OMONT Nicolas
LAPEYRIE Jean−Baptiste

Modélisation et simulation de réseaux de communnication

LE PROTOCOLE TCP :

ETUDE ET SIMULATION

Table des matières

Introduction

Le protocole TCP

Il a pour but de permettre une régulation du débit de chaque source dans un réseau. L'un des points importants pour un réseau comme Internet est que ce protocole permette un passage à l'échelle sans problème ; cela implique donc d'avoir un contrôle décentralisé. Même si l'on peut introduire des rétroactions, la source doit se trouver en mesure de gérer d'elle−même son débit dans le réseau, sans le surcharger…

Une régulation par gestion de la fenêtre

Comme on se trouve sur un réseau à commutation de paquets, la source va envoyer des paquets sur celui−ci. Comme on vient de le voir, il faut trouver un mécanisme de contrôle du débit. En fait, TCP effectue un contrôle sur la « fenêtre ». Cette fenêtre correspond aux nombres de paquets envoyés et pour lesquels on n'a toujours pas reçu d'accusé de réception. On introduit ici le RTT (Round Trip Time), qui correspond au temps que mette le message pour aller de la source au destinataire et l'accusé pour revenir à la source. C'est un paramètre très important d'un réseau comme on pourra le voir. En reprenant la formule de LITTLE on obtient que : fenêtre = débit * RTT (ce qui se comprend facilement en reprenant les définitions de chacun). TCP agit donc bien sur le débit en contrôlant cette « fenêtre ».

TCP – Reno

Rentrons un peu plus dans les détails en étudiant le protocole TCP−Reno. On définit une variable *v_seuil*, seuil de la fenêtre, et une variable *v_fenetre*, correspondant à la valeur de la fenêtre autorisée. Tout d'abord, on n'est autorisé à émettre un nouveau paquet que si le nombre de paquets émis sans accusé est inférieur strictement à *v_fenetre*.

A l'instant initial, *v_fenetre* est initialisée à 1 et *v_seuil* à une valeur quelconque. Dans une première phase, on va essayer d'atteindre assez rapidement la valeur *v_seuil*. Pour cela, à chaque paquet reçu, on va augmenter *v_fenetre* de un, tant que : *v_fenetre* < *v_seuil*. Cela signifie qu'en l 'espace d'un RTT, on va pouvoir doubler notre fenetre. Dans cette phase de démarrage (appelée *slow−start*), on obtient une croissance exponentielle de notre fenêtre et donc du débit (exponentielle dépendant du RTT…..) . A partir de là, on entre dans une nouvelle phase, appelée *congestion−avoidance*. Pour chaque accusé reçu, on augment la fenêtre de 1 / |*v_fenetre*| : à chaque RTT, on augmente ainsi *v_fenetre* d'une unité. Cette phase est ainsi linéaire en fonction du temps (on retrouve de nouveau l'influence du RTT dans le coefficient de linéarité).

Obligatoirement, on ne peut augmenter indéfiniment le débit sans saturer un élément du réseau. On va maintenant découvrir comment l'on réagit en fonction des informations reçues du circuit. Sans rentrer pour l'instant dans les détails, il existe deux types d'événement indiquant une congestion du réseau. Tout d'abord, on peut détecter la perte de paquets, en regardant les accusés reçus ; sinon, dans un cas plus grave, où le réseau est vraiment surchargé, on peut ne plus recevoir d 'accusé du tout pendant un temps relativement long : on appelle cela un *timeout*. Dans le premier cas, on réinitialise d'abord *v_seuil* à : |*v_fenetre*| / 2 ; ensuite, on reprend la phase linéaire, à partir de *v_fenetre* = *v_seuil* en retransmettant d'abord les paquets apparemment perdus (on parle de *multiplicative decrease*). Dans le deuxième cas, on réinitialise de la même manière v_seuil, mais on repart du début cette fois−ci, avec v_fenetre = 1, et on retransmet tous les paquets qui étaient en attente d'un accusé.

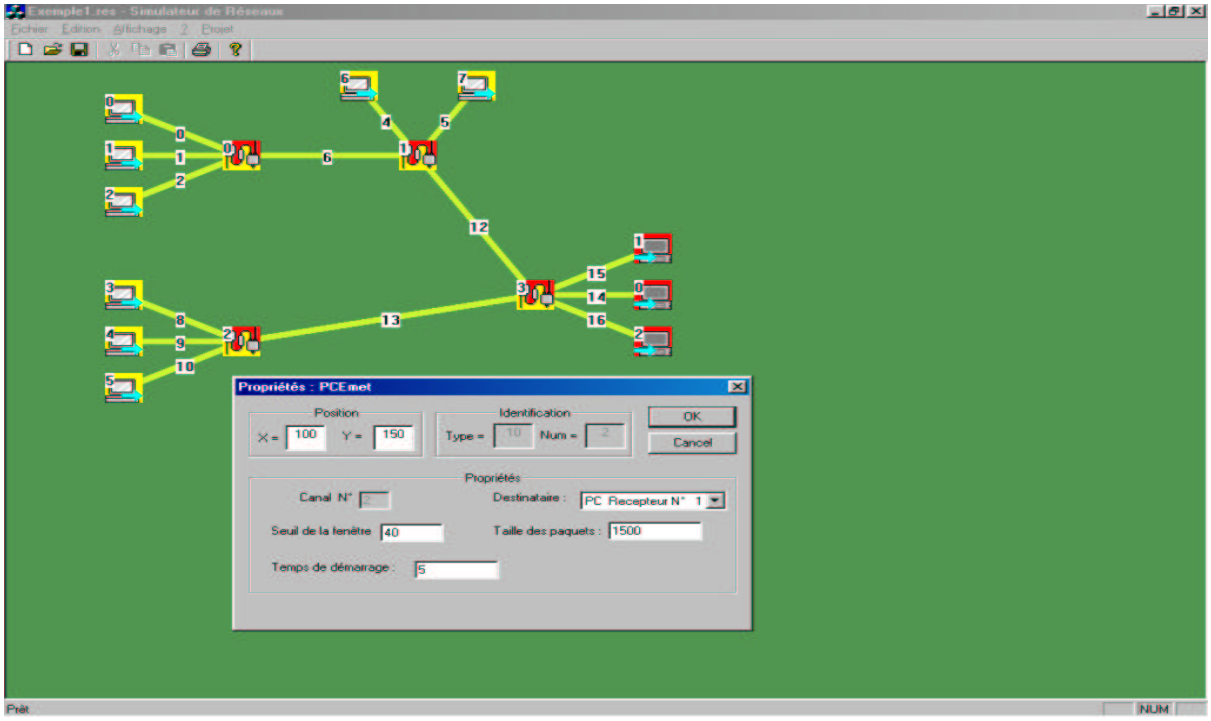
Ce protocole a l'avantage de toujours s'adapter aux conditions en fonction du temps : le réseau peut changer, la source devrait dans une certaine mesure s'y adapter. De plus, un point très important et même primordial pour l'application de ce protocole est qu'il est STABLE.

Il devient désormais intéressant d'essayer de tester ce protocole par rapport aux différents paramètres du réseau, d'étudier le partage du débit entre les différents utilisateurs, et par rapport au débit total possible…

La simulation effectuée est une simulation en temps discret, dont on précisera le schéma de Matthes dans la prochaine partie.

I L'implémentation

1 L'interface graphique



Bien que ce ne soit pas le point le plus important du projet, une interface graphique assez simple a été réalisée, afin de construire de manière visuelle le réseau désiré et de pouvoir modifier directement les paramètres sans avoir à écrire dans un fichier. Elle permet de plus de sauvegarder rapidement une configuration de réseau pour pouvoir la réutiliser en la rechargeant. L'inconvénient est que cette interface réduit la portabilité du programme, qui ne marche que sous Windows.

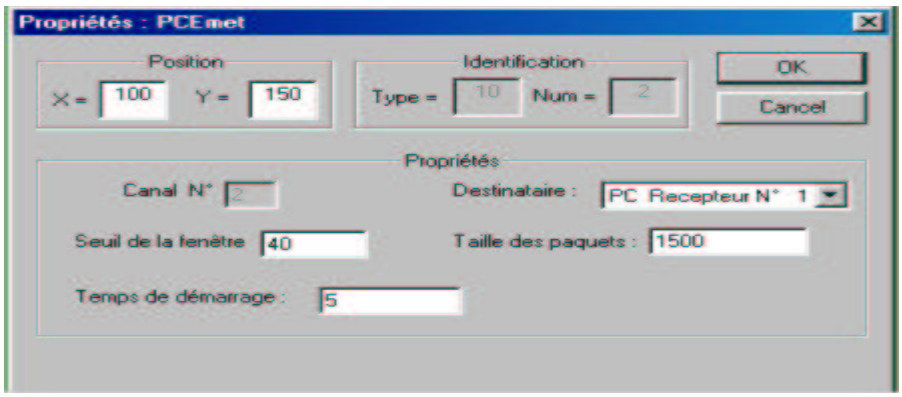
On peut avoir un aperçu de l'affichage dans l'exemple ci−dessus.

a.Les différents éléments

Nous allons maintenant présenter les différents éléments, correspondant à des classes C++, qui composent le réseau, et que l'on peut découvrir sur l'interface graphique. Cela va permettre en même temps de préciser un peu plus le schéma de Matthes sous−jacent. Nous avons décider de profiter de l'orientation objet de C++ pour modéliser les éléments sous forme de classe. Tous les liens entre éléments sont décentralisés et réalisés via des pointeurs (et non sous la forme d'un tableau), ce qui alourdit un peu ces classes, mais permet un accès local aux éléments « voisins ».

