Mémoire de Master en Informatique

Spécialité Recherche, mention Informatique Fondamentale et Appliquée

---

Modèle de coût des communications TCP à un niveau applicatif

---

présenté à

l'Université Louis Pasteur – Département d'Informatique

effectué au

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection

sous la direction de

Stéphane Genaud, Guillaume Latu et Jean-Jacques Pansiot

Rapporté par M. Jean-Michel Dischler

Constantinos Makassikis

Juin 2006

Remerciements

Je remercie tout d'abord mes maîtres de stage, M. Stéphane Genaud, M. Guillaume Latu et M. Jean-Jacques Pansiot, qui m'ont suivi et guidé durant mes cinq mois de stage et sans lesquels le travail présenté dans ce rapport eut été difficile à accomplir.

Par ailleurs, je remercie M. Fabrice Heitz, directeur du LSIIT, pour m'avoir permis d'effectuer ce stage au sein de l'équipe ICPS, ainsi que tous les membres de cette équipe pour m'avoir accueilli et accompagné de leur bonne humeur tout au long du stage.

Parmi les personnes que je voudrais également remercier, figurent M. Pascal Schreck sans l'autorisation duquel je n'aurai pas pu m'inscrire en Master 2, mes camarades de promo et ma famille qui m'a soutenu.

Table des matières

Introduction

Une grille de calcul désigne un ensemble de ressources de calcul situées sur des sites différents et reliées par un réseau rapide. Ces ressources de calcul, qui peuvent être de natures très variées (du simple ordinateur de bureau au supercalculateur parallèle), sont fédérées par un intergiciel et mettent à la disposition des utilisateurs une puissance de calcul jusqu'à présent inégalée.

L'exploitation de ce type de plateforme passe par la programmation d'applications parallèles et distribuées. Ces dernières sont amenées à communiquer et bien souvent les volumes de données échangés sont tels que les temps de transferts rivalisent avec les temps de calcul. L'importance que revêtent les communications lors de l'exécution de telles applications a été soulignée dans de nombreux travaux relatifs aux grilles [6] et elle influence grandement la manière dont sont abordés des problèmes classiques tels que l'équilibrage de charge. Ainsi, par exemple, une machine de puissance moyenne mais connectée via un réseau rapide sera préférée à une machine plus puissante mais connectée via un réseau plus lent (à condition bien sûr que la plus puissante des machines n'arrive pas à compenser). Il en résulte qu'une bonne connaissance des coûts aussi bien de calcul que de communications est nécessaire pour réaliser un bon équilibrage de charge.

En ce qui concerne les coûts liés aux calculs, la tâche est relativement aisée. En effet, moyennant une connaissance des caractéristiques de la machine telles que son architecture, la vitesse du(des) processeur(s) et sa quantité de mémoire vive, il est possible de prévoir le temps d'exécution de façon fiable. Par contre, pour l'estimation des coûts des communications, la tâche s'avère être plus ardue. La consultation de la littérature correspondante dans le domaine des grilles de calcul, révèle rapidement que les modèles de coût de communications sont quasi inexistants. Par ailleurs, ceux qui sont utilisés dans la pratique sont trop naïfs par rapport à la réalité, dans le sens où ils ne prennent pas vraiment en compte le comportement de TCP (Transport Control Protocol) : le protocole réseau qui assure les transferts. Il s'ensuit que ces modèles restent pour le moins très approximatifs, voire irréalistes. Pour ce qui est de la littérature réseau, il existe quelques travaux de modélisation du protocole de communication réseau TCP. Ces derniers sont intéressants, dans la mesure où ils mettent en évidence les paramètres qui influencent le comportement de ce protocole. Cependant, certaines de ces informations, telles que les taux de pertes, sont liées à l'état du réseau et ne sont pas accessibles au niveau applicatif. Or, ce qui serait intéressant pour les personnes travaillant sur des grilles, serait de pouvoir disposer d'un modèle de coûts des communications TCP suffisamment fiable et exploitable facilement au niveau applicatif. Un tel modèle permettrait de fournir une meilleure solution au problème de l'équilibrage de charge et profiterait à d'autres problèmes. L'ordonnancement des communications, qui préoccupe une partie de la communauté scientifique, en est un exemple [12]. En somme, on pourrait optimiser et prévoir le temps de fin de ce type d'applications.

Dans le cadre du stage de Master 2 d'Informatique Fondamentale et Appliquée, nous nous sommes intéressés à l'étude du comportement des communications TCP afin de voir s'il serait possible d'apporter aux modèles existants des simplifications qui conduiraient à un modèle de coût dont les paramètres sont plus accessibles au niveau applicatif. L'étude a été menée sur la plateforme Grid'5000. Il s'agit d'une grille de calcul d'envergure nationale, dédiée aux expérimentations et qui présente par ailleurs des caractéristiques sur le plan des machines et du réseau d'interconnexions très favorables aussi bien en termes de stabilité que de performances.

Le présent rapport s'organise comme suit : après une première partie consacrée au protocole TCP et à la présentation des principaux modèles existants pour effectuer des prédictions, suit une seconde partie dans laquelle nous présentons l'environnement Grid'5000 ainsi que différents choix effectués quant à l'implémentation TCP. Enfin, dans une troisième et dernière partie nous rendrons compte de diverses expériences effectuées dans notre recherche d'un modèle ainsi que des différents résultats obtenus.

Chapitre 1

Le protocole TCP ou le besoin de transmission fiable et efficace

TCP est un protocole de communication réseau qui se situe au niveau de la couche transport (couche 4 du modèle OSI[1]) et dont l'objectif principal est de fournir à la couche application (couche 7 du modèle OSI) un moyen fiable de transmission des données. En effet, au niveau le plus bas, les données ne sont pas à l'abri d'erreurs et de corruption liées aux supports de transmission, aux équipements réseau ou encore à la charge réseau. De plus, comme le choix des routes (i.e. routage) s'effectue dynamiquement, des données issues d'un même émetteur et destinées à un hôte commun n'empruntent pas toujours le même chemin pour l'atteindre. Il s'ensuit que les délais d'acheminement peuvent varier de manière importante et donner lieu à des déséquencements (i.e. les données arrivent dans le désordre), voire des duplications (i.e. les données arrivent plusieurs fois). De plus, la transmission de données nécessite, suivant les équipements utilisés, de prendre en considération et donc de gérer un certain nombre de paramètres supplémentaires tels que la taille optimale des données à envoyer.

En réponse à ces difficultés, TCP offre une interface commune à la couche application permettant de faire abstraction de la plupart des considérations liées à la transmission fiable et optimale de données. Par ailleurs, en plus du contrôle d'erreur, TCP assure également un contrôle de flux et de congestion : selon la capacité du récepteur à recevoir et consommer les données qui lui sont envoyées ainsi que la charge du réseau, TCP adaptera la vitesse de transmission.

Depuis son introduction en 1981 dans par J.Postel [19], TCP a rapidement évolué jusqu'à une forme très similaire à celle qu'on lui connaît maintenant et que l'on va essayer de présenter. Après quoi, nous exposerons différentes évolutions que TCP a connues avant de finalement présenter les modèles de coûts.

Mais avant tout et dans le but de faciliter l'exposé, nous allons fournir quelques explications concernant certains termes que l'on a déjà utilisés ou que l'on utilisera par la suite.

1.1 Rappels et terminologie

Dans le présent rapport, nous allons nous intéresser à des communications réseau *point-à-point*. Ces dernières s'effectuent entre un *émetteur* et un *récepteur* qui désignent respectivement l'*entité* qui envoie des données et celle qui les reçoit. Le terme entité, selon le contexte

[1] Open Systems Interconnection

d'écriture, pourra désigner la machine ou le logiciel accomplissant la tâche d'envoi et/ou de réception.

Deux entités qui sont amenées à communiquer doivent pouvoir se comprendre : il s'agit du *protocole*. Dans notre cas particulier, le protocole utilisé est TCP que nous avons déjà situé précédemment dans le modèle OSI.

Lorsque des données sont transmises au protocole TCP depuis une application, elles sont recopiées, puis maintenues dans un zone mémoire appelée *tampon d'émission*[2] (ou *send buffer*) tant qu'elles n'ont pas été transmises avec succès au destinataire. Avant de les envoyer, TCP les *encapsule*. Cette opération consiste à leur rajouter une entête contenant différentes informations liées au protocole avant de les transmettre au protocole de la couche inférieure qui en fera de même.

Ainsi, le niveau d'encapsulation des données diffère selon la position d'un protocole dans la hiérarchie du modèle OSI. À chaque niveau est associée une terminologie précise. En toute rigueur, il convient de parler de *segment* TCP, de *paquet* IP et de *trame* réseau.

La taille maximale des données qui peut être prélevée du tampon démission, encapsulée, puis envoyée en une seule fois dépend des équipements réseau. Le MTU[3] correspond à la plus grande taille de paquet (en octets) que peut transmettre un réseau. Il peut varier au sein d'un même réseau selon le chemin emprunté à cause de l'hétérogénéité des équipements réseau. La détermination du MTU se fait dynamiquement grâce à une technique connue sous le nom de *Path MTU Discovery* [17].

Le MSS[4] désigne la taille maximale autorisée pour un segment. Elle est déduite à partir du MTU en soustrayant la taille des différentes entêtes intermédiaires.

Le temps mis pour effectuer l'aller-retour entre un émetteur et un récepteur s'appelle le RTT[5]. Il correspond à deux fois le délai (ou latence). Le RTT dépend de :

- la rapidité des systèmes émetteurs/récepteurs à respectivement encapsuler/décapsuler les données,
- la vitesse de transmission des interfaces et du support de transmission réseau (Ethernet 10/100/1000, fibre optique ...),
- la capacité des équipements intermédiaires (routeur/switch) à transmettre les paquets.

Finalement, le *débit* correspond à une vitesse moyenne d'envoi des données d'un point à un autre et la *bande passante* désigne une vitesse observable à ne pas confondre avec la vitesse maximum théorique.

1.2 Transmission fiable

Pour assurer la transmission fiable de données, TCP utilise une technique d'*acquittements positifs avec retransmission* qui consiste dans sa forme la plus simple à envoyer un paquet, puis à attendre de la part du récepteur un acquittement positif qui confirme la bonne réception avant d'émettre le paquet suivant (cf. Fig. 1.1, p. 11).

La détection d'une perte[6] d'un paquet de données repose sur l'utilisation d'un temporisa-

[2] On parlera de *tampons en réception* (ou *receive buffer*) pour désigner la zone mémoire dans laquelle TCP stocke les données qu'il reçoit en attendant que l'application les lui réclame.

[3] Maximum Transmission Unit

[4] Maximum Segment Size

[5] Round Trip Time

[6] La corruption d'un paquet est également considérée comme une perte et est ainsi détectée par le même mécanisme.

