MPTCP
Performances et optimisation de la sécurité avec un ordonnancement réparti dans les topologies virtualisées OpenFlow

Encadrants : **S. Secci,**
Y. Benchaïb, M. Coudron,

Etudiants : **R. Ly, K. Lam, Q. Dubois, S. Ravier**

Table des matières

1 Cahier des charges

1.1 Objectifs

Les objectifs du projet sont de :

- mesurer les performances de MPTCP sur différentes topologies de réseaux virtuels.
- modifier l'ordonnanceur de MPTCP pour privilégier une répartition équilibrée sur les différents sous-flots.

1.2 Contexte

MPTCP est une extension de TCP qui permet pour une connexion TCP donnée d'utiliser plusieurs chemins pour l'échange de données. La multiplicité des sous-flots a pour but d'améliorer le débit et d'augmenter la résilience de la connexion [1–3].

Les performances de MPTCP ne doivent pas être inférieures à celles de TCP et son l'utilisation ne doit pas diminuer le débit des autres utilisateurs s ur le méme réseau. Les performances de MPTCP dépendent en partie de l'algorithme utilisé pour la répartition des données entre les différents sous-flots ouverts [4]. Pour caractériser les performances de l'ordonnanceur de MPTCP, nous allons le tester dans différents réseaux virtualisés en utilisant dans un premier temps l'algorithme implémenté dans le kernel MPTCP de linux [1].

L'emploi de MPTCP améliorerait la sécurité si les données transitaient de manière équilibrée entre les différents sous-flots, ce qui complexifient les attaques. Le débit global de la connexion serait affecté car les chemins les plus lents vont ralentir le débit des chemins les plus rapides, ce qui, en contre partie, peut s'avérer moins performant qu'une simple connexion TCP. Nous allons modifier l'ordonnanceur afin de garantir la répartition équitable des charges puis analyser l'influence de cette modification sur les performances de MPTCP dans les topologies réseaux utilisées précédemment.

1.3 Méthodes

La réalisation du projet peut être subdivisé en trois parties :

- la simulation de réseaux à topologies différentes,
- l'analyse des performances de MPTCP,
- l'adaptation de l'ordonnanceur pour l'aspect sécurité.

Nous utiliserons Mininet afin de simuler les topologies réseaux où nous pourrons mesurer les performances de MPTCP à l'aide de l'API Python fournie par Mininet. Pour caractériser l'influence de l'ordonnanceur sur les performances, nous utiliserons des réseaux simples où les différents sous-flots sont asymétriques et diffèrent par une propriété à la fois : latence, débit, pertes... Nous testerons différents algorithmes de répartition de charge entre sous-flots : l'algorithme implémenté par défaut (LIA), celui qui satisfait l'optimum

1. `mptcp.org`

de pareto par rapport aux objectifs de MPTCP, ou encore un algorithme de répartition équilibrée de la charge réseau entre les différents sous-flots.

2 Plan de développement

La première partie est de consuitre les topologies virtualisées et de tester les performances de MTPCP en faisant varier les paramètres des sous-flots. La seconde partie est de construire un alogrithme d'ordonnancement répondant à des critères de sécurité.

Les étapes du développement suivront les points suivants :

- Préparation d'une machine mininet avec le noyau MPTCP compilé pour l'ensemble de l'équipe :
- Lecture, compréhension et commentaires du code de MPTCP ;
- Préparation de plusieurs topologies : *fat tree* pour simuler un *data center* et une topologie permettant de tester la concurrence entre MPTCP et TCP ;
- Préparation d'une bibliothèque de tests et de mesures via l'API python ;
- Écriture d'un algorithme d'ordonnancement dans le noyau ;
- Mesures de performance des différents algorithmes.

