Wireless TCP

Xavier KRANTZ
Thibaud LEHMANN
Matthieu THOMASSIN

18 *décembre* 2008

Sous la direction de :
Abdelmajid BOUABDALLAH

utc
Université de Technologie
Compiègne

Table des matières

1 Les réseaux sans-fil et le protocole TCP **4**
- 1.1 Bref rappel sur les réseaux sans-fil . . . 4
  - 1.1.1 Le réseau cellulaire . . . 4
  - 1.1.2 Le réseau Ad-Hoc . . . 4
  - 1.1.3 Les réseaux satellites . . . 5
- 1.2 Le protocole TCP Classique . . . 6
  - 1.2.1 format d'un segment TCP . . . 6
  - 1.2.2 Echange de données . . . 7
  - 1.2.3 Contrôle de flux . . . 9
- 1.3 Problèmes liés aux réseaux sans-fil . . . 10
  - 1.3.1 Problème de congestion ou problème de transmission ? . . . 10
  - 1.3.2 Une transmission plus aléatoire . . . 10
  - 1.3.3 Des liaisons asymétriques . . . 11

2 TCP et les mécanismes de Contrôle de Congestion **12**
- 2.1 Congestion . . . 12
- 2.2 Algorithmes de Contrôle de Congestion . . . 12
  - 2.2.1 Slow Start . . . 12
  - 2.2.2 Congestion Avoidance . . . 15
  - 2.2.3 Fast retransmission . . . 16
  - 2.2.4 Fast recovery . . . 19
- 2.3 Les protocoles TCP d'optimisation du contrôle de congestion . . . 21
  - 2.3.1 TCP Tahoe et Reno . . . 21
  - 2.3.2 TCP New Reno . . . 22
  - 2.3.3 TCP Vegas . . . 24
- 2.4 Réseau à très haut BDP . . . 28
  - 2.4.1 Principe . . . 28
  - 2.4.2 TCP CUBIC . . . 28
- 2.5 Conclusion . . . 29

3 Mécanismes d'amélioration du protocole TCP **30**
- 3.1 Mécanismes de bout-en-bout . . . 30
  - 3.1.1 TCP Probing . . . 31
  - 3.1.2 Freeze-TCP . . . 31
  - 3.1.3 Wireless TCP . . . 32
  - 3.1.4 TCP Westwood (TCPW) . . . 33
- 3.2 Mécanismes de fractionnement de la connexion . . . 33
  - 3.2.1 Indirect TCP (I-TCP) . . . 33
  - 3.2.2 Mobile TCP (M-TCP) . . . 34
  - 3.2.3 Explicit Bad State Notification (EBSN) . . . 35
- 3.3 Protocoles de couche liaison - Récupération locale . . . 35

3.3.1 Snoop Protocol 36

Bibliographie **38**

Table des figures **40**

Introduction

C'est en septembre 1969 que pour la première fois deux ordinateurs ont pu communiquer entre eux, avec la naissance d'ARPANET (Advanced Research Projects Agency Network). Il fut dans un premier temps utilisé par la défense américaine en 1972. C'est l'année suivante que fut créer le concept d'adresse IP, afin de fiabiliser la transmission des données, grâce à une meilleure authentification des machines. De là, on dériva un protocole qui allait révolutionner le monde de la réseautique : le TCP/IP. Celui-ci permettait notamment de connecter à ARPANET des réseaux n'utilisant pas de câbles, tels que les réseaux radio ou satellite. Et ce fut le début de l'ère nouvelle des communications : envoi de messages électroniques, téléchargement de fichiers, connexion à distance et même la transmission de la voix entre deux ordinateurs. Ce standard, approuvé sur ARPANET, est à la base de l'Internet tel que nous le connaissons aujourd'hui.

Ce protocole TCP fonctionne très bien dans les milieux où les erreurs de transmissions sont rares comme les réseaux câblés par exemple. En revanche, nous nous rendons compte que ce protocole n'est pas bien adapté aux environnements où les taux d'erreurs sont grands, les déconnexions régulières, la bande passante changeante, etc. Or les stations mobiles utilisent des solutions sans-fil qui sont généralement sujettes à ces problèmes. Heureusement, il existe différents types de mécanismes qui ont été développés et qui permettent de pallier aux limites de TCP que nous allons présenter ultérieurement.

Dans un premier temps nous allons présenter les différents types de réseaux sans-fils et rappeler le fonctionnement classique du protocole TCP. Cela nous permettra d'identifier les différents problèmes que rencontre TCP sur les réseaux sans-fil. Dans une seconde partie nous présenterons les algorithmes de contrôle de congestion qui sont déjà implémentés dans le TCP, puis nous en montrerons quelques variations. Dans une dernière partie nous présenterons les différentes solutions existantes permettant d'améliorer les connexions avec des stations mobiles.

Chapitre 1

Les réseaux sans-fil et le protocole TCP

1.1 Bref rappel sur les réseaux sans-fil

On peut distinguer 3 principaux types de réseaux sans-fil :

1.1.1 Le réseau cellulaire

Utilisé notamment dans la téléphonie, il désigne les types de réseau dans lesquels un récepteur mobile est relié au réseau filaire par l'intermédiaire d'une base fixe, appelé point d'accès. Il existe beaucoup de nouveaux protocoles dérivant de TCP et permettant d'améliorer les performances de la connexion pour ce type de réseau. Ils cherchent à optimiser les performances du point d'accès, qui sert de relai entre le récepteur et le reste du réseau.

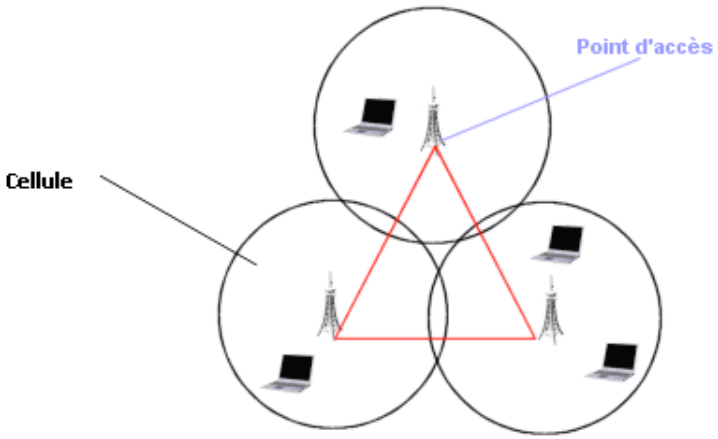


FIGURE 1.1 – Réseau cellulaire

1.1.2 Le réseau Ad-Hoc

Composé uniquement de récepteurs mobiles, le réseau AD-HOC est la méthode la plus simple à mettre en oeuvre pour la création d'un réseau sans fil. Sa principale contrainte est que deux noeuds ne peuvent échanger des informations que s'ils sont à portée de réception l'un de l'autre. Ainsi sur l'exemple 2, la machine A devra jouer le rôle de routeur pour assurer la connexion entre les machines B et C, ce qui pose de nombreux problèmes pratiques. Ce type de réseau reste d'ailleurs assez peu employé car ses performances restent sensibles à la qualité de propagation des ondes et aux interférences. Il ne sera donc utilisé que pour des réseaux ayant des noeuds

proches géographiquement et de nombre limité.

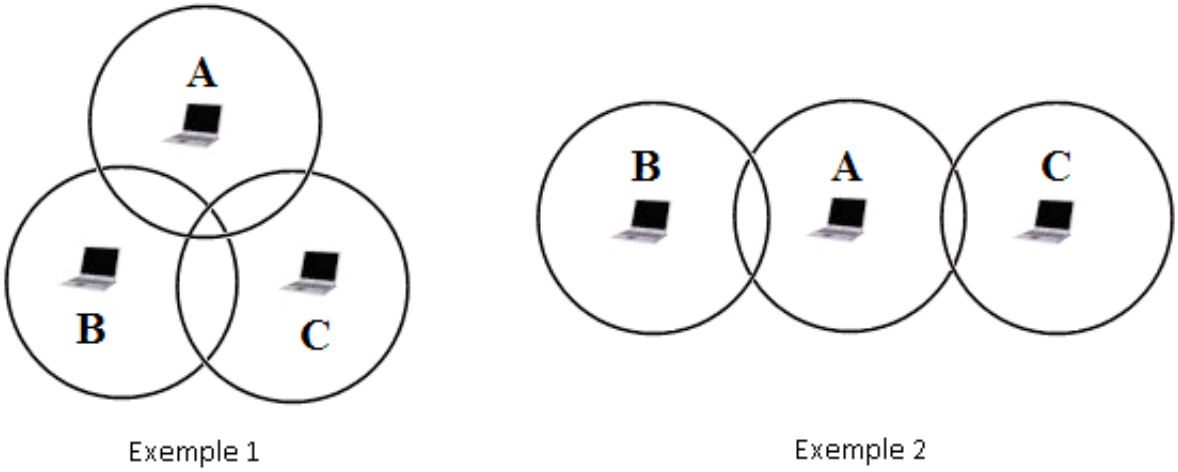


FIGURE 1.2 – Réseau Ad-Hoc

1.1.3 Les réseaux satellites

Bien que très utiles pour les liaisons longues distances, les liaisons par satellite sont encore assez peu utilisé car la grande distance qui sépare les machines du satellite provoque un ralentissement de la connexion et un taux d'erreur relativement important.

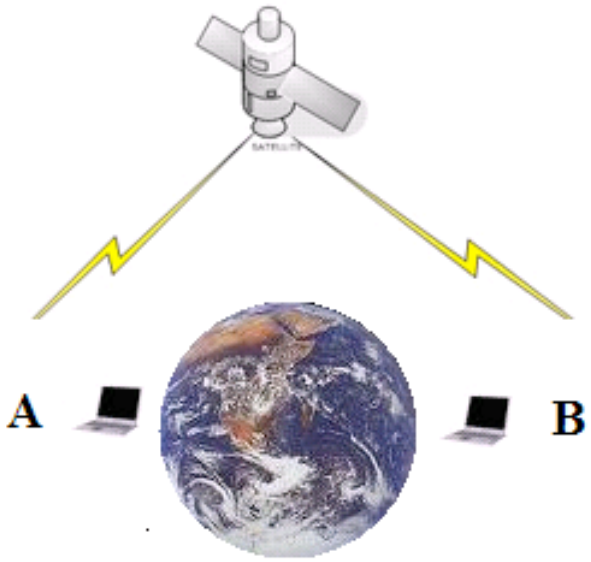


FIGURE 1.3 – Réseau satellite

1.2 Le protocole TCP Classique

Le protocole TCP (Transport Control Protocol) est un protocole orienté transport qui se caractérise par la fiabilité qu'il apporte au transfert des données. TCP fractionne les données applicatives en segments de taille variable. La taille des segments, calculée avant la transmission, fait en sorte d'assurer un débit de transfert de données optimal, tout en évitant autant que possible les pertes de données ou le transfert de données erronées.

Le protocole TCP appartient à la 3ème couche du modèle OSI comme nous le voyons sur la figure suivante :

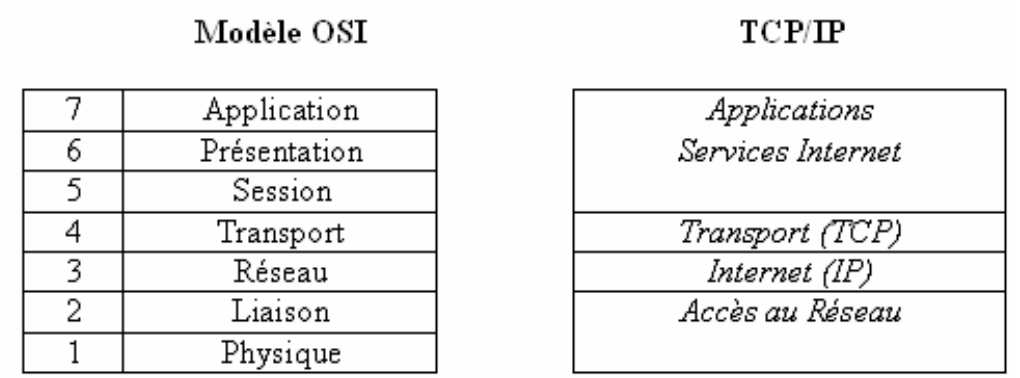


FIGURE 1.4 – Modèle OSI

1.2.1 format d'un segment TCP

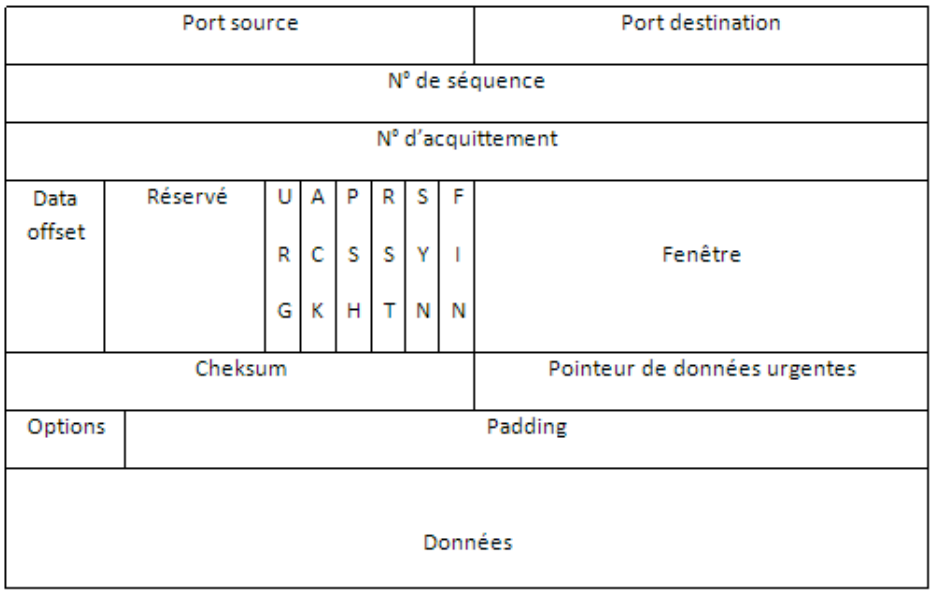


FIGURE 1.5 – Format d'un segment TCP

- Les numéros de ports sources et destinations permettent de référencer les applications
- Le numéro de séquence est le numéro du premier octet transmis dans le paquet