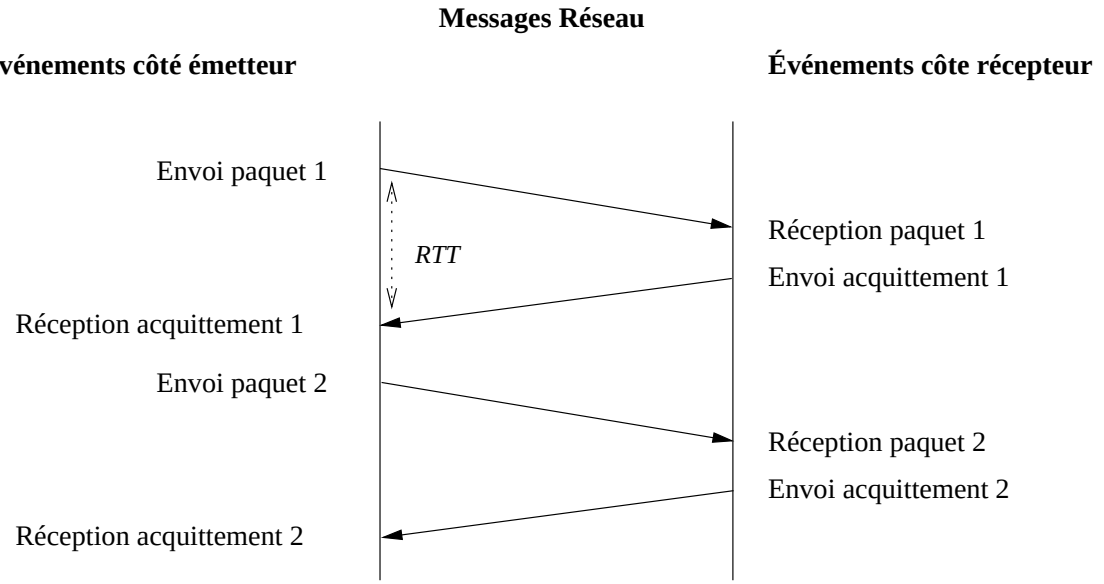

FIG. 1.1 – Acquittements positifs : Scénario idéal sans pertes. Schéma d'après [2].

teur[7] qui est armé après l'envoi d'un paquet. L'expiration du temporisateur avant la réception de l'acquittement est interprétée comme la perte du paquet et entraîne sa retransmission (cf. FIG. 1.2, p. 12).

Telle quelle, cette manière de procéder est loin d'être fiable, étant donné qu'un retard du paquet et/ou de son acquittement (voire simplement une perte de l'acquittement) peuvent mener à une expiration du temporisateur sans pour autant qu'il y ait de perte. Ceci peut déboucher sur des situations avec des paquets dupliqués sur le réseau que TCP résout en utilisant des numéros de séquence. TCP attribue ces derniers à chaque octet de données envoyé et exige du récepteur qu'il mémorise d'une part les séquences déjà reçues et d'autre part qu'il mette dans un acquittement le numéro de séquence de l'octet reçu. Ceci établit une association unique entre les octets envoyés et leur acquittement et permet d'éviter toute confusion liée à des retards ou des duplications[8].

En observant la figure FIG. 1.1 (p. 11), on peut se rendre compte que l'utilisation seule de la technique d'acquittements positifs avec retransmission n'est pas très efficace, étant donné qu'on est limité à envoyer un unique paquet à la fois. Le gâchis de bande passante est d'autant plus vrai que le temps mis pour recevoir l'acquittement (environ un RTT) est extrêmement long comparé au temps mis pour envoyer un paquet. En effet, le premier est de l'ordre de la milliseconde tandis que le second est de l'ordre de la microseconde. La solution à ce problème consiste à utiliser des *fenêtres glissantes*.

[7] timer

[8] En réalité, TCP envoie dans l'acquittement le numéro de séquence du prochain octet attendu. Comme nous le verrons par la suite, cet acquittement acquitte également tous les octets de numéro de séquence inférieur. (cf. nature cumulative des acquittements TCP, 1.2.1, p. 12)

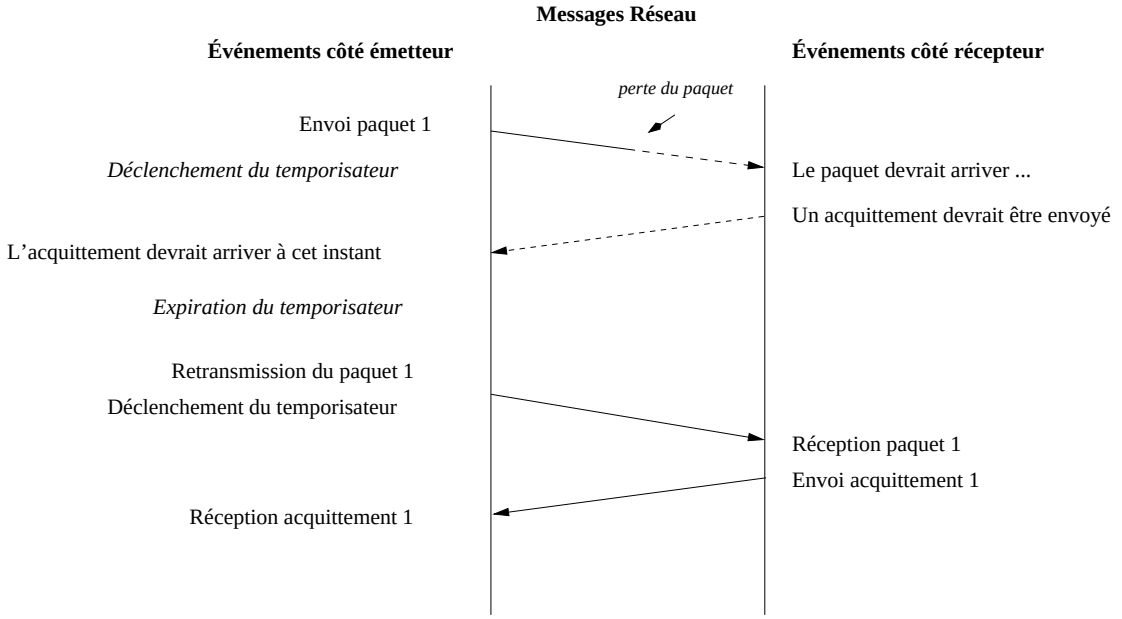

FIG. 1.2 – Acquittements positifs : scénario avec perte et retransmission. Schéma d'après [2]

1.2.1 Fenêtres glissantes

En pratique, la technique d'acquittements positifs avec retransmission s'utilise conjointement avec le protocole dit des fenêtres glissantes[9]. Ce protocole consiste à placer au-dessus de la suite de données à expédier une *fenêtre* de taille fixe et petite (cf. FIG. 1.3, p. 13). La fenêtre permet de délimiter le tampon d'émission en trois zones. Les données contenues dans la fenêtre sont des données qui ont été envoyées mais qui n'ont pas encore été acquittées. À gauche de la borne gauche de la fenêtre se trouvent les données acquittées et à droite de la borne droite les données prêtes à être envoyées. L'acquittement de la première série de données de la fenêtre, entraîne le déplacement de la fenêtre vers la droite (cf. FIG. 1.3, p. 13).

Ainsi, il est possible d'envoyer plusieurs paquets avant de se mettre en attente d'un acquittement, ce qui permet de mieux exploiter la bande passante disponible. La performance de la fenêtre est fonction de sa taille. En effet, en choisissant une taille de fenêtre suffisamment grande, le temps d'inactivité peut être réduit considérablement.

L'utilisation de cette technique offre plusieurs approches quant au choix du type d'acquittement et à leur fréquence d'envoi. TCP utilise des *acquittements cumulatifs* : la réception de l'acquittement renseigne l'émetteur sur la séquence du prochain octet attendu par le récepteur et acquitte tous les octets de numéros de séquence inférieurs non encore acquittés. Cette pratique permet de rendre le protocole plus robuste à d'éventuelles pertes d'acquittements, mais également de réduire la fréquence de leur envoi afin de ne pas surcharger le réseau.

Sur le protocole des fenêtres glissantes reposent deux mécanismes de TCP que nous allons présenter ci-après.

Le premier mécanisme de TCP à avoir été implémenté en utilisant le concept de fenêtres

[9] sliding windows protocol

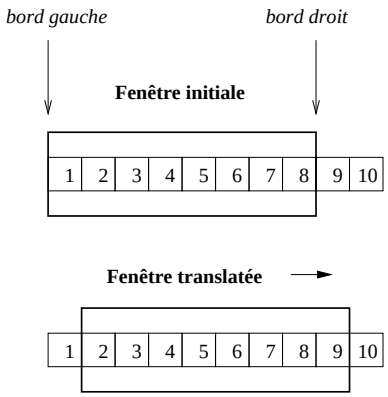

FIG. 1.3 – Visualisation du mécanisme de fenêtre glissante. Schéma d'après [2]

glissantes est le *contrôle de flux*.

1.2.2 Contrôle de flux

Le *contrôle de flux* désigne le mécanisme assurant que l'émetteur n'envoie pas plus de données que le récepteur ne peut en stocker dans sa mémoire tampon[10]. Ceci permet d'éviter la situation où le récepteur se voit forcer de jeter des paquets qui arrivent parce qu'il n'a pas assez de place dans son tampon pour les accueillir. Ces situations étaient particulièrement fréquentes dans les années 1980. À l'époque, le tampon du récepteur constituait le facteur limitant lors d'un transfert et ne devait en aucun cas être dépassé. La solution adoptée repose sur l'utilisation de fenêtres glissantes. Le récepteur et l'émetteur maintiennent chacun une fenêtre. Celle du récepteur délimite la zone du tampon de réception pouvant accueillir des paquets. Celle de l'émetteur contient l'ensemble des données envoyées qui attendent d'être acquittées. Le contrôle de flux consiste alors à tenir au courant l'émetteur de la quantité de place libre sur le tampon du récepteur afin qu'il puisse faire varier en conséquence la taille de sa fenêtre : soit l'agrandir en cas d'augmentation de la place libre sur le récepteur, soit la réduire dans le cas contraire. La quantité de place disponible est transmise à l'émetteur dans chaque acquittement. Elle est connue sous le nom de *fenêtre d'annonce* (ou advertisement window que l'on note *aw*).

Le deuxième mécanisme est le *contrôle de congestion*.

1.2.3 Contrôle de congestion

Le *contrôle de congestion* désigne le mécanisme permettant de réguler le trafic afin d'éviter de surcharger le réseau au point qu'il s'effondre.

Contrairement au contrôle de flux, le contrôle de congestion ne faisait pas partie de la spécification initiale du protocole TCP. Il fut rajouté vers la fin des années 1980 lorsque le trafic, avec la croissance rapide de l'ARPANET (futur Internet), avait commencé à excéder la capacité des réseaux. Les routeurs de ces derniers n'arrivaient pas à transmettre les paquets au rythme où ils arrivaient, entraînant ainsi de nombreux effrondrements du réseau (congestion

[10] buffer

collapses) en octobre 1986 [11]. Ces événements furent à l'origine des travaux de Jacobson, lesquels aboutirent en 1988 à la proposition de deux algorithmes. Ces derniers délèguent à TCP une tâche supplémentaire : celle de réguler le débit des paquets en fonction de l'état du réseau. Ces algorithmes connus sous le nom de *slow-start* et *congestion avoidance* reposent sur l'utilisation d'une fenêtre supplémentaire par l'émetteur appelée *fenêtre de congestion* (ou congestion window que l'on note *cwnd*). L'approche consiste à augmenter additivement la taille de cette fenêtre tant que le réseau et les possibilités des machines émettrice et réceptrice le permettent[11], puis à la réduire multiplicativement en cas de signes de congestion : il est question d'une approche AIMD[12].

La phase additive contient deux sous-phases : celle du démarrage lent et celle d'évitement de congestion correspondant respectivement aux algorithmes slow-start et congestion avoidance (vf. FIG. 1.4, p. 14).

Pendant la phase du démarrage lent, l'augmentation de la fenêtre de congestion suit une croissance exponentielle : pour chaque acquittement reçu pendant cette phase, la taille de la fenêtre de congestion est augmentée de la taille d'un segment. En l'occurrence, si la taille initiale de la fenêtre est d'un segment, elle prendra la suite de valeurs suivante : 2, 4, 8, 16, 32 ... au bout de chaque RTT. Cette phase se poursuit jusqu'à atteindre une valeur seuil bien choisie et fixée par une variable communément appelée *ssthresh* [1, 24]. Suit alors la phase d'évitement de congestion qui est régie par le deuxième algorithme. Pendant cette phase, la croissance de la fenêtre de congestion est linéaire : elle augmente à raison d'un segment pour une fenêtre acquittée (i.e. un segment par RTT).