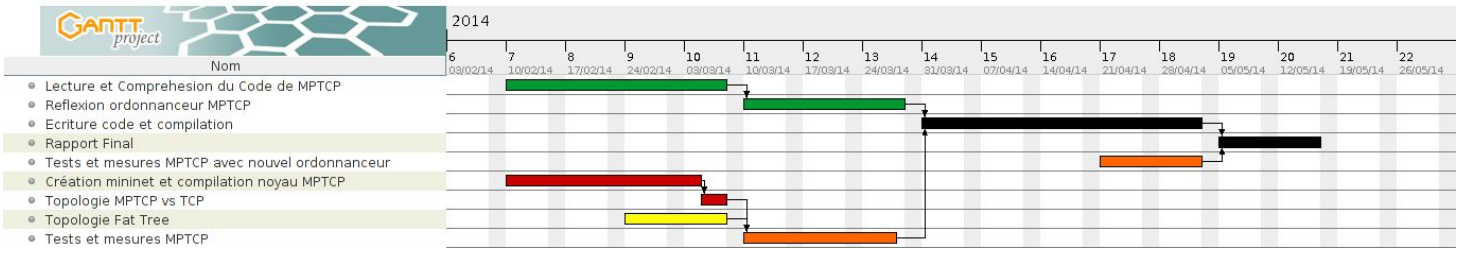

Figure 1 – **Diagramme de Gantt général**. Les couleurs correspondent à la répartition entre les membres de l'équipe : en *rouge* M. Ly, en *jaune* M. Ravier et en *vert* M. Dubois et M. Lam, en *orange* M. Ravier et M. Ly et en *noir* par tout le monde.

