Rapport de stage de L3

Protocoles de transport hautes performances

Equipe RESO Laboratoire de l'Informatique du Parallélisme
ENS Lyon

sous la direction de Laurent Lefèvre et Dino Lopez-Pacheco

Anne-Cécile Orgerie

Juin - juillet 2006

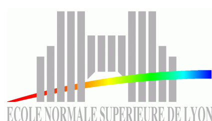


Table des matières

Introduction **2**

1 Les protocoles de transport jusqu'à XCP **2**
1.1 Principe, rôle et insuffisances de TCP 2
1.2 Fonctionnement de XCP 2
1.3 Présentation de XCP-i : XCP inter-opérable 2

2 Le protocole XCP et son adaptation pour arriver à XCP-i **3**
2.1 Le protocole XCP 3
L'en-tête XCP 3
La mise à jour du *feedback* 3
L'importance des ACKs 4
2.2 Adaptation de cet algorithme : l'algorithme de XCP-i 4
Détection d'un nuage non-XCP 4
Détection des routeurs XCP de bordure 5
Évaluation de la bande passante dans un nuage non-XCP 5
Le routeur virtuel XCP-i 6
Adaptation des clients 6

3 Implémentation de XCP-i **6**
3.1 L'option XCP 6
3.2 Définition de l'en-tête 7
3.3 Opérations sur les deux TTL 7
3.4 Estimation de la bande-passante 8
3.5 Le routeur virtuel 8

4 Validation de XCP : tests sur un réseau local **8**
4.1 Conditions expérimentales 9
4.2 Résultats des tests 9

Conclusion **11**
Problèmes rencontrés 11
Ce que je n'ai pas eu le temps de faire 12
Ce que ce stage m'a apporté 12
Remerciements 12

Bibliographie **13**

Introduction

J'ai effectué mon stage à l'ENS de Lyon au LIP dans l'équipe RESO. Il porte sur l'étude et l'implémentation d'un protocole de transport hautes performances basé sur XCP (eXplicit Control Protocol) sur un réseau local.

Les objectifs du stage sont la compréhension de XCP, son déploiement sur une plateforme locale, la réalisation de tests et le commencement d'une implémentation de XCP-i qui est une version de XCP développée par Laurent Lefèvre et Dino Lopez-Pacheco.

Nous allons donc tout d'abord introduire et expliquer le fonctionnement de ce protocole à partir de TCP. Puis, nous regarderons l'implémentation effective de ce protocole en partant d'une implémentation de XCP. Enfin, nous effectuerons des tests de performance pour valider ce protocole.

1 Les protocoles de transport jusqu'à XCP

1.1 Principe, rôle et insuffisances de TCP

TCP (Transport Control Protocol) est un protocole de la couche transport (n°4) du modèle OSI. Il permet une transmission sécurisée par flux entre deux machines. Ses segments ont une entête de 40 octets lui permettant d'assurer différents contrôles de trafic et de sécurité.

Il est utilisé par 90% des applications réseau car il offre une certaine souplesse d'utilisation mais surtout parce qu'il garantit l'émission des données.

TCP se base sur des ACKnowledges "positifs" avec retransmission possible des paquets invalidés et sur une fenêtre glissante : l'émetteur peut envoyer plusieurs paquets avant réception d'un aquittement.

Mais, la théorie et les expériences montrent que l'augmentation du produit *bande passante* × *délai* ne lui est pas favorable. Il devient alors instable (perte continue de paquets), inefficace et non équitable.

Or l'arrivée des réseaux optiques à très grande bande passante et des réseaux satellites à grands délais va entraîner immanquablement de grandes difficultés pour TCP. De plus, les nouvelles versions de TCP pour des réseaux à haut débit n'arrivent pas à acquérir de bonnes performances. C'est pour remédier à cela qu'a été développé XCP.

1.2 Fonctionnement de XCP

XCP (eXplicit Control Protocol) est un protocole de transport proposé par Dina Katabi [2]. XCP représente une approche différente dans le domaine des algorithmes de contrôle de congestion, car il utilise l'assistance des routeurs.

Il fournit une signalisation précise de la congestion. Le réseau annonce en effet à l'émetteur le niveau de la congestion et comment y remédier. De plus, l'émetteur maintient et communique sa *cwnd* (congestion window) et son RTT (round-trip time) via un en-tête de congestion spécial dans chaque paquet.

Cette approche est fondamentale pour les réseaux à grand produit bande passante × délai, où TCP est loin d'obtenir de bons résultats. XCP permet ainsi d'obtenir de hautes performances dans une large gamme d'infrastructure de réseaux.

1.3 Présentation de XCP-i : XCP inter-opérable

XCP nécessite la collaboration de tous les routeurs depuis l'émetteur jusqu'au récepteur. Le déploiement de routeurs spécialisés est donc indispensable pour pouvoir utiliser XCP. Cependant, ce déploiement est très coûteux et pratiquement impossible à réaliser à grande échelle.

De plus, les performances de XCP dans un environnement non-XCP sont beaucoup moins bonnes que celles de TCP (cf [3]). Cela limite donc considérablement l'intérêt de développer XCP sur une portion de réseau.

C'est pourquoi nous allons nous intéresser à une extension de XCP : XCP-i où le 'i' signifie inter-opérable (cf [3] et [4]). Cette version, développée par l'équipe *RESO*, va nous permettre

d'interconnecter des nuages XCP avec des nuages non-XCP tout en gardant l'algorithme précis de calcul de la fenêtre de congestion de XCP.

