le marquage du bit ACK
   - Si ACK est marqué
     - Si SEG.ACK =\< ISS, ou SEG.ACK \> SND.NXT,
       - envoyer un RST (sauf si le segment est lui même un RST auquel cas il doit être ignoré et Retour) [SEQ=SEG.ACK][CTL=RST]
       - détruire le segment.
       - **Retour.**
     - Si SND.UNA =< SEG.ACK =< SND.NXT l'acquittement est acceptable.

2. Vérifier le marquage du bit RST
   - Si RST est marqué
     - Si l'acquittement est conforme,
       - avertir l'application par "erreur: abandon distant",
       - détruire le segment,
       - passer la connexion en état CLOSED,
       - effacer le TCB
       - **Retour.**
     - Sinon (non ACK)
       - Ignorer le segment.
       - **Retour.**

   - Vérifier la sécurité et la priorité
     - Si la valeur de sécurité compartiment ne correspond pas à celle inscrite dans le TCB,
       - Si ACK est marqué
         - Envoi d'un RST : [SEQ=SEG.ACK][CTL=RST]
       - Sinon
         - Envoi d'un RST : [SEQ=0][ACK=SEG.SEQ+SEG.LEN][CTL=RST,ACK]
     - Sinon
       - Si ACK est marqué
         - Si la priorité dans le segment ne correspondre pas à celle inscrite dans le TCB
           - envoi d'un segment RST: [SEQ=SEG.ACK][CTL=RST]
       - sinon

         Si la priorité du segment est supérieure à celle inscrite dans le TCB et

- si le système et l'application le permettent,
  - remonter la priorité de la connexion inscrite dans le TCB au niveau de celle trouvée dans le segment
- Si il n'est pas autorisé de monter le niveau de priorité
  - envoyer un RST [SEQ=0][ACK=SEG.SEQ+SEG.LEN][CTL=RST,ACK]
- Si la priorité dans le segment est inférieure à celle inscrite dans le TCB
  - continuer.
- Si un RST a été envoyé,
  - détruire le segment
  - **Retour.**

3. Vérifier le marquage du bit SYN

Cette étape ne peut être atteinte que si l'ACK est valide, ou il ne s'agit pas d'un ACK ni d'un segment RST.

- Si le bit SYN est marqué et les conditions de sécurité/compartiment et priorité sont vérifiées
  - RCV.NXT prend la valeur SEG.SEQ+1,
  - IRS prend la valeur SEG.SEQ.
  - Si ACK est marqué : SND.UNA est incrémenté jusqu'à la valeur SEG.ACK tous les segments de la pile de retransmission qui ont de ce fait été acquittés sont supprimés.
    - Si SND.UNA > ISS (notre propre SYN a été acquitté),
      - passer la connexion à l'état ESTABLISHED,
      - Emettre l'accusé de réception [SEQ=SND.NXT][ACK=RCV.NXT][CTL=ACK] Les données et contrôles en attente dans la pile d'émission peuvent être transmis dans ce segment.
        - Si des données sont contenues dans le segment
          - Aller au pas 6 ci-après, où le bit URG est vérifié,
        - Sinon
          - **Retour.**
    - Autrement (notre SYN n'est pas encore acquitté)
      - passer en mode SYN-RECEIVED,
      - former un segment SYN,ACK sous la forme [SEQ=ISS][ACK=RCV.NXT][CTL=SYN,ACK] et l'envoyer.
      - Si d'autres contrôles ou données sont présentes dans le segment,
        - les empiler en attendant le passage à l'état ESTABLISHED.
        - **Retour.**
- Si aucun des bits SYN ou RST n'est marqué
  - rejeter le segment
  - **Retour.**

Autrement,

Etats précédents : SYN-RECEIVED STATE,ESTABLISHED STATE, FIN-WAIT-1 STATE,FIN-WAIT-2 STATE,CLOSE-WAIT STATE,CLOSING STATE,LAST-ACK STATE,TIME-WAIT STATE

1. Tester le numéro de séquence

Les segments sont traités dans l'ordre de la séquence. Les premiers tests servent à éliminer les anciennes données dupliquées, les traitements suivants sont faits selon l'ordre des SEG.SEQ. Si segment transporte des données dont une partie est ancienne et l'autre nouvelle, seule la partie nouvelle sera prise en compte.