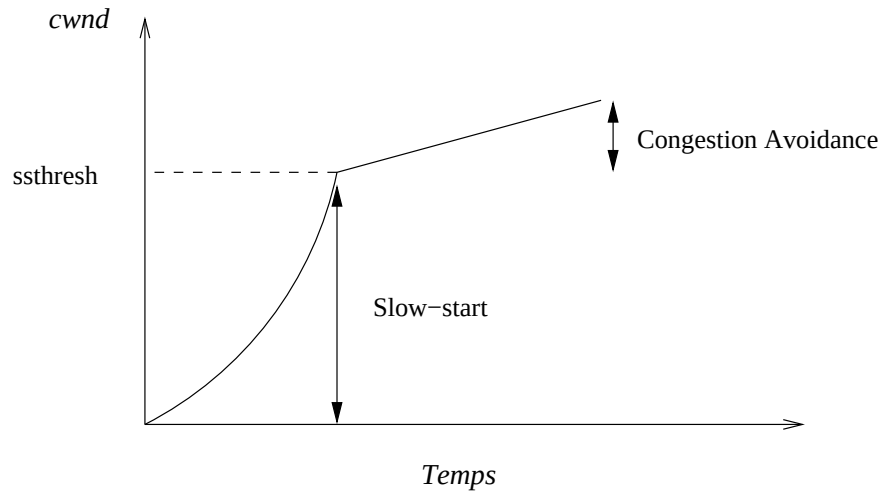

FIG. 1.4 – Algorithmes slow-start et congestion avoidance.

Cette croissance est interrompue lorsqu'un temporisateur associé à un paquet expire ou lorsque des signes de congestion apparaissent. Dans les deux cas, la fenêtre de congestion est réduite au minimum permis (en général un à deux segments) et la transmission reprend sur un slow-start.

[11] Clairement la taille maximum des tampons et celle de la fenêtre d'annonce *aw*.
[12] Additive Increase Multiplicative Decrease

Le schéma décrit précédemment et dans lequel interviennent tour à tour les algorithmes de Jacobson, peut se répéter plusieurs fois au cours d'une connexion (cf. FIG. 1.5, p. 15).

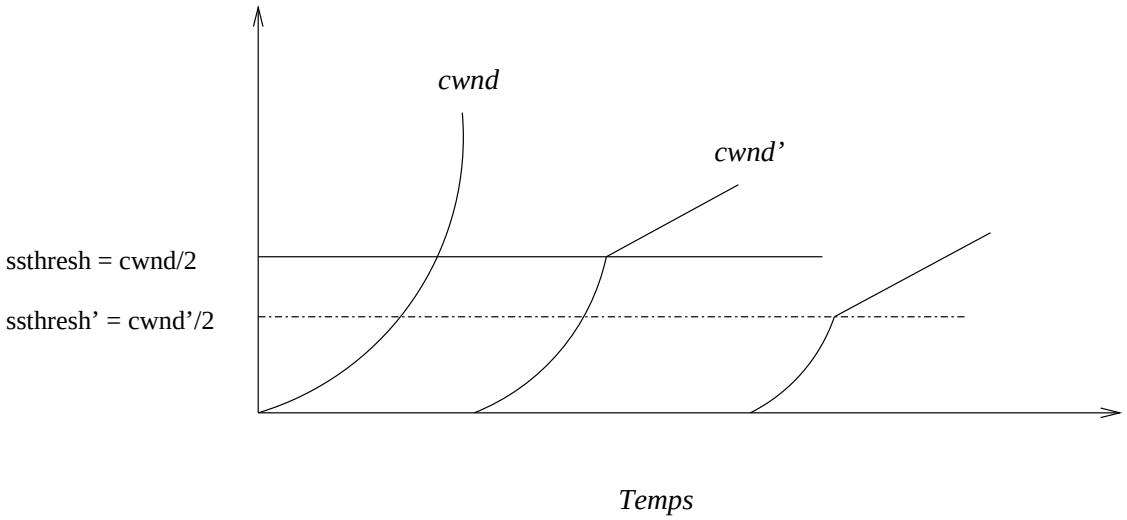

FIG. 1.5 – Algorithmes slow-start et congestion avoidance au cours d'une connexion.

Dans cette figure, on peut remarquer que le schéma n'est effectif qu'après une phase de slow-start interrompue par une expiration de temporisateur. Cette pratique est couramment utilisée car elle permet d'initialiser la variable *ssthresh* avec une valeur assez réaliste.

Avec l'introduction du contrôle de congestion, TCP a connu un certain nombre de changements, en particulier l'introduction d'une nouvelle fenêtre, la fenêtre de congestion qui est dorénavant prise en compte pour déterminer la taille de la fenêtre d'émission notée *swnd*. Cette dernière est donnée par :

$$swnd = min(cwnd, aw, snd\_buff) \tag{1.1}$$

où snd_buff désigne la taille du tampon d'émission.

En résumé, TCP est un protocole de transmission qui assure fiabilité, contrôle d'erreur et contrôle de congestion. Son fonctionnement repose sur le mécanisme d'acquittements positifs et cumulatifs ainsi que sur le concept de fenêtre glissante.

1.3 Améliorations

Au fil des années, TCP a connu un certain nombre de changements visant à l'améliorer et à l'adapter aux diverses évolutions des réseaux. Ces derniers ont eu tendance à devenir de plus en plus rapides, à augmenter en capacité[13] ce qui a introduit de nouvelles difficultés (problématiques). Ces changements portaient sur plusieurs mécanismes de TCP et plus particulièrement sur l'algorithme de congestion.

[13] et à se diversifier avec l'apparition des réseaux mobiles et connexions satellites

1.3.1 Fast Retransmit et Fast Recovery

Il s'agit de deux modifications proposées sur l'algorithme de congestion dans les années 1990 [9, 24].

TCP peut émettre des acquittements pour des paquets déséquencés mais ces derniers contiennent toujours le numéro du plus ancien paquet non encore acquitté. Il en résulte que TCP peut recevoir plusieurs fois le même acquittement. Cette réception est une indication ambiguë étant donné qu'il peut s'agir d'une perte ou simplement d'un déséquencement. *Fast retransmit* propose de lever l'ambiguité en utilisant l'heuristique du *triple ACK*. Cette dernière contraint TCP à attendre trois acquittements dupliqués avant de considérer qu'un segment est perdu. Le segment qui semble perdu est alors réémis sans attendre l'expiration du temporisateur. Les premières implémentations de TCP incluant cet algorithme étaient nommées *TCP Tahoe*.

Après *Fast retransmit*, TCP passe dans une phase transitoire dans laquelle il reste jusqu'à qu'une expiration de temporisateur survienne ou que le segment censé perdu soit acquitté. Durant cette phase, TCP continue à envoyer un nouveau segment pour chaque acquittement dupliqué reçu. Ceci est réalisé dans le but de maintenir le flot de données sur le réseau. Effectivement, tant que des acquittements sont reçus, cela signifie qu'un paquet vient de quitter le réseau. Dans le cas où le segment perdu est acquitté, la fenêtre est ramenée à la valeur de *ssthresh*, puis continue en phase d'évitement de congestion plutôt qu'en slow-start : il s'agit de *Fast Recovery*. *TCP Reno* fut le nom utilisé pour désigner les implémentations intégrant ces deux algorithmes.

Une autre amélioration bien connue qui fut implémentée et donna *TCP NewReno* est *NewReno*. Elle permet d'augmenter la robustesse de l'algorithme de congestion en cas de pertes multiples au sein d'une même fenêtre TCP [4, 22]. Elle s'utilise lorsque l'option des acquittements sélectifs n'est pas employée.

1.3.2 Les acquittements sélectifs : *SACK*

Avec les acquittements sélectifs [14], le récepteur utilise une partie du segment TCP contenant l'acquittement pour indiquer à l'émetteur quels segments sont arrivés à bon port. Ainsi, pendant la phase de *Fast Recovery*, l'émetteur peut déduire exactement quels segments n'ont pas atteint leur destination et les retransmettre. Ceci est particulièrement utile pour des pertes multiples.

D'autres améliorations telles que *FACK*[14] [15] et les acquittements sélectifs dupliqués[15] [5] sont venues compléter le mécanisme initial.

1.3.3 Quelques améliorations supplémentaires

Parmi les nombreuses autres évolutions, nous avons retenu celles introduites dans [17], [10] et [21].

Le *path MTU discovery* [17] consiste à découvrir dynamiquement le plus grand MTU sur un chemin donné afin d'optimiser les transferts. Effectivement, plus les paquets IP sont grands, meilleur sera le débit. Cependant, comme tous les équipements intermédiaires ne supportent pas forcément le même MTU, il est important de bien déterminer la taille maximale admissible,

[14] Forward ACKnowledgements
[15] Duplicate SACK

sinon les routeurs se voient contraints de fragmenter les paquets. Cette opération est coûteuse en temps et ralentit l'acheminement des paquets[16].

Dans [10] sont introduits trois mécanismes adaptant TCP aux réseaux haut-débit que l'on rencontre de plus en plus souvent. Le premier, *WSCALE*, permet de contourner la limitation de 64 Ko imposée à la taille des fenêtres TCP. Le deuxième, *RTTM*, améliore le calcul du RTT en utilisant le mécanisme des *timestamps* (ou estampillage). Finalement, le troisième *PAWS* se fonde également sur les *timestamps* et protège contre le bouclage des numéros de séquences. En effet, dans un réseau très rapide le risque d'avoir deux numéros de séquences identiques (pour une même connexion) est considérablement accru.

Dans [21] est introduit le mécanisme de notification explicite de congestion *ECN*[17]. Ce dernier permet à TCP d'obtenir directement de la part des équipements réseau intermédiaires un retour sur l'état de congestion.

1.4 Modèles prédictifs de coûts

Au-delà du modèle trivial qui consiste simplement à déduire le temps de transfert d'un volume de données à partir de la bande passante théorique ou disponible d'une connexion, il existe dans la littérature réseau principalement deux types de modèles prédictifs [7, 8] :

- modèles statistiques basés sur des historiques,
- modèles analytiques.

Le premier type de modèle nécessite beaucoup de prises de mesures sur les connexions, ce qui les rend très particuliers et donc difficilement exportables sur une autre connexion.

Parmi les modèles de coût basés sur une formule, nous allons présenter ci-après les deux plus classiques.

1.4.1 Le modèle de Mathis

Le plus ancien [16], qui date de 1997 et qui s'intitule "The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm", repose sur la formule suivante :

$$BW = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{1.2}$$

selon laquelle la bande passante BW s'approxime par :

- MSS : la taille maximale des segments.
- RTT[18] : le temps moyen du trajet aller-retour entre l'émetteur et le destinataire.
- p : la probabilité de perte d'un paquet.
- C : une constante de proportionnalité qui regroupe un certain nombre de paramètres constants pour une combinaison donnée d'implémentation TCP, de stratégie d'acquittement[19] et de mécanisme de perte.

Par exemple, C est estimée à $\sqrt{\frac{3}{2}}$ dans le cas de pertes périodiques et de l'utilisation d'une stratégie d'acquittement consistant à acquitter tous les paquets. Pour une stratégie avec des

[16] Avec le passage à IPv6, les routeurs ne se chargent plus de fragmenter les trop grands paquets. Ils notifient simplement l'expéditeur de le faire.