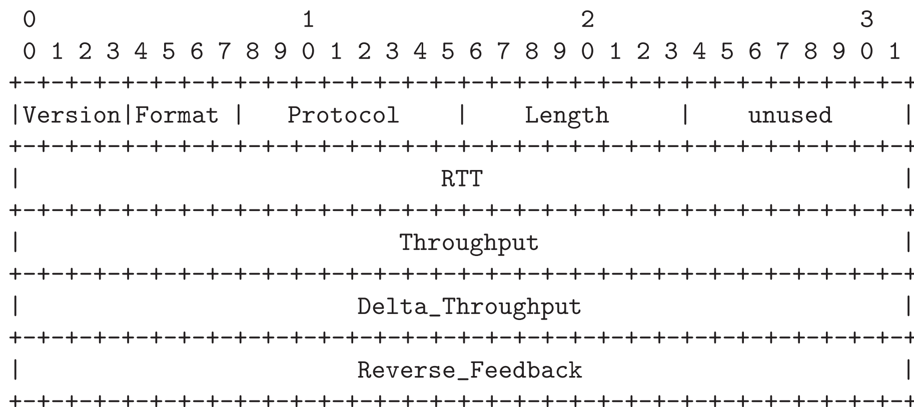
2 Le protocole XCP et son adaptation pour arriver à XCP-i

2.1 Le protocole XCP

XCP utilise l'assistance des routeurs pour informer constamment l'émetteur de l'état de congestion du réseau. Chaque routeur surveille en permanence le trafic entrant, la capacité du lien de sortie et la taille instantannée de la file d'attente. Ces informations vont servir au calcul du *feedback* et sont renvoyées à l'émetteur dans l'ACK.

L'en-tête XCP

Elle est normalement placée entre l'en-tête IP et l'en-tête TCP (cf [1]). La voici :



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|Format |   Protocol    |    Length     |    unused     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              RTT                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Throughput                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Delta_Throughput                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Reverse_Feedback                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

`Version` désigne la version de XCP utilisée. `Format` renseigne sur le format ; il en existe deux possibles : le format standard pour les paquets allant de l'émetteur au récepteur et le format minimal pour les ACKs. `Protocol` indique le protocole utilisé au niveau suivant. `Length` est la longueur de l'en-tête. Puis, on a 8 bits inutilisés.

`RTT` signifie round-trip time, il est mesuré en secondes par l'émetteur et c'est un entier non signé. `Throughput` désigne le temps calculé par l'émetteur entre deux émissions de paquet. `Delta_Throughput` indique de combien l'émetteur voudrait modifier son `Throughput`, cette valeur peut être modifiée par les routeurs traversés. `Reverse_Feedback` est la valeur du `Delta_Throughput` reçue par le récepteur ; le récepteur copie le champ `Delta_Throughput` dans le champ `Reverse_Feedback` du prochain paquet sortant du même échange.

Le *feedback* est donc essentiel et doit constamment être remis à jour pour tenir compte de l'évolution du réseau.

La mise à jour du *feedback*

Ce sont les routeurs XCP qui mettent à jour à chaque étape le *feedback* grâce à un contrôle d'efficacité (Efficiency Controller) qui a pour but de maximiser l'utilisation des ressources et un contrôle d'équité (Fairness Controller) qui a pour but de partager équitablement les ressources.

Concrètement, l'EC cherche à maximiser l'utilisation des liens et à minimiser le taux de perte de paquets. Il sert à déterminer le *feedback*. Le FC, quant à lui, traduit le *feedback* fourni par l'EC pour le mettre dans les en-têtes des paquets.

Ainsi, à la réception de l'ACK, l'émetteur met la taille de sa fenêtre de congestion à jour :

$$cwnd = max(cwnd + feedback, taille\ des\ paquets)$$

Le *feedback* est lui calculé grâce à l'équation suivante :

$$feedback = \alpha \times RTT \times S - \beta \times Q$$

avec :

$\alpha = 0{,}4$ et $\beta = 0{,}226$ (ce sont des constantes déterminées dans [2])

S : la bande passante disponible (spare bandwidth), cette valeur peut être négative en cas de surexploitation du lien

et Q : la taille résiduelle de la file d'attente (persistent queue size)

L'importance des ACKs

En cas de congestion sévère, XCP doit réagir comme TCP. Cependant, la perte d'un ACK peut entraîner une surestimation de la part de l'émetteur de la bande passante disponible et donc une congestion du réseau. On n'avait pas ce problème avec TCP ; ici la perte d'un ACK a des conséquences bien plus importantes.

Avec TCP, l'ACK sert juste à indiquer à l'émetteur la bonne réception des données par le récepteur. Donc, si un ACK se perd, cela va simplement ajouter un délai pour l'émission de nouvelles données chez l'émetteur.

Avec XCP, l'ACK contient en plus la valeur dont doit être modifiée la fenêtre de congestion de l'émetteur. La perte d'un ACK peut donc provoquer une mauvaise perception par l'émetteur des conditions réelles de trafic sur le réseau et donc peut-être une surestimation de la bande-passante disponible ce qui se traduit par une *cwnd* trop grande.

Cela peut également causer l'effet inverse : la *cwnd* peut être plus petite que ce qu'elle devrait être optimalement, ce qui diminuerait le flux de données émises alors que la bande passante disponible en permettrait plus.