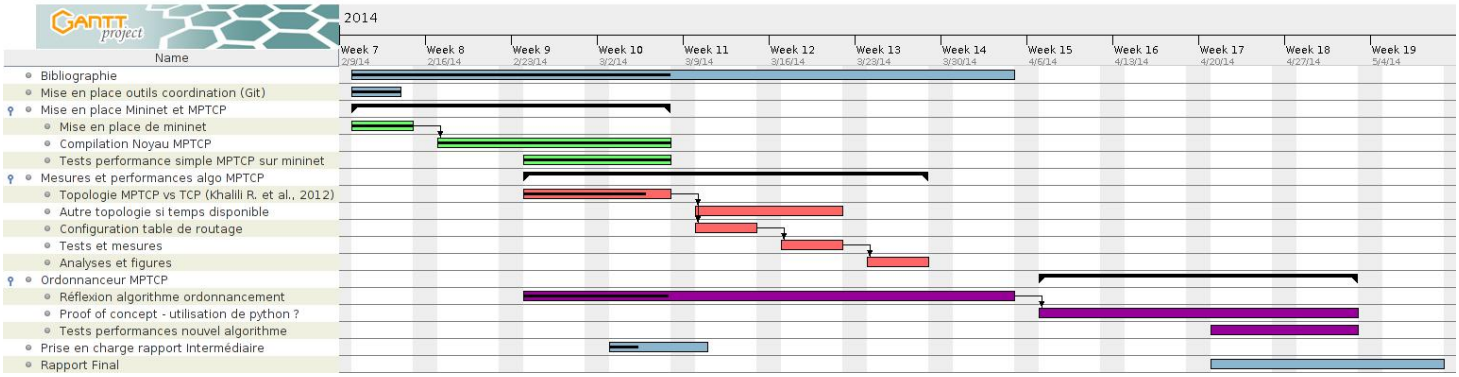

Figure 2 – **Diagramme de Gantt personnalisée Romain Ly.**

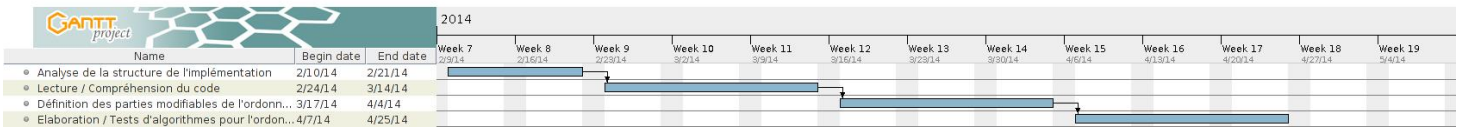

Figure 3 – **Diagramme de Gantt personnalisée Kevin Lam et Quentin Dubois.**

3 Contexte technologique

L'élaboration de MPTCP a été motivée par l'observation de l'existence dans les réseaux de plusieurs chemins entre deux machines A et B. L'utilisation de ces différents chemins entre les deux hôtes pourrait être un atout non négligeable pour augmenter le débit de la connexion et/ou la résilience de la connexion si l'un des chemins venait à ne plus pouvoir acheminer les paquets (congestion, panne de routeur, etc). De plus le multi-chemin permet d'équilibrer la répartition des charges sur les sous-flots utilisés. TCP n'a pas été conçu pour exploiter plusieurs chemins d'où la nécessité de concevoir des protocoles multi-chemins comme MPTCP permettant d'utiliser les chemins disponibles pour transmettre les paquets d'une connexion entre A et B via les sous-flots connectés.

Il existe déjà plusieurs protocoles proposant d'utiliser plusieurs chemins. Nous en citerons que deux : SCTP et ECMP. SCTP (*Stream Control Transmission Protocol*) allie l'avantage de TCP et UDP et permet de multiplexer les flux sur plusieurs interfaces [5]. ECMP (*Equal Cost MultiPath*) est un protocole qui semblait prometteur dans les data center. Lors d'une connexion entre deux hôtes, le routeur peut transférer les paquets sur plusieurs meilleurs chemins à coûts « égaux » [6]. L'inconvénient de SCTP est la nécessité que tous les hôtes terminaux puissent comprendre le protocole ; il est donc nécessaire de modifier la couche application pour pouvoir l'utiliser. ECMP nécessite le travail des routeurs pour connaître les chemins et l'augmentation de performance n'est pas forcément significative. L'avantage de MPTCP est d'être transparent par rapport à TCP, c'est à dire que si un hôte n'est pas compatible avec MPTCP, la connexion retournera vers une connexion TCP classique. L'autre avantage est qu'il est totalement transparent pour les routeurs, c'est une connexion *end to end*.

3.1 Fonctionnement de MPTCP

MPTCP utilise dans un premier temps une connexion TCP pour créer des sous-flots similaire à TCP avec des chemins différents. La couche TCP est alors remplacée par la couche MPTCP qui est divisée en deux parties : la couche supérieure correspond aux fonctions nécessaires à MPTCP de fonctionner (découverte et gestion des chemins, ordonnancement des paquets, contrôle de congestion) et la couche inférieure correspond aux sous-flots établis.

Lorsque MPTCP utilise plusieurs sous-flots (si les chemins différents existent), le débit est augmenté de manière à ce qu'il soit supérieur à une connexion TCP unique sur le meilleur des chemins utlisés. MPTCP augmente aussi la résilience de la connexion, si un sous-flot est congestionné, le traffic est alors réparti et/ou réexpédié sur les autres sous-flots sans qu'il y a it besoins de rétablir une connexion MPTCP entre les deux terminaux. Cette approche permet d'utiliser les ressources disponibles par une approche bout en bout.

Mais pour pouvoir profiter de ces deux avantages, le gestionnaire des chemins est essentiel car c'est en découvrant de multiples chemins, s'il en existe, que la résilience, la résistance aux pannes et le débit sont accrus. Il permet de découvrir de nouveaux sous-flots, les supprimer en cas de panne d'un sous-flot ou si celui-ci est beaucoup trop lent

(car cela ralentirait le débit global de MPTCP).

L'ordonnanceur des paquets, quant à lui, est également utile pour cette augmentation en débit. En effet, c'est la fonction qui permet de gérer les différents buffers des sous-flots, d'organiser le nombre de paquets à envoyer dans chaque sous-flot et de modifier cette quantité, en cas de nécessité. C'est ce qui permet d'utiliser de façon synchrone plusieurs sous-flots TCP.