[17] Explicit Congestion Notification

[18] Round Trip Time

[19] Il s'agit de la fréquence à laquelle sont renvoyés les acquittements. Généralement, la plupart des implémentations envoient un acquittement tous les deux segments reçus

acquittements retardés, la valeur de C est inférieure à 1, ce qui simplifie la formule et permet d'avoir la majoration suivante :

$$BW < (\frac{MSS}{RTT})\frac{1}{\sqrt{p}} \tag{1.3}$$

L'applicabilité de ce modèle est soumise à un certain nombre de conditions. En particulier,

1. il ne concerne que les transferts de données assez longs (i.e. l'émetteur a toujours des données à envoyer) afin que TCP atteigne l'équilibre,
2. le réseau devra être soumis à des pertes de paquets modérées et périodiques (les timeout de retransmissions devront être évités),
3. la fenêtre du récepteur devra être suffisamment grande afin que le débit soit limité par le réseau,
4. la présence du support des acquittements sélectifs en cas de pertes multiples,
5. les implémentations TCP utilisant des stratégies d'ouverture de fenêtre différentes de *TCP Reno* ne suivent pas le modèle. En l'occurrence, *TCP Vegas* qui se fonde sur la variation du RTT plutôt que sur les *timeout* pour détecter les pertes, n'adhère pas au modèle.

Dans des applications réelles, la troisième contrainte n'est pas admissible car on est souvent amené à ouvrir plusieurs connexions ce qui peut conduire à des situations de famine mémoire (même pour de machines bien pourvues) et à une détérioration des performances.

1.4.2 Le modèle de Padhye

L'article de Padhye [18] paru en 2000 et qui s'intitule "Modeling TCP Reno Performance : A Simple Model and Its Empirical Validation" présente un modèle approximant le débit de TCP en émission par la formule :

$$BW = min(\frac{W_m}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)}) \tag{1.4}$$

Cette formule comprend deux composantes correspondant aux opérandes d'un minimum. La composante de gauche qui dépend uniquement de W_m, la taille maximale d'ouverture de la fenêtre en émission, et du RTT, correspond au cas où la probabilité de perte p est nulle. La composante de droite, un peu plus complexe, correspond au cas où il y a des pertes. Elle dépend d'un plus grand nombre de paramètres dont p, RTT, b le nombre de paquets attendus par le récepteur avant d'envoyer un acquittement et T_0 la durée d'une expiration de temporisateur[20] isolée[21].

Cette approximation ne s'applique, tout comme la formule de Mathis, qu'aux flux TCP longs impliquant donc un large volume de données transmis tels que les transferts ftp.

[20] timeout

[21] En cas de pertes multiples de paquets, plusieurs temporisateurs peuvent expirer.

Les deux modèles brièvement exposés précédemment sont très connus par la communauté scientifique réseau, d'une part car ce sont les seuls de leur catégorie qui existent, et d'autre part à cause de leur performance correcte. Néanmoins, ils possèdent l'inconvénient majeur de dépendre de paramètres, notamment la probabilité de perte p, T_0 et le RTT qui sont difficilement accessibles au niveau applicatif ou très variables au cours d'une connexion.

Chapitre 2

Environnement d'expérimentation

2.1 Grid'5000

2.1.1 Présentation rapide

À l'origine, Grid'5000 est le nom d'un programme français initié par l'ACI Grid. Ce programme, d'envergure nationale, vise à promouvoir la mise en place d'une grille expérimentale de recherche constituée d'une dizaine de centres équipés de fermes d'ordinateurs personnels. L'objectif d'une telle plateforme est de fournir aux scientifiques les moyens expérimentaux pour mener à bien des recherches dans le domaine des grilles.

Actuellement, Grid'5000 ne compte que 2000 processeurs sur les 5000 prévus. Ils sont répartis de manière relativement équilibrée sur un ensemble de neuf sites présents dans toute la France : Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis et Toulouse.

Les machines hébergées par ces sites sont des machines modernes équipées de biprocesseurs d'AMD (Opterons), de processeurs Intel (Itanium/Xeon) ou de processeurs IBM (PowerPc) cadencés à des fréquences d'au moins 2 GigaHertz. Elles disposent également d'une quantité de mémoire vive de l'ordre du gigaoctet et de disques durs de capacité supérieure à 70 gigaoctets (IDE/SATA/SCSI). Côté connectivité, elles sont dotées d'interfaces réseau Gigabits Ethernet en standard ce qui leur permet d'avoir une connectivité intrasite de 1 Gigabits.

Initialement, l'interconnexion des sites reposait sur une solution *Ethernet sur MPLS*[1] au-dessus de l'épine dorsale[2] de RENATER[3] et utilisait les tunnels MPLS (LSP) afin d'établir un maillage complet[4] entre les POP [5] de RENATER sur lequels sont reliés les sites. Ceci permettait aux sites d'être interconnectés à travers des VLAN [6] d'une capacité de 1 Gigabits.

Avec le passage à la version 4 de Renater (RENATER-v4) et l'exploitation des fibres noires, l'interconnexion des sites a été revue à la hausse et devrait atteindre 10 Gigabits. Pour l'instant, les sites de Grenoble, Lille, Nancy, Rennes, Sophia, et Toulouse sont les premiers à avoir effectué la migration vers la nouvelle infrastructure. Parmi ces sites, seuls ceux de Nancy, Rennes et Sophia disposent des équipements nécessaires pour communiquer en 10 Gigabits. Les

[1] Multi Protocol Label Switch

[2] backbone

[3] Réseau National de télécommunications pour la Technologie, l'Enseignement et la Recherche, `http://www.renater.fr`

[4] full mesh topology

[5] les points de présence

[6] réseaux virtuels Virtual LAN

autres restent à 1 Gigabits. Quant aux sites de Bordeaux, Lyon et Orsay, ils restent connectés via l'ancienne infrastructure en 1 Gigabits.

2.1.2 Utilisation de Grid'5000

La plateforme Grid'5000 est un environnement fermé auquel on peut accéder via les frontaux présents sur chacun des sites. Une fois connecté, l'utilisateur peut commencer à exploiter la plateforme. Habituellement, la démarche qu'il suivra avant de débuter ses expérimentations sera de :

- *créer* un ou plusieurs environnements[7] et
- *déployer* un ou plusieurs environnements.

Phase de création d'environnement

Le terme environnement désigne dans ce contexte un système d'exploitation que configure un utilisateur en vue de mener ses expériences. Il se présente sous la forme d'une archive[8] (*tar* par exemple). Selon ses besoins, l'utilisateur sera amené à en créer plusieurs. Ce sera le cas s'il projette d'utiliser plusieurs systèmes d'exploitation ou des machines d'architectures différentes (Intel/AMD).

Afin de faire gagner du temps à l'utilisateur, un certain nombre d'environnements minimaux sont fournis. Ces derniers comportent tous les réglages nécessaires pour les rendre fonctionnels sous Grid'5000. La plupart sont basés sur des noyaux Linux[9] mais il en existe aussi pour FreeBSD et Solaris.

Finalement, la procédure de création consiste à :

- *réserver* une machine cible contenant plusieurs partitions et *déployer* sur l'une de ses partitions un de ces environnements minimaux, c'est-à-dire décompresser l'archive correspondante dans une partition de la machine cible et rebooter la machine sur le système contenu dans l'archive,
- le *modifier* selon ses besoins (installer logiciels, ...) et
- le *sauvegarder* sous la forme d'une nouvelle archive.

Phase de déploiement

Cette phase se déroule en deux étapes :

- la *réservation* des machines cibles et
- le *déploiement* de l'image (ou des images) sur les machines réservées,

qui sont assurées respectivement par deux suites logicielles : *OAR* et *Kadeploy*.

Utilisation courante

Habituellement, un utilisateur de Grid'5000 est amené à effectuer de nombreuses réservations et de nombreux déploiements. À la longue, ces opérations deviennent extrêmement fastidieuses pour plusieurs raisons :

- Les réservations et les déploiements peuvent concerner des centaines de machines.
- La gestion d'autant de machines n'est pasà l'abri de problèmes matériels.

[7] Si ce n'est pas déjà fait.
[8] Appelée également image.
[9] Debian, Fedora, Ubuntu pour en citer quelques-uns.

– L'homogénéité entre sites n'est pas parfaite. Par exemple, le nom et le numéro de la partition de déploiement ne sont pas identiques.

À cet effet, un ensemble de scripts automatisant ces tâches ont été écrits.

Cette plateforme est extrêmement intéressante car elle ouvre les portes sur la réalisation d'expériences à grande échelle dans un environnement très contrôlé et hautement configurable. Globalement, son fontionnement est très satisfaisant et prometteur. Néanmoins, elle connaît parfois des perturbations liées à de mauvais réglages ou à des mises à jour.

2.2 Choix de l'implémentation et de l'algorithme de contrôle de congestion TCP

Après nous être documentés sur le protocole TCP et sur Grid'5000, nous avons étudié les implémentations TCP de différents systèmes d'exploitation afin de voir quel en était l'état. Ce qui nous intéressait, notamment, c'était de déterminer quels types d'algorithmes de congestion sont utilisés, quelles fonctionnalités sont implémentées, quelles sont celles qui sont activées par défaut, comment et par qui elles peuvent être modifiées. Cette étude semblait requise pour plusieurs raisons :

– d'une part, à cause de notre objectif. En effet, nous souhaitons obtenir un modèle assez général qui puisse être utilisé par une majorité de systèmes. Ceci est d'autant plus important que l'utilisation d'une option se décide lors de l'établissement d'une connexion TCP sur un commun accord des deux extrémités et dépend naturellement de sa présence et de son état d'activation.
– d'autre part, parce que les implémentations TCP ne constituent qu'une solution possible qui est censée respecter un certain nombre de principes énoncés dans les RFC.
– enfin, car on devait faire un choix parmi la pléthore de systèmes d'exploitation déployables sur Grid'5000.

Pour pouvoir mener à bien notre tâche, nous avons été amenés à étudier les implémentations TCP des systèmes d'exploitation FreeBSD, AIX 5.2, IRIX 64 ainsi que de systèmes basés sur les noyaux Linux 2.4 et 2.6[10].

L'étude s'est concentrée sur deux aspects : premièrement la gestion des tampons utilisés par TCP et deuxièmement les fontionnalités implémentées. Les sources d'informations consultées sont variées et vont des pages de manuel TCP (AIX, IRIX) jusqu'aux codes sources et documentations des noyaux en passant par les *Changelogs* et *Errata*, pour les systèmes Open Source (Linux, FreeBSD).

2.2.1 Gestion des tampons

La taille des tampons constitue une des principales sources de réglage permettant d'affecter les performances de TCP en termes de débit [20]. En émission, la taille du tampon détermine la quantité potentielle d'octets qui pourront être transmis en une seule fois avant que l'émetteur ne se mette en attente d'un acquittement. Habituellement, les spécialistes souhaiteraient attribuer aux tampons en émission une taille égale à :

[10] Linux n'est pas un système d'exploitation.

$$\text{débit} * RTT \tag{2.1}$$

Ce produit est communément appelé BDP[11]. Il peut atteindre plusieurs dizaines de méga-octets sur des réseaux très rapides.

Les tampons en réception devront être dans la mesure du possible suffisamment grands pour ne pas limiter l'émetteur en exerçant du contrôle de flux.

En général, tous ces systèmes d'exploitation possèdent trois variables présentes dans le système de fichiers système qui sont modifiables par le superutilisateur via une commande de type *sysctl*. Il existe une variable pour la taille par défaut des tampons en émission, une pour la taille des tampons en réception et une pour la taille maximale autorisée. Cette dernière est utilisée par le système pour limiter la taille maximale permise de tampons que peut demander une application via la primitive *setsockopt()* de l'interface des *socket*. Font exception à la règle IRIX 64 et les noyaux Linux. IRIX 64, bien qu'ayant une valeur maximale définie, ne permet pas de la modifier. Heureusement, elle est suffisamment grande (cf. Tab. 2.1). Quant aux noyaux Linux, ils intègrent en plus deux mécanismes de réglage automatique (autotuning) : un pour chaque type de tampon. Ces mécanismes sont désactivés lorsque l'application impose la taille de ses tampons avec la primitive *setsockopt()*.