- Le numéro d'acquittement est le numéro du prochain octet attendu par l'émetteur du segment
- *Data Offset* représente la taille de l'en-tête du segment. En effet, la taille du champ option est variable. L'unité de longueur utilisé est le mot de 32 bits.
- On trouve ensuite 6 flags :
  - *URG* : Les données envoyées sont urgentes
  - *ACK* : Prise en compte ou non du numéro d'acquittement
  - *PSH* : Les données doivent être transmises à la couche supérieure
  - *RST* : Fermeture de la connexion (en cas d'erreur)
  - *SYN* : Établissement de la connexion
  - *FIN* : Fin de la connexion
- Le champ Fenêtre correspond au nombre d'octets que le récepteur peut accepter.
- *Cheksum* : calculé par l'émetteur, il permet de mettre en place un contrôle d'erreur.
  algorithme de calcul

  1. le champ *checksum* est initialement mis à 0,
  2. la suite à protéger est considérée comme une suite de mots de 16 bits,
  3. les mots de 16 bits sont additionnés un à un, modulo 65 535,
  4. le *checksum* est le complément à 1 (inverse bit à bit) de la somme trouvée,
  5. le récepteur fait la somme modulo 65 535 de tous les mots concernés et vérifie qu'il obtient FF FF ou 00 00

  Cependant ce contrôle d'erreur n'est effectué que sur 16 bits, ce qui le rend assez peu fiable. Il est donc préférable de l'associer à d'autres types de contrôle d'erreur (situé sur d'autres couches du modèle OSI en général).

- Le pointeur de données urgentes permet de préciser si certains octets des données doivent être traités en priorité.
- Options : De nombreuses options peuvent être précisées dans l'en-tête du segment TCP, mais on trouve parmi les plus répandues :

  1. *Timestamps* : contrôle du délai d'acheminement ;
  2. *Window Scaling* : Augmente la capacité de la fenêtre ;
  3. *No Operation* : Alignement de la fin d'une seule option ;
  4. *End of option list* : Permet à la partie variable de l'en-tête de se terminer en frontière de mot ;
  5. *MSS option* : Permet de négocier la taille maximale des segments envoyés.

- *Padding* : Zéros ajoutés pour aligner les champs suivants du paquet sur 32 bits, si nécessaire.

1.2.2 Echange de données

Le protocole TCP fonctionne en mode connecté, ce qui permet d'assurer une meilleure fiabilité de la connexion. L'établissement d'une connexion entre deux hotes A et B s'effectue d'après le schéma suivant :

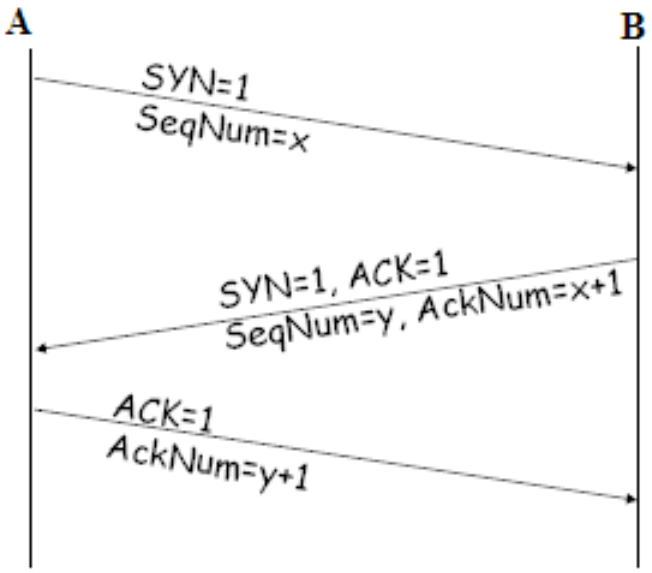


FIGURE 1.6 – Schéma d'une connexion TCP

L'activation du bit $SYN = 1$ permet de préciser qu'il s'agit d'une demande de connexion. L'émetteur transmet également le numéro de la séquence envoyée *NSeq*. Par la suite, chacun des deux hotes enverra également un accusé de réception *AckNum* qui permettra d'informer l'autre de la prochaine séquence attendue. Cela permet, entre autre, de repérer la perte d'un paquet.

De même, l'arrêt de la connexion se fait d'après le schéma suivant, le bit *FIN* permettant de préciser qu'il s'agit d'une demande de libération de connexion :

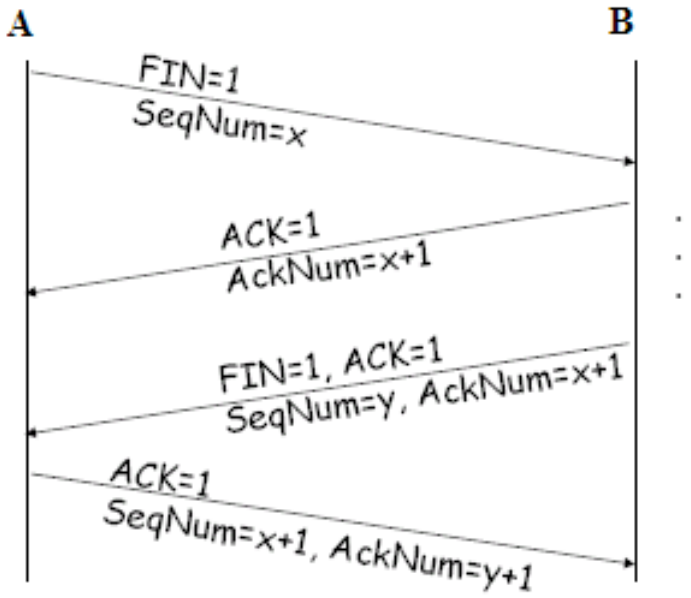


FIGURE 1.7 – Schéma d'une déconnexion de transaction TCP

1.2.3 Contrôle de flux

A et B disposent de buffers de tailles limitées pour envoyer et recevoir les données. Chaque segment TCP contient la taille disponible dans le buffer de l'hôte qui l'a envoyé. En réponse, l'hôte distant va limiter la taille de la fenêtre d'envoi afin de ne pas le surcharger (régulation du nombre de paquet à envoyer).

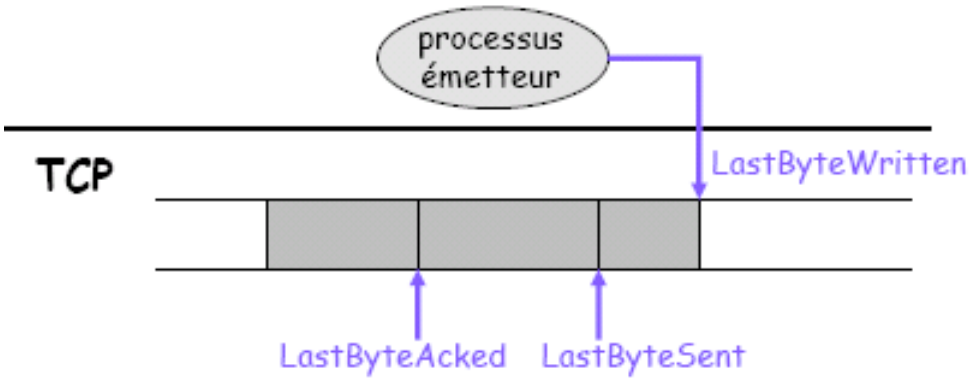


FIGURE 1.8 – Buffer d'émission TCP

TCP calcule régulièrement la différence *LastByteSent* - *LastByteAcked* pour définir la taille de la fenêtre de la congestion : cette taille correspond au nombre maximum de paquets que l'émetteur peut envoyer sans recevoir aucun accusé.

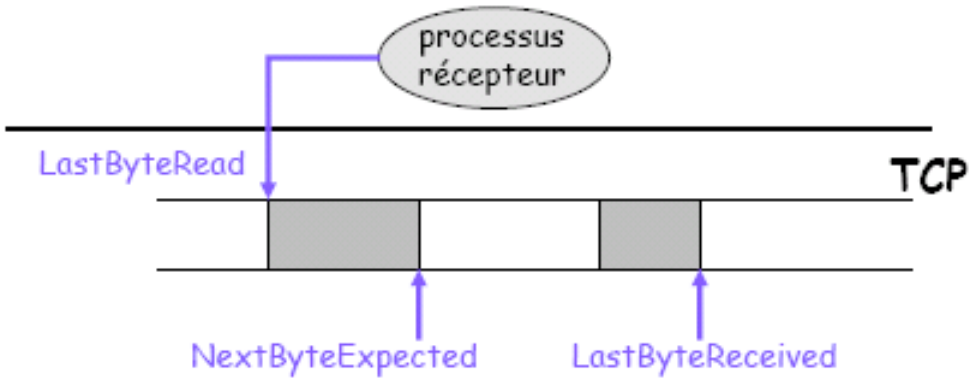


FIGURE 1.9 – Buffer de réception TCP

De même, TCP calcule régulièrement la différence *LastByteReceived* - *LastByteRead* afin de calculer la taille de la fenêtre de réception : cette taille correspond au nombre maximum de paquets que le récepteur est capable de recevoir à un moment donné. Au fur et à mesure que les données arrivent, TCP les acquitte si tous les octets précédents ont été reçus

1.3 Problèmes liés aux réseaux sans-fil

A l'origine, le protocole TCP a été élaboré pour assurer la fiabilité des connexions filaires uniquement. Les caractéristiques des réseaux sans-fil n'ont donc pas été prises en compte dans l'élaboration des algorithmes de contrôle de flux et de contrôle de congestion de TCP, ce qui pose de nombreux problèmes :

1.3.1 Problème de congestion ou problème de transmission ?

Le principal problème lié au réseau sans fil est que TCP est incapable de différencier les pertes de données liées à la congestion (saturation du réseau dû à un nombre trop important d'utilisateurs) de celles liées à la dégradation de la connexion (qui peut être dû à des bruits ou à la mobilité de l'utilisateur par exemple). Or la réaction doit être différente dans les deux cas : En cas de problème de connexion par exemple, la réduction de la taille de la fenêtre de congestion, bien que complètement inutile, aboutit à une forte diminution du débit, et donc des performances globales du réseau.

1.3.2 Une transmission plus aléatoire

Il existe d'autres sources de problèmes qui affectent les performances de TCP sur les réseaux sans fils : on peut citer l'ordre de réception des paquets successifs qui est parfois aléatoire sur les réseaux sans-fil. Or le protocole TCP classique va avoir tendance à considérer injustement certains paquets comme perdu, car les accusés de réception n'auront pas été correctement reçus. On peut également citer les délais de transmission qui seront beaucoup plus variables sur les réseaux sans-fil que sur les réseaux filaires, à cause notamment des handovers (processus qui permet à un terminal mobile d'effectuer le passage entre deux points d'attachement à un réseau).

1.3.3 Des liaisons asymétriques

Le temps de transfert de données entre une machine A et une machine B peut-être différent du temps de transfert de A vers B (on parle dans ce cas de liaison asymétrique sur une partie du réseau). TCP, qui ne tient compte que du temps d'aller-retour, utilisera donc une valeur erronée du temps de transfert dans ses différents calculs (calcul du RTT par exemple). Là encore, cela peut réduire inutilement les performances globales de TCP.

Nous présenterons donc tout au long de ce dossier les différents algorithmes et les différentes améliorations de TCP susceptibles de résoudre ces problèmes.

Chapitre 2

TCP et les mécanismes de Contrôle de Congestion

2.1 Congestion

Comme nous avons pu le voir dans la partie précédente, la perte d'un paquet sur le réseau est interprétée par TCP comme un unique problème de congestion. Le contrôle de congestion implémenté dans TCP est géré exclusivement du côté de l'émetteur, le récepteur ne fait que renvoyer des accusés de réception.

Rappelons rapidement le principe de congestion :

La congestion désigne un état où le réseau est saturé et subit un ralentissement global du trafic. Dans ce cas, certaines données sont perdues dû au fait que la file d'attente du ou des noeuds intermédiaires est pleine.

2.2 Algorithmes de Contrôle de Congestion

Le protocole TCP classique utilise des algorithmes de contrôle de congestion du réseau qui incluent différents aspects d'un mécanisme de type AIMD (additive-increase-multiplicative-decrease). Cela veut dire que la taille de la fenêtre de congestion sera augmentée par incrémentation et diminuée par division d'un certain coefficient.

Les algorithmes basiques de contrôle de congestion utilisés par TCP sont référencés par la RFC 2581 et sont au nombre de 4 :

- "Slow start" ;
- "Congestion Avoidance" ;
- "Fast restransmit" ;
- "Fast Recovery" ;

Dans les faits, ces 4 algorithmes sont complémentaires. Les différentes versions de TCP sont en fait des évolutions de ce dernier, qui intègrent au fur et à mesure ces différents algorithmes.