Il y a quatre cas d'acceptabilité des segments:

Longueur du Segment	Fenêtre de Réception	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	non acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

Si RCV.WND vaut zéro, aucun segment ne peut être accepté, mais des dérogations doivent être prévues pour des segments ACK, URG et RST valides.

- Si le segment n'est pas acceptable,
  - s'il s'agit d'un segment RST
    - Jeter le segment
    - **Retour.**
  - sinon,
    - Emettre un acquittement [SEQ=SND.NXT][ACK=RCV.NXT][CTL=ACK]
    - Jeter le segment
    - **Retour.**

Dans la suite, nous traitons le cas d'un segment "idéal" de premier numéro de séquence RCV.NXT et dont la longueur n'excède pas la largeur de la fenêtre. Il serait possible, si cette longueur était supérieure, de découper ce segment en deux parties, l'une de longueur égale à la largeur fenêtre (y compris pour les segments SYN et FIN), et traiter le segment commençant à RCV.NXT. L'autre partie "hors fenêtre" serait alors empilée pour traitement ultérieur.

2. Vérifier le bit RST,

Etat précédent : SYN-RECEIVED STATE

- Si le bit RST est marqué
  - Si l'ouverture de connexion était passive (c'est-à-dire, était passée par un état LISTEN) Retourne la connexion à l'état passif LISTEN. L'application n'en est pas nécessairement avisée.
- Si l'ouverture de la connexion était active (c'est-à-dire, étant passée par l'état SYN-SENT) alors la requête de connexion a été refusée par le distant.
  - TCP signale à l'application "Connexion rejetée".
  - Passer la connexion à l'état CLOSED
  - Supprimer le TCB.
- Dans les deux cas,
  - Supprimer tous les segments de la pile de retransmission.
  - **Retour.**

Etats précédents : ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT

- Si le bit RST est marqué
  - Signaler "erreur : abandon" pour toute commande SEND ou RECEIVE en instance.

- Vider toutes les piles internes.
- L'application peut de plus recevoir un signal "abandon" global.
- Passer la connexion à l'état CLOSED,
- Supprimer le TCB.
- **Retour.**

Etats précédents : CLOSING STATE, LAST-ACK STATE, TIME-WAIT

- Si le bit RST est marqué
  - Passer la connexion à l'état CLOSED
  - Supprimer le TCB
  - **Retour.**

3. Vérifier la sécurité et la priorité

Etat précédent : SYN-RECEIVED

- Si la valeur de sécurité/compartiment ne correspond pas exactement à celle inscrite dans le TCB,
  - mettre un segment RST
  - **Retour.**

Etat précédent : ESTABLISHED STATE

- Si la valeur de sécurité/compartiment ne correspond pas exactement à celle inscrite dans le TCB,
  - Emettre un segment RST
  - Signaler "erreur : abandon" pour toute commande SEND ou RECEIVE en instance.
  - Vider toutes les piles internes.
  - L'application peut de plus recevoir un signal "abandon" global.
  - Passer la connexion à l'état CLOSED,
  - Supprimer le TCB.
  - **Retour.**

Notez que ce test est placé après le test de séquence, afin d'éviter qu'un segment d'une ancienne connexion entre ces deux ports, avec un niveau de priorité et de sécurité distinct ne vienne provoquer l'abandon intempestif de la connexion courante.

4. Vérifier le bit SYN

Etats précédents : SYN-RECEIVED, ESTABLISHED STATE, FIN-WAIT STATE-1, FIN-WAIT STATE-2, CLOSE-WAIT STATE, CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

- Si le segment SYN est dans la fenêtre, ce ne peut être qu'une erreur,
  - Emettre un RST,
  - Signaler "erreur : abandon" pour toute commande SEND ou RECEIVE en instance.
  - Vider toutes les piles internes.
  - L'application peut de plus recevoir un signal "abandon" global.
  - Passer la connexion à l'état CLOSED
  - Supprimer le TCB
  - **Retour.**

Le cas où le segment SYN n'est pas dans la fenêtre correspond à celui où un ACK a été renvoyé dans le premier pas (vérification des numéros de séquence).