Le premier cas est bien plus défavorable que le second. Il peut en effet avoir un impact conséquent sur les réseaux haut-débit en provoquant des congestions sévères. Ce qui implique de nouvelles pertes de paquets, donc des retransmissions coûteuses et des timeout pénalisants.

2.2 Adaptation de cet algorithme : l'algorithme de XCP-i

Nous allons expliquer le fonctionnement de XCP-i à travers l'exemple de la figure 2.1.

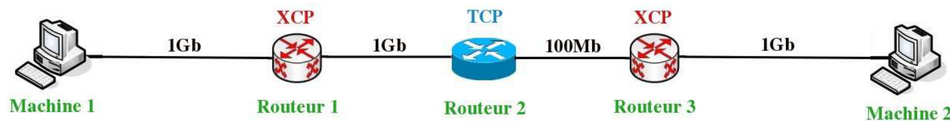


Fig. 2.1 – Exemple

On a deux machines reliées via trois routeurs dont uniquement les deux routeurs des bouts sont des routeurs XCP ; l'autre étant un routeur simple (TCP).

Tous les liens sont à 1Gb sauf le lien entre le routeur du milieu et le routeur XCP de droite.

Détection d'un nuage non-XCP

Le premier problème qui se pose est de détecter les routeurs non-XCP, ici le routeur du milieu. En effet, sinon les routeurs XCP ne pourront pas indiquer à l'émetteur une bonne estimation de la bande passante et les performances seront très mauvaises par rapport à celles de TCP. On va utiliser ici le compteur TTL qui est indiqué dans l'en-tête du paquet IP.

La fonction principale de ce compteur est d'éviter qu'un paquet ne circule à l'infini dans le réseau. Chaque fois qu'un routeur transmet un paquet, il décrémente le compteur TTL de celui-ci.

On va donc rajouter un nouveau champ dans l'en-tête du paquet XCP que l'on va nommer `xcp_ttl_`. Il va fonctionner comme le compteur TTL, mais seuls les routeurs XCP-i vont le décrémenter à chaque passage. Ce compteur sera initialisé à la même valeur que le compteur TTL par l'émetteur.

Ainsi, à la réception d'un paquet, le destinataire regardera ces deux compteurs. S'ils sont identiques, c'est que le paquet n'a traversé que des routeurs XCP ; sinon, cela signifie qu'il a rencontré des routeurs non-XCP.

Et donc quand un routeur reçoit un paquet dont le TTL est plus petit que le `xcp_ttl_`, il sait que le paquet vient de traverser un nuage non-XCP.

Sur notre exemple, si *Machine 1* envoie un paquet à *Machine 2*, supposons que *Machine 1* a initialisé le TTL et le `xcp_ttl_` à la valeur k. Après le premier routeur, TTL vaut $k-1$ et `xcp_ttl_` aussi. Par contre après le second routeur, TTL vaut $k-2$ et `xcp_ttl_` vaut $k-1$. Donc lorsque le paquet arrive au destinataire, après le dernier routeur, TTL vaut $k-3$ et `xcp_ttl_` vaut $k-2$. Le *Routeur 3* sait donc que le paquet a traversé un nuage non-XCP.

Cette solution est simple et ne nécessite quasiment aucun temps de traitement.

Détection des routeurs XCP de bordure

Nous allons également avoir besoin de connaître les routeurs XCP-i de bordure, c'est-à-dire les deux routeurs XCP-i qui encadrent un nuage non-XCP.

Il s'agit donc sur notre exemple du *Routeur 1* et du *Routeur 3*. Nous avons vu que le *Routeur 3* savait que le paquet venait de traverser un nuage non-XCP, donc on sait que c'est un routeur de bordure. Il faut également détecter le dernier routeur XCP-i traversé.

Pour pouvoir découvrir son identité, on va rajouter un autre champ dans l'en-tête du paquet XCP appelé `last_xcp_routeur_` qui contiendra l'adresse IP du dernier routeur XCP-i qui a traité le paquet.

Donc le routeur XCP-i devra mettre son adresse IP dans ce champ avant de retransmettre le paquet. Ainsi, lorsque le *Routeur 3* reçoit le paquet et qu'il voit qu'il a traversé un nuage non-XCP, il peut lire dans le champ `last_xcp_routeur_` l'adresse IP du *Routeur 1*.

Cette solution est elle aussi simple et ne nécessite quasiment aucun temps de traitement. Elle va nous permettre d'évaluer la bande passante dans un nuage non-XCP.

Évaluation de la bande passante dans un nuage non-XCP

Reprenons notre exemple. L'idée ici est que le *Routeur 3* lance une procédure d'estimation de la bande passante disponible entre le *Routeur 1* et lui-même.

Pour cela, le *Routeur 3* va envoyer une requête au *Routeur 1* et attendre pendant `xcp_req_timeout` un accusé de réception de la part du *Routeur 1*. Si celui-ci n'arrive pas à temps, le processus est relancé.

Après trois requêtes sans réponse, le *Routeur 3* conclut à la rupture du lien entre lui et le *Routeur 1*. La procédure d'estimation ne peut alors être relancée que par la réception d'un nouveau paquet de données.

Donc si le *Routeur 1* a bien reçu la requête du *Routeur 3*, il lui envoie un accusé de réception et essaye de déterminer la bande passante entre lui-même et le *Routeur 3*.

Il existe pour cela plusieurs algorithmes, on peut notamment citer *Iperf*, *Pathload* ou encore *Pathchirp*.