2.2.1 Slow Start

Principe

Slow Start est la première phase de contrôle de congestion durant laquelle le protocole va chercher à obtenir le débit maximum de transmission de données. Ainsi, pour chaque connexion, TCP maintient une fenêtre de

congestion, limitant le nombre total de paquets en cours d'acheminement (n'ayant pas encore reçu d'ACK). Cela suit le même principe que la fenêtre glissante de contrôle de flux de données. L'algorithme Slow Start augmente la taille de la fenêtre de congestion de manière régulière une fois que la connexion est initialisée.

Initialisation :

Concrètement, la fenêtre de congestion commence avec une taille initiale égale à la taille maximale du segment (MSS). Cette taille de MSS est négociée pendant la phase d'ouverture de la connexion TCP, effectuée en 3 temps entre l'émetteur et le récepteur. De manière générale, cette phase permet de déterminer la taille maximale que le segment peut avoir afin d'arriver en entier (MTU pour IP) au destinataire sans subir de fragmentation en cours de route. Si cette information n'est pas déterminée, elle prend la valeur par défaut de 536 Octets (avec une entête TCP de 40 Octets, soit une taille totale de 576).

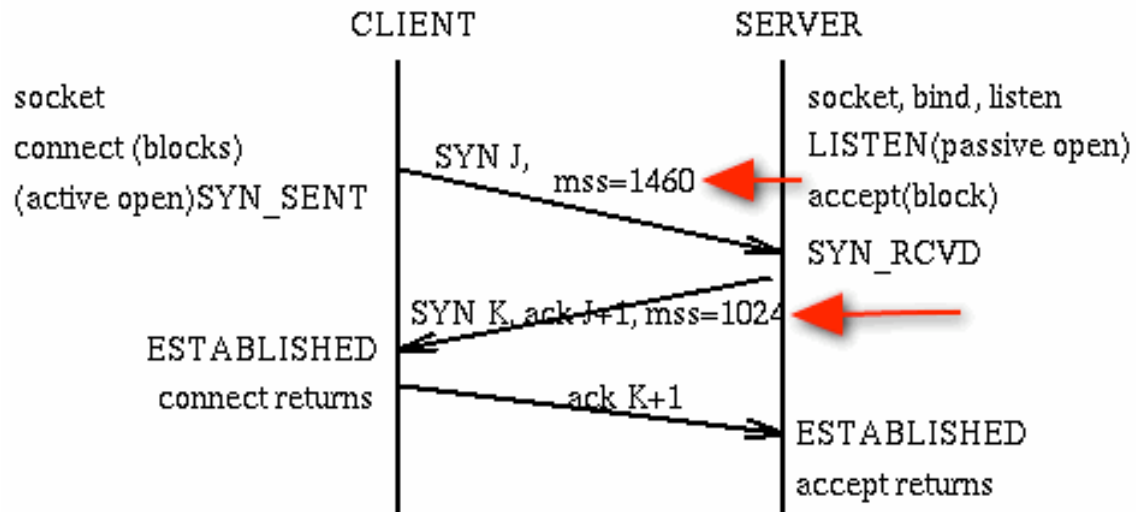


FIGURE 2.1 – Schéma de demande de connexion TCP avec négociation du MSS

Evolution :

Bien que cette ouverture initiale soit petite, son taux d'augmentation est très rapide : pour chaque paquet dont l'accusé (ACK) est reçu, la taille de la fenêtre augmente de 1 MSS. Puisque l'acquittement régénère le crédit dépensé, la taille passe ainsi à 2 dès le premier acquittement. L'émission de 2 paquets acquittés donne une taille de 4 (chaque ACK régénère un crédit et augmente la taille de la fenêtre de 1 MSS), et ainsi de suite.

Taille fenêtre <– Taille fenêtre + (nb ACK * MSS)

Si le récepteur renvoi un ACK pour chaque paquet émis, l'effet direct de l'algorithme est un doublement du taux de données transmissible simultanément à chaque RTT (temps entre le début d'envoi du segment et la réception de l'ACK).

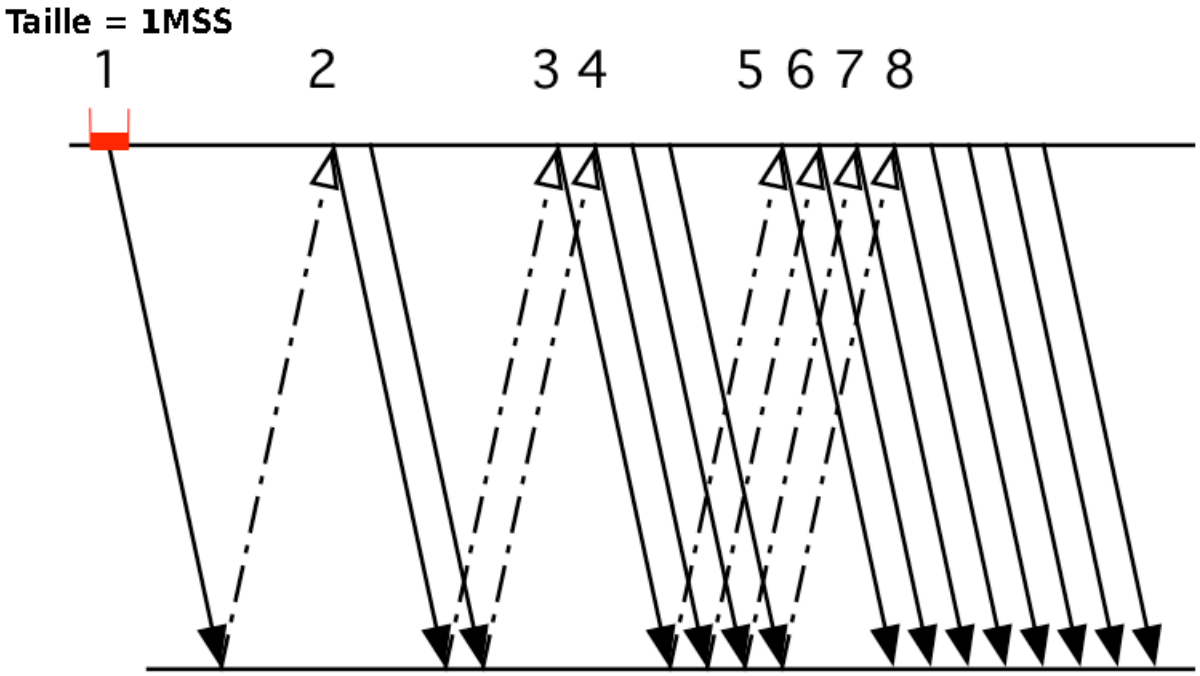


FIGURE 2.2 – Evolution de la fenêtre de congestion en nombre de MSS

Par contre, si le récepteur supporte les ACKs "retardés", c'est à dire qu'il possède un temps de délai (timeout) nettement plus important que le RTT alors le taux d'augmentation sera légèrement plus faible. Néanmoins, la taille de la fenêtre augmente au moins d'1 MSS à chaque RTT.

Seuil limite "ssthresh" (Slow-Start Threshold) :

Evidement, cette augmentation ne peut pas être infinie. Si elle l'était, soit la taille de la fenêtre de congestion dépasserait la taille de la fenêtre de réception du destinataire, soit la capacité du réseau serait excédée (congestion). Dans tous les cas il y aurait des pertes de paquets alors que c'est justement ce que nous cherchons à éviter en utilisant ce type d'algorithme.

Ainsi est définie un "seuil de croissance" : *ssthresh*.

La valeur de *ssthresh* est fixée initialement, et de manière arbitraire, à la valeur de la taille de fenêtre de réception du destinataire.

Lorsque la fenêtre de congestion dépasse cette valeur seuil, l'algorithme entre dans un nouvelle phase : "Congestion Avoidance", et cherche à éviter la congestion. Dans certaines implémentations, (sur Linux notamment) le seuil initial est très important, ainsi la première phase d'augmentation s'arrête après la perte d'un premier paquet.

Cette valeur seuil est mise à jour à la fin de chaque phase d'augmentation de la taille de la fenêtre. En effet, lorsque l'état de congestion est remarqué, *ssthresh* prend la valeur de la moitié de la taille de la fenêtre de congestion actuelle. Ce seuil représente pour TCP un point de repère où le phénomène de congestion n'était

pas encore rencontré mais pouvait être prévu.

2.2.2 Congestion Avoidance

Principe

Comparée à *Slow Start*, l'algorithme d'évitement de congestion utilise une méthode de sondage du réseau afin de découvrir le débit limite de perte de paquets. Ainsi, là ou *Slow Start* présentait plus une croissance exponentielle de la taille de la fenêtre de congestion, *Congestion Avoidance* présente une croissance linéaire.

En effet, au risque de nous répéter, nous venons de voir que cet algorithme entre en action une fois que le seuil *ssthresh* est dépassé par la fenêtre de congestion. Ce seuil représentant la taille de la fenêtre de réception du destinataire, nous pouvons donc présumer que nous ne sommes pas loin du seuil de tolérance d'acceptation de données. Au lieu de simplement s'arrêter à cette valeur, TCP cherche toujours à augmenter son débit de transmission.

Evolution

Pendant cette phase, l'émetteur augmente la taille globale de la fenêtre de congestion de l'émetteur d'une valeur constante à chaque RTT.

Taille_Globale <- Taille_globale + MSS (1)

Comme nous l'avons vu précédemment, cette action est gérée à la réception d'un ACK non dupliqué.

Donc, si la taille de la fenêtre de congestion à un instant t est *Taille* et que chaque segment possède une taille de MSS, nous pouvons définir que le nombre de paquets en transit est :

NbPaquet = Taille / MSS (2)

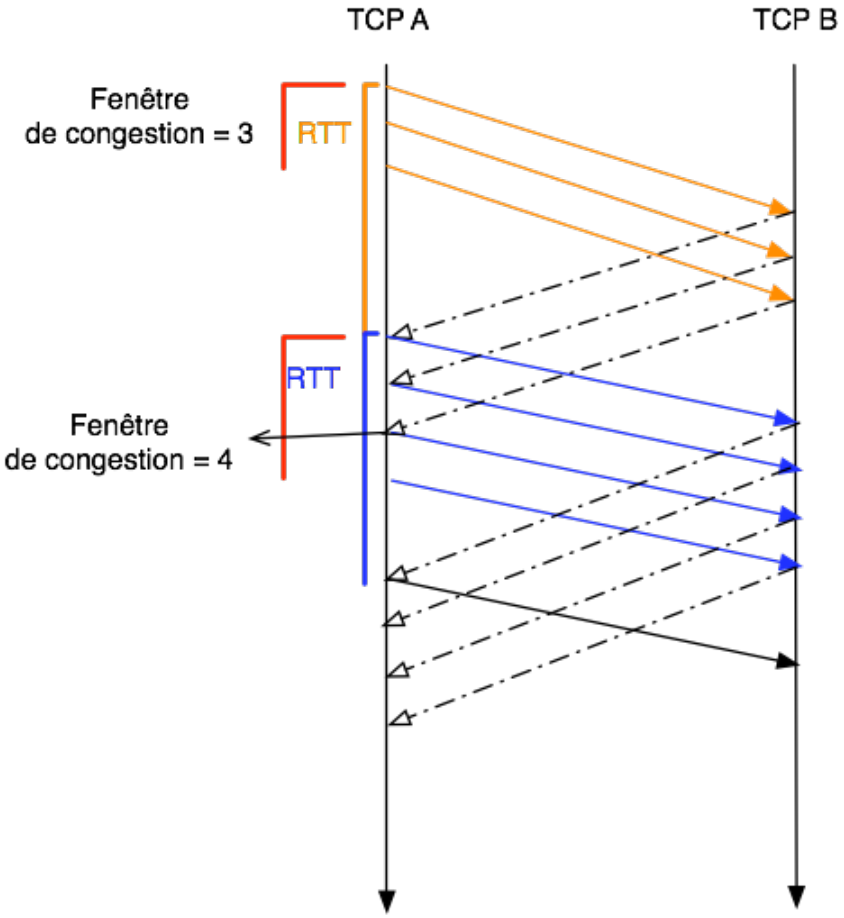
Ainsi, c'est également le nombre d'ACK non dupliqués que nous pouvons nous attendre à obtenir au bout d'un temps égal à RTT.

En référence à la formule que nous avons écrite pour *Slow Start*,

Taille <- Taille + (NbACK * i) (3)

Nous pouvons ainsi en déduire le coefficient i et la formule finale telle que :

Taille <- Taille + MSS * MSS / Taille



La fenêtre suit cette évolution jusqu'à la perte d'un paquet.

2.2.3 Fast retransmission

Principe :

Le But de *Fast Restransmission* est de détecter le plus rapidement possible la perte de paquets afin de :
- Retransmettre le paquet perdu ;
- Ajuster le débit et réduire la probabilité d'une prochaine perte ;

Détection de perte

De manière générale, TCP détecte une anomalie en utilisant uniquement un mécanisme de délai. Après avoir envoyé un paquet, TCP déclenche son propre timer pour le paquet émit. De manière générale, le timer est initialisé à la valeur du RTO (*Retransmission Timeout Period*) calculé par un autre algorithme en fonction des RTT précédents. Si TCP reçoit correctement l'ACK correspondant au paquet émiT et ce avant l'expiration du timer, alors TCP considère qu'il n'y a pas eu de problème sur le réseau. Il réinitialise le timer et attend les autres ACKs. Cependant, si TCP ne reçoit pas l'ACK attendu pendant le RTO, TCP retransmettra le paquet dont le RTO a expiré. De plus, TCP rentre de nouveau en phase *Slow Start*, réinitialise la taille de la fenêtre à 1 MSS et met à jour le *ssthresh*. Il a vite était découvert qu'utiliser uniquement le mécanisme de délai conduisait à un trop long temps mort avant de réagir au problème.