Enfin, le contrôle de congestion est un outil pour les deux fonctions précédentes. C'est la fonction qui permet d'adapter le débit de chaque sous-flot et de définir si un chemin est trop lent par rapport au meilleur sous-flot, ce qui est une optimisation. Il permet aussi de renvoyer l'information au gestionnaire s'il y a une panne.

Le contrôle de congestion nécessite un algorithme performant pour que l'utilisation de MPTCP à la place de TCP puisse effectivement augmenter le débit de l'utilisateur sans influencer le débit des autres utilisateurs sur les mêmes chemins, c'est à dire qu'il doit garantir l'optimalité de pareto. L'algorithme de MPTCP est donc un point central dans les performances de MPTCP sur le réseau.

Lors des choix des sous-flots, l'algorithme doit effectuer un compromis entre équilibre des charges dans les différents sous-flots et responsivité (*responsiveness*) en cas de modification de la latence des sous-flots ou de découvertes de nouveaux chemins. Une priorité vers l'équilibre des charges entraîne l'envoi des données sur les meilleurs routes (selon la métrique utilisé, par défaut la latence du chemin) mais peut déclencher un changement constant de route produisant un effet de type ping-pong (*flappiness*) : si plusieurs chemins possèdent le même coût, l'algorithme aura tendance à changer très souvent de chemins. Si la priorité utilisée est la responsivité (par exemple augmentation de la taille de la fenêtre d'un des sous-flot), l'utilisation de toutes les ressources disponibles peut ne pas être optimale car on aura tendance à utiliser qu'un seul sous-flot. Les paramètres de l'algorithme doit être déterminer efficacement pour équilibrer les charges sur les sous-flots et ne pas être aggressif (augmentation trop rapide de la taille de la fenêtre sur un des sous-flots) pour garantir l'optimalité de Pareto.

Dans l'algorithme par défaut, le critère priviliégé par l'algorithme est le RTT. Il serait intéressant de modifier les caractéristiques du réseau pour mesurer les performance de MPTCP sur le choix des chemins utilisées ou en cas de modification du chemins sur des critères de latence, pertes, débit ...

3.2 Utilisation réelle de MPTCP

Dans la pratique, l'utilisation de MPTCP est difficile. L'utilisation de plusieurs sous-flots ne garantie pas l'augmentation de débit. Pour celà, il est nécessaire que les sous-flots empruntent des chemins physiques différents et aujourd'hui il n'est pas possible pour un utilisateur de contrôler le routage de ses paquets de bout en bout. Une méthode pour contourner le problème serait d'utiliser la conjonction de MPTCP et de LISP (*Locator/Identifier Separation Protocol*) qui permet de découvrir la diversité de chemins existant entre routeurs de bordures (A-MPTCP) [3].

Cependant il existe des cas où MPTCP est utilisable à son plein potentiel et suscite l'intérêt : dans les data center et en utilisation mobile. Par l'intérmédiaire d'une stratégie de routage par SDN *Software Defined Network* par exemple openFlow, le contrôleur peut établir des chemins différents entre deux hôtes sur tout son réseau. Le transfert de données au sein d'un data center nécessite des débits très importants. L'utilisation de MPTCP pourrait équilibrer les charges entre les différents noeuds. Des expériences sur différents topologies de data center denses ont permis de montrer que MPTCP égale et surpasse même la performance d'un ordonnanceur centralisé et est de surcroit plus robuste [7]. En mobile, le terminal pourra utiliser le réseau 3G/4G et le réseau wi-fi environnant. MPTCP permettra de décharger le réseau téléphonique de l'opérateur tout en augmentant le débit et la résilience de la connexion.

3.3 MPTCP et sécurité

À l'heure d'Eric Snowden, l'utilisation de plusieurs sous-flots pourrait être un avantage non négligeable en terme de sécurité. Pour pouvoir épier une connexion entre A et B, il faudrait à l'attaquant de pouvoir *sniffer* les paquets qui sont émis sur les sous-flots utilisés, c'est à dire sur autant de chemins physiques différents. L'intérêt du multi-chemin prend alors tout son sens. Cependant ce n'est pas le seul avantage.