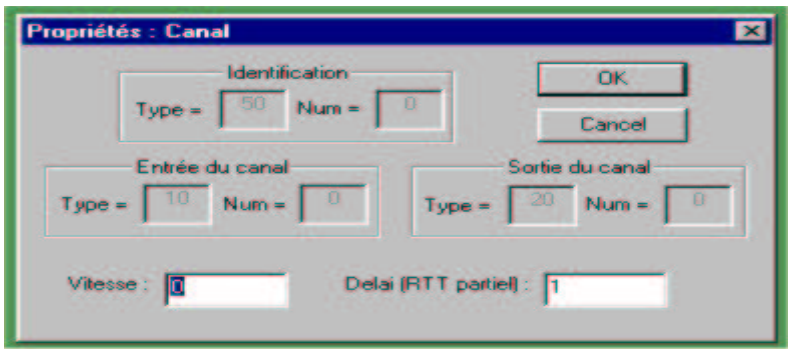
b.Les PC Emetteurs 

Ils correspondent à la source des paquets. Ce sont donc eux qui vont émettre les paquets et vont devoir appliquer le protocole TCP dans le contrôle de leur débit (plus précisément dans celui de leur fenêtre). Ils sont reliés au reste du réseau via un canal de sortie. Les paramètres réglables sont leur position graphique (comme tous les autres éléments), le destinataire de leurs paquets (un PC Récepteur), la valeur initiale de *v_seuil*, la taille des paquets émis, et le temps au bout duquel il commence à émettre (émission retardée). Comme tous les autres éléments, il possède son propre fichier pour y écrire les statistiques à chaque fois qu'il est appelé par le programme



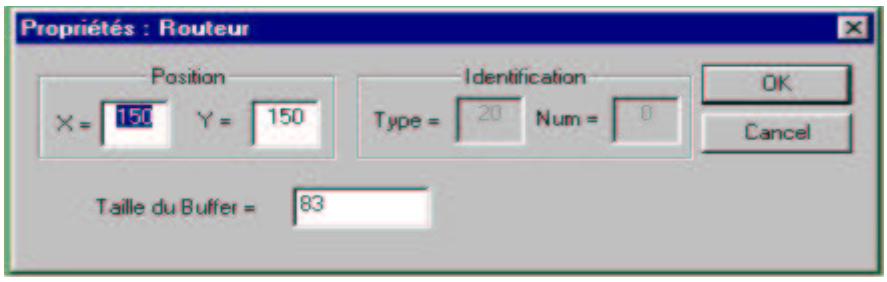
c.Les Canaux

Ils permettent de relier tous les autres éléments entre eux (via bien−sûr des pointeurs). Leurs paramètres réglables sont leur vitesse, et un *« RTT partiel »*, qui permet de constituer une partie du vrai RTT en introduisant un délai dans le transfert du paquet. En fait, ils permettent de remplacer une partie de réseau, ayant une vitesse de transfert apparente égale à celle de ce canal et un délai égal à son *« RTT partiel »*.



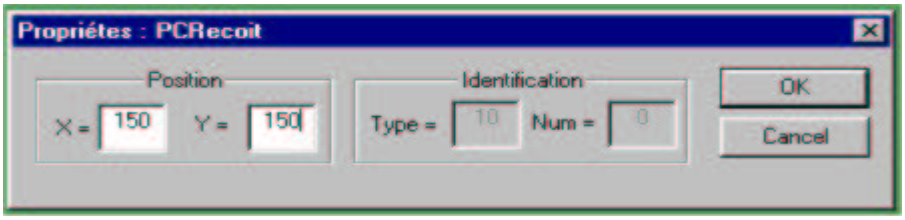
d.Les Routeurs 

Comme un routeur normal, il doive aiguiller les messages vers le destinataire, en disposant d ’un mémoire afin de stocker des messages en attente. Le paramètre à ajuster devient alors la taille de cette mémoire, mais représentée ici sous le nombre de paquets que l’on peut stocker. Comme les paquets auront tous la même taille (ce que nous allons voir), cela est équivalent à fixer la taille totale de la mémoire du routeur. L’autre réglage très important du routeur est la table de routage. Celle−ci est à initialiser juste avant la simulation en rentrant les informations via l’interface ; il convient donc de bien donner les bonnes données…



e.Les PC Récepteurs 

A part leur position graphique, ils n’ont aucun autre paramètre à régler. Comme on va l ’aborder dans la prochaine partie, leur rôle se limite à recevoir des paquets, indiquer le temps d’arrivée et envoyer aussitôt ce paquet à l’émetteur, pas via le réseau, mais directement dans le programme, via un pointeur…



2 Trajet d'un paquet dans ce réseau : schéma de Matthes du simulateur

Nous allons étudier dans cette partie un peu plus précisément le rôle de chaque élément, et les adaptations qui ont été faites pour la programmation.

a.Au départ : un PC Emetteur

Tout commence avec nos PC Emetteurs. Supposons que la fenêtre soit telle que l'on puisse envoyer un nouveau paquet (ce qui est nécessairement le cas au tout début de la simulation) et que l'on n'est pas déjà en train d'en envoyer un. Dans cette situation−là, le PC Emetteur positionne un drapeau à 1 (pour signaler qu'il transmet un paquet) le temps que met la transmission du paquet : (taille du paquet) / (vitesse du canal). Après ce temps−là, il peut donner un pointeur du paquet au Canal auquel il est relié. Dans une liste des paquets envoyés, et sans réponse, il place ce paquet. Il met alors à jour les paramètres de sa fenêtre et, comme des paquets ont pu arriver entre−temps, il commence ou non la transmission d'un nouveau paquet, suivant les valeurs de la fenêtre (la gestion de la fenêtre sera reprise plus loin).

b.Puis dans un Canal

Le Canal vient ainsi de recevoir un paquet d'un PC Emetteur. Or, celui−ci doit introduire un délai dans la propagation de ce paquet, simulant une portion de réseau. Ainsi, il le place dans une salle d'attente (considérée comme infinie), et ne l'enverra à l'élément suivant qu'au bout du temps paramétré par le *RTT partiel*. La salle d'attente est considérée comme infinie, mais on voit tout de suite qu'elle admet un seuil : la vitesse de ce canal limite les arrivées des paquets, qui restent tous le même temps fini dans cette salle d'attente. On peut en déduire le nombre maximum possible de paquets dans cette salle.

c.Dans un routeur

Dans la suite logique, le paquet finit par arriver au routeur, qui le place immédiatement dans sa salle d'attente, de capacité limitée cette fois−ci. Si celle−ci est pleine, le paquet est perdu. Sinon, le routeur traite les paquets un par un, dans une discipline PAPS, et les envoie sur le bon canal grâce à la table de routage. Pour envoyer un paquet sur le canal correspondant, le routeur se comporte de la même manière que le PC Emetteur : il attend le temps d'émission du paquet (suivant la vitesse du canal) pour passer à ce dernier un pointeur sur le paquet. Ceci étant fait, il peut commencer à transmettre le prochain paquet de son buffer, si celui−ci n'est pas vide.

Si tout se passe bien, le paquet continue son chemin sur un canal (dont on connaît le fonctionnement). Il peut alors encore passer par plusieurs routeurs et canaux pour arriver enfin à sa destination : un PC Récepteur.

d.Retour au PC Emetteur et gestion de la fenêtre

Parvenu à sa cible, le paquet est donc traité par un PC Récepteur. Celui−ci lui remplit seulement son temps d'arrivée et le transmet aussitôt (sans délai) au PC Emetteur qui l'a émis. Cela joue le rôle d'accusé de réception pour le PC Emetteur. On néglige alors le retour du paquet via le réseau ; toutefois, ce temps de retour est inclus dans le RTTpartiel du dernier canal. Ce simulateur permet donc bien d'intégrer correctement le retour de l'accusé de réception.

Il reste maintenant à voir la partie la plus importante qui est le traitement de l'accusé de réception en vue de la gestion de la fenêtre.

Le premier travail consiste en le calcul du RTT de la part de ce PC Emetteur. En effet, celui−ci ne peut le mesurer que grâce aux mesures effectuées sur les paquets émis et reçus. Pour le TCP, la formule prise est : $RTT_{estimé}(n+1) = 0{,}9 * RTT_{estimé}(n) + 0{,}1 * RTT_{mesuré}(paquet)$, qui doit converger vers la bonne valeur, sans trop tenir compte des variations ponctuelles des valeurs. On verra dans la suite l 'intérêt de la connaître.

Le PC Emetteur regarde d'abord s'il arrive bien dans l'ordre prévu, grâce à sa liste de messages envoyés et sans réponse.

Si tout est normal, on met à jour la fenêtre selon la phase où l'on se trouve : *slow start* ou *congestion avoidance.* Il enlève le paquet de sa liste. Si aucun paquet n'est en train d'être transmis, et que la fenêtre le permet, il commence alors la transmission d'un nouveau paquet.

Sinon, il se peut qu'il arrive alors que des paquets partis avant ne sont pas encore arrivés : étant donné le fonctionnement de notre simulateur, (un seul chemin possible, et une politique PAPS pour les routeurs) cela implique nécessairement la perte des paquets précédents. Ceux−ci sont alors placés dans la liste des paquets à envoyer, on réinitialise les variables de la fenêtre : *v_seuil=|v_fenetre/2|*, puis *v_fenetre=v_seuil.* Il va ainsi falloir attendre le retour des autres messages sans pouvoir émettre pendant un certain temps..

Dans la réalité, les paquets peuvent prendre des chemins différents, et donc ne pas arriver à destination dans le bon ordre. Cela n'est pas très gênant car les paquets sont numérotés, mais complique la tâche pour savoir si on a perdu des paquets. En fait, le Récepteur renvoie dans l'accusé de réception le plus grand numéro du paquet reçu formant une suite continue : s'il reçoit les paquets 1, 2, 3, 4, 6, 7, 8, il retournera alors les valeurs 0, 1, 2, 3, 4, 4, 4 . C'est justement lorsque la source reçoit trois fois le même numéro que celle−ci considère qu'un paquet a été perdu, et applique ensuite les même règles que ci−dessus.