Fast Retransmit est un algorithme qui a été mis au point afin de détecter plus rapidement la perte de paquets. Le principe se base sur la réception d'ACK dupliqués. Cependant, un unique ACK dupliqué n'est pas suffisant pour déterminer la perte d'un paquet. En effet, lorsqu'un paquet de numéro de séquence x arrive correctement au récepteur, ce dernier émet un ACK contenant le numéro de séquence du prochain paquet attendu (x+1). Ainsi, lorsqu'un paquet arrive dans le désordre (par exemple 1, 2, 3, 5), le récepteur ré-émet l'ACK du paquet attendu. L'émetteur reçoit donc un ACK "dupliqué" dans le sens où il a déjà reçu un ACK de cette valeur. De manière générale, le récepteur génère quelques ACK dupliqués au court d'une transmission avant de générer un ACK de la séquence de données entière une fois qu'il aura reçu le paquet errant. Ainsi, après avoir reçu 3 ACKs dupliqués de suite, l'émetteur considère que le paquet concerné est perdu et n'attend pas l'expiration du RTO pour retransmettre le paquet en question.

Fast Retransmit ne remplace pas le mécanisme de délai mais vient au contraire le compléter. Dans le cas de petite transmission ou si le paquet perdu est en fin de séquence, il n'y aura pas assez d'ACKs dupliqués pour détecter la perte de paquets.

A noter que la raison pour laquelle l'émetteur attend 3 ACKs dupliqués est expliquée dans la RFC 2001 :

> *Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost.*

Action en cas perte

A la réception d'un troisième ACK dupliqué, ou à l'expiration du RTO, l'émetteur retransmet immédiatement le segment référencé par la valeur de l'ACK.

Puisque TCP interprète toute perte de donnée comme étant un problème de congestion, *Fast Retransmit* ré-initialise la fenêtre de congestion à 1 MSS, met à jour le *ssthresh* à 1/2 de la taille de l'ancienne fenêtre de congestion et retourne en mode *Slow Start*.

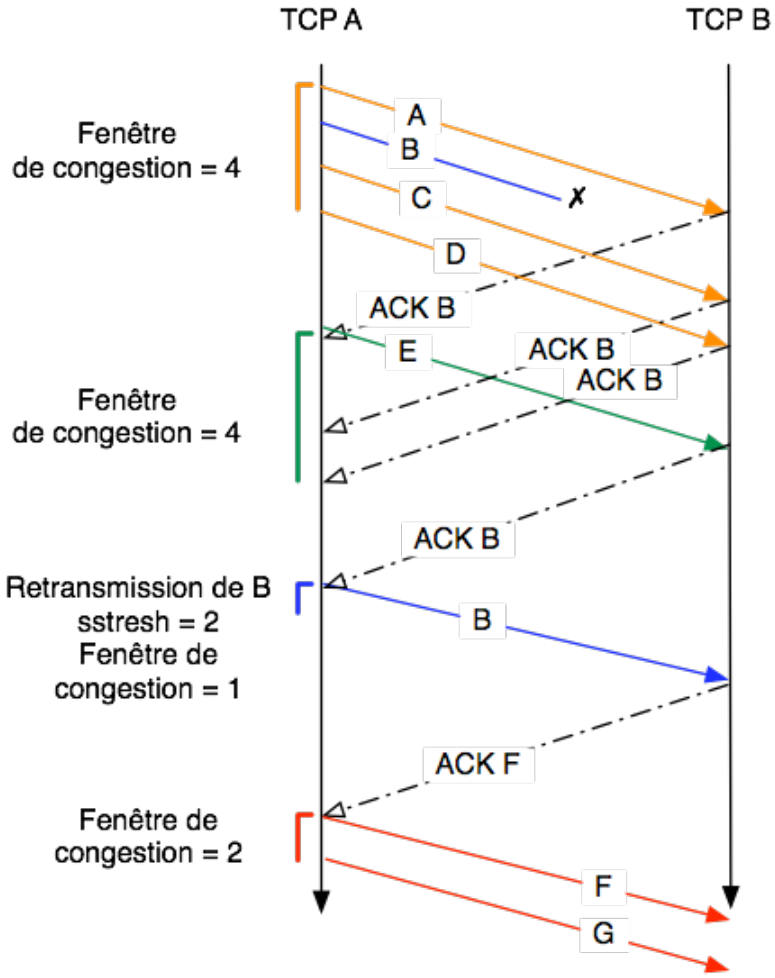
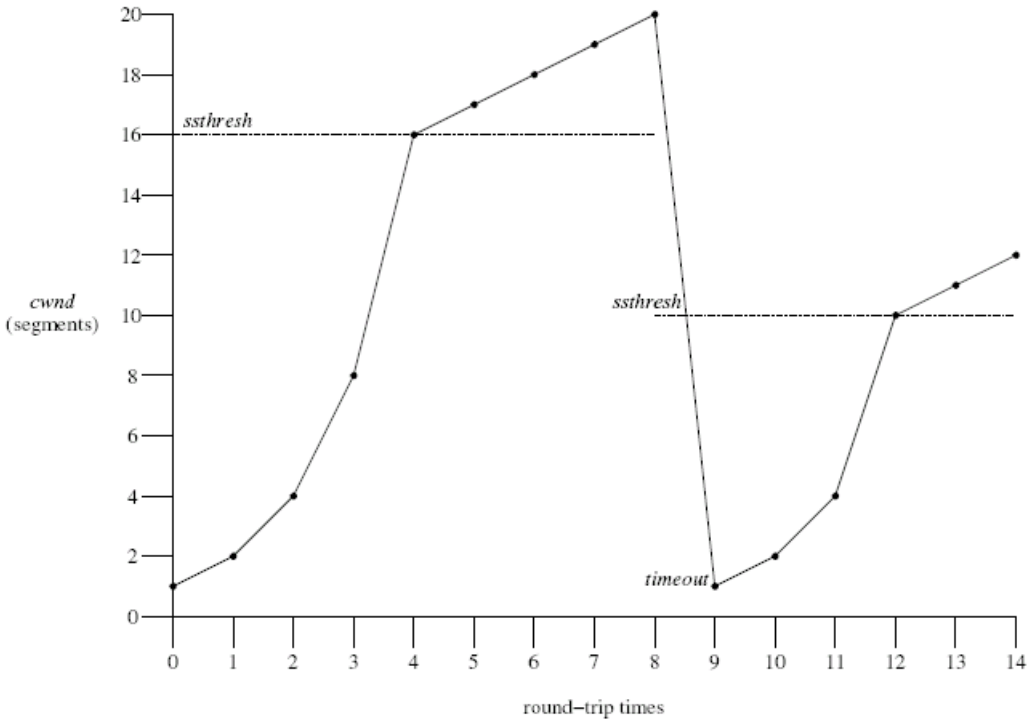


FIGURE 2.3 – Fonctionnement de Fast Restransmit sur 3 ACKs dupliqués

Remarques

- L'ACK de A permet de ré-ouvrir une nouvelle fenêtre de congestion, mais le premier ACK dupliqué de A demandant B empêche l'émission d'un nouveau paquet.
- Puisque l'émetteur ne reçoit pas tous les ACKs des précédents paquets, la taille de la fenêtre n'augmente pas selon *Congestion Avoidance*.
- Après la retransmission (segment B), la fenêtre de congestion est ré-initialisée à 1 MSS. Si le paquet retransmis n'arrive pas, c'est le RTO qui déclenchera la retransmission.

Nous pouvons ainsi résumer les 3 phases, *Slow Start*, *Congestion Avoidance* et *Fast retransmission* :



2.2.4 Fast recovery

Principe :

Le problème avec *Fast Retransmit* est qu'à chaque fois qu'une perte de paquet est détectée, comme nous l'avons vu par l'expiration du RTO ou par la réception de 3 ACKs dupliqués, l'émetteur retourne en phase *Slow Start*.

Cependant, nous pouvons nous demander pourquoi utiliser cet algorithme de récupération de débit de transmission et ré-initialiser la fenêtre de transmission à 1 MSS si nous savons que des paquets peuvent encore être transmis ? (en effet, si l'émetteur reçoit toujours des ACKs, cela veut dire que des paquets ont été reçu).

Ainsi, pour optimiser cette transmission, l'algorithme *Fast recovery* à été développé.

Fonctionnement

Comme cela a été dit précédemment, la réception d'un ACK dupliqué pour le paquet de numéro de séquence X fourni plus d'informations que l'expiration d'un simple *timeout*. En effet, cela signifie que le segment en question n'est pas arrivé de l'autre côté, mais que d'autres segments après lui ont pu être transmis et traités. De ce fait, il y a toujours un flux de données qui peut être préservé.

Nous pouvons ainsi résumer en 5 étapes les actions de *Fast Recovery*, utilisé avec *Fast Retransmit*. Après avoir reçu 3 ACKs dupliqués à la suite :

1. Mise à jour de la valeur seuil *ssthresh* à 1/2 de la taille de la fenêtre de congestion,
2. Retransmission du paquet perdu,
3. Ré-initialisation de la taille de la fenêtre à *sshtresh+3*,

4. Pour chaque ACKs dupliqués reçu, incrémentation de la taille de la fenêtre de 1 MSS et transmission d'un nouveau paquet si possible,
5. Si un ACK non dupliqué est reçu, la fenêtre est ré-initialisée à *ssthresh* et on entre en mode *Congestion Avoidance*.

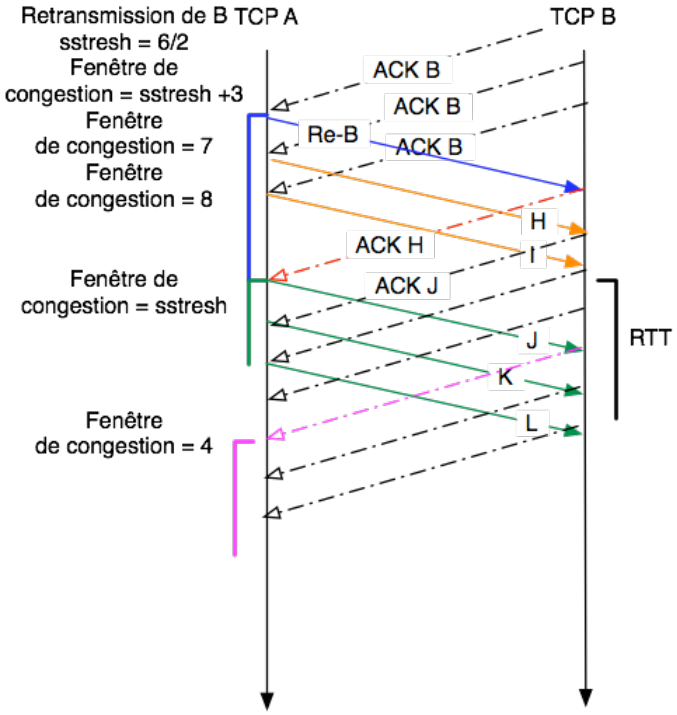


FIGURE 2.4 – Fonctionnement de Fast Recovery après 3 ACKs dupliqués

Explications

Les deux premières opérations sont réalisées par *Fast Retransmit*.

La troisième met la taille de la fenêtre de congestion à la valeur de *ssthresh* nouvellement mis à jour + 3 MSS (nombre de segment qui ont été émis après la perte du paquet et qui génèrent les ACKs dupliqués utilisés pour la détection de cette dernière).

Dans notre cas, le troisième ACK dupliqué reçu provient du paquet E. Comme la fenêtre avait une taille de 6, le paquet F a pu être transmis et son ACK est également un ACK dupliqué demandant B. La réception de l'ACK de A demandant B a permis de recommencer une nouvelle fenêtre et d'émettre le paquet G avant la réception du 3ème ACK dupliqué.

L'étape numéro 4 permet d'envoyer un segment supplémentaire en remplacement de celui qui est acquitté (générant un ACK dupliqué). Ceci permet de ne pas arrêter la transmission de données pendant l'attente de l'ACK du paquet retransmis.

Ici, nous pouvons constater l'émission des paquets H et I avec l'augmentation de la taille de la fenêtre de congestion.

Dans l'étape 5, et dans le cas d'une perte d'un seul paquet, le premier ACK non dupliqué reçu (celui du paquet retransmis) ré-initialisera la taille de la fenêtre à *ssthresh* et TCP recommencera un mode *Congestion*

Avoidance pour retrouver le débit maximal de la connexion.

Cas du RTO :
Encore une fois, il existe un autre signal de perte de paquets. Lors de l'arrêt complet de réception d'ACK, l'émetteur attend jusqu'à l'expiration du RTO (Restransmission Timer Out).

Lorsque le RTO a expiré, les actions misent en place sont assez similaire à la réception d'un ACK dupliqué. Dans un premier temps, l'émetteur met à jour *ssthresh* avec la valeur de 1/2 la taille de la fenêtre d'émission actuelle. Cependant, l'émetteur ne peut réaliser d'hypothèse concernant l'état actuel du réseau :

Congestion ou simple perte d'un paquet en fin de séquence ?

Dans ce cas, l'expéditeur réduit la fenêtre de congestion à un seul segment, et redémarre en mode *Slow Start.*

2.3 Les protocoles TCP d'optimisation du contrôle de congestion

2.3.1 TCP Tahoe et Reno

TCP Tahoe et Reno sont deux variations du protocole TCP classique. Ces deux versions ont été nommés ainsi lors de leur première apparition dans la distribution 4.3 du système Unix BSD.