L'utilisation de chiffrement de type CBC (*Cipher Block Chaining*) compliquera l'attaquant car il est nécessaire d'avoir le bloc n-1 pour déchiffrer le bloc n. AES-CBC est utilisé couramment dans des communications de type HTTPS/SSL. Il sera nécessaire à l'attaquant de disposer de tous les paquets sans exception pour pouvoir déchiffrer le message en supposant qu'il possède la clé adéquate.

De plus, si les protocoles de sécurité sont conscients de l'utilisation de MPTCP, il pourrait y avoir une entente *cross-layer*. Par exemple, en distribuant les informations des MAC (*Message Authentication Code*) de chaque paquet entre les différents sous-flots de manière à éviter les *man in the middle* : sous flot 1 = message 1 + HMAC (message 2) ; sous flot 2 = message 2 + HMAC (message 1). Un autre exemple serait de négocier les clés pour le chiffrement de la communication d'un sous-flot (par exemple en utilisant IPSec) dans le sous-flot adjacent.

Il est donc nécessaire que MPTCP dans une optique sécurité utilise au mininum deux sous-flots. Dans une première approche simpliste, il serait intéressant de forcer l'algorithme de MPTCP à répartir les paquets équitablement sur plusieurs sous-flots, quitte à diminuer les performances de MPTCP.

4 Analyse

5 Conception

5.1 Topologies virtualisées

Nous allons simuler des topologies openFlow en utilisant mininet. Les switchs seront virtualisés par open Vswitch (OVS) qui est installé par défaut dans mininet. Pour utiliser le multi chemin, le noyau de MPTCP sera compilé dans la machine virtuelle et chaque hôte sera configuré de manière adéquate pour pouvoir utiliser MPTCP.

Nous créerons et testerons les topologies virtuelles grâce à l'API python.

5.1.1 Multi-chemins simple

La topologie simple est composé de deux hôtes et de N switchs. Les N switchs composeront les N chemins disponibles. Cette topologie simple servira principalement de test du fonctionnement de MPTCP.

5.1.2 FatTree

Afin de tester MPTCP de manière réaliste, nous avons simulé une topologie FatTree, souvent utilisée dans les Datacenters qui sont les premiers nécessiteux des performances offertes par MPTCP. Cette topologie repose sur le principe d'établir plusieurs liens physiques entre deux équipements réseau, en l'occurrence des switches. Tous les switches du réseau ont le même nombre de ports ; ils sont organisés par couches : une couche « coeur », une couche « frontière » et une couche « hôtes ». Les couches hôtes et coeur sont directement connectées à la couche frontière, mais pas entre elles. Chaque switch de la couche coeur est connecté à chaque switch de la couche frontière par de multiples liens. Le nombre de ports disponibles sur les switches coeurs est équitablement réparti entre chaque switch frontière ; ainsi, avec deux switches coeurs et quatre switches frontières à 36 ports, on disposera de 9 liens entre chaque paire de switches de couches différentes. Le reste des ports disponibles sur les switches frontières sont utilisés pour y connecter les hôtes, à raison d'un lien par hôte. Notons que deux équipement d'une même couche ne sont jamais interconnectés.

5.1.3 MPTCP vs TCP

Pour déterminer les critères de l'ordonnanceur (celui par défaut, ou l'OLIA) à respecter les principes de MPTCP (équitabilité avec les utilisateurs TCP et performances supérieures à TCP), nous allons reproduire le *testbed* utilisé dans l'article de Khalili (Fig. 4).

Si nous pouvons reproduire les résultats obtenus par Khalili avec notre configuration, nous reproduirons le cas avec N1 utilisateurs MPTCP et N2 utilisateurs TCP (voir Fig. 4).