Une fois celle-ci connue, le *Routeur 1* doit envoyer la valeur obtenue au *Routeur 3*. Le *Routeur 3* créera alors une entrée dans une table de hachage en utilisant l'adresse IP du *Routeur 1* et y conservera la bande passante calculée précédemment.

La procédure d'estimation de cette bande passante sera réexécutée périodiquement par le *Routeur 1* et elle sera arrêtée après une période d'inactivité du *Routeur 1* et l'entrée correspondante dans la table de hachage du *Routeur 3* sera alors supprimée.

Il est indispensable que ce soit le *Routeur 3* qui stocke la bande passante disponible et non le *Routeur 1* car le *Routeur 1* n'a aucun moyen de distinguer les paquets qui arriveront au *Routeur 3* de ceux qui arriveront à un autre routeur XCP-i après avoir traversé le nuage non-XCP.

Ainsi, sur notre exemple, la capacité du réseau sera estimée proche de 100 Mb et non de 1 Gb comme l'aurait fait l'algorithme de XCP.

Cette solution nécessite la conservation d'un état par routeur XCP-i de bordure, soit un nombre assez faible.

Le routeur virtuel XCP-i

Lorsque le *Routeur 3* reçoit un paquet qui a traversé un nuage non-XCP, il regarde le champ `last_xcp_routeur_` et il regarde s'il existe une entrée dans la table de hachage correspondant à cette IP. Si c'est le cas, il va utiliser un routeur virtuel *Routeur v* pour calculer le *feedback* relatif aux conditions du réseau dans le nuage non-XCP.

Ce routeur virtuel va servir à simuler un routeur XCP-i placé juste avant le *Routeur 3* avec un lien virtuel de sortie relié à ce dernier et avec une capacité égale à la bande passante disponible dans le nuage non-XCP.

Le *Routeur v* peut donc être considéré comme une entité logique figurant le nuage non-XCP. L'équation pour calculer le *feedback* dans le *Routeur v* est la même que celle utilisée par n'importe quel routeur XCP : $feedback = \alpha \times RTT \times Bande\ passante\ estim\acute{e}e - \beta \times Q$.

Le *Routeur v* n'a pas besoin de connaître le trafic entrant car une fois qu'il aura mis le *feedback* à jour, le *Routeur 3* effectuera les calculs nécessaires comme un routeur XCP normal.

Adaptation des clients

Sur notre exemple, si le *Routeur 1* et le *Routeur 2* étaient inversés (le *Routeur 1* serait un routeur TCP au lieu d'être un routeur XCP et inversement pour le *Routeur 2*), l'émetteur, c'est-à-dire la *Machine 1* ne serait plus connectée directement à un routeur XCP.

C'est pourquoi des parties de l'algorithme de XCP-i doivent également être intégrées dans les noeuds terminaux.

En effet, dans ce cas, il faut que la *Machine 1* se comporte comme un routeur XCP. C'est-à-dire qu'elle doit initialiser la procédure d'estimation de la bande passante à la réception de la requête du *Routeur 2*.

De même, si on inverse le *Routeur 2* et le *Routeur 3* dans l'exemple initial, il faut que le récepteur, c'est-à-dire la *Machine 2*, soit capable de détecter le nuage non-XCP placé juste avant lui et qu'il calcule le *feedback*.

Ainsi il va falloir dupliquer des parties de code concernant les routeurs XCP-i dans la pile du protocole XCP des clients.

3 Implémentation de XCP-i

Nous allons nous appuyer sur une version de XCP suivant l'algorithme de Dina Katabi et implémentée par Yongguang Zhang pour Boeing Company en 2004 [6].

Nous nous sommes procuré le code source écrit en C et nous allons le modifier pour rajouter XCP-i. Je propose donc ici l'implémentation de XCP-i que j'ai réalisé dans le cadre du stage.

Le code fourni par Yongguang Zhang se divise en deux parties :
- un patch pour le noyau à appliquer aux clients
- et deux modules noyau à rajouter au serveur.

Nous devons donc modifier les deux parties.

Le patch s'applique sur cinq fichiers du noyau (`include/net/sock.h`, `include/net/tcp.h`, `include/linux/tcp.h`, `net/ipv4/tcp.c`, `net/ipv4/tcp_input.c` et `net/ipv4/tcp_output.c`), et y rajoute un fichier (`net/ipv4/fp16.c`).

Les deux modules à installer dans le noyau des routeurs s'appellent `xcp_core.o` et `sch_xcp.o`.

Chaque modification sera encadrée par `#ifdef XCP_I` et `#endif` ; ainsi l'implémentation sera réversible et claire. On utilisera également `#else` quand on aura à modifier la déclaration même de fonctions déjà existantes.

3.1 L'option XCP

L'en-tête TCP est définie dans le fichier `include/linux/tcp.h`. Yongguang Zhang a implémenté XCP comme une option de TCP. Il déclare donc cette option dans le fichier précédemment nommé :

```
#define TCP_XCP 20 /* XCP mode */
```

Cette façon d'implémenter XCP lui permet de réutiliser toutes les structures existantes de TCP.

3.2 Définition de l'en-tête

Lorsque l'option `TCP_XCP` est activée, les paquets TCP doivent avoir un format d'option spécifique. Les options font partie de l'en-tête, cela revient donc à définir un en-tête spécifique pour XCP. Cette définition se fait dans le fichier `/include/net/tcp.h`.