Dans le tableau Tab. 2.1 (p. 24), nous avons reporté les différentes valeurs que nous avons pu observer sur les différentes architectures utilisées.

OS/Kernel	**Valeur max.**	**Émission**	**Réception**
Linux 2.4.x	?	16384	87380
Linux 2.6.x	262144	16384	87380
FreeBSD < 4.5	262144	16384	16384
FreeBSD 4.5	262144	32768	65536
FreeBSD > 4.5	262144	32768	57334
AIX	1048576	16384	16384
IRIX	134217728	61440	61440

Tab. 2.1 – Valeurs par défaut de la taille des tampons d'émission et de réception ainsi que leur taille maximale

On s'aperçoit que les valeurs par défaut sont assez conservatives et faibles, particulièrement pour des réseaux possédant un BDP élevé[12]. Une intervention de l'administrateur système sera bien souvent nécessaire pour pouvoir utiliser des tampons plus grands dans les applications et bénéficier de débits convenables relativement à la connexion utilisée.

[11] Bandwidth Delay Product

[12] Les réseaux BDP sont caractérisés soit par une grande latence (réseaux satellites), soit par un grand débit (Grid'5000)

2.2.2 Algorithmes de congestion et options TCP

Dans le tableau TAB. 2.2 (p. 25) sont reportées les différentes options et leur état d'activation par défaut selon le système utilisé. Les notations utilisées sont :

- on, off : indiquent si l'option est activée ou non par défaut.
- ? : l'option existe mais on ne connaît pas sa valeur par défaut.
- - : l'option n'a pas été trouvée.

OS/Kernel	Alg. de Cong.	RFC1323	SACK	ECN	NAGLE	DACK	DSACK	FACK	PMTU_DISC
Linux 2.4.x	NewReno	on	on	off	on	-	on	on	on
Linux 2.6.[0-7]	NewReno	on	on	off	on	-	on	on	on
Linux 2.6.[8-12]	BIC TCP	on	on	off	on	-	on	on	on
Linux 2.6.[13-15]	NewReno	on	on	off	on	-	on	on	on
FreeBSD $\geq$ 4.3	NewReno	?	?	?	on	?	-	-	on
FreeBSD 4.3	NewReno	on	?	?	on	?	-	-	on
FreeBSD 4.5	NewReno	on	-	-	?	?	-	-	on
FreeBSD $\geq$ 4.6	NewReno	on	-	-	on	?	-	-	on
FreeBSD $\geq$ 5.1	NewReno	on	-	?	on	?(on)	-	-	on
FreeBSD $\geq$ 5.3	NewReno	on	on	?	on	?(on)	-	-	on
AIX $\geq$ 4.3.3	NewReno	off	off	off	on	off	-	-	on
IRIX 64	NewReno	on	on	-	on	on	-	-	on

TAB. 2.2 – Disponibilité et valeur par défaut d'options TCP sous plusieurs systèmes d'exploitation analysés.

On s'aperçoit que les noyaux Linux implémentent plus de fontionnalités nouvelles par rapport aux autres, notamment avec la présence d'options telles que *FACK*, *Dup SACK* ou *ECN*. Par ailleurs, il est important de noter que Linux offre la possibilité de choisir entre plusieurs implémentations TCP dans les version récentes des séries 2.4 et 2.6.

De cette étude, il ressort qu'actuellement l'algorithme de congestion le plus répandu reste *NewReno* conjointement utilisé avec les améliorations de [10], le *path MTU discovery* et les acquittements sélectifs. Pour nos expérimentations, notre choix s'est finalement porté sur l'utilisation d'un système d'exploitation utilisant le noyau Linux 2.6.12 qui est actuellement fréquemment utilisé dans plusieurs distributions Linux.

2.3 Conditions d'expérimentation

Avant de nous lancer dans des expérimentations, nous avons effectué un certain nombre de tests en vue de vérifier les qualités de Grid'5000 et celles des outils de mesure.

Pour mener à bien nos expérimentations, nous avons construit une image pour laquelle un certain nombre de paramètres réseau (cf. 2.2) ont été fixés.

2.3.1 Sélection des sites et mise au point d'une image

Grid'5000 présente une grande diversité au niveau des architectures utilisées, mais la majorité des sites possède un cluster composé de machines bi-opterons. Nous avons pu utiliser huit sites : Bordeaux, Lille, Lyon, Nancy, Orsay, Rennes, Sophia et Toulouse.

Ceci nous a conduit à construire notre image à partir de *Debian4All*, une image minimale déjà existante, basée sur une *Debian* avec un noyau adapté à ce type d'architecture[13].

Après avoir testé qu'on pouvait effectivement la déployer, nous l'avons paramétrée en respectant les choix de la section précédente en termes d'options TCP. Le noyau Linux 2.6.12 présent sur l'image n'a pas été recompilé. Les modifications apportées sont décrites ci-après.

Nous avons désactivé tous les algorithmes de congestion alternatifs pour avoir du *TCP NewReno* :
- *net/ipv4/tcp_bic=0*
- *net/ipv4/tcp_vegas_cong_avoid=0*
- *net/ipv4/tcp_westwood=0*

Le mécanisme de tuning automatique en réception a été explicitement désactivé[14]
- *net/ipv4/tcp_moderate_rcvbuf=0*

Les options *ECN*, *FACK* et *DSACK* ont également été désactivées :
- *net/ipv4/tcpi_dsack=0*
- *net/ipv4/tcp_ecn=0*
- *net/ipv4/tcp_fack=0*

Nous avons conservé avec SACK les améliorations de [10] et [17] :
- *net/ipv4/tcp_sack=1*
- *net/ipv4/tcp_window_scaling=1*
- *net/ipv4/tcp_timestamps=1*
- *net/ipv4/ip_no_pmtu_disc=0*

Finalement, la taille maximale autorisée pour les tampons en émission et en réception a été fixée à 8 mégaoctets[15] :
- *net/core/rmem_max=8388608*
- *net/core/wmem_max=8388608*

Ces valeurs ont été rajoutées telles quelles dans le fichier de configuration */etc/sysctl.conf* afin qu'elles soient prises en compte à chaque redémarrage.

Outre ces réglages système, nous avons également installé un certain nombre d'outils qui allaient nous servir. La suite d'outils comprenait :
- *iperf 2.0.2* : outil très complet de mesure de bande passante.
- *tcpdump 3.9.4* : analyseur réseau.
- *tcptrace 6.6.1* : analyseur de traces de tcpdump.

[13] SMP

[14] La désactivation du mécanisme sur les tampons en émission ne se fait que lorsqu'on fixe la valeur de notre *socket* TCP via la primitive *setsockopt()*.

[15] Attention : cette valeur signifie que l'on pourra allouer une valeur d'au maximum 16 Mo pour les tampons. En effet, le noyau Linux alloue dans la mesure du possible, le double de la taille demandée par un utilisateur via la primitive *setsockopt()*.

Volume (Ko)	vsftpd	ftp	$\frac{ftp}{vsftpd}$	iperfs	iperfc	$\frac{iperfc}{iperfs}$	$\frac{iperfs}{ftp}$	$\frac{iperfc}{vsftpd}$
8	3.52	6.98	50.47%	6.5	7.11	109.35%	93.13%	201.79%
16	4.69	7.02	66.86%	9.9	10.66	107.65%	141.01%	227.03%
32	7.05	9.38	75.11%	11.2	11.82	105.49%	119.41%	167.70%
64	10.56	13.14	80.36%	13.8	14.16	102.59%	105.02%	134.08%
128	15.6	18.18	85.80%	18.7	22.52	120.41%	102.87%	144.36%
512	22.94	23.76	96.55%	26.5	28.21	106.45%	111.55%	122.99%
1024	25.6	26.45	96.78%	28.18	28.22	100.13%	106.53%	110.22%
2048	27.69	28.16	98.31%	29.1	29.12	100.07%	103.32%	105.16%
4096	28.84	29.11	99.07%	29.3	29.6	101.01%	100.67%	102.63%
8192	29.33	29.48	99.47%	29.72	29.78	100.21%	100.80%	101.55%
16384	29.64	29.73	99.70%	29.85	29.91	100.18%	100.41%	100.90%
32768	29.74	29.81	99.75%	29.8	29.87	100.24%	99.96%	100.45%

TAB. 2.3 – Comparaison des débits mesurés par différents outils sur des tranferts de données aux volumes variés. Les transferts se font d'une machine du cluster de Nancy vers une machine du cluster de Lille. Les débits reportés sont en Mégabits par seconde (*Mbps*)

- *tracepath* : outil qui parcourt un chemin et découvre le MTU ainsi que le nombre de sauts.
- *vsftpd 2.0.3* : serveur ftp.
- *netcat 1.10* : couteau suisse de TCP/IP.

2.3.2 Validation croisée des outils de mesure

Afin de vérifier la justesse des mesures effectuées par l'outil *iperf*, nous avons comparé les résultats de mesure fournis par le client et le serveur *iperf* au cours d'un certain nombre de tranferts avec celles d'un client et d'un serveur ftp. Les transferts ont été faits pour des volumes de données compris entre 8 kilooctets et 256 mégaoctets.

Comme nous n'avons pas réussi à trouver de serveur et de client ftp non-payants qui en plus d'afficher les vitesses de transfert permettent d'ajuster facilement à l'image d'*iperf* la taille de tampons, nous n'avons eu d'autre choix que de laisser l'autotuning en émission actif. Celui en réception a été maintenu désactivé (cf. 2.3.1, p. 25).

La comparaison des mesures s'est faite entre émetteurs et entre récepteurs. Ainsi, les mesures du serveur *iperf* ont été comparées à celles du client ftp. Quant aux mesures du client *iperf*, elles l'ont été avec celles du serveur ftp. Les résultats sont reportés dans le tableau TAB. 2.3 (p. 27).

Pour des volumes de données supérieurs à 1 Mo, la différence entre les mesures prises par les quatre outils diffèrent d'au plus 10%. Pour les volumes de données plus faibles, l'écart peut être très important. En effet, il atteint 127% pour les 16 Ko. Ceci est d'autant plus vrai que les tampons sont grands[16]. Dans de telles conditions, les mesures sont aberrantes surtout sur l'outil de mesure situé sur l'émetteur.

La différence importante entre les mesures de *ftp* et *vsftpd* s'attribue au fait que les deux programmes n'utilisent pas forcément la même méthode de mesure du temps.

[16] Testé avec *iperf*.

Le serveur et le client *iperf* correspondent à un seul et même programme. Ainsi, que ce dernier agisse en tant que client ou en tant que serveur, la méthode de mesure reste la même. C'est ce qui explique que les mesures effectuées par le client *iperf* et le serveur *iperf* soient plus proches que celles mesurées par le client ftp et le serveur ftp.

Les mesures prises par *iperf* peuvent être considérées fiables pour des volumes de données importants. Dans les cas que nous avons anvisagé ils doivent être supérieurs à 16 Mo.

2.3.3 Vérification de la stabilité de Grid'5000

Après nous être assurés de la fiabilité de notre logiciel de mesure, nous avons entamé un certain nombre d'expériences qui ont confirmé les qualités de l'environnement en termes de stabilité de latence et la capacité du réseau.

Description des tests

Le test prévoyait d'effectuer plusieurs mesures de débits, espacés par des mesures de RTT, des mesures du nombre de sauts et de la taille de MTU selon le schéma de base suivant :

- 10 mesures de RTT.
- 1 mesure du MTU et du nombre de sauts.
- 1 mesure de débit pour chacun des transferts de volume de données compris entre 8 Ko et 512 Mo.