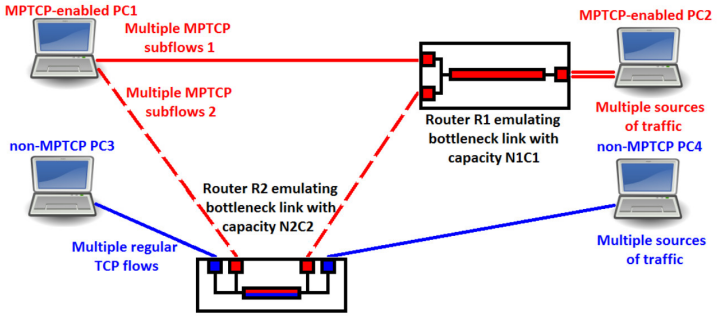

Figure 4 – **Testbed MPTCP vs TCP [4]**.

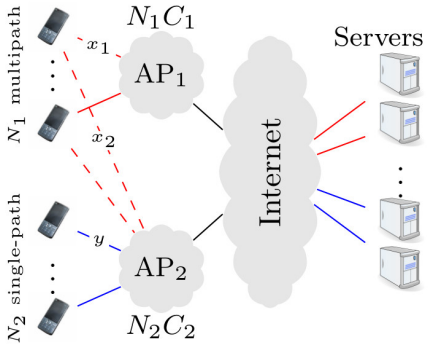

Figure 5 – **Testbed MPTCP vs TCP [4]**. Les N1 utilisateurs MPTCP (rouge) utilisent deux points d'accès pour se connecter à un serveur distant dont un qui est partagé avec les N2 utilisateurs TCP (bleu).

5.2 Performances de MPTCP

Pour mesurer les performances de MPTCP, nous allons faire varier les propriétés de chaque sous-flots empruntés en modifiant les chemins de manière asymétrique. Le but est de créer des conditions de stress qu'on pourra tester à la volée avec les différents algorithmes gérant MPTCP (celui par défaut, l'OLIA et le notre si celui-ci est opérationnel) et sur les différents topologies virtuels construites.

Les contraintes appliquées auront comme critères la latence (critère actuellement priviliégé par l'ordonnanceur pour les choix de sous-flots), la capacité, le taux d'erreur, la gigue, etc. Nous testerons quelle est l'influence de ces paramètres sur le choix des sous-flots par l'ordonnanceur.

5.3 Conception algorithme sécurité

Le but est de rendre une connexion plus sécurisée par la complexité de l'analyse des paquets de données échangés entre deux utilisateurs. Nous chercherons à faire une méthode simple et non performante pour efffectuer des tests et savoir comment MPTCP réagit au

nouvel algorithme de répartition des paquets dans les sous-flots TCP. Cette méthode consiste à prendre le nombre de sous-flots total et de répartir les paquets équitablement entre les sous-flots. Le débit de chaque sous-flot correspondra au débit le plus faible des sous-flots. Cela reste une solution de l'objectif noté dans le cahier des charges.

Néanmoins il sera nécessaire d'avoir un algorithme plus intelligent. En effet, il est nécessaire d'avoir un meilleur algorithme que celui expliqué ci-dessus car le débit serait bien plus faible si un des chemins à un débit beaucoup plus faible ou s'il est congestionné. Or même si nous voulons accroître la sécurité il est préférable d'avoir au moins le débit d'une connexion TCP simple. Bien sûr, la difficulté dans cette partie est de pouvoir adapter l'ordonnanceur selon le nombre de sous-flots disponibles car s'il n'y a pas de chemins différents et que l'on veut sécurisé la connexion, cela n'est pas possible et, s'il n'y en a pas assez ou qui ne sont pas suffisamment corrects, cela aurait des répercussions sur le débit mais il pourrait y avoir une option pour que l'utilisateur fasse lui-même ses propres choix selon ses besoins.

5.4 Test de l'algorithme d'ordonnancement

L'écriture et le test de l'algorithme d'ordonnancement dans le noyau linux peut s'avérer une tâche difficile en si peu de temps. Pour tester la validité de notre algorithme d'ordonnancement, nous réfléchissons à effectuer d'abord un *proof of concept* en utilisant directement python qui utilisera des fonctions de *callback* pour certaines fonctions du noyau nécessaire à MPTCP. On utilisera alors UDP pour la transmission des données.