5. Vérifier le marquage du bit ACK
    - Si le bit ACK n'est pas marqué
        - Jeter le segment
        - **Retour.**
    - Si le bit ACK est marqué

Etat précédent : SYN-RECEIVED STATE

- Si SND.UNA =< SEG.ACK =< SND.NXT
    - Passer la connexion à l'état ESTABLISHED
    - **continuer**
- Si l'accusé de réception n'est pas acceptable,
    - Emettre un RST : [SEQ=SEG.ACK][CTL=RST]

Etat précédent : ESTABLISHED STATE

- Si SND.UNA < SEG.ACK =< SND.NXT
    - SND.UNA = SEG.ACK.
    - Retirer de la pile de retransmission tous les segments acquittés.
    - Renvoyer un signal d'aquittement à l'application pour chaque commande SEND complètement envoyée et
    - acquittée (en renvoyant le tampon SEND avec la mention "OK").
- Si le segment ACK est un doublon (SEG.ACK < SND.UNA)
    - Ignorer le segment
- Si l'accusé de réception acquitte des données non encore émises (SEG.ACK > SND.NXT)
    - Emettre un ACK
    - Jeter le segment
    - **Retour.**
- Si SND.UNA < SEG.ACK =< SND.NXT,
    - Remettre à jour la fenêtre d'émission
- Si (SND.WL1 < SEG.SEQ ou (SND.WL1 = SEG.SEQ et SND.WL2 =< SEG.ACK)),
    - SND.WND = SEG.WND,
    - SND.WL1 = SEG.SEQ,
    - SND.WL2 = SEG.ACK.

Notez que SND.WND représente un décalage relatif par rapport à SND.UNA, que SND.WL1 enregistre le numéro de séquence du dernier segment ayant servi à remettre SND.WND à jour, et que SND.WL2 enregistre le numéro d'acquittement du dernier segment ayant permis de remettre SND.WND à jour. Ce test permet d'éviter de considérer des segments "anciens" pour remettre à jour la fenêtre.

Etat précédent : FIN-WAIT-1 STATE

En plus du traitement effectué pour l'état ESTABLISHED,

- Si le segment FIN émis (local) a été acquitté
    - Passer la connexion à l'état FIN-WAIT-2
    - Continuer dans cet état.

Etat précédent : FIN-WAIT-2 STATE

En plus du traitement effectué pour l'état ESTABLISHED,

- Si la pile de retransmission est vide
    - La commande CLOSE de l'application peut être acquittée ("OK"), mais le TCB doit rester en place.

Etat précédent : CLOSE-WAIT STATE

- Id. ESTABLISHED.

Etat précédent : CLOSING STATE

En plus du traitement effectué pour l'état ESTABLISHED,

- Si le segment ACK acquitte le segment FIN émis (local)
  - Passer la connexion à l'état TIME-WAIT
- Sinon
  - Ignorer le segment.

Etat précédent : LAST-ACK STATE

Le seul événement attendu dans cet état est l'acquittement de notre segment FIN.

- Dès que cet événement se produit,
  - Supprimer le TCB,
  - Passer la connexion à l'état CLOSED
  - **Retour.**

Etat précédent : TIME-WAIT STATE

Le seul segment attendu dans cet état est la retransmission du segment FIN distant.

- Dès que cet événement se produit,
  - L'acquitter
  - Relancer la temporisation de 2 MSL.
  - **Retour.**

6. Vérifier le bit URG

Etats précédents : ESTABLISHED STATE, FIN-WAIT-1 STATE, FIN-WAIT-2 STATE

- Si le bit URG est marqué
  - RCV.UP = max(RCV.UP,SEG.UP),
    - Si le pointeur urgent (RCV.UP) pointe sur des données non consommées
      - Si l'application n'est pas déjà dans le mode "urgent"
        - Signaler à l'application la présence de données urgentes.

Etats précédents : CLOSE-WAIT STATE, CLOSING STATE, LAST-ACK STATE, TIME-WAIT

Ce cas ne devrait pas se produire, car un segment FIN a été reçu du distant.

- Ignorer l'URG.