Si le réseau devient très encombré, il se peut que beaucoup de paquets se perdent, et que le PC Emetteur ne reçoive plus pendant un long moment d'accusés de réception : il doit alors vraiment beaucoup baisser son débit. Ceci correspond à la détection de *timeouts*, qui peut se faire en comparaison du *RTTestimé*. Dans le protocole TCP, on considère qu'il y a un tel *timeout*, lorsque l'on n'a plus reçu d'accusés pendant un intervalle de : 2 * *timeout.* Ceci est logique, dans la mesure où un paquet met approximativement un temps égal au *RTTestimé* pour revenir au PC Emetteur. Ici, notre simulateur se comporte comme dans la réalité. Maintenant qu'il a détecté un timeout, le PC Emetteur réinitialise ses variables de fenêtre : *v_seui l= |v_fenetre/2|*, puis *v_fenetre = 1*. De plus, on considère tous les paquets émis comme perdus : tous les paquets envoyés sont replacés dans la liste des paquets à envoyer, et on repart comme au début avec une phase de *slow−start*.

e.Chiffres Utilisés

Maintenant que l'on connaît le fonctionnement du simulateur, il ne reste plus qu'à le faire fonctionner afin d'étudier ce protocole TCP. Dans la réalité, les valeurs utilisées sont :

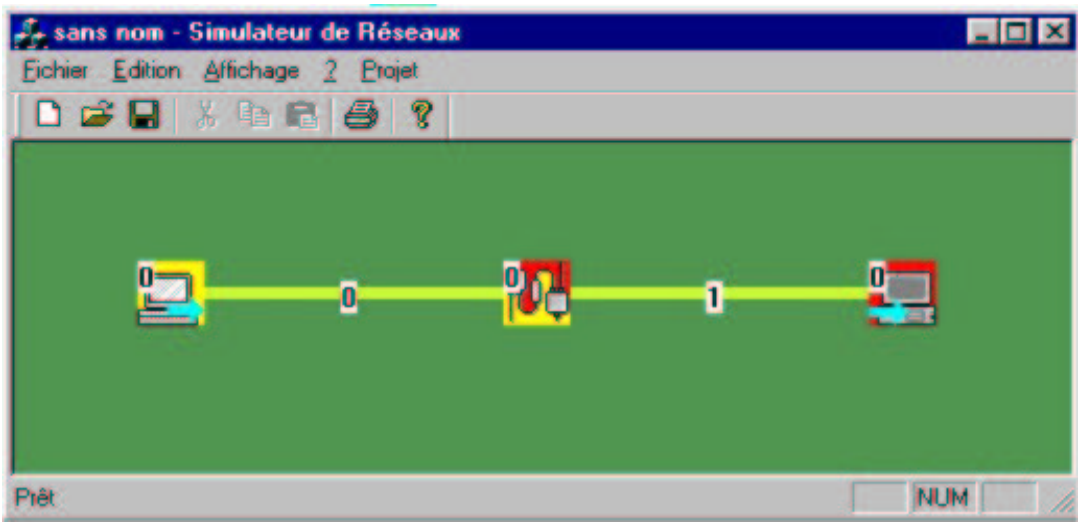
- 1,5 Koctets soit 12 Kbits pour les paquets,
- 100 Mbits/s pour les débits,
- une mémoire de 10 Mbits pour les routeurs, soit une salle d'attente d'environ 833 paquets.

On s'aperçoit tout de suite que cela va faire un très grand nombre de paquets à simuler. Nous avons donc décidé de simuler des groupes de 10 paquets. Nous avons ainsi adapté la mémoire des routeurs. De plus, comme nous voulons étudier l'adaptation de la source en fonction des conditions en aval, en fonction du protocole, le premier canal en amont du PC Récepteur est le plus lent du circuit, avec une vitesse de l'ordre de 10Mbits/s : cela simule une limitation de bande passante en aval due au reste du traffic. On utilise donc les chiffres suivants pour notre simulation :

- 15 Koctets soit 120 Kbits pour les paquets,
- 100 Mbits/s puis environ 10 Mbits/s pour les vitesses des canaux,
- une mémoire de 83 paquets (soit environ 10 Mbits) pour les routeurs

II Le test : la configuration un émetteur−un routeur−un récepteur

On s'intéresse tout particulièrement à la simulation du réseau 1 émetteur−1 routeur−1 récepteur avec les caractéristiques suivantes : paquets de taille 15 ko, 1er lien de 100 MB, 2ème lien de 10 MB, RTT totale 0.5 s, et mémoire tampon du routeur de 83 paquets. On trace les graphiques grâce à Scilab. On constate la présence de trois échelles de temps d'observation distinctes.



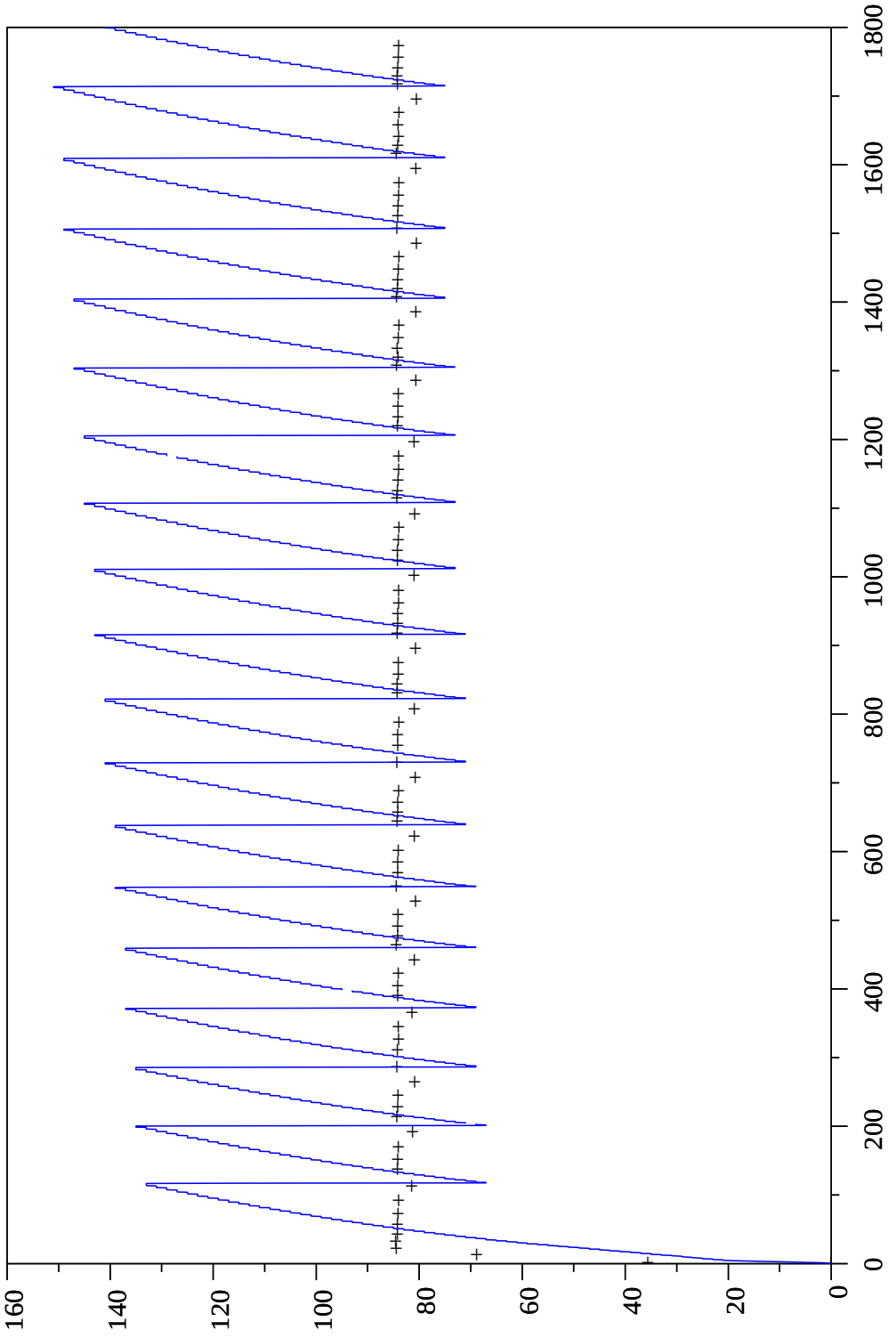
1 L’échelle globale

On réalise la simulation sur 1800 secondes soit une demi−heure. Le volume de données générées est très important. Le fichier des événements de l’émetteur compte 300 000 lignes, il a envoyé 150 000 paquets. Ainsi, même si tracer un graphe de 300 000 points est possible en augmentant la taille de la pile de Scilab à l’aide de la fonction stacksize(), on se contentera de tracer les graphes avec un point sur 50 seulement par la suite. Nous avons comparé les deux graphes pour constater la coincidence.

Après avoir pris une ligne sur 50, on obtient le graphique I sur lequel les lignes bleues représentent le nombre de paquets dans la fenêtre d’émission et les points noir le débit moyen d’émission des paquets (en paquets par seconde). Dans la suite, tous les graphique à cet ordre d’échelle de temps sont réalisé sur ce modèle

On voit sur le graphique I du nombre de paquets dans la fenêtre d’émission deux périodes bien distinctes. D’abord, une montée rapide dans les 100 premières secondes, c’est−à dire un régime transitoire que nous allons étudier particulièrement à l’aide de zooms adaptés. Ensuite, une série de pics à croissance quasi−linéaires suivis par de brusques chutes qui constituent une deuxième phase à analyser.