Ce schéma a été réitéré vingt fois pour chaque couple de machines (a, b) considéré et ce dans les deux sens : de a vers b mais également de b vers a. Comme les tests ont été effectués dans le cas de communications intersite mais aussi intrasite, les machines a et b appartiennent soit aux clusters de deux sites différents soit au cluster d'un même site. Comme les machines au sein d'un même cluster ont une configuration matérielle exactement identique, il était indifférent qu'on n'utilisât toujours la même machine ou non.

Les trois étapes du schéma ont été réalisées respectivement à l'aide des commandes *ping*, *tracepath* et *iperf*.

La taille des tampons d'émission et de réception étaient, dans un premier temps, laissées à leur valeur par défaut[17] afin de mesurer les performances par défaut. Puis, dans un second temps, elles ont été fixées à 16 mégaoctets afin de lever toute limitation éventuelle liée à leur taille. Ainsi, on pouvait facilement atteindre les limites du réseau.

Résultats intrasites

Aussi bien pour les tailles de tampons par défaut que pour les tailles de 16 Mo, on observe un débit moyen de 940 Mbps, soit 94% du débit nominal (cf. Tab. 2.5, p. 30). Le RTT est par ailleurs très stable avec un écart type inférieur à 0.01 et une valeur moyenne inférieure à 0.12 millisecondes (ms) pour tous les sites (cf. Tab. 2.4, p. 30).

[17] respectivement 16 et 85.3 Ko

De plus, comme annoncé dans la section précédente, les mesures sur des tranferts impliquant des volumes de données faibles sont imprécises et donc peu significatives. Les valeurs données précédemment se vérifient pour des volumes supérieurs à 128 Mo.

Résultats intersites

Peu importe la taille des tampons, les RTT mesurés entre deux sites restent très stables mais sensiblement plus élevés qu'en intrasite (cf. TAB. 2.4, p. 30).

Les débits mesurés sont également très stables pour différentes tailles de tampons. Ce qui change, c'est qu'avec les tampons de 16 Mo on s'approche du débit théorique de 1 Gbps, mais les valeurs demeurent néanmoins en retrait par rapport aux valeurs intrasites (cf. TAB. 2.6, p. 30).

De l'observation des débits pour les tailles de tampons par défaut, il ressort une forte corrélation avec la valeur du RTT.

Les débits mesurés se stabilisent à partir de volumes de 2 Mo pour les tampons par défaut et 128 pour ceux de 16 Mo. Les mesures peuvent être considérées fiables pour des transferts dont le volume excède ces valeurs.

Ces expériences confirment :

- d'une part les qualités du réseau en termes de stabilité des RTT et des débits mesurés à partir d'une certaine taille de données. Pour des volumes faibles, l'influence du RTT et l'interaction système deviennent prépondérantes et ne facilitent ni la prise de mesure ni la prédiction. C'est pour cela que dorénavant nous ne nous intéresserons qu'à des prédictions sur des volumes de données importants.
- d'autre part, à la vue de ces premiers résultats, il semblerait qu'une construction de modèle à partir de la partie gauche de la formule de Padhye [18] :

$$\text{débit} \simeq \frac{W_m}{RTT} \tag{2.2}$$

est réaliste, sous l'hypothèse qu'il n'y ait pas de saturation du réseau et donc de la congestion et des pertes. Nous allons vérifier l'adéquation de la formule dans le chapitre suivant.

—	Bordeaux	Lille	Nancy	Orsay	Rennes	Sophia	Toulouse
Bordeaux	0.09/0.01	1.09/0.01	3.09/0.01	4.09/0.01	5.09/0.01	6.09/0.01	7.09/0.01
Lille	23.20/0.00	–	4.09/0.01	4.53/0.02	11.61/0.03	18.47/1.02	–
Nancy	12.05/0.71	8.50/1.09	5.09/0.01	5.79/0.95	11.60/0.00	19.83/0.05	–
Orsay	14.31/0.03	4.53/0.02	6.09/0.01	0.14/0.25	9.00/0.02	16.01/0.03	–
Rennes	7.98/0.01	11.63/0.26	7.09/0.01	9.04/0.03	18.10/0.00	14.61/12.55	–
Sophia	10.30/0.01	18.40/0.01	8.09/0.01	16.02/0.04	18.10/0.00	0.09/0.01	–
Toulouse	–	–	–	11.66/0.28	0.09/0.00	–	–

TAB. 2.4 – RTT moyen/écart type (en millisecondes) observés en intersite et intrasite sur Grid'5000.

—	Bordeaux	Lille	Nancy	Orsay	Rennes	Sophia	Toulouse
Bordeaux	940.14/3.23	11.85/0.01	22.68/0.14	19.07/0.03	34.10/0.02	26.27/0.01	–
Lille	11.85/0.00	–	32.30/0.01	58.92/0.01	23.44/0.00	14.87/0.00	–
Nancy	22.37/0.03	32.02/0.63	941.04/0.16	47.12/0.02	23.72/0.04	13.75/0.07	–
Orsay	19.08/0.00	58.95/0.06	47.12/0.11	880.66/20.72	30.18/0.00	17.04/0.00	–
Rennes	34.09/0.01	23.43/0.00	23.74/0.00	30.13/0.00	–	15.10/0.00	25.23/0.00
Sophia	26.29/0.00	14.87/0.00	13.82/0.00	17.04/0.00	15.11/0.00	940.34/2.20	–
Toulouse	–	–	–	–	25.24/0.00	–	941.45/0.04

TAB. 2.5 – Débits moyens/écart type (en Mbps) observés en intersite et en intrasite sur Grid'5000. Les mesures ont été effectuées avec un tampon en émission de 16 Ko et un tampon en réception de 85.3 Ko. Ce sont les valeurs de tampons par défaut de Linux.

—	Bordeaux	Lille	Nancy	Orsay	Rennes	Sophia	Toulouse
Bordeaux	936.68/11.17	735.27/102.90	794.34/181.37	850.63/45.40	–	831.10/9.27	–
Lille	752.21/103.39	937.95/7.99	486.10/35.42	931.57/6.98	–	854.54/2.70	–
Nancy	902.92/0.25	906.53/25.19	941.21/0.01	923.54/0.45	–	–	–
Orsay	811.45/46.86	907.40/18.80	698.43/48.07	–	887.30/11.85	851.82/6.02	–
Rennes	–	–	–	873.28/63.32	–	937.61/5.84	–
Sophia	717.30/66.33	829.10/2.43	–	821.55/28.42	–	–	941.43/0.01
Toulouse	–	–	–	–	–	–	–

TAB. 2.6 – Débits moyens/écart type (en Mbps) observés en intersite et en intrasite sur Grid'5000. La valeur des tampons en émission et en réception est de 16 Mo.

Chapitre 3

Vers l'élaboration d'un modèle

Afin de mettre au point notre modèle, nous avons choisi de suivre une approche incrémentale. Cette approche consiste à élaborer le modèle sur des cas simples et à l'étendre à des configurations plus complexes *a posteriori*.

Suite aux conclusions tirées du chapitre précédent, nous avons décidé d'utiliser comme point de départ la partie de la formule de Padhye [18] correspondant au cas où les pertes sont absentes (Éq. 2.2, p. 29).

La réalité sous-jacente à cette équation a été confirmée dans plusieurs études. Néanmoins, elle nécessite de garantir l'absence de perte et d'être capable de déterminer la valeur de la fenêtre d'émission. L'absence de perte peut aisément se vérifier. Pour cela, il suffit de capturer le trafic lors du transfert pour ensuite l'analyser. Le nombre de retransmissions renseigne clairement sur le taux de pertes, surtout lorsque l'option *SACK* est utilisée. Par contre, la détermination de W_m est loin d'être aussi simple.

De manière générale, la taille de la fenêtre d'émission est donnée par l'équation 1.1 (p. 15). En supposant qu'il n'y ait ni de congestion (i.e. pas de pertes) ni de limitation de la part du récepteur via la fenêtre qu'il annonce régulièrement à l'émetteur, la fenêtre d'émission *wnd* serait simplement bornée par la taille allouée au tampon d'émission *snd_buff* :

$$swnd = min(cwnd, aw, snd\_buff) \Longrightarrow swnd = snd\_buff \tag{3.1}$$

Or, il se trouve que sous *Linux*, les tampons contiennent également un certain *overhead* lié à la bufferisation des données[1]. Il en résulte que lorsque la taille du tampon d'émission devient le facteur limitant, la place occupée par l'*overhead* se traduit par une diminution de la taille de la fenêtre.

La documentation du noyau Linux explique bien comment se calcule la quantité de mémoire reservée pour l'*overhead* dans le cas du tampon de réception. Cependant, aucune information utile permettant d'en faire autant pour le tampon d'émission n'est fournie.

[1] Il semblerait que cela ne soit pas le cas sous *FreeBSD* par exemple, où la mémoire utilisée pour l'*overhead* n'est pas puisée dans celle des tampons.

3.1 Calcul de l'*overhead* côté récepteur

Pour les tampons en réception, la quantité de mémoire qui sera utilisée pour l'*overhead* se calcule à partir de la valeur de la variable *tcp_ adv_ win_ scale* selon la formule suivante issue de [13] :

$$\begin{cases} overhead = \frac{BR}{2^{tcp\_adv\_win\_scale}} & \text{Si } tcp\_adv\_win\_scale > 0 \\ overhead = BR - \frac{BR}{2^{-tcp\_adv\_win\_scale}} & \text{Sinon} \end{cases} \quad (3.2)$$

avec BR la taille du tampon de réception allouée par le noyau Linux.

Par défaut, cette variable vaut 2. Ainsi, un quart du tampon de réception alloué sera pour de l'*overhead*.

Au cours des expériences menées pour déterminer la taille de l'*overhead*, nous en avons également profité pour vérifier la formule 3.2. Il s'est avéré pour une raison inconnue qu'en réalité la taille maximale[2] de la fenêtre annoncée par le récepteur est :

$$aw = \frac{(BR - overhead)}{2} \quad (3.3)$$

3.2 Détermination de l'*overhead* côté émetteur

Comme nous l'avons déjà énoncé, il n'existe pas de formule équivalente à celle des tampons en réception.

L'objectif de cette section est donc de déterminer la part de mémoire occupée par l'*overhead* au niveau des tampons d'émission pour un certain nombre de tailles de tampons différentes. Ceci permettrait d'étudier les variations de la taille de l'*overhead* et voir si elle présente une certaine régularité, voire une constance.

L'idée principale pour y parvenir était de déterminer la taille moyenne de la fenêtre d'émission afin d'en déduire la part occupée par l'*overhead*.

Une fois encore, nous n'avons pas trouvé d'outil donnant directement les valeurs de la taille de la fenêtre d'émission au cours d'une communication TCP. La méthode que nous avons utilisée, repose sur le comportement que nous avons observé chez l'émetteur à partir de l'analyse de nombreuses traces de *tcpdump*.

3.2.1 Comportement sur l'émetteur

L'analyse de traces de *tcpdump* permet de mettre en évidence deux caractéristiques quant au comportement de TCP :

- La stabilisation très rapide de la taille de la fenêtre d'émission.
- Le comportement *transmit and wait*.

[2] La fenêtre annoncée grandit au fur et à mesure jusqu'à atteindre la valeur donnée par la formule 3.3. Probablement, il s'agit là d'un mécanisme de protection contre des émetteurs sans scrupules qui ne respecteraient pas le slow-start.