TCP Tahoe apparu dans la version BSD 4.3 du même nom (conçut pour supporter le CCI "Tahoe" mini-computer). TCP Tahoe a été distribué sous licence BSD et non AT&T afin d'assurer sa large distribution et implémentation. Quelques améliorations, ont été ensuite inclues dans 4.3 BSD-Reno.

Les comportement de Tahoe et Reno sont diffèrents dans la manière dont ils détectent et réagissent à la perte de paquet :

TCP Tahoe

Il existait deux versions de Tahoe. La première n'intégrait que *Slow Start* et *Congestion Avoidance.*

La situation lorsque plusieurs paquets étaient perdus dans une séquence d'émission est presque la même que lorsqu'un seul paquet est perdu. La perte du premier paquet n'est détectée qu'après l'expiration du RTO correspondant. Cette implémentation n'est pas très performante car en plus du temps d'attente de détection de la perte, le protocole retourne en mode *Slow Start* et recommence l'émission avec une fenêtre d'1 MSS.

La seconde version intègrait *Fast Retransmit.* Avec ce dernier, TCP pouvait détecter les pertes de paquets avant l'expiration du RTO et effectuer la retransmission directement. Mais cette version de TCP n'est plus très utilisée, au profit d'autre version plus efficace à d'autres niveaux.

TCP Reno

TCP Reno est en fait une évolution de Tahoe intégrant Fast Recovery. Ainsi, si 3 ACK identiques sont reçu (ACKs d'un même paquet, n'utilisant pas le principe de "piggybacking" et qui ne modifient pas la fenêtre du récepteur), Reno réduit de moitié la taille de la fenêtre de congestion et réalise une retransmission rapide avant d'entrer en phase "Fast Recovery". Cette phase permet en plus d'exploiter un maximum la connexion établie lorsqu'il n'y a pas de congestion (émission de nouveaux paquets en même temps que la phase de retransmission).

Par contre, si un ACK n'est pas reçu avant l'expiration du délai, il considère l'éventualité d'un problème de congestion et entre dans une phase de *Slow Start* comme Tahoe.

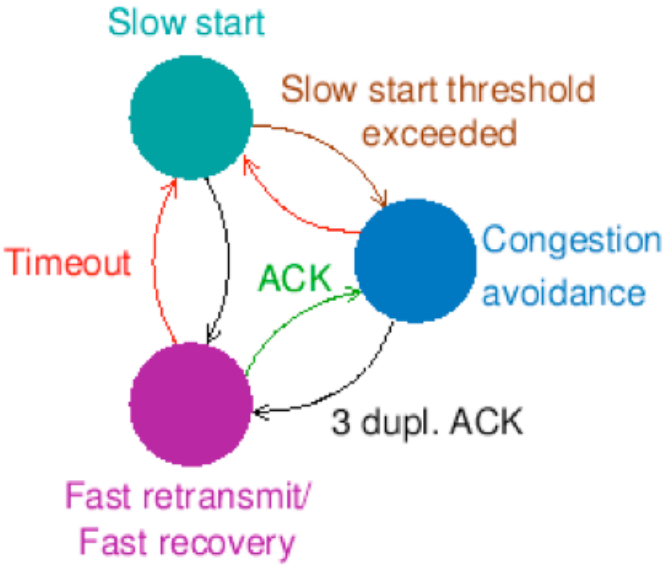


FIGURE 2.5 – Schéma d'interaction entre les différents algorithmes de contrôle de congestion

2.3.2 TCP New Reno

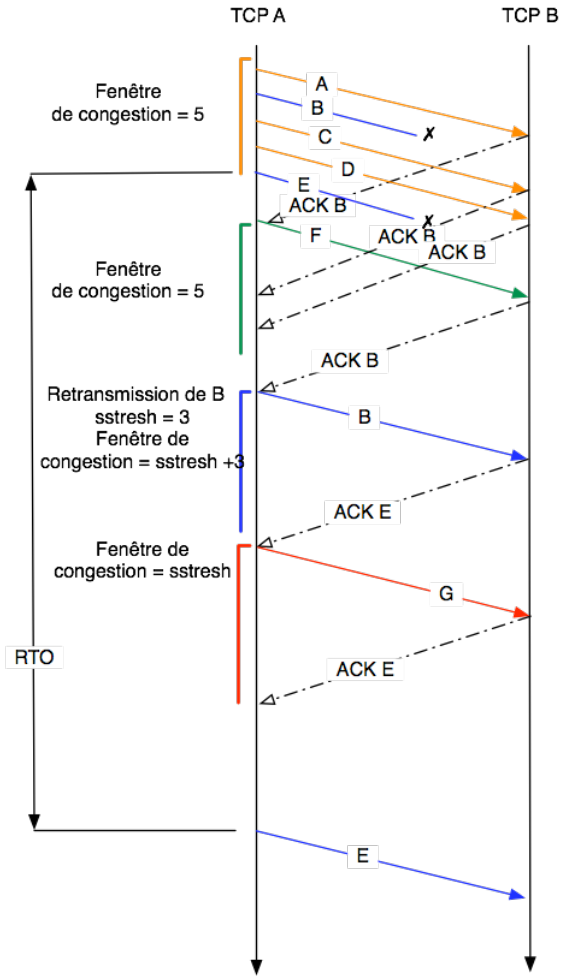
Le problème de TCP Reno

TCP New Reno est une variante de Reno avec une petite modification au niveau de l'algorithme *Fast Recovery*. Cette modification a été effectuée dans le but de résoudre un problème du *timeout* lorsque plusieurs paquets étaient perdus dans la même séquence de transmission.

En effet, dans ce cas, la première nouvelle information que l'émetteur va recevoir est celle de l'ACK du paquet retransmis. Celui-ci va indiquer quel "nouveau" paquet est en attente.

Si il n'y avait qu'un seul paquet de perdu et pas de problème d'ordonnancement, alors l'accusé de réception de ce paquet va concerner tout les paquets envoyés avant *Fast Retransmit*. Cependant, s'il y avait plusieurs paquets de perdu, alors l'ACK du premier paquet retransmis ne concernera pas tous les paquets transmis avant *Fast Retransmit*.

Exemple de fonctionnement :

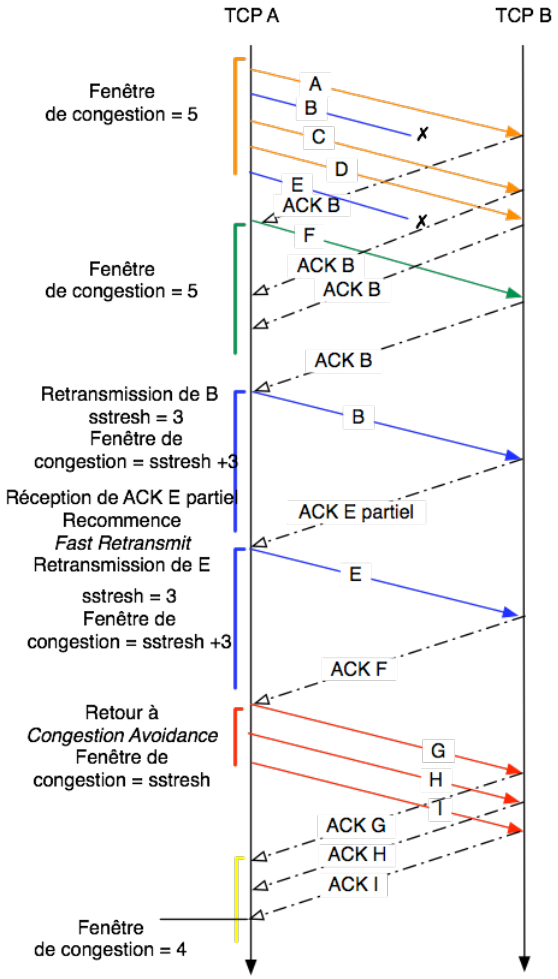


1. Les premiers paquets sont acquittés correctement ;
2. Les paquets B et E, respectivement paquets 2 et 5 d'une fenêtre de taille 5, sont perdus ;
3. Les segments C, D et F génèrent donc des ACKs dupliqués du segment A. Ces 3 ACKs déclenchent *Fast Retransmit* et *Fast Recovery* ;
4. Le paquet B est retransmis ;
5. L'ACK reçu confirme les paquets B, C et D et demande le segment E. C'est un ACK nouveau et différent des 3 précédents. La fenêtre de congestion est ré-initialisée à *ssthresh* et un nouveau paquet est émis (G)
6. Le segment G est donc retransmis et génère un ACK pour le segment E ;
7. De part la taille de la fenêtre de congestion au moment du problème, il n'y a plus de paquet à émettre, donc plus d'ACKs et il faut attendre l'expiration du RTO du segment E pour continuer.

ACK partiel et ACK complet

Il a été suggéré dans la RFC 3782 que, pendant la phase *Fast Recovery* de New Reno, l'émetteur réponde à un ACK partiel. A partir de cet ACK, TCP doit en déduire que le prochain paquet désigné par son numéro de séquence a également été perdu et qu'il doit le retransmettre.

L'algorithme de TCP New Reno diffère de celui de Reno (RFC 2581) par l'utilisation d'une variable "recover" dont la valeur initiale est fixée au premier numéro de séquence émit. Cette variable permet de faire la différence entre un ACK partiel et complet.



ACK Complet :

Si cet ACK permet d'acquitter toute les données >= à "recover", alors, cet ACK acquitte tout les segments intermédiaires entre la transmission originale du paquet perdu et la réception du 3eme ACK dupliqué. A ce moment on ne change rien.

ACK Partiel :

Si cet ACK ne permet pas d'acquitter tout les segments >= à "recover", alors, c'est un ACK partiel. Dans ce cas, on rentre de nouveau en mode *Fast Retransmit* suivi de *Fast Recovery*.

Bien que New Reno résolve le problème du *timeout*, lorsque plusieurs paquets sont perdus dans une même fenêtre de congestion, il ne peut retransmettre qu'un seul paquet par RTT.

2.3.3 TCP Vegas

Principe

TCP Vegas est une version du protocole TCP, développé par l'université d'Arizona, qui se base sur TCP Reno et ses algorithmes de contrôle de congestion. Vegas modifie la plupart de ces algorithme et possède la particularité d'anticiper les retards des paquets plutôt que d'uniquement traiter leur perte.

Pour ce faire, il repose sur le calcul d'une estimation plus précise du RTT et d'autres délais. En effet, jusqu'au milieu des années 90, toutes les versions de TCP fixaient et mesuraient les valeur des RTT en se basant seulement sur les informations issues du dernier paquet transmis. Dans Vegas, les délais sont déterminés pour tous les paquets présents dans le buffer de transmission.

De plus, TCP Vegas utilise une gestion plus fine de la taille de la fenêtre de congestion (augmentation et diminution additive), de manière à utiliser au mieux la bande passante disponible dans le réseau.

Calcul plus précis du RTT

Dans TCP Reno, le RTT et ses variantes sont calculées à partir d'une valeur initiale d'environ 500ms. Cette valeur étant élevée, l'estimation du RTT ne se révèle donc pas très précise. La mesure du RTT influence à la

fois la précision du calcul lui-même, puisqu'il se base sur les valeurs antérieures de RTT, mais également la fréquence à laquelle le protocole TCP vérifie l'expiration de ce délai pour un segment en transit. De manière générale, TCP Reno estimait la valeur du RTT à plus de 3 fois la valeur réel pour un paquet.

Vegas présente donc une autre manière de calculer cette valeur de RTT. Au lieu de diminuer l'effet du RTT mesuré, il conserve la valeur minimale entre le RTT estimé et le RTT mesuré. Le RTT mesuré est déterminé par l'enregistrement de l'intervalle de temps entre l'émission du segment et la réception de son ACK.

"Congestion Detection and Avoidance"

Principe Les mécanismes de détection et de contrôle de congestion de TCP Reno utilisent la perte d'un segment comme signal de congestion dans le réseau. Ils ne possèdent aucun mécanisme d'anticipation de congestion, avant que des pertes n'arrivent, pour prévenir cet événement. Ainsi, TCP Reno est réactif plutôt que proactif. Il est obligé de "créer", de forcer la perte de données afin de déterminer la bande passante de la connexion.

L'approche de Vegas pour détecter un état de congestion se base sur la valeur du débit d'émission. Il compare le débit d'émission mesuré avec celui attendu. L'idée que Vegas exploite est simple. Le nombre d'octets en transit est directement proportionnel au débit attendu, et donc, comme la taille de la fenêtre augmente, causant l'augmentation du nombre d'octets en transit, le débit de la connexion devrait également augmenter.

Vegas utilise ce principe pour mesurer et contrôler la quantité de données supplémentaires en transit. Par données supplémentaires, il faut comprendre des données qui n'auraient pas à être émise si la bande passante utilisée par la connexion était exactement celle disponible sur le réseau. Le but de TCP Vegas est de maintenir la "bonne" quantité de donnée supplémentaire sur le réseau.

Evidement, si une connexion envoit trop de données, cela causera un problème de congestion. Moins évident cette fois, si une connexion envoit trop peu de données, TCP Vegas ne peut pas répondre assez rapidement aux variations de la bande passante disponible du réseau.

Pour gérer la taille de la fenêtre (donc le débit d'émission), Vegas définit deux valeurs seuil arbitraires α et β (avec $\alpha < \beta$), correspondant respectivement aux seuils de données supplémentaires minimal et maximal sur le réseau.

Les actions réalisées par Vegas au cours de *Congestion Avoidance* sont basées sur l'évolution de l'estimation de cette quantité de données supplémentaires sur le réseau et pas seulement sur les segments perdu.