7. Récupérer les données,

Etats précédents : ESTABLISHED STATE, FIN-WAIT-1 STATE, FIN-WAIT-2 STATE

Une fois la connexion entrée dans l'état ESTABLISHED, il est possible de livrer des données dans les tampons RECEIVE de l'utilisateur. Les données peuvent être extraites du segment et rangées dans le tampon jusqu'à ce que celui-ci soit plein, ou le segment soit vide. Si le segment est vide (moins de données arrivées que d'espace libre dans le tampon) et que le flag PUSH est marqué, alors l'application en est informée lorsque le tampon lui

est renvoyé.

Lorsque TCP prend la responsabilité de transmettre des données vers l'utilisateur, il est supposé acquitter la réception de ces données.

Une fois que TCP a pris cette responsabilité, il doit avancer RCV.NXT du nombre d'octets acceptés, et ajuste RCV.WND de façon à tenir compte de l'espace de tampon restant. La somme de RCV.NXT et RCV.WND ne peut être réduit.

Reportez vous aux suggestions sur la gestion de fenêtre au paragraphe 3.7.

Envoyer un accusé de réception sous la forme: <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

Cet accusé de réception pourra profiter de la transmission d'un segment d'émission, dans la mesure où cela n'engendre pas un retard trop important.

Etats précédents : CLOSE-WAIT STATE, CLOSING STATE, LAST-ACK STATE, TIME-WAIT STATE

Ce cas ne devrait pas survenir, dans la mesure où un segment FIN a été reçu du côté distant. Ignorer les données du segment dans ce cas.

8. vérifier le bit FIN,
    - Ne traitez pas le bit FIN si l'état est CLOSED, LISTEN ou SYN-SENT puisque SEG.SEQ ne peut pas être validé
        - Jeter le segment
        - **Retour.**
    - Si le bit FIN est marqué,
        - signaler à l'application "déconnexion en cours"
        - Renvoyez un message identique pour chaque RECEIVE en instance,
        - Avancer RCV.NXT au delà de FIN,
        - Envoyer un acquittement pour ce FIN.

Notez que ce FIN force le PUSH de tout segment non encore délivré à l'application.

Etats précédents : SYN-RECEIVED STATE, ESTABLISHED STATE

- Passer dans l'état CLOSE-WAIT.
- **Retour.**

Etat précédent : FIN-WAIT-1 STATE

- Si notre propre FIN a été acquitté (peut être dans ce segment), alors
    - Passer dans l'état TIME-WAIT,
    - Amorcer la temporisation d'attente
    - Arrêter toutes les autres temporisations;
    - **Retour.**
- Sinon
    - Passer dans l'état CLOSING.

Etat précédent : FIN-WAIT-2 STATE

- Passer dans l'état TIME-WAIT.
- Amorcer la temporisation d'attente

- Arrêter toutes les autres temporisations;
- **Retour.**

Etat précédent : CLOSE-WAIT STATE

- Reste dans l'état CLOSE-WAIT.

Etat précédent : CLOSING STATE

- Reste dans l'état CLOSING.

Etat précédent : LAST-ACK STATE

- Reste dans l'état LAST-ACK.

Etat précédent : TIME-WAIT STATE

- Reste dans l'état TIME-WAIT.
- Relance de 2 MSL la temporisation.
- **Retour.**

Temporisation : USER TIMEOUT

Dans tous les états, si la temporisation utilisateur s'écoule complètement,

- Vider toutes les piles
- Renvoyer "erreur: abandon sur non réponse" à l'application, en général, et pour chacune des commandes en instance.
- Supprimer le TCB
- Passer à l'état CLOSED
- **Retour.**

Temporisation : RETRANSMISSION TIMEOUT

Si la temporisation de retransmission expire pour un segment de la pile de retransmission,

- Renvoyer de nouveau ce segment en tête de pile.
- Réinitialiser la temporisation
- **Retour.**

Temporisation : TIME-WAIT TIMEOUT

Lorsque la temporisation d'attente de déconnexion expire,

- Supprimer le TCB
- Passer dans l'état CLOSED
- **Retour.**

GLOSSAIRE

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". La spécification de l'interface entre un ordinateur et ARPAnet.

Accusé de réception

Le numéro de séquence indiqué dans le champ d’accusé de réception du segment entrant.

ACK

Un bit de contrôle (acquittement) ne consommant pas d’espace séquence, indiquant que le champ d’accusé de réception de ce segment indique le prochain numéro de séquence attendu par l’émetteur de ce segment, acquittant implicitement tous l’espace de séquence inférieur à ce nombre.