Le format d'option défini par Y. Zhang (cf [6]) pour les paquets est donné par la figure 3.1.

14	8	H_feedback
H_rtt		H_cwnd

Fig. 3.1 – En-tête d'un paquet

Et celui des acknowledgements est donné par la figure 3.2.

15	4	H_feedback

Fig. 3.2 – En-tête d'un ACK

Le premier nombre est codé sur un octet et correspond au numéro d'option TCP (14 et 15 étaient libres). Le second correspond à la taille de l'option.

J'ai dû modifier le premier format pour ajouter le `xcp_ttl_` et le `last_xcp_router_`, comme on peut le voir dans la figure 3.3.

14	8	H_feedback
H_rtt		H_cwnd
xcp_ttl_		
last_xcp_router_		

Fig. 3.3 – Nouvelle définition de l'option 14

Ces deux nouveaux champs de quatre octets chacun nous obligent à modifier la taille de l'option XCP dans le fichier `/include/net/tcp.h` :

```
#define TCPOLEN_XCP            16
```

Il faut également initialiser ces deux nouveaux champs dans le fichier `/net/ipv4/tcp.c`. Pour cela, on définit `NO_XCP_ROUTER` qui va être l'état initial du `last_xcp_router_` ; il n'y a effectivement pas de routeur XCP précédemment rencontré lorsqu'on est dans l'émetteur. On initialise `xcp_ttl_` à `IPDEFTTL` qui est le TTL par défaut de l'en-tête IP (il vaut 64).

On a également déclaré ces deux nouveaux champs dans le bloc de contrôle de TCP (fichier `/include/net/sock.h`).

3.3 Opérations sur les deux TTL

Le nouveau TTL ne doit pas simplement être décrémenté. En effet, si on décrémente simplement le `xcp_ttl_` à chaque routeur XCP, tous les routeurs XCP situés après un nuage non-XCP auront le comportement d'un routeur de bordure, alors qu'ils ne le sont pas forcément.

J'ai donc choisi de remettre le `xcp_ttl_` à jour dans chaque paquet sortant du routeur XCP en lui affectant la valeur du TTL de l'en-tête IP. Cette opération est effectuée dans la fonction `xcp_update_new_fields` (dans le module `xcp_core`) qui a pour but de mettre à jour les deux champs ajoutés. Elle met donc les deux TTL à égalité et met l'adresse du routeur dans le champ `last_xcp_router_`.

Elle est appelée dans la fonction `xcp_do_departure`. Cette fonction est la dernière concernant XCP appelée par le routeur lors de la retransmission d'un paquet.

3.4 Estimation de la bande-passante

Tout d'abord, il a fallu ajouter une structure de données pratique permettant de stocker des informations relatives aux routeurs XCP situés avant les nuages non-XCP voisins : `close_router`. Puis on fait un tableau de structures : `router_table` de taille `NB_OF_CLOSE_ROUTERS` qu'on a défini auparavant.

```
#ifdef XCP_I
/*
The following structure will define the usefull parameters of a close router.
Then the router_table will contain these close routers.
*/
#define NB_OF_CLOSE_ROUTERS 100

struct close_router {
__u32 address;   //IP address
long estimation;  //value of bandmidth estimation
__u32 birth_time;  //date to which estimation was made in jiffies
};

struct close_router router_table[NB_OF_CLOSE_ROUTERS];

/* Retransmit timer = time we wait before considering that the link is broken (in s)*/
#define XCP_REQ_TIMEOUT 1

/* Time of validity of a bandwidth estimation */
#define VALIDITY_TIME_OF_AN_ESTIMATION 600
```

Le champ `birth_time` va nous permettre de voir si la mesure est encore valide au moment où on en aura besoin, c'est-à-dire dans la fonction `init_bandwidth_estimation`. En effet, avant d'utiliser une mesure, cette fonction vérifie qu'elle n'a pas plus de `VALIDITY_TIME_OF_AN_ESTIMATION`. Si c'est le cas, elle utilise cette mesure ; sinon, elle relance la mesure et écrase la précédente.

Deux fonctions ont été ajoutées au module `xcp_core` pour réaliser l'estimation de la bande passante si les deux TTL sont différents à la réception d'un paquet.

La première, `init_bandwidth_estimation`, est statique et sert à initialiser l'estimation. C'est elle qui va envoyer la requête au routeur désigné par l'adresse IP `last_xcp_router_`. Ensuite, elle attend la réponse pendant `XCP_REQ_TIMEOUT` et relance la requête au maximum trois fois si nécessaire. Et elle finit par mettre à jour la `router_table`.

La seconde, `start_bandwidth_estimation`, effectue l'estimation puis renvoie la réponse dans un ACK particulier qui ne peut être confondu avec un ACK normal par la fonction de parsage du module (`xcp_parse_packet`).

3.5 Le routeur virtuel

La fonction `virtual_xcp_router` se charge de simuler le routeur virtuel. C'est elle qui effectue tous les calculs nécessaires à la mise à jour du *feedback* selon l'algorithme XCP-i (décrit dans [3]) avec l'estimation de bande-passante précédemment trouvée.

Il aurait fallu dupliquer une partie du code pour faire aussi un module noyau pour les clients qui doivent être capables de lancer une procédure d'estimation de bande passante. Mais, il nous a parru plus simple d'installer directement les deux modules noyau destinés aux routeurs dans les clients, cela leur assure exactement les mêmes fonctionnalités que les routeurs.