Enfin, on constate que l’amplitude des oscillations augmente, mais que cela n’a aucune influence sur le débit moyen qui reste au maximum possible, c’est à dire la vitesse du second lien. Cela est certainement dû au fait que la mémoire tampon se vide de plus en plus à chaque diminution de moitié de la taille de la fenêtre, ce qui implique ensuite une montée plus grande, puis une chute encore plus forte. Cet accroissement de la fenêtre maximum se prolonge tant que la mémoire tampon du routeur n’est pas complètement vidée à l’occasion de la diminution par deux de la taille de la fenêtre



Graphique I : Simulation 1 émetteur−1 routeur−1 récepteur de 1800 secondes

2 La phase transitoire

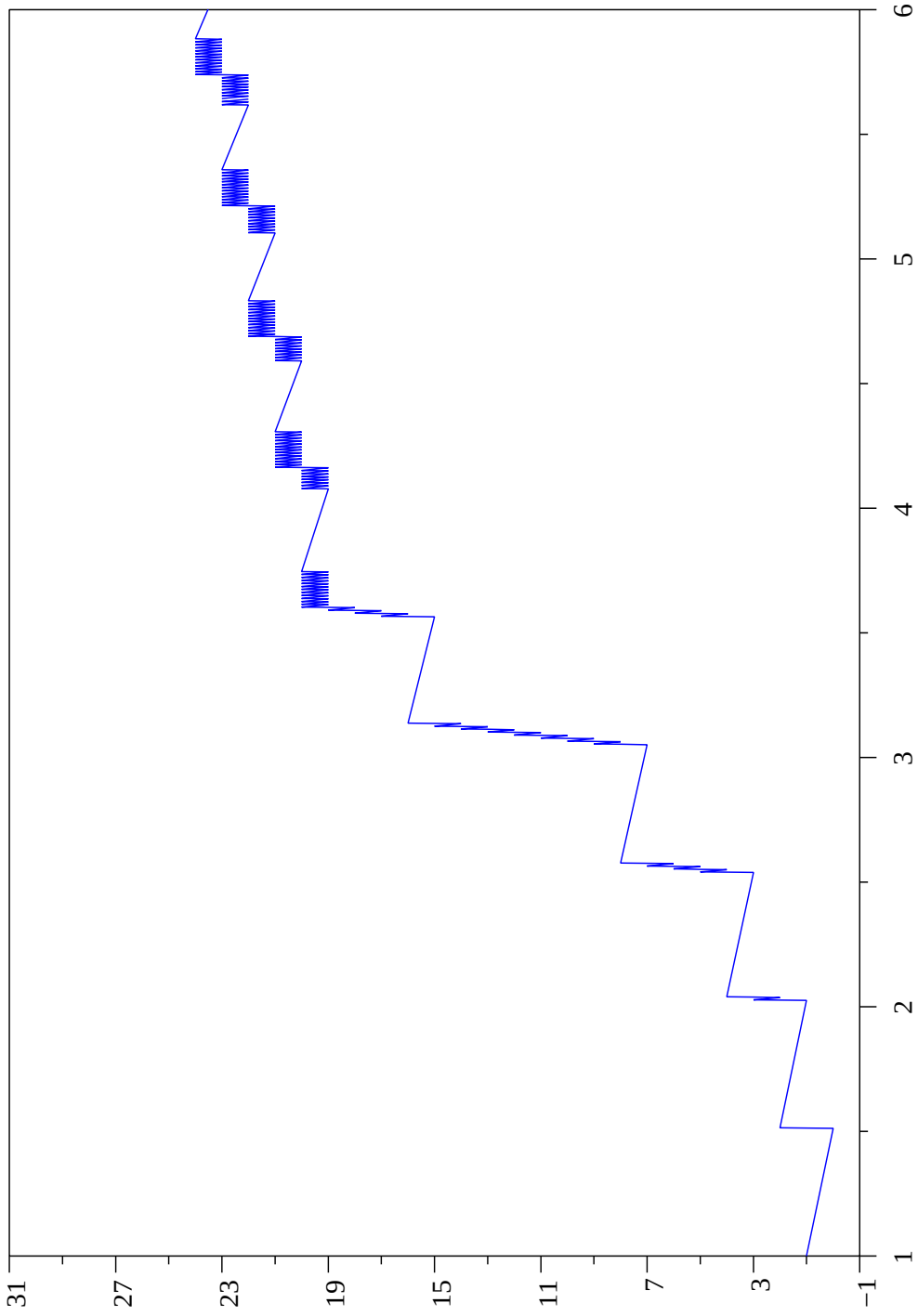
Le graphique II représente les 6 premières secondes de la simulation, soit une partie représentative de la phase transitoire.

On distingue très bien deux phases : d'abord une phase exponentielle jusqu'à 3,5 secondes qui correspond au slow−start du protocole TCP, puis le début de la phase linéaire de ce protocole.

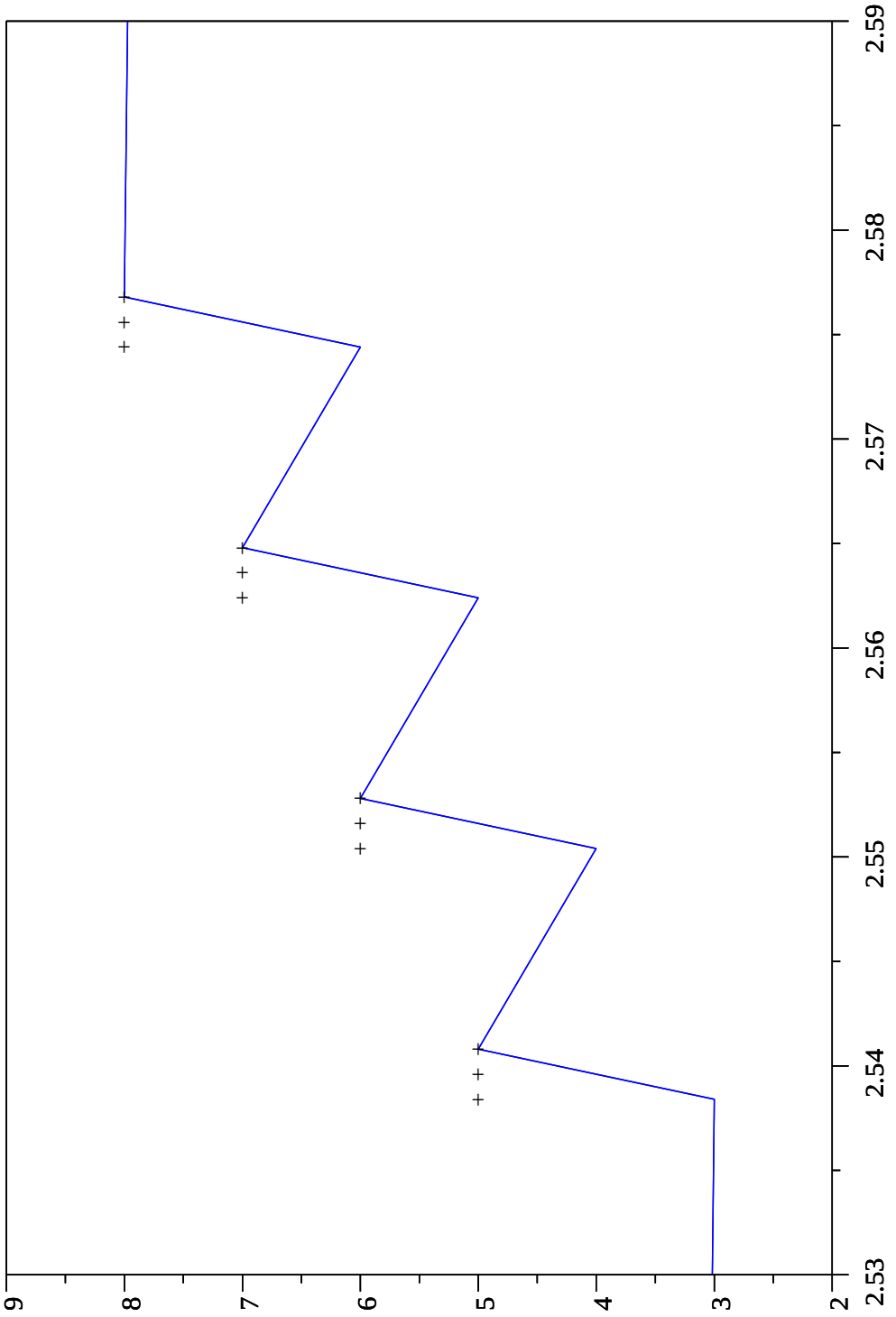
A l'aide d'un nouveau gros plan, regardons le détail au niveau de chaque événement de ce qui ce passe au niveau de l'émetteur. On obtient le graphique III à l'aide d'un gros plan dans la phase exponentielle entre 2,5 et 2,6 secondes. La courbe bleue représente toujours le nombre de paquets présent dans la fenêtre. Nous avons préféré représenter des lignes entre les points correspondant à chaque événement afin de faciliter la compréhension du chemin suivi, car certains événements sont très proches les uns des autres. Les points noirs représentent la taille de la fenêtre telle qu'elle est calculée par l'émetteur. C'est un réel. Le nombre de paquets dans la fenêtre ne peut donc dépasser la partie entière de la taille de la fenêtre.

Voici comment on interprète le graphique III : le 1er événement est l'arrivée d'un accusé de réception d'un paquet, ce qui a pour effet de diminuer d'un le nombre de paquets dans la fenêtre qui passe de 4 à 3, tandis que la taille de la fenêtre autorisée passe elle de 4 à 5. Ainsi, l'émetteur émet 2 paquets successivement afin d'occuper la fenêtre autorisée, ce qui est marqué par la superposition du troisième point et d'un sommet de la courbe bleue. En 2,55 un nouvel accusé de réception arrive, provoquant un processus identique.

C'est le fait d'augmenter de 1 la fenêtre à chaque arrivée d'un accusé de réception qui donne l'allure exponentielle observée sur le graphe II.



Graphique II : Simulation 1 émetteur−1 routeur−1 récepteur, gros plan sur la phase transitoire



Graphique III : Simulation 1 émetteur−1 routeur−1 récepteur, détail de la phase exponentielle initiale

3 La partie linéaire par morceaux

Un gros plan sur l'une des dents de scie observée donne le graphique IV.

