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 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 contenant MPTCP pour l'ensemble de l'équipe.
- Lecture et compréhension du code de MPTCP et écriture de commentaires.
- 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 performances des différents algorithmes

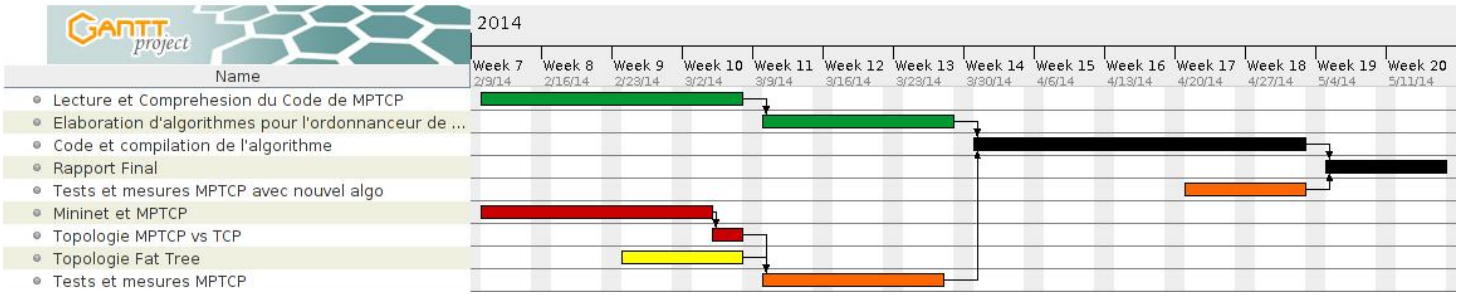

Figure 1 – **Diagramme de Gantt général**. Les couleurs correspondent à la répartition grossière entre les membres de l'équipe : en *rouge* M. Ly, en *jaune* M. Ravier et en *vert* M. Dubois et M. Lam.