Algorithme

1. Vegas commence par définir une valeur de base du RTT d'un segment lorsque la connexion n'est pas en état de congestion. En pratique, Vegas fixe cette valeur de base au minimum de toutes les mesures de RTT effectuées.
2. A partir de là, Vegas calcule le débit de données attendu :

   $$Expected = \frac{WindowSize}{BaseRTT}$$

   Où *WindowSize* est la taille de l'actuelle fenêtre de congestion, qui doit être égale au nombre d'octets en transit.

3. Vegas calcule ensuite la valeur réelle du débit en enregistrant l'heure d'émission du paquet et le nombre d'octets transmis entre le moment où le segment est émis et celui où son ACK est reçu. Il calcul également le RTT pour le segment en question lors de l'arrivée de son ACK et divise le nombre d'octets transmis

par le RTT mesuré. Ce calcul est effectué une fois par RTT.

$Actual = \frac{Bytestransmited}{RTTmesured}$

4. Vegas compare les valeurs des débits actuels et attendus et ajuste la fenêtre de congestion en conséquence.

   $Diff = (Expected - Actual)$

   - Si *Diff* < 0, alors *BaseRTT* prend la valeur de *RTT Mesuré*.
   - Si *Diff* $< \alpha$, Vegas augmente la fenêtre de congestion linéairement (de 1 MSS) pendant le prochain RTT.
   - Si Diff $> \beta$, la fenêtre diminue linéairement.
   - Sinon, Vegas ne modifie pas la fenêtre.

Explication Puisque BaseRTT est le plus petit de tous les RTT, nous avons $1/BaseRTT > 1/RTTmesur$.

Comme le nombre d'octets transmis sur le réseau ne peut pas dépasser la taille de la fenêtre de congestion, nous pouvons donc dire que *Bytes Transmited* <= *WindowsSize*.

Donc, par définition *Diff* >= 0.
- si *Diff* < 0, cela implique que *Actual* > *Expected*, que *RTT mesuré* < *BaseRTT*. Il faut donc mettre à jour la valeur de BaseRTT.
- Intuitivement, si *Diff* $> \beta$, le débit réel est alors très différent du débit attendu. Puisque *Diff* > 0, alors *Actual* « *Expected*, il y a donc congestion et il faut réduire le débit d'émission.
- D'un autre côté, lorsque le débit réel est trop proche du débit attendu ($0 <$ *Diff* $< \alpha$), la connexion risque de ne pas utiliser toute la bande passante disponible, nous pouvons donc augmenter la taille de la fenêtre.

Congestion Detection dans Slow-Start

L'algorithme Slow Start utilisé dans TCP Reno est très coûteux en terme de pertes. En effet, si la taille de la fenêtre de congestion double à chaque RTT, quand il n'y a pas de perte (ce qui est équivalent à doubler le débit), lorsque celle-ci dépasse finalement la quantité de données transmissible par la bande passante de la connexion, il est possible d'obtenir une perte de données pouvant s'élever jusqu'à la moitié de la fenêtre.

Afin de ne plus avoir ce genre de pertes, Vegas incorpore son mécanisme de détection de congestion à *Slow Start*. Cela permet de détecter et d'éviter l'état de congestion.

La mesure de *Diff* est réalisée à taille de fenêtre constante pendant un certain RTT. La croissance exponentielle de la fenêtre est donc uniquement autorisée pendant les RTT suivants.

Lorsque le débit réel dépasse une certaine quantité la valeur minimale du débit attendu, valeur appelée seuil ϕ, Vegas passe de *Slow Start* à *Congestion Avoidance* (croissance / décroissance linéaire).

De la même manière que α et β, ϕ est déterminé expérimentalement.

Mécanisme de décision de retransmission

Nous avons vu que TCP Reno ne retransmettait pas seulement à l'expiration d'un délai, mais également lorsqu'il recevait 3 ACKs dupliqués d'un même paquet selon l'algorithme *Fast Retransmit*.

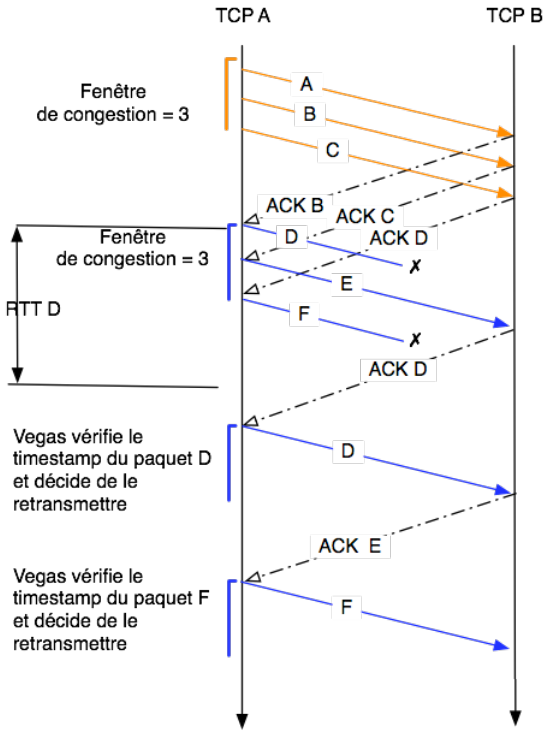


FIGURE 2.6 – TCP Vegas

TCP Vegas optimise encore la retransmission tel que, lorsqu'un ACK dupliqué est reçu, il calcule la valeur du RTT (vérifie la différence entre le temps courant et l'heure d'émission enregistré) pour le segment concerné.

Si le RTT mesuré est supérieur à la valeur du RTT attendu, alors Vegas retransmet directement le segment sans attendre les deux autres ACKs dupliqués.

Dans la plupart des cas, les pertes sont si importantes ou la fenêtre de congestion est si petite que l'émetteur ne recevra jamais les 3 ACKs dupliqués. De ce fait, TCP Reno ne pourra qu'attendre l'expiration du RTO avant de retransmettre le paquet perdu.

TCP Vegas gère également la perte de plusieurs paquets dans une même séquence d'émission, tout comme New Reno. En effet, Lorsqu'un ACK non dupliqué arrive et si c'est le premier ou le deuxième après une retransmission, Vegas re-vérifie si l'intervalle de temps depuis que le segment à été envoyé n'est pas plus grand que le délai autorisé. Si tel est le cas, Vegas retransmet le paquet. Ce mécanisme permet de prendre en compte tout les paquets qui ont été perdu avant la première retransmission et ce sans attendre d'autres ACKs dupliqués.

En d'autres termes, Vegas traite la réception d'un ACK dupliqués comme un signal pour vérifier si l'expiration d'un délai doit être prise en compte. Il conserve néanmoins le mécanisme de RTO de TCP Reno au cas où celui-ci ne fonctionnerait pas.

Ajustement de la fenêtre de congestion

La fenêtre de congestion devrait être seulement réduite lors de pertes qui ont eu lieu au débit d'émission actuel et pas à cause des pertes qui on eu lieu à un débit plus important précédent. Dans l'implémentation TCP Reno, il est possible de réduire la fenêtre de congestion plus d'une fois pour des pertes qui ont eu lieu pendant 1 unique intervalle RTT.

TCP Vegas est meilleur que Reno en ce qui concerne la détection des pertes de segments. Si Vegas utilisait la politique de re-dimensionnement de la fenêtre de congestion de TCP Reno, il diminuerait la fenêtre plus de fois que Reno simplement car il détecte les pertes plus tôt que Reno.

Pour éviter ces diminutions excessives, Vegas ne réduit la taille de la fenêtre que si le segment retransmis a été envoyé après la dernière diminution. Toute perte qui a eu lieux avant la dernière diminution de la fenêtre de congestion ne doit pas impliquer que le réseau est congestionné pour la taille actuelle de la fenêtre. De ce fait, cela ne doit pas non plus impliquer que la fenêtre soit diminuée encore un fois.

2.4 Réseau à très haut BDP

2.4.1 Principe

Cette partie, comme d'autre, pourrait faire office d'un étude à elle seule.

En effet, TCP présente quelques problèmes liés aux réseaux à très haut "BDP" également appelés "Long Fat Networks".

Le BDP, produit capacité-délai, est la quantité Débit*RTT qui désigne souvent la valeur "optimale" de la fenêtre d'émission, dans le sens où elle permet à TCP de transmettre en flux continu.

Ces problèmes concernent :

- Une sous-utilisation de la capacité de liaison du réseau ;

  Puisque le champ taille de la fenêtre de l'en-tête TCP porte sur 16 bits, il n'est pas possible d'avoir une fenêtre de réception $>$ 216 - 1 Octets.

  Les mécanismes de contrôle de congestion limiteront également le débit d'émission en fonction des appareils et noeuds intermédiaires qui peuvent représenter un goulot d'étranglement.

- Un problème d'estimation du RTT ;

  La méthode encore utilisée pour estimer la valeur du RTT comporte quelques erreurs qui peuvent être acceptables sur des fenêtres de petites tailles mais qui s'avère trop importantes pour des valeurs supérieures.

- Un problème de numérotation des séquences ;

  L'en-tête de TCP est codé sur des mots de 32 bits. Ainsi, il est possible de transporter des données d'environs 4Go (232). Dans les réseaux à haut BDP, il peut y avoir un problème de numérotation dans le sens où le compteur peut refaire un "cycle" d'attribution de numéros alors que des segments dupliqués sont encore en transit sur le réseau.

2.4.2 TCP CUBIC

TCP CUBIC, comme de nombreuses autres versions de TCP, a été développé pour faire face aux problèmes des réseaux à haut BDP.

CUBIC est une amélioration de BIC, "Binary Increase Congestion control", dans laquelle la gestion de la taille de la fenêtre est simplifiée et améliorée. Celle-ci est définie par une fonction temporelle cubique et ne

dépend plus de la valeur du RTT.

$W_{cubic} = C(t-K)^3 + W_{max}$

W taille de la fenêtre, C constante d'échelle, t temps écoulé depuis la dernière diminution de la fenêtre et $K = (W_{max} * \beta/C)^(1/3)$ (avec β facteur de diminution).

Le comportement de cette fonction est une diminution progressive de la croissance de la taille de la fenêtre plus W_{cubic} s'approche de la valeur de W_{max}.

Comme BIC, TCP CUBIC est capable d'augmenter la taille de la fenêtre de congestion au delà de sa valeur maximale initiale. Il permet ainsi d'utiliser au mieux la bande passante réelle.

La fenêtre arrête de croître au moindre paquet perdu, et la valeur W_{max} est ainsi définie.

TCP CUBIC est implémenté et utilisé par défaut dans le noyau Linux depuis la version 2.6.19, preuve de son efficacité.

2.5 Conclusion

TCP New Reno est toujours la version de TCP la plus communément implémentée.

La plupart des autres algorithmes en proposition sont en compétition les uns des autres, et ont encore besoin d'être évalués.

Cependant la version 2.6.18 du noyau Linux changea l'implémentation par défaut de TCP Reno par celle de BIC, qui fut encore changée par CUBIC dans la version 2.6.19.

Lorsque le besoin en bande passante par flux et le temps de latence augmentent, quel que soit le régime de mise en file d'attente utilisé, TCP devient inefficace et sujet à instabilité. Ceci devient de plus en plus important puisque Internet évolue et incorpore des connexions optiques à très grande bande passante.

Chapitre 3

Mécanismes d'amélioration du protocole TCP

Il existe plusieurs mécanismes permettant d'améliorer les performances du protocole TCP dans un milieu sans-fil. Les mécanismes que nous allons étudier peuvent être classés en 3 catégories : Les mécanismes de bout-en-bout, les mécanismes de fractionnement de la connexion et les mécanismes de récupération locale.

3.1 Mécanismes de bout-en-bout

La sémantique de bout-en-bout est un élément très important dans l'Internet. Il permet aux noeuds du réseaux d'envoyer des paquets à d'autres noeuds du réseau sans nécessiter d'éléments du réseau d'intermédiaire.

Les mécanismes de bout-en-bout tentent de corriger les problèmes de pertes de connexions au niveau de la couche transport de l'émetteur et du récepteur.

L'émetteur et le récepteur sont responsables du débit de transfert des données : vitesse d'émission, moment d'émission, régulation du débit.

3.1.1 TCP Probing

Avec TCP Probing, lorsqu'un segment de données est en retard ou est perdu, l'émetteur au lieu de le réémettre et de réduire la taille de la fenêtre de congestion, il entre dans un cycle de sondage (probe). Un cycle de sondage entraîne l'échange de segments tests entre l'émetteur et le récepteur afin d'écouter le réseau.

Ces segments tests sont des segments TCP contenant des header avec certaines options mais sans données. Cela permet d'aider à soulager le réseau car ces segments sont légers comparés aux segments à retransmettre. Le cycle se termine lorsque l'émetteur peut réaliser 2 mesures successives du RTT à partir de ces segments tests. Si les mesures sont réalisées avec succès, l'émetteur reprend ses envois avec la même taille de fenêtre qu'avant être entré dans le cycle. En revanche, en cas d'erreurs persistantes, TCP diminue sa fenêtre de congestion et son seuil.