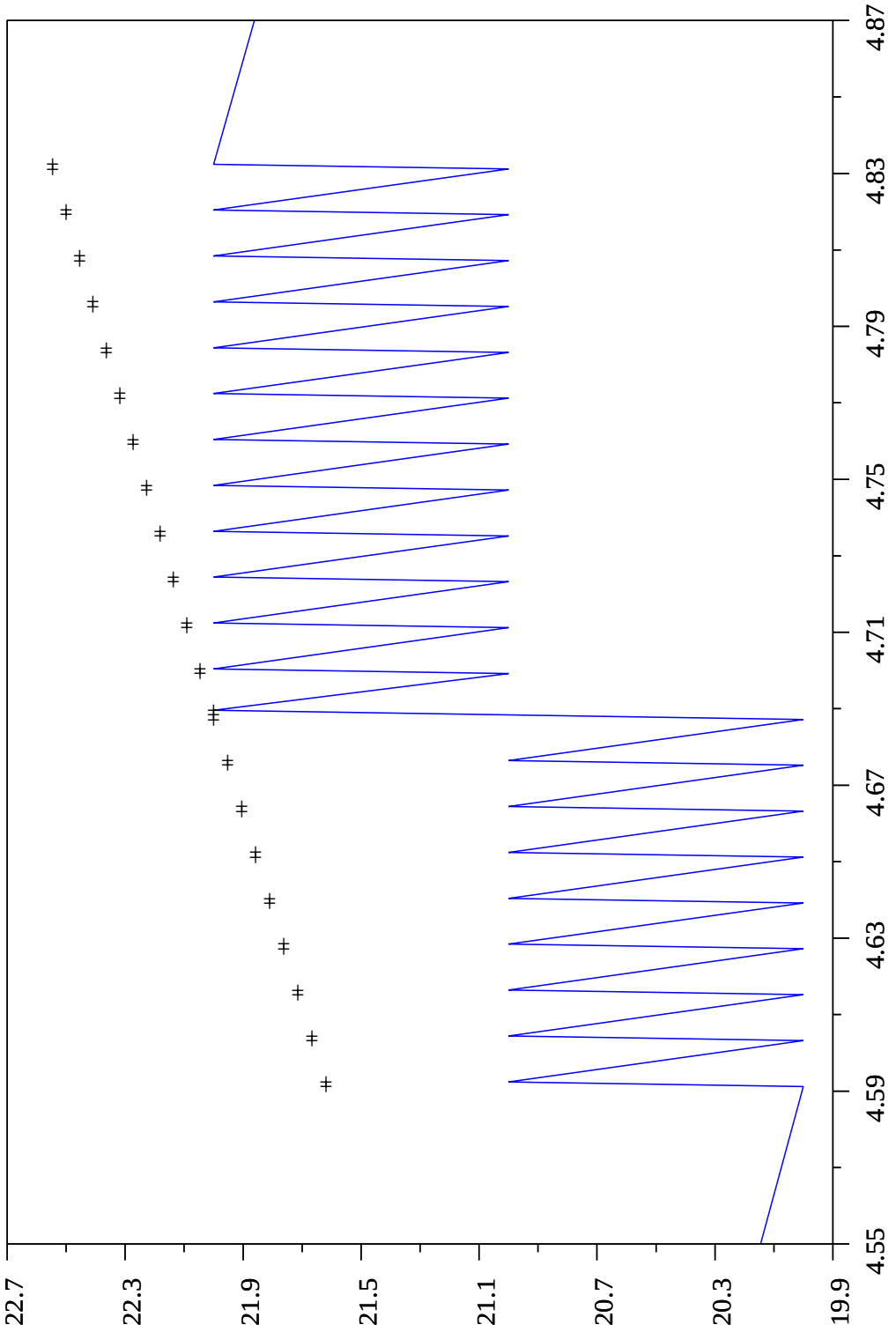
On constate d'abord que cette phase n'est pas strictement linéaire, contrairement à ce qu'on peut imaginer a priori, du fait de l'allongement du RTT lorsque la mémoire tampon du routeur se remplit, c'est−à−dire vers la fin des dents de scie.

Les chutes brutales du nombre de paquets dans la fenêtre correspondent bien à une division de taille par deux, comme prévu par le protocole.

Enfin, on constate la présence de structure plus fine qu'il est nécessaire d'analyser à l'aide d'un gros plan. Le graphique V en le résultat entre les temps 4,55 et 4,87. Cette période correspond à une phase linéaire mais pour lesquels la taille de la fenêtre est plus petite et donc le graphique plus facilement analysable. Les points noirs représentent la taille de fenêtre autorisée, calculée par l'émetteur.

D'abord, on constate que l'accroissement de la fenêtre est bien linéaire, conformément aux spécifications du protocole. On peut analyser ce graphe au niveau de chaque événement. Chaque accusé de réception d'un paquet fait diminuer le nombre de paquet dans la fenêtre d'un, et provoque donc l'émission d'un nouveau paquet, qui provoque à la fin de son émission le retour de la fenêtre à son maximum autorisé.

D'autre part, à chaque accusé de réception, la taille de la fenêtre autorisée augmente linéairement, et lorsqu'elle franchit un nouvel entier, un paquet supplémentaire est émis, ce qui correspond à l'augmentation de 2 paquets du nombre de paquets dans la fenêtre en 4.69.



Graphique V : Simulation 1 émetteur−1 routeur−1 Récepteur, gros plan sur un détail de la partie linéaire

III L’exploitation

Maintenant que l’on a testé le fonctionnement du simulateur sur un réseau simplifié à l’extrême, on l’utilise pour observer l’influence des différents paramètres et des différentes topologies. Tous les graphiques présents dans cette partie sont réalisés après sélection d’une valeur sur 50 des résultats de la simulation. Ils représentent le nombre de paquets dans la fenêtre d’émission (trait bleu) et débit moyen en paquets par seconde (points noirs).

1 L’influence de la taille de la mémoire tampon du routeur

Le graphe VI représente une simulation 1 émetteur−1 routeur−1 récepteur avec les mêmes caractéristiques que dans la partie précédente, sauf que l’on a doublé la mémoire tampon du routeur. On observe les mêmes dents de scie que précédemment mais beaucoup plus espacées, car la mémoire du routeur est deux fois plus longue à remplir. Du point de vue quantitatif, cela se traduit par une « période » d’oscillation qui passe de 100 à 200 secondes. On n’observe plus de lente augmentation de l’amplitude, ce qui traduit le fait que la mémoire tampon est vidée entièrement lors de la division par deux de la taille de la fenêtre.

On constate que le débit final est peu affecté par ce changement de la taille de mémoire tampon.

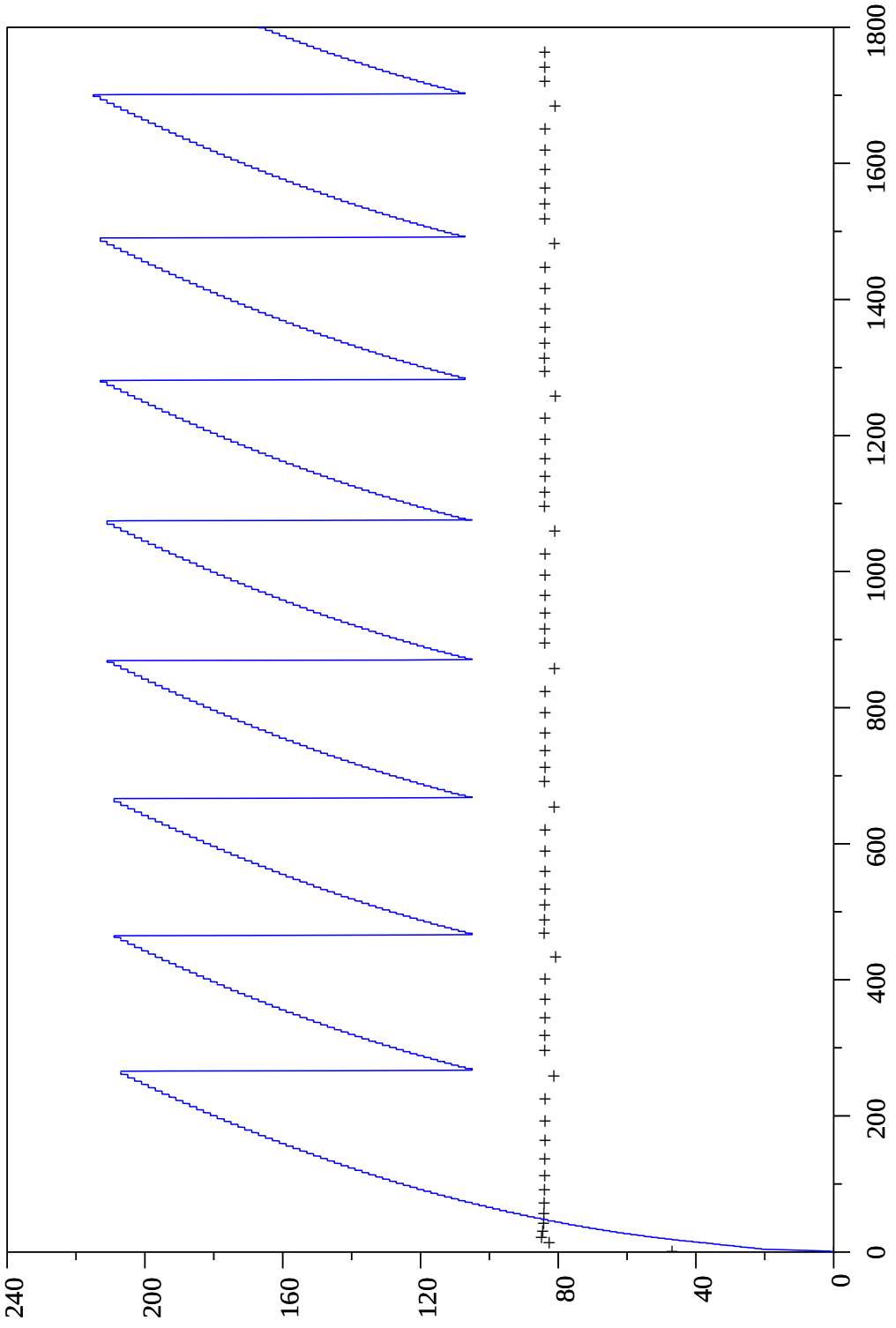
Une simulation en divisant par deux la taille de la mémoire tampon confirme les tendances décrites, à savoir que l’on a dans ce cas une nette augmentation de l’amplitude des oscillation, et une période d’oscillation qui passe à 50 secondes.

2 L’influence du débit du dernier canal

Le graphique VII présente une simulation 1 émetteur−1 routeur−1 récepteur identique à celle détaillée dans la partie II mise à part le débit du dernier lien qui a été divisé par 2. Comme on pouvait s’y attendre, la division par deux de la vitesse du canal limitant conduit à une division par deux du débit d’émission.