4 Validation de XCP : tests sur un réseau local

La version de Yongguang Zhang a été réalisée pour des noyaux linux 2.4. Nous avons donc installé ce noyau sur plusieurs machines pour réaliser nos tests.

4.1 Conditions expérimentales

On utilise *Nistnet* pour simuler une congestion dans le réseau. *Nistnet* permet notamment de fixer la bande passante disponible, le temps de latence et le pourcentage de perte entre deux machines. On peut voir sur la figure 4.1 la configuration choisie pour réaliser cette série de tests. On a installé *Nistnet* sur une machine qui se comporte comme un routeur.

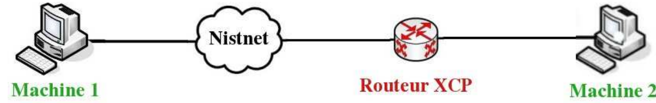


Fig. 4.1 – Exemple

4.2 Résultats des tests

Les résultats des tests de la version de Y. Zhang de XCP que j'ai effectués avec *ettcp* (ttcp adapté à XCP par Y. Zhang) sont résumés par la figure 4.2. J'ai fait varier le pourcentage de perte et le temps de latence.

Test	Délai en ms	Pourcentage de pertes	Débit XCP en Mb/s	Débit TCP en Mb/s	Test	Délai en ms	Pourcentage de pertes	Débit XCP en Mb/s	Débit TCP en Mb/s
1	0	0	9,2	9,26	***25***	0	1	8,87	9,17
2	5	0	9,19	9,26	***26***	5	1	8,75	8,03
3	10	0	9,18	9,26	***27***	10	1	8,28	4,76
4	20	0	9,06	9,24	***28***	20	1	7,46	2,51
5	50	0	4,89	4,9	***29***	50	1	3,9	1,06
6	100	0	2,45	2,42	***30***	100	1	1,87	0,62
7	0	0,1	9,18	9,26	***31***	0	2	8,14	8,59
8	5	0,1	9,15	9,19	***32***	5	2	7,77	6,17
9	10	0,1	9,08	8,85	***33***	10	2	7,63	3,31
10	20	0,1	8,98	6,46	***34***	20	2	6,32	1,69
11	50	0,1	4,76	3,27	***35***	50	2	3,23	0,75
12	100	0,1	2,36	1,29	***36***	100	2	1,65	0,35
13	0	0,2	9,1	9,25	***37***	0	5	5,26	3,99
14	5	0,2	9,09	9,17	***38***	5	5	4,89	2,57
15	10	0,2	9,02	8,32	***39***	10	5	4,68	1,71
16	20	0,2	8,79	5,79	***40***	20	5	3,98	0,91
17	50	0,2	4,54	2,67	***41***	50	5	2,1	0,36
18	100	0,2	2,29	1,07	***42***	100	5	1,18	0,22
19	0	0,5	9,07	9,24	***43***	0	10	0,8	0,7
20	5	0,5	8,91	8,94	***44***	5	10	1,87	0,69
21	10	0,5	8,8	6,7	***45***	10	10	2,26	0,36
22	20	0,5	8,35	3,71	***46***	20	10	1,79	0,35
23	50	0,5	4,27	1,33	***47***	50	10	1,08	0,22
24	100	0,5	2,17	0,89	***48***	100	10	0,74	0,11

xcp plus rapide
identiques
tcp plus rapide

Fig. 4.2 – Résultats des tests

Cependant, ces résultats ne concordaient pas avec ceux fournis par Y. Zhang [6]. Ces tests ont été effectués avec des machines possédant des noyaux 2.4.21 (dans la documentation, Y. Zhang dit avoir travaillé sur des noyaux 2.4.20 mais je n'ai pas réussi à faire fonctionner le patch sur ce noyau). Sur ces noyaux, la taille des buffers d'émission et de réception est limitée.

J'ai donc refait tous les tests après avoir modifié les quatre paramètres qui limitent cette taille en mettant des valeurs très grandes (`/proc/sys/net/core/wmem_max`, `/proc/sys/net/core/rmem_max`, `/proc/sys/net/ipv4/tcp_wmem` et `/proc/sys/net/ipv4/tcp_rmem`). Les résultats sont maintenant meilleurs et plus en accord avec ceux fournis par Y. Zhang [6] (voir la figure 4.3).

Tous les tests ont été effectués sur 60 secondes.

Test	Délai en ms	Pourcentage de pertes	Débit XCP en Mb/s	Débit TCP en Mb/s	Test	Délai en ms	Pourcentage de pertes	Débit XCP en Mb/s	Débit TCP en Mb/s
1	0	0	9,195	9,263	*25*	0	1	8,938	9,111
2	5	0	9,193	9,261	*26*	5	1	8,904	8,093
3	10	0	9,188	9,257	*27*	10	1	8,909	4,829
4	20	0	9,044	9,244	*28*	20	1	8,925	2,647
5	50	0	8,998	9,2	*29*	50	1	8,943	1,186
6	100	0	8,939	9,003	*30*	100	1	8,774	0,603
7	0	0,1	9,154	9,244	*31*	0	2	8,903	8,834
8	5	0,1	9,155	9,238	*32*	5	2	8,877	6,306
9	10	0,1	9,159	9,433	*33*	10	2	8,777	3,459
10	20	0,1	9,031	7,073	*34*	20	2	8,806	1,837
11	50	0,1	8,981	3,417	*35*	50	2	8,645	0,803
12	100	0,1	8,986	2,763	*36*	100	2	8,447	0,459
13	0	0,2	9,086	9,24	*37*	0	5	6,555	4,259
14	5	0,2	9,118	9,189	*38*	5	5	7,901	2,89
15	10	0,2	9,106	8,591	*39*	10	5	8,117	1,569
16	20	0,2	9,074	5,872	*40*	20	5	8,628	0,986
17	50	0,2	9,005	2,32	*41*	50	5	8,07	0,434
18	100	0,2	8,922	1,699	*42*	100	5	8,186	0,229
19	0	0,5	8,969	9,228	*43*	0	10	1,076	0,808
20	5	0,5	9,006	8,89	*44*	5	10	2,43	0,728
21	10	0,5	8,994	6,587	*45*	10	10	6,23	0,372
22	20	0,5	9,018	3,75	*46*	20	10	6,692	0,445
23	50	0,5	8,964	1,556	*47*	50	10	7,256	0,27
24	100	0,5	8,824	0,989	*48*	100	10	7,098	0,12