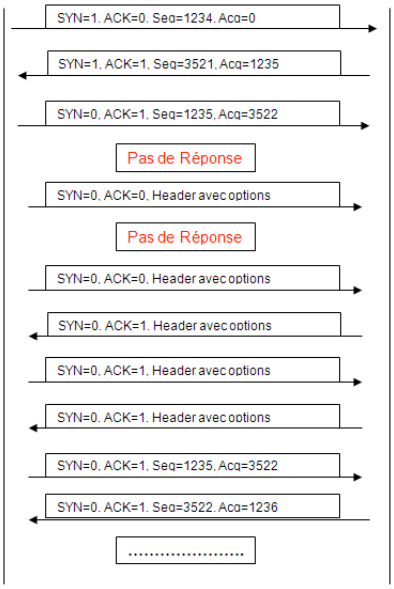


FIGURE 3.1 – Échanges avec TCP Probing

3.1.2 Freeze-TCP

Freeze-TCP est un pur protocole de bout-en-bout. Il suppose que la station mobile a la capacité de prévoir une imminente perte de liaison en étudiant la puissance du signal. Si la station réceptrice pressent une perte de liaison, elle va alors envoyer un ZWA (*Zero Window Advertisement*) ce qui va forcer l'émetteur d'entrer en mode persistant (TCP persist mode) sans qu'il ne croit à un problème de congestion. L'émetteur va alors geler tous les *timers* relatifs à la session actuelle et envoyer des sondes au récepteur (appelées *Zero Window Probes* (ZWP)) afin de repérer le moment où la fenêtre du récepteur se ré-ouvre.

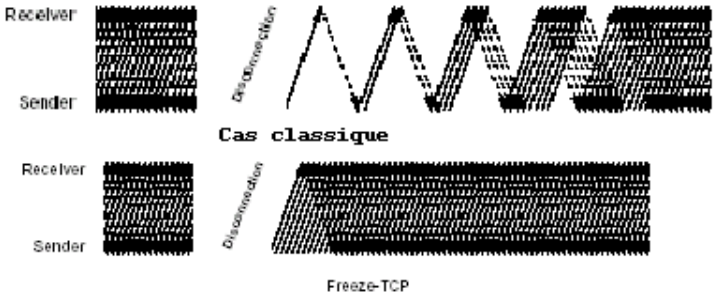


FIGURE 3.2 – Performance Freeze TCP

En théorie, chaque ZWP doit contenir exactement 1 octet de données mais généralement les implémentations de ce protocole TCP n'incluent aucune données dans le ZWP (notamment pour les distributions Linux et FreeBSD).

L'intervalle entre chaque envoi de ZWP augmente exponentiellement jusqu'à atteindre une durée maximale de 1 minute où il restera constant par la suite.

Lorsque la station mobile sera de nouveau à portée du signal, elle répondra aux ZWP par une taille de fenêtre non nulle et l'émetteur pourra alors continuer sa transmission en utilisant la taille de fenêtre correspondante.

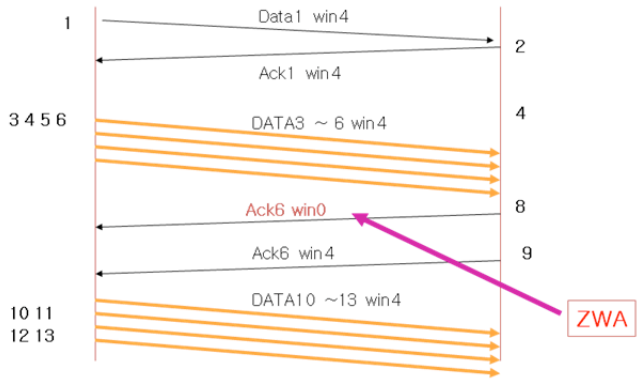


FIGURE 3.3 – Echanges Freeze 1

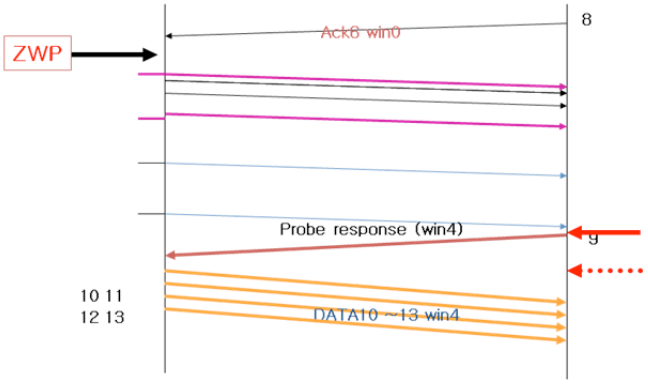


FIGURE 3.4 – Echanges Freeze 2

L'intérêt de cette méthode est qu'il n'y a aucun changement à effectuer sur la machine émettrice ni sur les routeurs intermédiaires. Seule la station mobile devra être modifiée. Freeze-TCP ne présente un intérêt que dans le cas où une déconnexion apparaît pendant l'échange de données. Ce protocole ne présente pas d'intérêt dans le cas de déconnexion sans transaction entre l'émetteur et le récepteur.

3.1.3 Wireless TCP

WTCP a été développé pour les réseaux WWAN où tous les algorithmes TCP échouent en considérant faussement les pertes de paquets comme des problèmes de congestion. Ce protocole tente de différencier les pertes de données aléatoires, dues à la connexion, des problèmes de congestion en se basant sur le taux de transmission et le délai inter-paquets.

WTCP propose une couche transport possédant ses propres mécanismes de fiabilité et de contrôle de congestion. Tout d'abord, le taux de transmission est déterminé par le récepteur en utilisant un ratio du délai moyen inter-paquet pour le récepteur et du délai moyen inter-paquet de l'émetteur. L'algorithme utilisé est *packet-pair* qui remplace *Slow Start*. L'émetteur transmet son délai inter-paquet courant dans chaque paquet de données.

Le récepteur va tenir à jour un historique des pertes de paquets et va calculer le taux moyen de transmission en cas de non-congestion en se basant sur les informations contenues dans les paquets et transmet ce taux à la station émettrice à l'intérieur d'un ACK.

En maintenant un historique de ces valeurs, WTCP arrive à faire la distinction entre les pertes de paquets dues à la mauvaise connexion et celles dues à un problème de congestion. WTCP va alors réagir en adaptant son taux de transmission soit de manière agressive en cas de congestion (réduction de 50%), soit de manière plus modérée si la perte provient de la liaison sans-fil.

Dans ce dernier cas, WTCP va utiliser un ratio entre le temps moyen inter-paquets du récepteur et le délai moyen inter-paquets de l'émetteur.

En pratique, WTCP va garder à jour 2 variables : lavg_ratio et savg_ratio pour les moyennes à long et court terme des délais moyens inter-paquets au niveau du récepteur.

De plus, WTCP s'assure de la bonne transmission des paquets non pas en utilisant des *timeouts* de retransmission mais en utilisant des acquittements cumulatifs et sélectifs. L'émetteur compare l'état des paquets non acquittés dans les ACK du récepteur avec ceux qui avaient été sauvegardé précédemment lors de la dernière retransmission afin de déterminer si un paquet est perdu ou est en cours d'acheminement. Au bout d'un intervalle prédéfini, si l'émetteur n'a toujours pas reçu d'ACK il va entrer en mode appelé *blackout* et comme TCP probing, va utiliser des sondes pour résoudre le problème.

L'inconvénient de ce protocole est qu'il nécessite des modifications pour toutes les stations qui désirent l'utiliser.

3.1.4 TCP Westwood (TCPW)

TCP Westwood estime la bande passante réelle au niveau de l'émetteur sans aucune modification pour le protocole TCP du récepteur. Cette estimation de la bande passante est basée sur les retours d'ACK et permet donc de limiter la quantité de données émises. Un ACK dupliqué (DUPACK) met à jour l'estimation de la bande passante (BWE) dès qu'il notifie l'émetteur qu'un paquet à été reçu à destination, même si le paquet a été défaillant. Lorsque n DUPACK sont reçus, le plus bas seuil de départ est mis à jour de la façon suivante :

$$ssthresh = \frac{(BWExRTTmin)}{SS}$$

Avec RTTmin la plus basse mesure du RTT et SS la taille du segment TCP.

La fenêtre de congestion est initialisée à *ssthresh* celui-ci est supérieur à la taille de la fenêtre actuelle. Ainsi la vitesse de connexion reste proche de la capacité du réseau si l'estimation de la bande passante est grande. Au cas où le *timer* expire, *ssthresh* est placé au minimum et la fenêtre de congestion est placée à 1 comme dans TCP Reno.

3.2 Mécanismes de fractionnement de la connexion

Les mécanismes de fractionnement de la connexion (*Split connexion mechanisms*) découpent la connexion en 2 au niveau du point d'accès. Le protocole TCP continue ainsi d'être utilisé par les machines connectées au réseau câblé jusqu'au point d'accès puis le relai est donné à un autre protocole plus performants en milieu sans-fil ou peut continuer à être TCP pour le transport des paquets entre le point d'accès et la station mobile. De cette manière, l'émetteur du paquet TCP est uniquement affecté par les problèmes de congestion dans la partie câblé du réseau et n'est plus concerné pas les pertes de liaisons au-delà du point d'accès. L'intérêt de ce fonctionnement est que le protocole de transport utilisé entre le point d'accès et la station mobile peut être spécialisé dans le transport de paquets dans un milieu sans-fil présentant de forts risques de perte de données. Mais le problème avec ce type de solution est une perte d'efficacité et de robustesse.

3.2.1 Indirect TCP (I-TCP)

Le protocole I-TCP est apparu en 1994-1995 et est un des premiers protocoles TCP pour les réseaux sans-fil.

Indirect TCP (I-TCP) suggère que toute connexion de la couche transport entre un hôte mobile et un hôte se trouvant sur le réseau câblé devrait être découpée en 2 connexions : une entre l'hôte mobile et la station de base via la connexion sans fil et une autre entre la station de base et l'hôte sur le réseau câblé. Au niveau de la station de base se trouve le MSR (Mobile Support Router), qui est un routeur spécifique. Pour cela, les données envoyées en direction de l'hôte mobile sont insérées dans le buffer du point d'accès et celui-ci s'occupe d'envoyer un ACK à la station émettrice. C'est ensuite de la responsabilité du point d'accès de délivrer correctement la donnée à l'hôte mobile.

Le point d'accès communique avec l'hôte mobile à l'aide d'une connexion séparée qui utilise une variante de TCP qui est optimisée pour les connexions sans-fil et qui prend en compte la mobilité de l'hôte.

Lorsque la station se déplace et vient à se connecter à une atre station de base, l'état de la connexion est transféré de SB1 vers SB2. L'indirection est transparente pour la station fixe et la station mobile étant donné que leurs ports de communication restent inchangés.

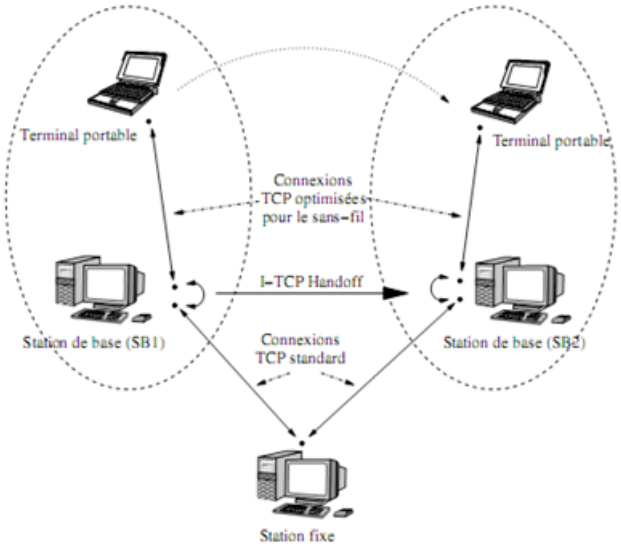


FIGURE 3.5 – Communications avec I-TCP

En utilisant I-TCP les performances sont meilleures qu'avec le TCP classique mais la sémantique bout-en-bout du TCP n'est pas conservée à cause de la découpe de la liaison en deux au niveau du point d'accès.

Les avantages de cette méthode sont :
- Les hôtes connectés au réseau câblé n'ont pas de changement à effectuer sur leurs protocoles TCP.
- Les erreurs de transmissions sur la liaison sans-fil ne se propagent pas dans le réseau câblé. La récupération d'erreurs est locale.
- La possibilité d'utiliser un protocole de transport optimisé dans les réseaux sans-fil pour la connexion entre le point d'accès et l'hôte mobile.

L'inconvénient majeur de cette technique est qu'elle nécessite d'importants changements pour la station de base et celle-ci devra posséder une mémoire tampon importante en cas de trafic important.

En cas de déplacements fréquents de la station mobile, celle-ci peut-être amenée à changer de nombreuses fois de station de base et la transmission des informations de connexions entre celles-ci peuvent entraîner des délais.

3.2.2 Mobile TCP (M-TCP)

M-TCP utilise également un fractionnement de la connexion mais tente de conserver la sémantique de bout-en-bout. Il est particulièrement adapté aux milieux où les stations mobiles rencontrent des périodes de

déconnexion régulières et aux changements dynamiques de la bande passante sur la liaison sans-fil.

M-TCP est très proche de I-TCP que nous avons étudié précédemment et découpe la connexion entre une machine sur le réseau câblé et une station mobile en 2 au niveau du point d'accès.

La différence est que la connexion entre le point d'accès et la station mobile passe par l'utilisation d'un protocole de la couche session. Lorsque l'hôte sur la liaison câblé émet un paquet, le point d'accès le réceptionne et le transmet à la station mobile. La station mobile va alors renvoyer un accusé de réception à la station émettrice que le point d'accès va copier en mémoire dans un buffer. Contrairement à d'autres méthodes, le point d'accès va sauvegarder uniquement l'ACK correspondant au dernier paquet transmis.