La première observation est liée à la qualité de l'environnement d'expérimentation et en particulier à l'absence de pertes. TCP connaît la phase du *slow-start* et celle d'évitement de congestion avant de voir la taille de la fenêtre se stabiliser à cause de la limitation du tampon d'émission FIG. 3.1.

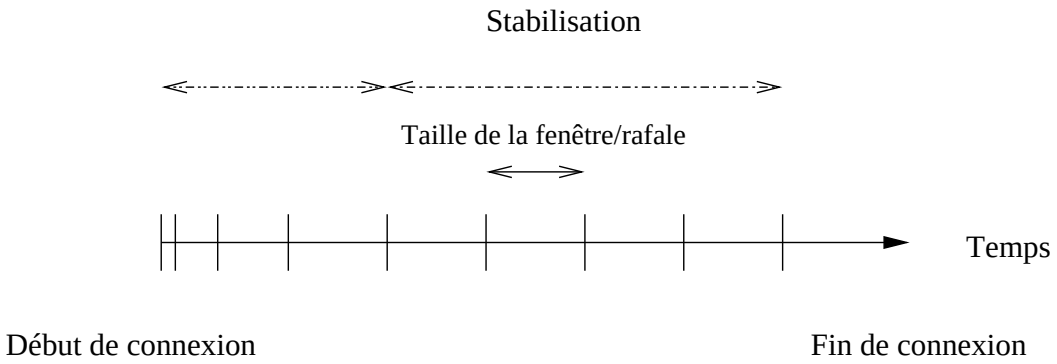

FIG. 3.1 – Évolution de la taille de la fenêtre d'émission au cours d'une connexion TCP en absence de pertes.

La deuxième observation désigne l'activité par rafales de TCP, aussi bien du côté de l'émetteur que celui du récepteur.

L'émetteur envoie (*transmit*) autant de paquets que le lui permet sa fenêtre[3], puis se met en attente d'acquittements (*wait*).

Le récepteur procède de façon similaire, mais envoie des acquittements en rafale (cf. FIG. 3.2, p. 34).

Pendant les rafales, TCP envoie à la vitesse maximale de l'interface réseau qui est voisine du Gigabits dans notre cas. Le temps qui s'écoule entre l'envoi de deux paquets consécutifs monte jusqu'à une dizaine de microsecondes, voire par moments une centaine. Le passage à la centaine de microsecondes est relativement rare et irrégulier. Nous l'attribuons à l'activité du système d'exploitation[4].

Le temps d'attente entre l'envoi de plusieurs rafales est très voisin du RTT de la connexion pour de petites tailles de tampons en émission et a tendance à diminuer pour des tailles de tampons suffisamment grandes. Pour de grands tampons, la connexion est saturée. Dans ce dernier cas, il n'y a pas de temps d'attente.

Ainsi, dans le cas de l'utilisation de tailles de tampon d'émission convenables, il est possible de discriminer au sein d'une trace *tcpdump* les rafales et par conséquent d'en déduire la taille de la fenêtre en émission. C'est sur ce principe que fonctionne notre script pour calculer la taille moyenne de la fenêtre en émission à partir d'une trace *tcpdump*.

[3] C'est ce que l'on désigne par rafale.

[4] Notamment à cause de l'ordonnancement des démons système.

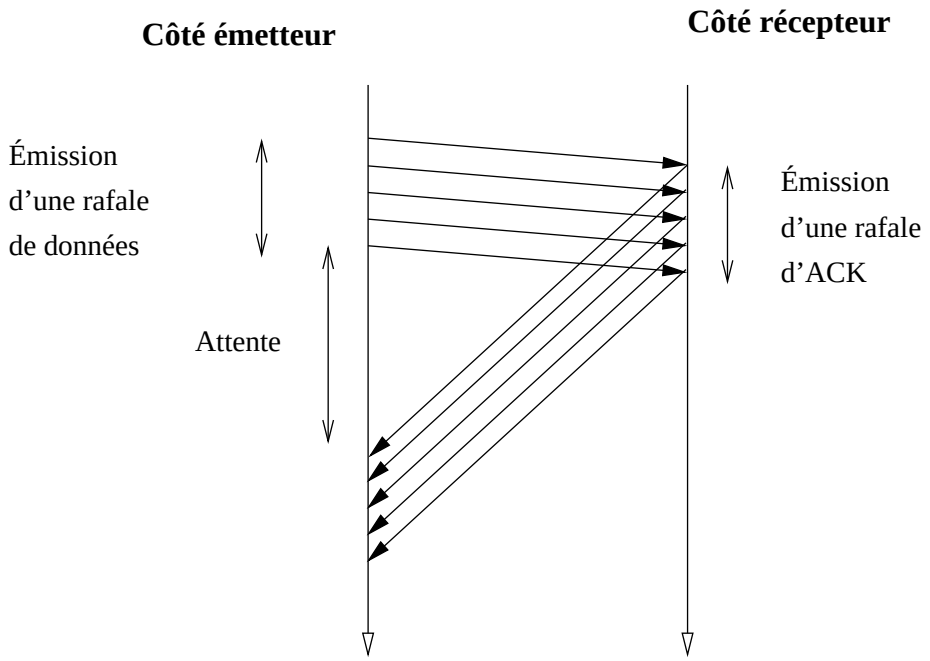

FIG. 3.2 – Illustration du comportement en rafale de TCP.

3.2.2 Étude/Détermination de l'*overhead* du tampon d'émission

L'étude de l'*overhead* du tampon d'émission s'est faite au travers de trois séries d'expériences. Chacune d'entre elles a consisté à effectuer des transferts intersite d'une durée de 10 secondes et ce, pour différentes tailles de tampons en émission et en réception.

La totalité de chaque transfert était capturée sur l'émetteur et sur le récepteur à l'aide de *tcpdump* pour être ensuite analysée par *tcptrace* et par un de nos scripts.

tcptrace fournit des statistiques d'une connexion TCP à partir d'une capture *tcpdump*. Ces dernières sont utiles d'une part, parce qu'elles constituent un moyen tiers de vérifier le débit affiché par *iperf* et d'autre part, parce qu'elles indiquent notamment le nombre d'octets/blocs retransmis ainsi que la présence de SACK au cours du transfert.

Première expérience

La première expérience s'est déroulée entre un client *iperf* situé sur un noeud du site de Nancy et un serveur *iperf* situé sur un noeud du site de Bordeaux. Pour chaque transfert, le tampon en réception était fixé suffisamment grand (16 Mo) pour éviter que le récepteur exerce du contrôle de flux et limite ainsi le débit. Le tampon en émission prenait une valeur différente comprise entre 8 Ko et 16 Mo. Le RTT était de l'ordre de 12 ms. Les résultats de cette expérience figurent dans le tableau TAB. 3.1 (p. 35). Les abréviations utilisées sont :

- BR : taille du tampon de réception allouée.
- BE : taille du tampon d'émission allouée.
- FE : taille moyenne de la fenêtre d'émission.

Débit (Mbps)	BR (Ko)	BE (Ko)	FE (Ko)	$\frac{FE}{BE}$
3,573	16384	8	4,22	0,527
7,332	16384	16	10,025	0,626
13,745	16384	32	20,085	0,627
26,67	16384	64	40,377	0,63
54,35	16384	128	84,718	0,661
105,312	16384	256	168,277	0,657
170,787	16384	512	271,173	0,529
388,215	16384	1024	693,773	0,677
687,166	16384	2048	1488,822	0,726
917,298	16384	4096	5542,668	1,353

TAB. 3.1 – Première expérience : transferts *iperf* de Nancy vers Bordeaux. $RTT = 12$ (ms), $BR = 16$ (Mo).

La quantité de mémoire occupée par l'*overhead* ($= 1 - \frac{FE}{BE}$) est relativement stable et varie entre 27 et 47%. Ainsi, la quantité de mémoire occupée par la fenêtre d'émission est au moins égale à 50% de la taille totale du tampon.

Le ratio de 1.353, calculé pour un tampon de 4 Mo, est attribué au manque de robustesse de notre méthode pour des tailles de tampons permettant d'atteindre le débit maximal de la connexion (cf. 3.2.1, p. 32).

Deuxième expérience

La seconde expérience suit exactement le même protocole que la première, sauf que l'émetteur est placé sur un site plus proche : celui de Toulouse. Ceci s'est fait pour observer une influence éventuelle liée au RTT qui avoisine dans ce cas les 4 ms. Les résultats de cette expérience figurent dans le tableau TAB. 3.2 (p. 35).

Débit (Mbps)	BR (Ko)	BE (Ko)	FE (Ko)	$\frac{FE}{BE}$
13,997	16384	8	-	-
26,436	16384	16	9,158	0,572
51,161	16384	32	18,752	0,586
99,828	16384	64	21,918	0,342
207,098	16384	128	77,95	0,608
381,242	16384	256	171,431	0,669
669,583	16384	512	340,541	0,665
924,73	16384	1024	19462,37	19,006
923,136	16384	2048	125213,105	61,139
888,721	16384	16384	25230,235	1,539

TAB. 3.2 – Deuxième expérience : transferts *iperf* de Toulouse vers Bordeaux. $RTT = 4$ (ms), $BR = 16$ (Mo).

La quantité de mémoire occupée par l'*overhead* se comporte de manière analogue à la première expérience. Elle est inférieure à 50%, sauf pour le tampon de 64 Ko.

Pour des tailles de tampons qui permettent d'atteindre le débit maximal de la connexion, on observe une fois encore des ratios supérieurs à 1. Cette fois-ci, le débit maximal est atteint dès l'utilisation de tampons en émission de 1 Mo, contre 4 Mo dans la première expérience. C'est directement lié au RTT de la connexion entre Toulouse et Bordeaux qui est trois fois plus faible que celui entre Nancy et Bordeaux.

Troisième expérience

La seule différence dans cette expérience par rapport à la deuxième se situe au niveau de la taille du tampon de réception. Elle vaut 256 Ko à présent. Il s'agit d'une valeur plus proche des valeurs que l'on rencontre par défaut (cf. 2.2.1). Les résultats de cette expérience figurent dans le tableau Tab. 3.3.

Débit (Mbps)	BR (Ko)	BE (Ko)	FE (Ko)	$\frac{FE}{BE}$
13,525	256	8	4,001	0,5
24,776	256	16	8,401	0,525
47,869	256	32	17,714	0,553
104,502	256	64	24,993	0,39
197,735	256	128	81,741	0,638
244,037	256	256	240,27	0,938

Tab. 3.3 – Troisième expérience : transferts *iperf* entre Toulouse et Bordeaux. $RTT = 4$ (ms), $BR = 256$ (Ko).

Pour cette expérience, hormis le cas du tampon en émission de 64 Ko, l'*overhead* reste inférieur à 50%.

Conclusions des expériences

Au vu des résultats des expériences précédentes, l'*overhead* occupe dans la plupart des cas entre 30% et 50% de la mémoire allouée au tampon d'émission. Il atteint un minimum de 10% dans la troisième expérience (cf. Tab. 3.3) pour un tampon en émission et en réception de 256 Ko. On observe un maximum de 70% pour un tampon en émission de 64 Ko (cf. Tab. 3.3 et Tab. 3.2). Il semblerait que ce soit un cas isolé qui mériterait d'être réexaminé.

Enfin, lorsque les tailles des tampons permettent d'atteindre le débit maximum, le calcul de la taille de la fenêtre d'émission n'est plus significatif pour les raisons évoquées précédemment (cf. 3.2.1, p. 32).

3.3 Vérification du modèle initial

Maintenant que l'on dispose d'un moyen de déterminer les tailles des fenêtres de réception et surtout de celles en émission, nous avons voulu vérifier l'applicabilité du modèle suggéré par la formule de Padhye en cas d'absence de pertes dans notre situation particulière.

La validation s'est faite sur plusieurs expériences réalisées :

- de Nancy vers Bordeaux.
- de Toulouse vers Bordeaux.