L’amplitude de la fenêtre d’émission diminue passant de 75−150 à 55−115, ce à quoi on s’attend puisque le protocole régule le débit de cette manière là. On ne peut s’attendre à une diminution par deux de cette amplitude car le RTT moyen varie entre les deux simulations du fait d’un remplissage différent du buffer.

Enfin, la période des oscillations est inchangée (100 secondes). En effet, une fois que la fenêtre oscille autour de sa valeur moyenne, seuls la taille de la mémoire tampon et le RTT (comme on va le voir dans la partie suivante) font varier cette période : le RTT en déterminant la « pente » de la dent de scie, et la taille de la mémoire tampon la date de la rupture.



Graphique VII: Simulation 1 émetteur - 1 routeur - 1 récepteur avec diminution de la taille du tampon du routeur

Graphique VII: Simulation 1 émetteur - 1 routeur - 1 récepteur avec doublement de la mémoire tampon du routeur

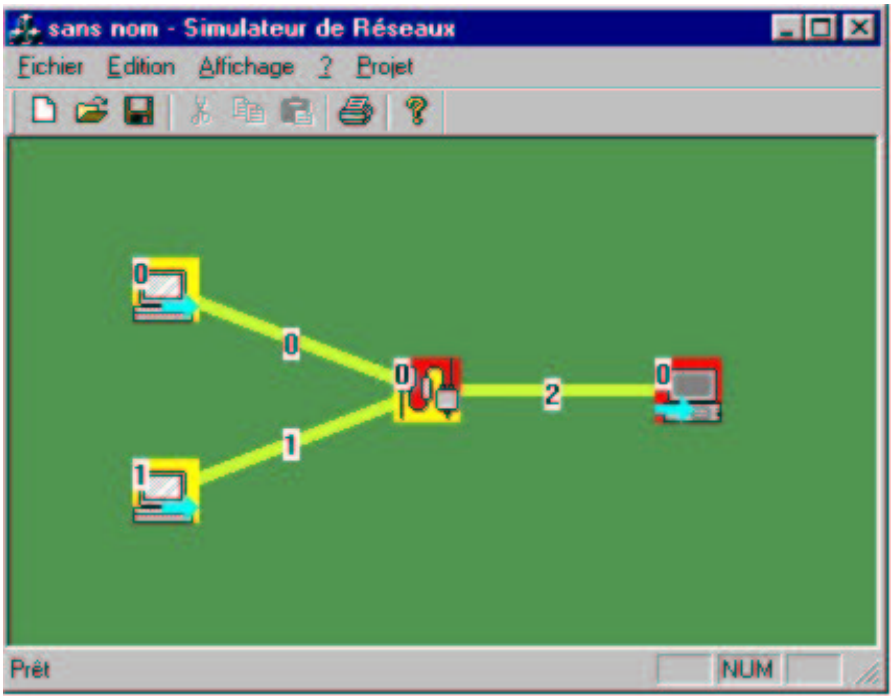
3 L’influence du RTT

a.Cas 1 émetteur−1 routeur−1 récepteur

Dans ce cas, on s’attend à un doublement de la fenêtre moyenne, lorsque l’on double le RTT. On obtient une augmentation mais pas un doublement, le RTT s’obtient en ajoutant le RTT dûs aux canaux avec le RTT dû au routeur, qui dépend de l’occupation moyenne de la mémoire du routeur.

La période des oscillations augmente. En effet, la taille de la fenêtre augmente moins vite, puisqu’elle augmente de 1 tous les RTT en régime stationnaire.

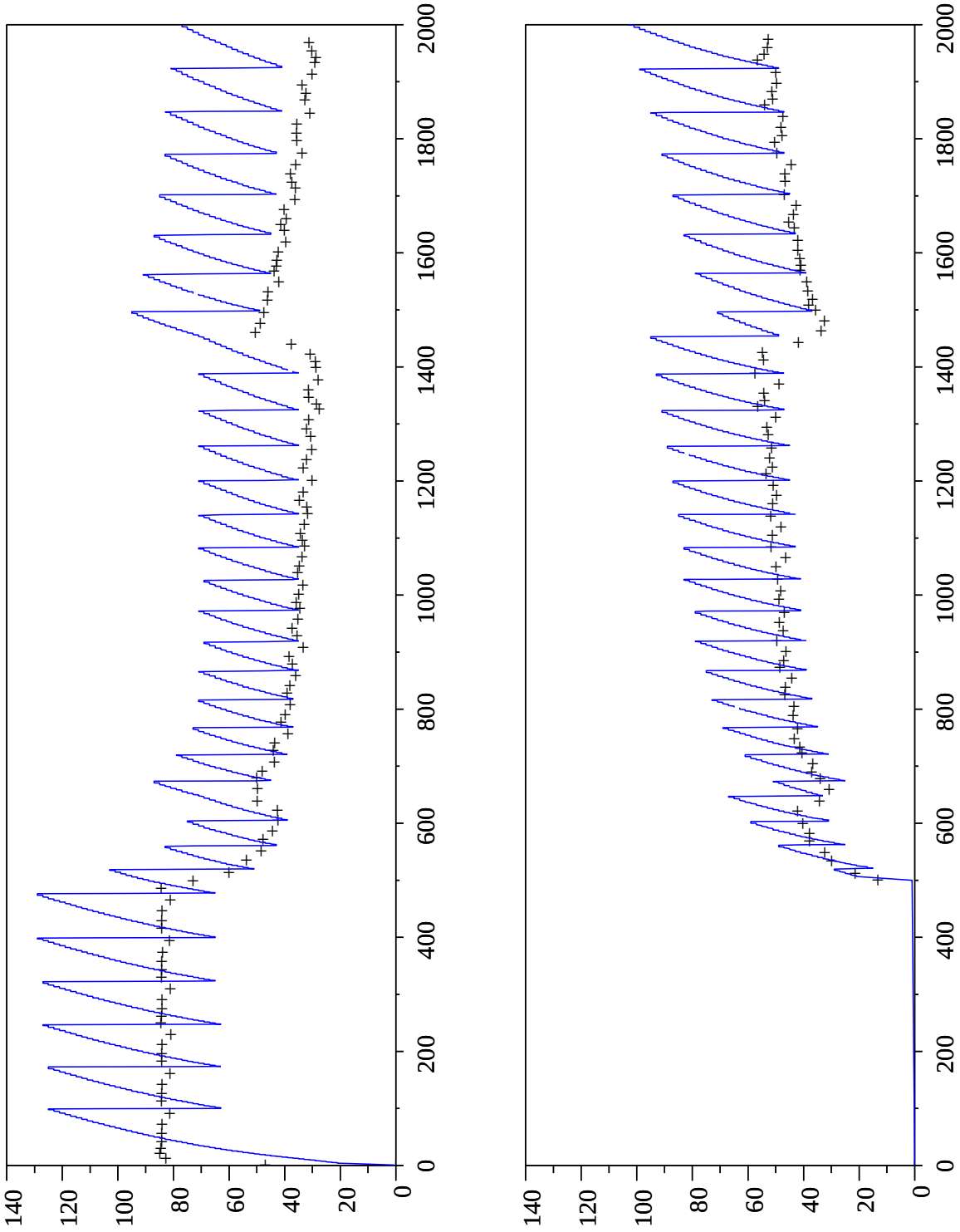
b.Cas 2 émetteurs−1 routeur−1 récepteur

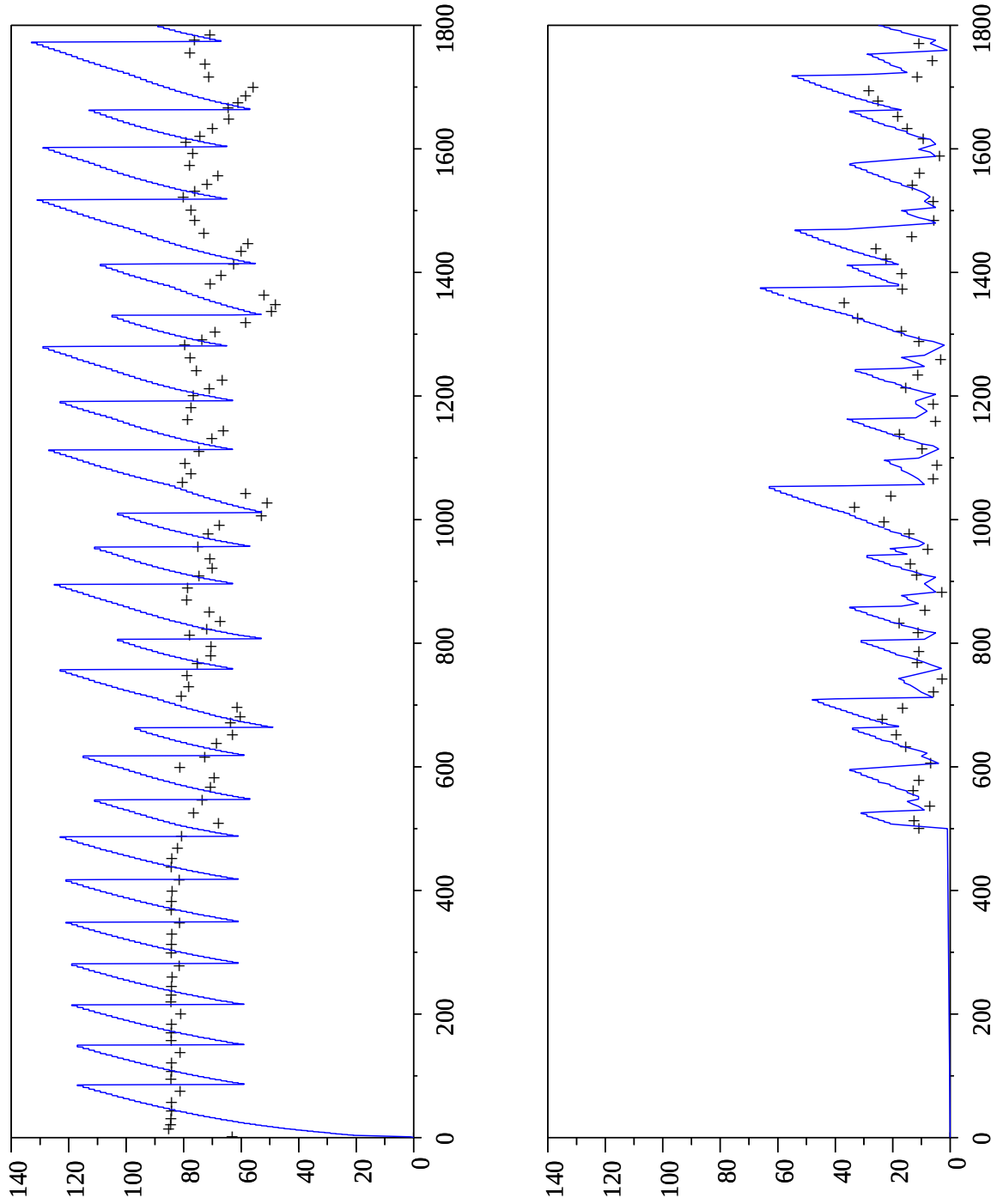


Ce cas permet d’étudier la répartition de la bande passante du canal limitant (le dernier canal après le routeur) entre les deux émetteurs. On étudie en particulier la manière dont un long RTT désavantage l’émetteur. L’émetteur 1 commence à émettre après l’émetteur 0.