Adresse de destination

L’adresse de destination spécifie le numéro de la "branche" de réseau et le numéro de l’ordinateur cible sur cette branche.

Adresse source

L’adresse Internet de la source, constituée d’une adresse réseau et de l’adresse de l’ordinateur sur ce réseau.

ARPAnet, message

L’unité de transmission entre un ordinateur et un IMP d’ARPAnet. La taille maximale du message est d’environ 1012 octets (8096 bits).

ARPAnet, paquet

Unité de transmission utilisée sur ARPAnet entre IMP. La taille maximale du paquet est de 126 octets (1008 bits).

Connexion

Une communication logique définie par une paire de sockets.

Datagramme

Un message envoyé sur un réseau de transmission de données à commutation de paquets.

Fenêtre d’émission

Représente la plage de séquence que le TCP distant est prêt à recevoir. Elle correspond à la valeur de fenêtre reçue dans les segments transmis par le TCP distant. La plage de numéro de séquence qu’il est possible d’émettre est nécessairement incluse dans l’intervalle [SND.NXT, SND.UNA + SND.WND - 1] et, en général commence sur SND.NXT. (Les retransmissions de segments dans une plage de SND.UNA à SND.NXT sont bien entendu acceptés).

Fenêtre de réception

Représente la plage de numéros de séquence (en réception) que TCP s’attend à recevoir. Ainsi, tout segment dont la plage est incluse dans l’intervalle [RCV.NXT, RCV.NXT + RCV.WND - 1] est recevable. Tout segment dont la plage est intégralement en dehors de cet intervalle est supposé être un doublon et est rejeté.

FIN

Un bit de contrôle (finis) occupant consommant un numéro de séquence, indiquant que l’émetteur n’enverra plus de données ni de contrôles occupant de l’espace de séquence.

Fragment

Une portion d’une unité de transmission de données, en particulier, un fragment Internet est une portion d’un datagramme Internet.

FTP

Protocole de transfert de fichiers.

Header (en-tête)

Ensemble d’informations de contrôle placé en tête d’un datagramme, segment, fragment, paquet ou bloc de données.

Host

Un ordinateur. Peut être source ou destination du point de vue d’un réseau de données. On distingue les Hosts des stations de travail ou terminaux, lesquels ne sont pas autonomes et nécessitent justement les ressources d’un Host.

Identification

Un champ du protocole IP. Cette valeur d’identification aide à recomposer les fragments d’un même datagramme.

IMP

Interface Message Processor, le "routeur" du réseau ARPAnet.

Internet, adresse

Une adresse de 32 bits spécifiant l’adresse unique d’un host sur un réseau.

Internet, datagramme

L’unité d’échange de données entre la couche Internet et les couches supérieures comprenant l’en-tête Internet.

Internet, fragment

Une portion d’un datagramme Internet complété d’une en-tête Internet.

IP

Protocole Internet

IRS

Initial Receive Sequence. Le tout premier numéro de séquence attendu par un récepteur.

ISN

Initial Sequence Number. Le premier numéro de séquence utilisé lors d’une connexion, (ISS ou IRS). Choisi en fonction d’une horloge.

ISS

Initial Send Sequence. Le tout premier numéro de séquence émis par un émetteur.

Leader (en-tête)

Information de contrôle placé en tête d’un bloc de données. En particulier, dans ARPAnet, l’information de contrôle de message ARPAnet à la hauteur de l’interface host/IMP.

Longueur de segment

La plage de numéro de séquence transmise par le segment, y compris les numéros consommés par certains bits de contrôle.

Module

L’implémentation, généralement logicielle, d’une fonction particulière.

MSL

Maximum Segment Lifetime, Le temps pendant lequel un segment TCP peut exister à l’intérieur du réseau de transmission. Arbitrairement : 2 minutes.

Numéro de séquence

Un index numérique servant à identifier un octet de données dans le flux de transmission. Cet index est calculé sur 32 bits, et reboucle modulo 2**32.

Octet

Huit bits.

Options

Un champ d’option peut en contenir plusieurs, chacune des options pouvant avoir une longueur de plusieurs octets. Les options sont à l’origine utilisées pour des fonctions de test; par exemple, pour véhiculer des étiquettes temporelles. Les deux protocoles Internet Protocol et TCP disposent de champs d’option.