Tout comme les expériences réalisées pour déterminer l'*overhead*, nous avons fait varier la taille du tampon en émission entre 8 Ko et 16 Mo. Celle en réception prenait, quant à elle, la valeur 256 Ko ou 16 Mo.

Par rapport aux expériences précédentes, nous avons également considéré le cas où la taille allouée au tampon d'émission est supérieure à celle du tampon en réception. Dans cette situation, la formule s'écrit :

$$BW = \frac{min(FR, FE)}{RTT} \tag{3.4}$$

Les tableaux Tab. 3.4 (p. 38) et Tab. 3.5 (p. 38) contiennent les résultats des mesures, ainsi que quelques prédictions, notamment :

- FE : prédiction de la taille de la fenêtre d'émission.
- FR : prédiction de la taille de la fenêtre de réception.
- D_p : Débit prédit.

Figurent également dans les tableaux :

- BE : taille du tampon d'émission.
- BR : taille du tampon de réception.
- D_m : Débit mesuré.
- $\frac{D_p - D_m}{D_m}$: l'écart relatif. Il permet d'évaluer la précision de la formule.

Quant à l'*overhead* sur la fenêtre en émission, il a été fixé à 45%. Ainsi, on estime que la taille de la fenêtre d'émission sera d'au plus 55% de la taille du tampon alloué en émission. Ce choix semble raisonnable, étant donné les résultats en 3.2.2.

Finalement, la formule utilisée pour effectuer la prédiction est :

$$BW = \frac{min(BR * C_r, BE * C_e)}{RTT} \tag{3.5}$$

avec $C_e = 0.55$ et $C_r = \frac{3}{8}$. C_e provient de nos statistiques. C_r est calculé d'après les équations 3.2 et 3.3 (p. 32).

Les prédictions obtenues sont très satisfaisantes dans la mesure où les valeurs prédites diffèrent d'au plus 30%, par excès ou par défaut, des débits mesurés[5]. La plupart des valeurs diffèrent généralement de moins de 20%.

[5] Mis à part le cas limite (cf. 3.2.1).

RTT (ms)	BE (Ko)	BR (Ko)	D_m (Mbps)	FE (Ko)	FR (Ko)	D_p (Mbps)	$\frac{D_p - D_m}{D_m}$
12	8	16384	3.84	4.4	6144	3	-21.78%
12	16	16384	7.36	9.6	6144	6.55	-10.96%
12	32	16384	14.2	19.2	6144	13.11	-7.70%
12	64	16384	24.9	38.4	6144	26.21	5.28%
12	128	16384	51.9	76.8	6144	52.43	1.02%
12	256	16384	97.7	153.6	6144	104.86	7.33%
12	512	16384	196	307.2	6144	209.72	7.00%
12	1024	16384	396	614.4	6144	419.43	5.92%
12	2048	16384	693	1228.8	6144	838.86	21.05%
12	8	256	3.57	4.8	96	3.28	-8.21%
12	16	256	6.64	9.6	96	6.55	-1.30%
12	32	256	13.4	19.2	96	13.11	-2.19%
12	64	256	24.8	38.4	96	26.21	5.70%
12	128	256	51.6	76.8	96	52.43	1.61%
12	256	256	64.2	153.6	96	65.54	2.08%
12	512	256	64.2	307.2	96	65.54	2.08%
12	16384	256	70.3	9830.4	96	65.54	-6.78%
12.7	8	16384	3.57	4.8	6144	3.1	-13.27%
12.7	16	16384	6.9	9.6	6144	6.19	-10.26%
12.7	32	16384	13.7	19.2	6144	12.38	-9.60%
12.7	64	16384	26.7	38.4	6144	24.77	-7.23%
12.7	128	16384	54.4	76.8	6144	49.54	-8.94%
12.7	256	16384	105	153.6	6144	99.08	-5.64%
12.7	512	16384	171	307.2	6144	198.16	15.88%
12.7	1024	16384	382	614.4	6144	396.31	3.75%
12.7	2048	16384	673	1228.8	6144	792.62	17.77%

TAB. 3.4 – Comparaison entre les mesures et les prédictions de débits pour des transferts de Nancy vers Bordeaux avec différentes tailles de tampons.

RTT (ms)	BE (Ko)	BR (Ko)	D_m (Mbps)	FE (Ko)	FR (Ko)	D_p (Mbps)	$\frac{D_p - D_m}{D_m}$
4	8	16384	14	4.8	6144	9.83	-29.78%
4	16	16384	26.4	9.6	6144	19.66	-25.53%
4	32	16384	51.2	19.2	6144	39.32	-23.20%
4	64	16384	98.8	38.4	6144	78.64	-20.40%
4	128	16384	207	76.8	6144	157.29	-24.02%
4	256	16384	381	153.6	6144	314.57	-17.43%
4	512	16384	669	307.2	6144	629.15	-5.96%
4	8	256	13.5	4.8	96	9.83	-27.18%
4	16	256	24.8	9.6	96	19.66	-20.72%
4	32	256	47.5	19.2	96	39.32	-17.22%
4	64	256	104	38.4	96	78.64	-24.38%
4	128	256	198	76.8	96	157.29	-20.56%
4	256	256	244	153.6	96	196.61	-19.42%
4	512	256	245	307.2	96	196.61	-19.75%
4	16384	256	252	9830.4	96	196.61	-21.98%

TAB. 3.5 – Comparaison entre les mesures et les prédictions de débits pour des transferts de Toulouse vers Bordeaux avec différentes tailles de tampons.

Conclusion

Durant ce stage, nous nous sommes intéressés à l'étude du comportement des communications TCP et à celle de modèles de coûts existants, dans le but d'en extraire un modèle destiné à être utilisé par des applications s'exécutant sur des grilles de calcul.

Le modèle recherché devait être d'une part fiable et d'autre part suffisamment simple pour être exploitable au niveau applicatif. En particulier, il devait être le moins dépendant possible de paramètres liés à l'état du réseau, tel que le taux de pertes que l'on retrouve dans les travaux de modélisation de TCP.

Pour mener à bien notre étude, nous avons pris comme point de départ Grid'5000 et un système d'exploitation fondé sur un noyau Linux.

Grid'5000 semblait être le terrain idéal pour commencer à expérimenter car, comme nous avons pu le constater et le vérifier, il s'agit d'un environnement qui a été pensé pour conduire des expériences de manière contrôlée.

En revanche, le noyau Linux a été à l'origine de nombreuses interrogations liées à la façon dont il implémente le protocole TCP et notamment à la gestion des tampons. En effet, la mémoire occupée par l'*overhead* est puisée dans celle des tampons, ce qui alourdi l'étude.

Cet obstacle a été surmonté et par la même occasion, nous avons pu vérifier ce qui peut être considéré comme le fondement de notre modèle : débit $= \frac{min(FE,FR)}{RTT}$. Ce dernier ne s'applique pour l'instant qu'à une unique communication point-à-point et nécessite d'une part que les volumes de données transférés soient suffisamment importants et d'autre part que le réseau soit dépourvu de congestion. Dans ces conditions, la vérification du modèle est très satisfaisante.

La prochaine étape serait dans un premier temps de confirmer le résultat en considérant pour émetteurs d'autres systèmes d'exploitation, puis dans un second temps, de s'attaquer à des configurations plus complexes et plus intéressantes impliquant d'abord trois machines, puis graduellement plus et ce, pour différents types d'opérations de communications collectives telles que : *all-to-all*, *all-to-one* ... Dans ces conditions, la probabilité d'être en congestion à cause de connexions concurrentes est accrue, les limites du réseau se voient atteintes et les performances du modèle que nous avons pu vérifier en fin de stage risquent de se dégrader mais c'est à ce moment qu'il pourra évoluer.

À ce stade, un modèle plus complet est très envisageable et sa vérification à l'échelle de Grid'5000 serait extrêmement intéressante pour les applications de ses utilisateurs.

Finalement, au vu de l'évolution des réseaux en termes de fiabilité, de capacité et de rapidité, il ne serait pas non plus irréaliste dans un futur proche de voir ce modèle s'appliquer à des réseaux plus généralistes que ceux utilisés dans Grid'5000.

Bibliographie

[1] M. Allman, V. Paxon, and W. Stevens. RFC 2581 : TCP congestion control, April 1999. Status : PROPOSED STANDARD.

[2] Douglas E. Comer. *Internetworking with TCP/IP Volume 1 : Principles, Protocols, and Architectures*. Prentice Hall, 2000.

[3] Sonia Fahmy and Tapan Prem Karwa. Tcp congestion control : Overview and survey of ongoing research. Technical Report CSD-TR-01-016, Purdue University, 2001.

[4] S. Floyd and T. Henderson. RFC 2582 : The newreno modification to tcp's fast recovery algorithm, April 1999. Status : EXPERIMENTAL.

[5] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. RFC 2883 : An extension to the selective acknowledgement (sack) option for tcp, July 2000.

[6] Stéphane Genaud, Arnaud Giersch, and Frédéric Vivien. Load-balancing scatter operations for grid computing. *Parallel Computing*, 30(8) :923–946, August 2004.

[7] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer tcp throughput. In *SIGCOMM '05 : Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–156, New York, NY, USA, 2005. ACM Press.

[8] Qi He, Constantinos Dovrolis, and Mostafa Ammar. Prediction of tcp throughput : formula-based and history-based methods. In *SIGMETRICS '05 : Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 388–389, New York, NY, USA, 2005. ACM Press.

[9] V. Jacobson. Modified tcp congestion avoidance algorithm, 1990.

[10] V. Jacobson, R. Braden, and D. Borman. RFC 1323 : TCP extensions for high performance, May 1992. Obsoletes RFC1072, RFC1185. Status : PROPOSED STANDARD.

[11] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[12] E. Jeannot and F. Wagner. Fast and efficient message scheduling algorithms for data redistribution through a backbone. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. IEEE Computer Society, 2004.

[13] A. Kleen and N. Singhvi. Linux programmer's manual tcp(7) : tcp - tcp protocol, June 2005.

[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018 : TCP selective acknowledgment options, October 1996. Status : PROPOSED STANDARD.

[15] Matthew Mathis and Jamshid Mahdavi. Forward acknowledgment : Refining TCP congestion control, August 21 1996.

[16] Matthew Mathis, Jeffrey Semke, and Jamshid Mahdavi. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3) :67–82, 1997.

[17] J. C. Mogul and S. E. Deering. RFC 1191 : Path MTU discovery, November 1990. Obsoletes RFC1063. Status : DRAFT STANDARD.

[18] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose. Modeling tcp reno performance : a simple model and its empirical validation. *IEEE/ACM Trans. Netw.*, 8(2) :133–145, 2000.

[19] J. Postel. RFC 793 : Transmission control protocol, September 1981. See also STD0007. Status : STANDARD.

[20] Ravi S. Prasad, Manish Jain, and Constantinos Dovrolis. Socket buffer auto-sizing for high-performance data transfers. Technical report, College of Computing Georgia Tech, January 06 2004.

[21] K. Ramakrishnan, S. Floyd, and D. Black. RFC 3168 : The addition of explicit congestion notification (ecn) to ip, September 2001. Obsoletes RFC2481, Updates RFC2474, RFC2401, RFC793 [19]. Status : PROPOSED STANDARD.

[22] A. Gurtov S. Floyd, T. Henderson. RFC 3782 : The newreno modification to tcp's fast recovery algorithm, April 2004. Status : PROPOSED STANDARD.

[23] P. Sarolahti and A. Kuznetsov. Congestion control in linux tcp, 2002.

[24] W. Stevens. RFC 2001 : TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, January 1997. Status : PROPOSED STANDARD.

Liste des tableaux

Table des figures