Dans le cas symétrique (RTT égale à 0,5 seconde), on obtient le graphique VIII. On observe l’émetteur 0 à gauche et l’émetteur 1 à droite. Le principal constat est que la bande passante est bien partagée en moyenne, mais que de forts écarts sont constatés, par exemple autour de 1400 secondes, où l’écart à la moyenne (83/2 paquets par seconde) est de 15 paquets par seconde, soit plus de35% d’écart à la moyenne.

Dans le cas non symétrique (RTT égales à 0,4 et 0,75 seconde respectivement), on obtient le graphique IX. On voit que l’émetteur 1 est très fortement défavorisé par son RTT élevé. Les variations par rapport au débit moyen sont très grande, l’émetteur 1 se retrouvant parfois avec un débit quasiment nul.


0
20
40
60
80
100
120
140
0
200
400
600
800
1000
1200
1400
1600
1800
2000



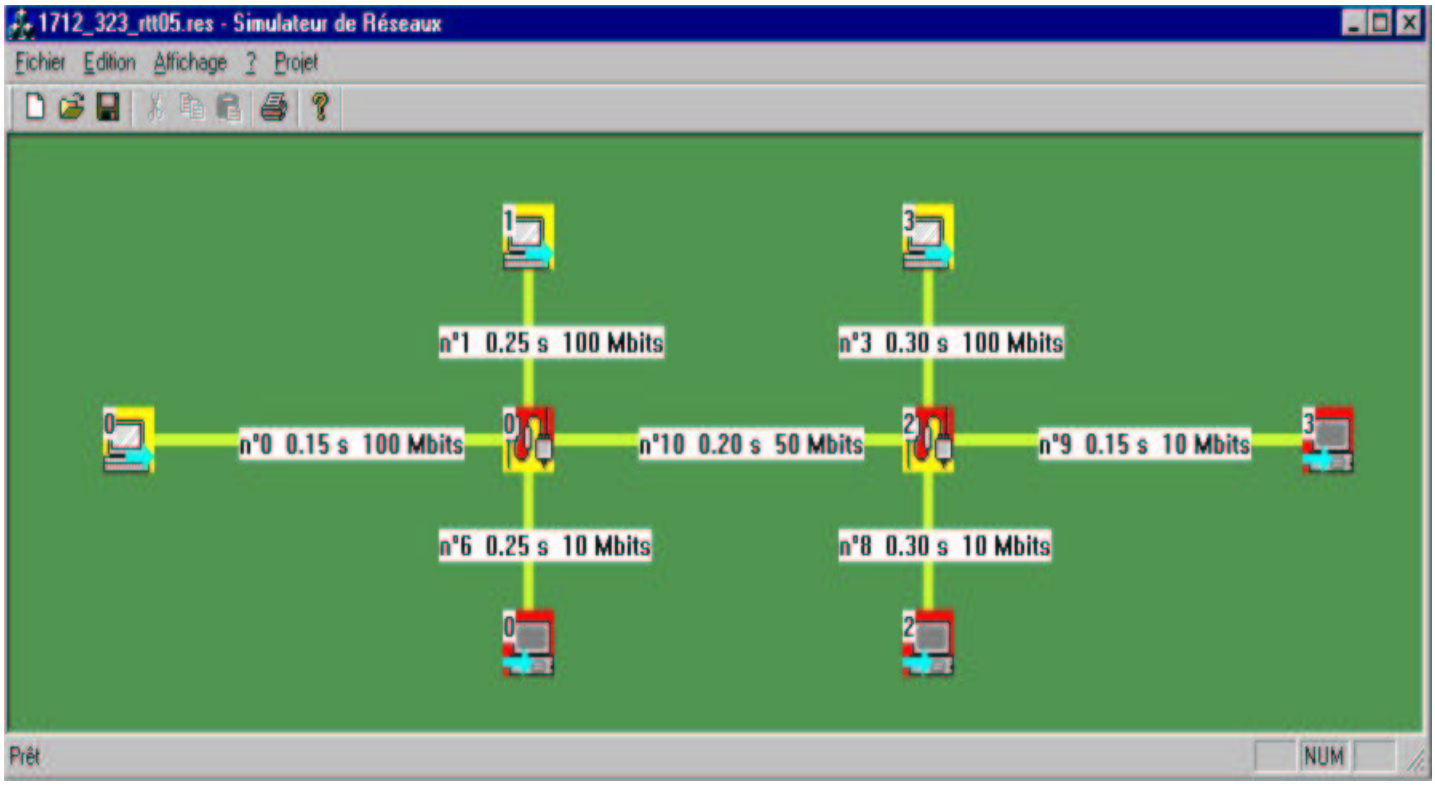
Graphique IX : Simulation asymétrique à deux éléments

c.Cas à 6 émetteurs pour un récepteur

On réalise un réseau à six émetteurs reliés à un récepteur par un routeur. Les RTT associées aux émetteurs 0 à 6 sont les suivantes : 0.5, 0.6, 0.4, 0.5, 0.7, 0.3 (en seconde). Un émetteur entre dans la simulation toutes les 500 secondes. On observe donc les partages successifs que fait le protocole pour un nombre d'émetteurs allant de 1 à 6.

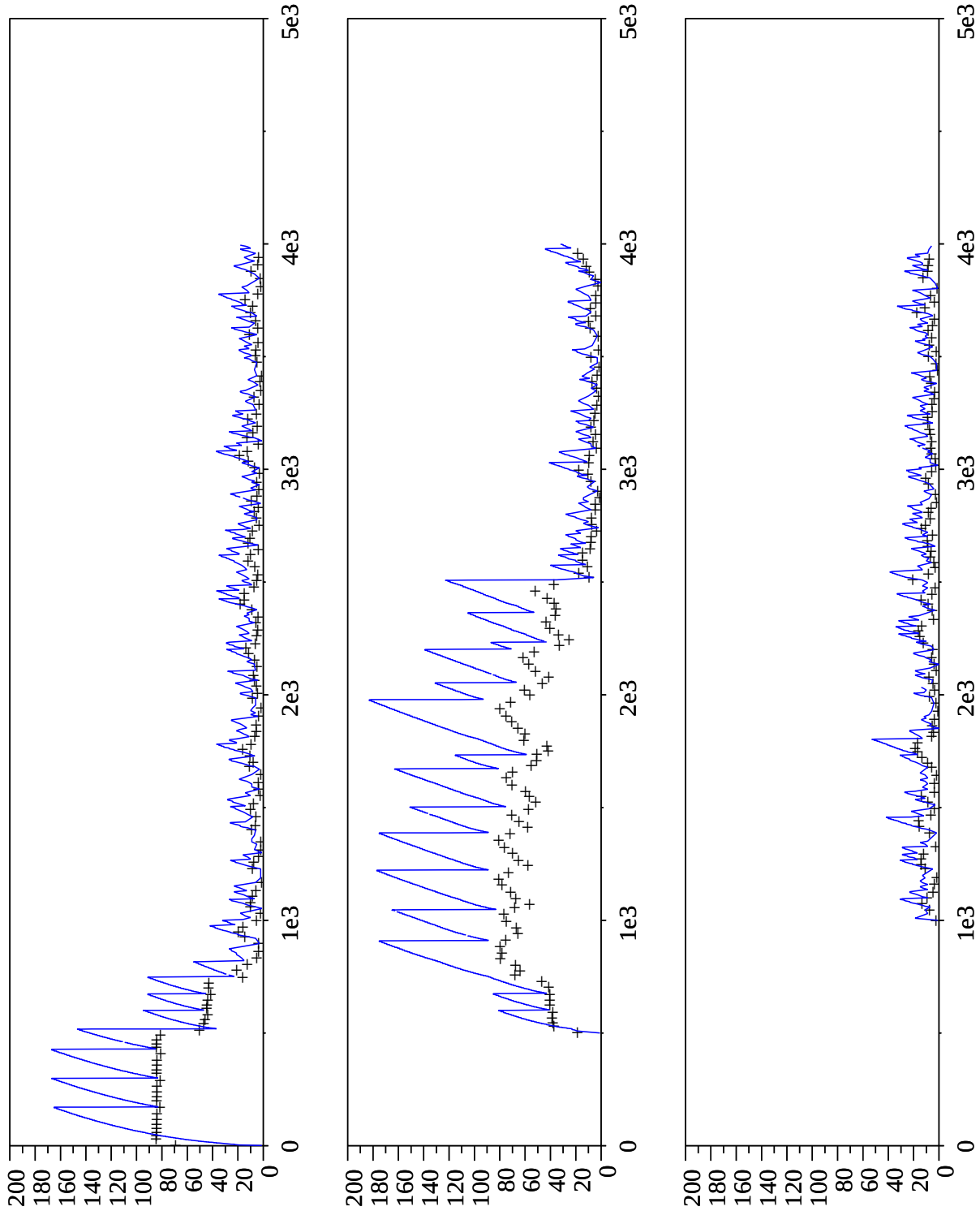
Comme on pouvait s'y attendre après la simulation à 2 émetteurs, les longs RTT sont particulièrement défavorisés dans leur accès à la ressource limitante. Cependant, l'émetteur 1 l'emporte sur les émetteurs 0, 2, 3, et 4 contrairement à ce que la RTT qui lui est associée nous permet de penser. On peut supposer que, la RTT totale étant de l'ordre de 1,7 du fait de la mémoire tampon du routeur, un écart de 0,2 seconde n'est pas le facteur déterminant de la communication..