xcp plus rapide
identiques
tcp plus rapide

FIG. 4.3 – Résultats des tests après modification de la taille des buffers

On peut remarquer que XCP est beaucoup plus performant que TCP au fur et à mesure qu'augmentent le taux de perte et le temps de latence. Cependant, *ettcp* ne nous permet pas de suivre l'évolution du flux.

Nous avons donc aussi effectué des tests avec *tcpdump* (et *tcptrace* et *xplot* pour tracer les graphiques de la figure 4.4).

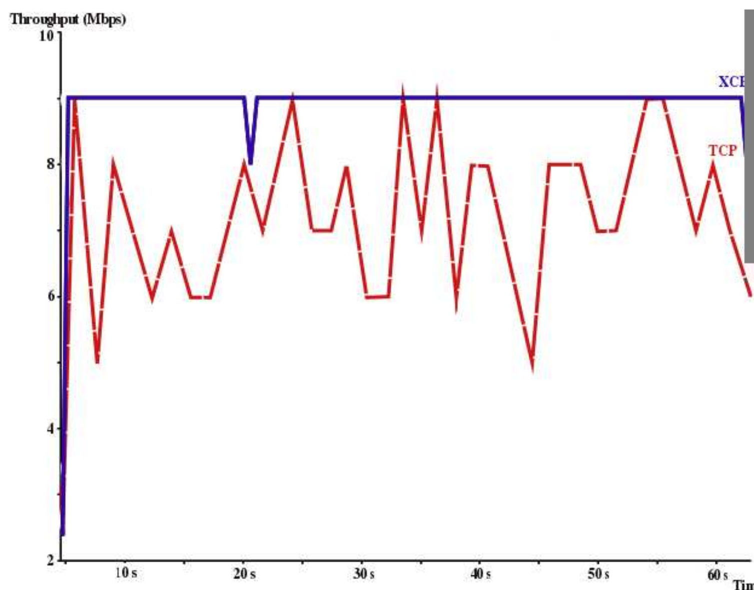


FIG. 4.4 – Graphique du flux avec XCP et avec TCP

Ceux-ci ont été effectués avec une bande passante de 10Mb/s, un délai de 10ms et un taux de perte de 0,2% assurés par *Nistnet*.

On peut constater que XCP a une conduite quasi optimale aux pertes de paquets près contrairement à TCP qui oscille beaucoup. Lors de ce test `Iperf` indique 9,2Mb/s de bande passante disponible.

L'inflexion du flux aux alentours de 20 secondes pour XCP peut s'expliquer par la perte d'un ou plusieurs ACKs, ce qui peut avoir des conséquences dramatiques comme on l'a vu au paragraphe 2.1.3.

Ainsi, en moyenne, sur ces tests, XCP obtient de meilleurs résultats que TCP. Cela tient notamment au fait que TCP augmente très progressivement la taille de la fenêtre de congestion lorsque c'est possible et dépasse le la taille maximale, alors qu'XCP cherche à avoir la taille optimale tout de suite. XCP est donc plus réactif aux variations de bande passante disponible. Cela lui permet de mieux réagir aux forts pourcentages de pertes de paquets que TCP.

Certaines expériences montrent cependant que TCP peut être plus performant que XCP. En effet, XCP nécessite un temps de traitement supplémentaire dans chaque routeur pour chaque paquet par rapport à TCP puisqu'on modifie l'en-tête de chaque paquet à chaque passage dans un routeur. C'est pour cela qu'en l'absence de pertes de paquets ou avec un très faible pourcentage de pertes et un faible délai entre les deux machines TCP obtient de meilleurs résultats.

Ces expériences ont permis de nous assurer que la version de XCP de Y. Zhang fonctionnait correctement, ce qui m'a permis de m'en servir comme point de départ pour l'implémentation de XCP-i.

Conclusion

Problèmes rencontrés

Les problèmes que j'ai rencontré ont été essentiellement d'ordre technique mais ils ont été assez nombreux. En effet, les sources de XCP fournies par Y. Zhang étaient sensées s'installer sans problème sur des machines possédant des noyaux 2.4.20, mais cela n'a pas été le cas. J'ai finallement réussi à tout installer sur des 2.4.21.

Le patch noyau ne fonctionnait pas, il a donc fallu d'abord s'en rendre compte puis modifier les fichiers concernés à la main. Ensuite, les modules noyau concernant le routeur ne compilaient pas avec la version de *gcc* nécessaire à la compilation du noyau (c'est-à-dire la 2.95 puisque ça ne marchait ni avec la 4.0 ni avec la 3.3).