6 État d'avancement

6.1 Outil de coordination : git

L'état des scripts utilisées par l'équipe est mise à jour par un système de version utilisant git `https://github.com/Romain-Ly/PRES`.

6.2 Compilation MPTCP [2]

La mise en place du noyau linux MPTCP (v0.88) dans une VM de mininet (v2.10) est à 100 % terminé.

Les paquets debian pour l'installation du noyau MTPCP sur une VM de mininet est disponible : (`https://www.dropbox.com/sh/y4ykck8rg6908ps/7V3lsV6Ggg`).

Pour tester la réussite de l'installation, une topologie deux hôtes et n switchs a été utilisé. L'utilisation de MPTCP montre un débit supérieur lorsque l'on compare à la même expérience où MPTCP a été désactivé dans le noyau.

6.3 FatTree [3]

Chargé de la conception du réseau de test, ma première préoccupation a été de maîtriser Mininet. Après documentation, je me suis penché sur l'API Python offert par cet outil. Après cette prise en main concrétisée par quelques tests de connectivité sur des topologies simples, j'ai commencé à coder ma propre topologie, un FatTree à 2 niveaux avec des switches à 36 ports. N'ayant pas trouvé de définition formelle du FatTree, je me suis contenté d'une instance particulière, relativement simple mais permettant tout de même à MPTCP d'emprunter plusieurs sous-flots différents pour se rendre d'un hôte à l'autre. Suite au travail de Romain, la prochaine tâche sera d'installer le noyau MPTCP sur la machine virtuelle Mininet, de le configurer, puis faire en sorte qu'il soit correctement utilisé dans notre réseau FatTree. J'effectuerai ensuite plusieurs tests de performance sur cette topologie, probablement en collaboration avec Romain.

6.4 Topologies virtualisées [4]

J'ai reproduit la topologie où MPTCP est en concurrence avec un flux TCP [4]. Il reste à établir les tables de routage de chaque hôte pour pouvoir tester les performances de MPTCP.

2. par M. Ly
3. par M. Ravier
4. par M. Ly

6.5 Code [5]

Nous avons regardé les fichiers de MPTCP pour avoir une vision globale de l'implémentation dans le noyau linux et essayer de déterminer les fichiers qui concernent l'ordonnancement des sous-flux. Nous avons ensuite essayé de déterminer où nous pouvions modifier le code afin d'adapter l'ordonnanceur aux besoins du projet. Nous avons avancé sur cette phase de compréhension du code mais il nous reste toujours à savoir où nous pouvons modifier le code sans rendre MPTCP non fonctionnel ou non performant. Pour cela, il faudra tester sur des topologies virtuelles simples et comparer les différences de performances. Bien sûr, dans les tests nous ne codons que des ordonnanceurs idiots : ils effectueront uniquement une répartition équitable des sous-flux sachant qu'ils ont tous le même débit.

5. par M. Lam et M. Dubois

Références

[1] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath tcp development," *RFC 6182*, March 2011.

[2] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," *RFC 6824*, January 2013.

[3] M. Coudron, S. Secci, G. Pujolle, P. Raad, and P. Gallard, "Cross-layer cooperation to boost multipath tcp performance in cloud networks," in *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*, pp. 58–66, IEEE, 2013.

[4] R. Khalili, N. Gast, M. Popovic, U. Upadhya, and J.-Y. Le Boudec, "Mptcp is not pareto-optimal : Performance issues and a possible solution," *Networking, IEEE/ACM Transactions on*, vol. 21, no. 5, pp. 1651–1665, 2013.

[5] R. Stewart, "Stream control transmission protocol," *RFC 4960*, September 2007.

[6] D. Thaler, Microsoft, C. Hopps, and N. Technologies, "Multipath issues in unicast and multicast next-hop selection," *RFC 2991*, November 2000.

[7] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, M.ndley, "Data center networking with multipath tcp," in *Proceedings of the 9th ACM SIGCOMM*, pp. 58–66, IEEE, 2010.