Désormais dans le cas où la station mobile est déconnectée, le point d'accès ne recevra plus d'ACK de la part de la station mobile et va en déduire que celle-ci a été temporairement déconnectée du réseau. Le point d'accès renverra alors à la station émettrice l'ACK du dernier paquet transmis correctement qu'il avait précédemment sauvegardé en modifiant dans l'en-tête de celui-ci la taille de la fenêtre qu'il place à zéro (*ZWA*). Ceci va provoquer l'entrée de la station émettrice en mode persistant et va donc geler tous ses timers relatifs à la session actuelle et commencera à envoyer au point d'accès des *Zero Window Probes* (ZWP). Le point d'accès répondra à ces requêtes par un ZWA jusqu'à ce qu'il reçoive par la station mobile une taille de fenêtre non nulle. Dès la réception d'un tel paquet, le point d'accès répond aux requêtes ZWP de la station émettrice en indiquant la taille de fenêtre correcte. L'émetteur pourra alors recommencer ses émissions avec une vitesse élevée à partir du dernier paquet qui n'a pas été transmis correctement.

Le fait que le protocole TCP du point d'accès n'envoie pas à l'émetteur d'ACK avant que le récepteur n'en ait émis un à M-TCP permet de conserver la sémantique de bout-en-bout qui est perdue dans I-TCP.

3.2.3 Explicit Bad State Notification (EBSN)

De la même manière, EBSN utilise une méthode de retransmission à partir de la station de base afin d'optimiser la liaison sans-fil en réduisans les erreurs sur celle-ci.

Pour cela, à chaque fois que le point d'accès ne parvient pas à transmettre le paquet, envoyé par l'émetteur, au récepteur via la connexion sans-fil, elle va envoyer à la station émettrice un message EBSN.

A chaque fois que l'émetteur reçoit un message EBSN, celui-ci va réinitialiser le timer à sa valeur initiale ce qui va empêcher de réduire la fenêtre de l'émetteur.

Ce principe nécessite des modifications chez l'émetteur mais ces changements sont faibles.

3.3 Protocoles de couche liaison - Récupération locale

La motivation principale d'une récupération locale est que les pertes de liaisons sont locales et que donc leur récupération devrait être locale.

Ces types de mécanismes peuvent être considérés comme un compromis entre les 2 précédents types.

Le but est de cacher les caractéristiques de la liaison sans-fil de la couche transport aux autres couches et tenter de récupérer les données à partir de la couche liaison.

3.3.1 Snoop Protocol

Le protocole Snoop n'entraîne aucun changement pour la couche transport des stations fixes et mobiles. Seul un nouveau module est introduit pour la station de base appelé agent Snoop. Le principe est que cet agent surveille chaque paquet qui passe par le point d'accès via une connexion TCP quel que soit sa direction. Cet agent va sauvegarder dans le cache du point d'accès une copie des paquets TCP qui n'ont pas reçu d'ACK.

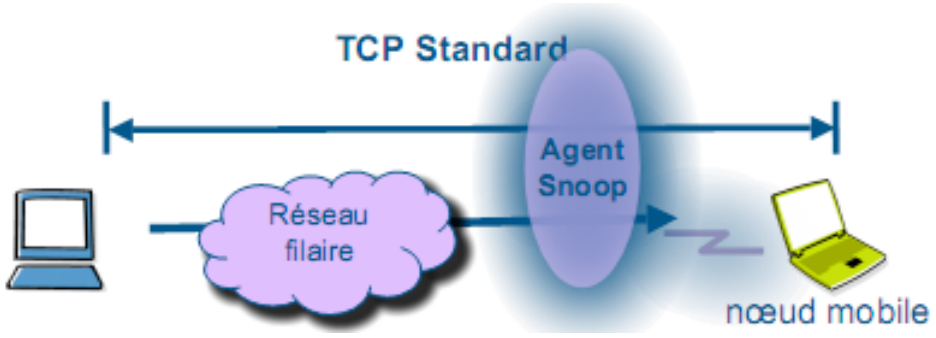


FIGURE 3.6 – Principe de Snoop Protocol

Pour les données transmises à l'hôte mobile : : Le point d'accès sauvegarde dans un buffer les données jusqu'à ce que l'hôte mobile n'envoie un ACK. L'agent Snoop déduit une perte de paquet sur la liaison sans-fil s'il reçoit plusieurs fois un acquittement sur le paquet précédent. Si la station de base possède encore dans son buffer le paquet n'ayant pas pu être remit elle se chargera de le réémettre à la station mobile. Au cas où celui-ci ne soit plus présent dans son buffer, elle acheminera les acquittements dupliqués à la station fixe qui pourra dans ce cas réémettre le paquet manquant. Les paquets perdus sont également déduit grâce aux *timers*.

L'hôte sur la liaison câblé ignore la perte de donnée et cela permet donc que celui-ci ne réduise pas sa fenêtre de congestion.

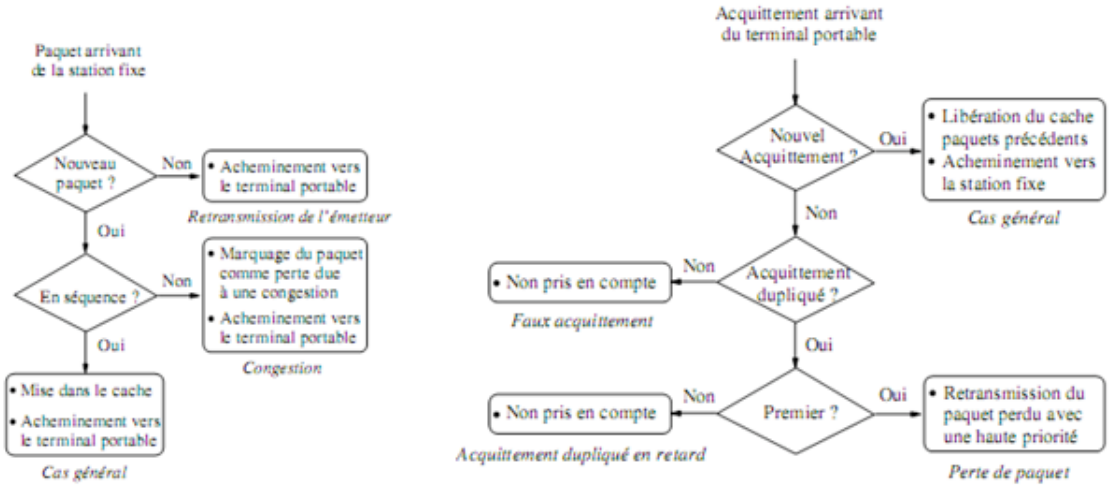


FIGURE 3.7 – Algorithme de Snoop Protocol

Pour les données envoyées par l'hôte mobile : Le point d'accès détecte les pertes de données sur la liaison sans-fil grâce aux numéros de séquences manquants. Le point d'accès répond directement avec l'hôte

mobile avec NACK (négative aknowledge). Cela permet à l'hôte mobile de retransmettre ses données avec un délai très court. En revanche, cela nécessite des modifications chez l'hôte mobile.

Avantages :
- Peut être implémenté sans modifications pour l'hôte mobile.
- La sémantique de bout-en-bout est conservée comme dans le protocole TCP classique.
- Si un crash apparaît quelque part, cela n'affecte pas la transmission mais uniquement la performance de la transmission

Invonvénients :
- Pour que le NACK fonctionne il faut modifier l'hôte mobile.
- Ne fonctionne pas avec des en-têtes TCP cryptés.
- Ne fonctionne pas pour des routages asymétriques.

Conclusion

Durant cette étude portant sur le protocle TCP et les environnements sans-fil, nous avons essayé de mettre en évidence et d'expliquer les problèmes que TCP pouvait rencontrer dans ce type de réseau avant de présenter quelques unes de ses adaptations. Après avoir rappelé le fonctionnement de base de TCP, nous avons pu identifier le principale problème qui résulte de sa conception initiale. En effet, ayant été inventé dans les années 70, le seul environnement existant était filaire. De ce fait, TCP considère toute perte de paquets comme étant un problème de congestion et non pas un problème fiabilité de la connexion.

Nous avons ainsi étudié les différents mécanismes de contrôle de congestion utilisés par TCP afin de comprendre la manière dont il aborde ce problème. TCP décompose cette gestion en trois phases principales. La première consiste à augmenter le plus rapidement possible la taille de sa fenêtre de congestion (quantité maximale de données en transit simultané) jusqu'à un certain seuil dont la valeur est fixée par les expériences antérieures. Dans un second temps, TCP entre dans une phase de progression plus fine afin de déterminer le débit maximal auquel il peut transmettre les données. Nous avons vu que dans la plupart des cas TCP n'anticipe pas la congestion, mais s'adapte après détection de celle-ci lors des premières pertes de paquets. C'est ainsi que la troisième phase des algorithmes de contrôle de congestion de TCP gère la retransmission des paquets perdus et le retour à un débit permettant d'éviter le phénomène de congestion.

Différentes évolutions de TCP ont été réalisées afin de détecter au plus vite ces pertes de paquets. Après l'utilisation de simples timers, de nombreuses versions se basent actuellement sur les accusés de réception sélectifs (New Reno, SACK ...). Les débits de transmissions évoluant sans cesse, de nouveaux problèmes d'optimisation de l'utilisation de la bande passante disponible sont apparus dans TCP. Afin de régler ce problèmes, les versions récentes de TCP utilisent des fonctions de gestion en "temps réel" de la taille de la fenêtre de congestion.

Dans la troisième partie de notre étude, nous avons développé les protocoles les plus utilisés et réputés dans l'amélioration de TCP dans un milieu sans-fil. Nous nous sommes rendu compte qu'il n'existe pas une unique solution mais que le problème peut-être résolu de plusieurs manières. Certains protocoles plus performants que d'autres mais qui généralement ont un coût, notamment celui de perdre la sémantique de bout-en-bout. Il n'existe donc pas une solutions meilleure que les autres mais en fonctions du contexte, de l'architecture réseau certains protocoles sont plus intéressants que d'autres. Il faut notamment faire attention au fait que ces protocoles peuvent demander des modifications du protocole TCP au niveau de l'émetteur ou du récepteur ou bien des deux et parfois également au niveau d'une interface tierce (point d'accès).

La plupart des protocoles de transport ayant été conçu pour des réseaux filaires présentent de nombreuses difficultés lors de leur utilisation dans les réseaux sans-fil. Mais TCP n'est pas le seul protocole à avoir été optimisé pour ces réseaux, le protocole IP a également été amélioré par MobileIP. De nombreux autres devraient subir quelques modifications afin de répondre au mieux aux nouveautés technologiques.

Bibliographie

[1] www-rp.lip6.fr/~spathis/teach/reseaux-tcp-slides.pps.

[10] nrlweb.cs.ucla.edu/publication/download/157/209.pdf.

[17] http://www.csm.ornl.gov/~dunigan/netperf/vegas.html.

[2] www-rp.lip6.fr/~kt/ist/tcp.pdf.

[3] http://www.clubic.com/article-14372-5-les-reseaux-locaux-sans-fil.html.

[4] lifc.univ-fcomte.fr/~dedu/teaching/tcp/cours.pdf.

[5] www.iro.umontreal.ca/~pift3320/files/Transparents/Mobile_Networks/reseauxsansfil.pdf.

[6] suraj.lums.edu.pk/~cs678s06/proposals/05030091-proposal.pdf.

[7] spider.postech.ac.kr/Seminar/Seminar2005/pptFile/FreezeTCP%20by%20shkimm.ppt.

[8] www.cs.umbc.edu/courses/graduate/CMSC628/spring2002/rlist/Freeze.ppt.

[9] www.cs.sunysb.edu/~samir/cse534/TCP-over-wireless.pdf.

[AWB] Stefan Alfredsson Annika Wennstrom and Anna Brunstrom, *Tcp over wireless networks*.

[Ded07] Mr Eugen Dedu, *Cours de contrôle de congestion dans le protocole tcp*, September 2007.

[Fal96] Kevin ; Sally Floyd Fall, *Simulation-based comparisons of tahoe, reno and sack tcp*, July 1996.

[Gui] Mohsen Guizani, *Wireless communications systems and networks*.

[Hus] Geoff Huston, *The isp column an occasional column on things internet*.

[Jac] Van Jacobson, *Congestion avoidance and control*.

[LSB] Sean W. O'Malley & Larry L. Peterson Lawrence S. Brakmo, *Tcp vegas : New techniques for congestion detection and avoidance*.

[Sha02] Hassan Shafazand, *Eurasia-ict 2002 : Information and communication technology*, October 2002.

Table des figures

1.1 Réseau cellulaire 4
1.2 Réseau Ad-Hoc 5
1.3 Réseau satellite 5
1.4 Modèle OSI 6
1.5 Format d'un segment TCP 6
1.6 Schéma d'une connexion TCP 8
1.7 Schéma d'une déconnexion de transaction TCP 9
1.8 Buffer d'émission TCP 9
1.9 Buffer de récéption TCP 10

2.1 Schéma de demande de connexion TCP avec négociation du MSS 13
2.2 Evolution de la fenêtre de congestion en nombre de MSS 14
2.3 Fonctionnement de Fast Restransmit sur 3 ACKs dupliqués 18
2.4 Fonctionnement de Fast Recovery après 3 ACKs dupliqués 20
2.5 Schéma d'interaction entre les différents algorithmes de contrôle de congestion 22
2.6 TCP Vegas 27

3.1 Échanges avec TCP Probing 31
3.2 Performance Freeze TCP 31
3.3 Echanges Freeze 1 32
3.4 Echanges Freeze 2 32
3.5 Communications avec I-TCP 34
3.6 Principe de Snoop Protocol 36
3.7 Algorithme de Snoop Protocol 36