J'ai appris à cette occasion comment compiler et installer un noyau et le rajouter dans le menu *lilo* et le menu *grub*. Les machines utilisées nécessitent en plus l'activation manuelle d'un certain nombre de modules car elles ont des disques SCSI et sont biprocesseurs. La version de XCP du japonais demande en plus l'activation du module noyau `q_disc`. Après quelques Kernel Panic, tout fonctionne.

De plus, il existe sur Internet une autre implémentation de XCP réalisée par P. Mosebekk (voir [7]) qui est elle prévue pour des noyaux 2.6.9. J'avais donc installé celle-ci au début puisque l'autre ne marchait pas, mais le code est beaucoup moins clair et structuré et il est donc moins facile de l'augmenter. J'ai aussi eu quelques soucis d'installation avec cette version.

Quelques machines possédaient des distributions *Red Hat*, ce qui rendait l'installation des paquets dont dépendaient les sources XCP et les différents outils utilisés beaucoup moins pratique que sur les autres machines qui possédaient des *Debian*.

Il y avait également deux machines dont le mot de passe root n'était pas connu, il a donc fallu l'effacer.

Ensuite, les tests ont pris pas mal de temps. Et au début, je les réalisais sans *Nistnet* au milieu ; et dans ces conditions (pas de latence, pas de pertes de paquets, juste avec un cable croisé entre le routeur et le client), TCP est plus performant que XCP. On ne pouvait donc pas savoir si XCP fonctionnait correctement ou non.

Je n'avais jamais installé de noyau auparavant, et là j'ai dû en installer complètement deux et en recompiler pas mal. J'ai donc appris beaucoup de choses très intéressantes et utiles dans ce domaine. Plusieurs stagiaires de l'équipe RESO m'ont aidé.

Ce que je n'ai pas eu le temps de faire

Je n'ai pas eu le temps de tester tous les outils de mesure de bande passante et d'en choisir un. J'ai utilisé `Iperf`, j'ai lu que `pathload` et `pathchirp` étaient assez performants mais je n'ai pas eu le temps de m'en rendre compte par moi-même. J'ai donc laissé deux trous dans mon code avec des "TODO" dans la fonction `init_bandwidth_estimation` qui doit envoyer la requête pour lancer l'estimation de bande passante et dans la fonction `start_bandwidth_estimation` qui doit réaliser l'estimation et renvoyer le résultat.

Une fois le code achevé, il aurait également fallu le tester et voir si cette implémentation de XCP-i est plus performante que TCP avec des nuages non-XCP. Il aurait également été intéressant de faire des tests "grandeur nature" sur la plateforme *Grid5000*.

Ce que ce stage m'a apporté

Ce stage a été très instructif et intéressant. Il m'a permis de me faire une idée plus précise du métier de chercheur. Il m'a également fait découvrir ce qu'est une équipe de recherche avec son organisation interne.

Ce stage m'a fait aborder plusieurs aspects des réseaux : l'installation de logiciels sur des machines, l'implémentation au niveau du noyau, la réalisation de nombreux tests et l'analyse et l'explication des résultats. J'ai également découvert plusieurs logiciels de mesure de performances dans les réseaux (*Iperf*, *ttcp*, *tcpdump*, *tcptrace*, *Nistnet*).

J'ai aussi pu examiner la structure interne d'un protocole de transport, son fonctionnement et son implémentation. Cela permet de se rendre mieux compte des difficultés de mise en oeuvre des mécanismes concernant les transports de données, et également de la complexité de leur implémentation.

Le sujet était très intéressant et cela m'a beaucoup plu de réfléchir dessus.

Remerciements

Je tiens à remercier mon maître de stage Laurent Lefèvre et Dino M. Lopez-Pacheco ; pour leur disponibilité et leur aide. Je tiens également à remercier Jean-Patrick Gelas, Damien Nicolet et Sébastien Soudan pour leur aide technique, notamment pour l'installation de plusieurs noyaux. Je remercie aussi Pascale Primet Vicat-Blanc pour son accueil.

Bibliographie

[1] Y. Pryadkin et D. Katabi A. Falk. Specification for the Explicit Control Protocol (XCP). *draft-falk-xcp-01*, octobre 2005.

[2] M. Handley et C. Rohrs D. Katabi. Internet Congestion Control for Future High Bandwidth-Delay Product Environments. *ACM Sigcomm 2002*, août 2002.

[3] C. Pham et L. Lefèvre D. M. Lopez-Pacheco. XCP-i : eXplicit Control Protocol pour l'interconnexion de réseaux haut-débit hétérogènes. *CFIP 2006*, octobre 2006.

[4] D. M. Lopez-Pacheco et C. Pham. Robust Transport Protocol for Dynamic High-Speed Networks : enhancing the XCP approach. *IEEE MICC-ICON*, 2005.

[5] Gary Wright et Richard Stevens. *TCP/IP Illustrated, Volume 2, chapitres 8, 9, 24, 26 et 28.* Addison-Wesley, 1995.

[6] Yongguang Zhang et Tom Henderson. An Implementation and Experimental Study of the eXplicit Control Protocol(XCP). *IEEE Infocom*, juillet 2004.

[7] Peter Mosebekk. A linux implementation and analysis of the eXplicit Control Protocol (XCP). Master's thesis, University of Oslo, mai 2005.