Paquet

Un paquet est un terme très général désignant très exactement une section d’un flux de données encapsulé dans divers protocoles. En général, on parlera de paquet une entité physique, plutôt que logique. Mais l’usage fait utiliser ce terme dans le cas le plus général.

Paquet local

L’unité de transmission dans un réseau local.

Pointeur de données urgentes

Un champ de contrôle qui n’a de signification que lorsque le bit URG est marqué. Ce champ communique le numéro de séquence du dernier octet ’urgent’ sur réception duquel l’application pourra repasser en mode "normal".

Port

La portion d’un socket qui définit l’entrée ou la sortie "logique" qu’utilise un processus pour véhiculer ses données.

Processus

Un programme en cours d’exécution. Une source ou destination, du point de vue du réseau ou de toute autre protocole

de communication "host-to-host".

PUSH

Un bit de contrôle ne consommant pas d'espace séquence, indiquant que ce segment contient des données à transférer immédiatement à l'utilisateur.

RCV.NXT

Prochain numéro de séquence à recevoir.

RCV.UP

Pointeur de données urgentes.

RCV.WND

Fenêtre de réception

RST

Un bit de contrôle (reset) ne consommant pas d'espace séquence, indiquant au récepteur que celui-ci doit abandonner la connexion sans autre interaction ultérieure. Le récepteur doit déterminer, selon le numéro de séquence et la valeur de l'accusé de réception, s'il doit prendre en compte cette demande ou l'ignorer. En aucun cas la réception d'un segment RST n'entraîne une réponse du type RST.

RTP

Real Time Protocol: un protocole de communication inter-processus pour la transmission de données à contraintes temporelles fortes.

SEG.ACK

Accusé de réception du segment.

SEG.LEN

Longueur du segment.

SEG.PRC

Valeur de priorité du segment.

SEG.SEQ

Séquence du segment.

SEG.UP

Pointeur de données urgentes du segment.

SEG.WND

Fenêtre dans le segment.

Segment

Une unité logique de base, un segment TCP est l'unité de transmission entre deux agents TCP.

Séquence d'émission

Prochain numéro de séquence émis par l'émetteur. Il est initialisé par une fonction génératrice renvoyant un numéro de séquence initial (ISN) et est incrémenté pour chaque octet ou bit de contrôle (certains seulement) émis.

Séquence gauche

Le prochain numéro de séquence à être acquitté par le TCP récepteur (ou le numéro d'acquittement courant le plus bas) parfois référencé comme le bord gauche de la fenêtre de transmission.

SND.NXT

Séquence à l'émission.

SND.UNA

Numéro de séquence du premier octet non acquitté.

SND.UP

Pointeur de données urgentes à l'émission.

SND.WL1

Numéro de séquence du dernier segment ayant servi à remettre à jour l'information de fenêtre.

SND.WL2

Numéro d'acquittement du dernier segment ayant servi à remettre à jour l'information de fenêtre.

SND.WND

Fenêtre d'émission.

Socket

Une adresse réseau constituée par la concaténation d'une adresse Internet avec un numéro de port TCP.

SYN

Un bit de contrôle consommant un numéro de séquence, utilisé pendant la phase d'initialisation de la connexion pour indiquer à quel numéro de séquence la transmission de données débute.

TCB

Transmission Control Block. La structure de données enregistrant les paramètres d'une connexion.

TCB.PRC

Le niveau de priorité d’une connexion.

TCP

Transmission Control Protocol: Un protocole de communication inter-processus fiabilisé dans un environnement multiréseaux.

TOS

Type Of Service. Un champ IP indiquant le type de service du fragment IP.

URG

Un bit de contrôle ne consommant pas d’espace séquence, utilisé pour indiquer que des données urgentes sont transportées par le segment. L’application réceptrice doit en être informée, basculer en mode "d’urgence", et y rester tant que le numéro de séquence des données reste inférieur à la valeur du pointeur de données urgentes.

BIBLIOGRAPHIE

[1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974. [2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981. [3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978. [4] Postel, J., "Assigned Numbers", RFC 790, USC/Information Sciences Institute, September 1981.