d.Simulation de trajets longs à travers plusieurs routeurs

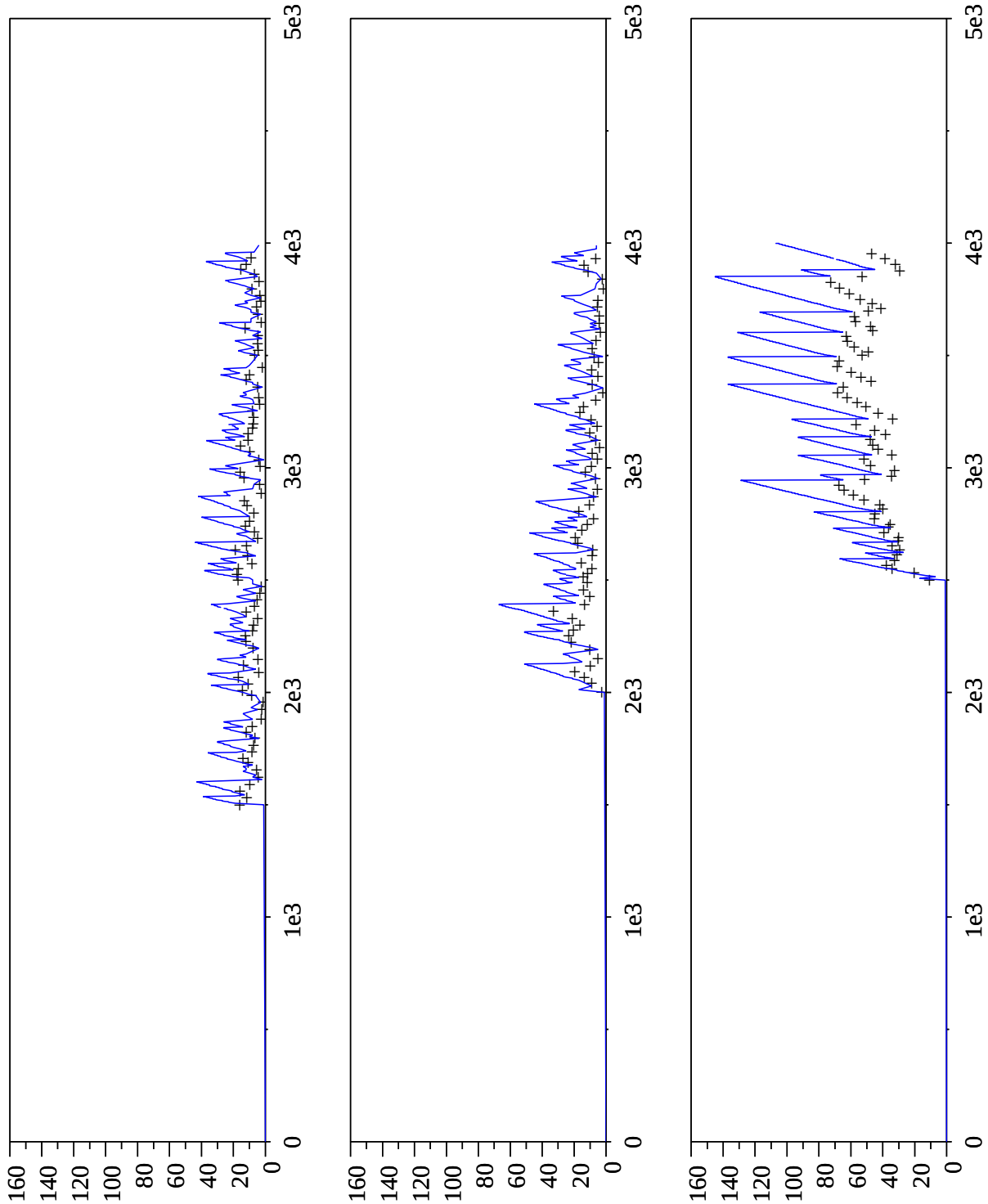


On simule l'envoi de paquet vers une destination lointaine à l'aide du réseau décrit ci−dessus. Le paquet est gêné dans sa course par des flux transverses, d'autant plus que ces flux correspondent à des connections à RTT faible.

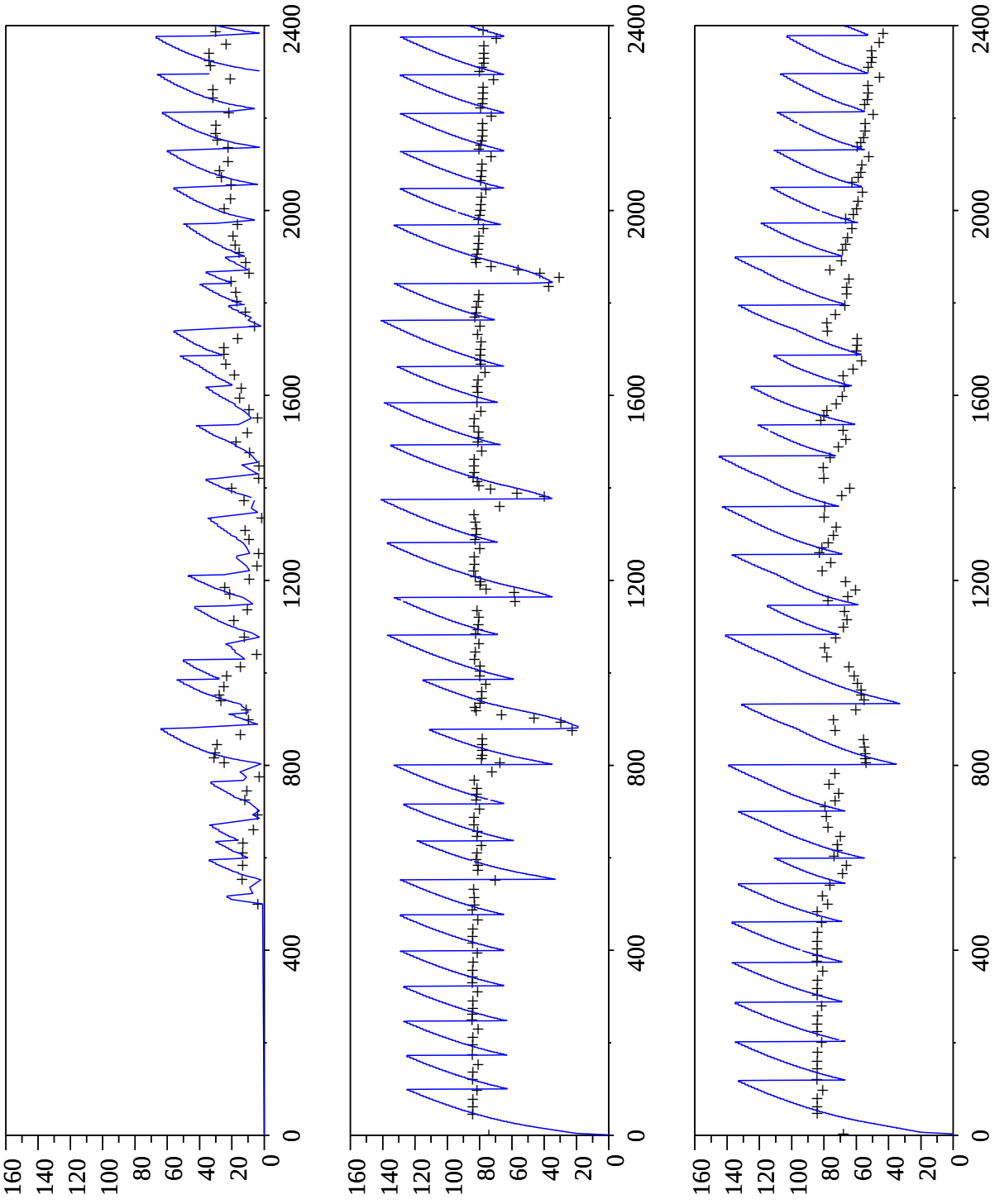
Comme on peut s'y attendre, l'émetteur 0 est particulièrement défavorisé. On remarque que l'émetteur 2 est plus gêné que l'émetteur 1 par l'arrivé de l'émetteur 0. En effet, la bande passante de sortie du 2ème routeur est de 20 MB, ce qui est inférieur aux 60 MB du premier routeur.



Graphique X : Simulation à 6 éléments, émetteurs 0 à 2



Graphique XI : Simulation à 6 éléments, émetteurs 3 à 5



Graphique XII : Simulation de longs parcours

Conclusion

La conception d'un tel simulateur est la base d'une étude plus approfondie des réseaux. Les simulations que nous avons effectuées sont loin de faire le tour des possibilités que nous offre ce type de programme. Néanmoins, nous avons déjà pu mettre en lumière des caractéristiques essentielles de la régulation par le protocole TCP :

- La grande variance du débit autour de sa valeur moyenne sur des périodes de l'ordre de la dizaine de minutes dès qu'il y a deux émetteurs
- Le désavantage important des connections à RTT élevé dans le partage du débit par TCP.

D'autre part, on voit la difficulté de la modélisation du protocole TCP, dans la mesure où plusieurs échelle de temps imbriquées sont représentatives.

Ainsi, ce rapport met−il en lumière plusieurs problèmes importants lors de l'étude expérimentale d'un protocole. Elle pourra nous servir de base dans nos travaux ultérieurs.

Bibliographie

- **C++, Micro Application, collection PC poche**
- **Visual C++ 6, de M. Williams, Campus press, c**
- **An Engineering Approach to Computer Networking, de S. Keshav, Addison−Wesley**