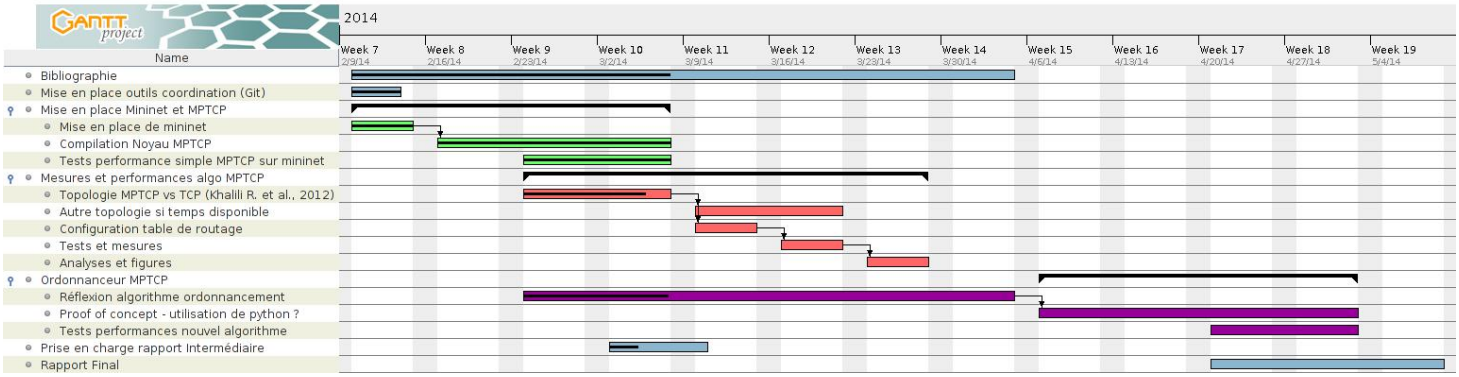

Figure 2 – **Diagramme de Gantt Romain Ly.**

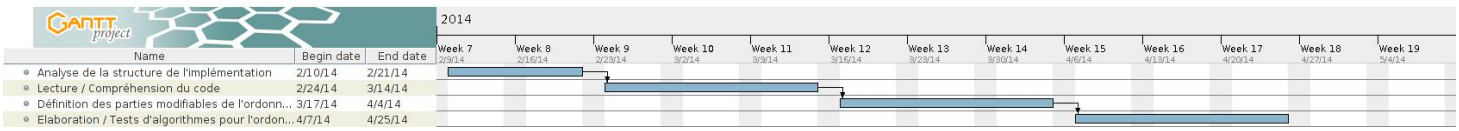

Figure 3 – **Diagramme de Gantt Kevin Lam et Quentin Dubois.**

2 Contexte technologique

L'élaboration de MPTCP a été motivée par le fait que l'on s est aperçu qu il y existait dans l'internet plusieurs chemins entre un utilisateur A et un utilisateur B. Une connexion entre A et B pourrait utiliser ces différents chemins pour augmenter le débit ou la résilience de la connexion si l'un des chemins venait à ne plus pouvoir acheminer les paquets (congestion, panne de routeur, etc.). TCP n'a pas été prévu pour qu'on puisse utiliser plusieurs chemins d'où la création de protocoles permettant d'utiliser les chemins disponibles d'où la création de MPTCP qui permet d'utiliser plusieurs sous-flots (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 [1]. 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 » [2]. 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*.

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 [4]. 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.

Au moment où les révélations de Snowden montrent que les états puissent *sniffer* les paquets au niveau de certains *backbone*. 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 voir les paquets qui sont émis sur deux chemins différents, c'est à dire sur deux chemins physiques différents. L'utilisation de chiffrement de type CBC (*Cipher Block Chaining*) compliquera d'avantage 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.

De plus, si les protocoles de sécurité sont conscients de l'utilisation de MPTCP, il pourrait y avoir une synergie entre la couche transport et la nécessité de sécuriser la connexion. 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). Il est donc nécessaire que MPTCP dans une optique sécurité utilise au mininum deux sous-flots et que les paquets sont plus ou moins répartis équitablement entre les différents sous-flots ce qui implique nécessaire un compromis entre besoin de sécurité et besoin de débit.

Pour pouvoir utiliser une version sécurisée de MPTCP, il s'avère nécessaire nécessaire de créer un algorithme répondant au besoin sécuritaire en compromettant les performancesde MPTCP.

3 Analyse

4 Conception

4.1 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.

4.1.1 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.

4.1.2 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 perfor-

mances 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.

4.1.3 MPTCP vs TCP

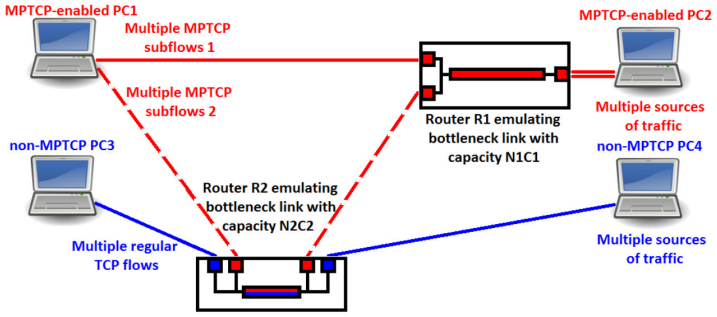

Figure 4 – **Testbed MPTCP vs TCP [5]**.

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).

4.2 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,

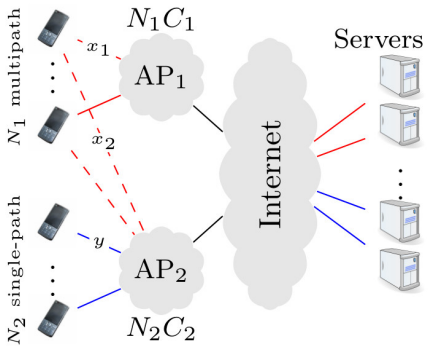

Figure 5 – **Testbed MPTCP vs TCP [5]**. Les N1 utilisateurs MPTCP (rouge) utilisent deux points d'accès pour se connecter à un serveur distant dont un qui est partagé avec les N2 utilisateurs TCP (bleu).

etc. Nous testerons quelle est l'influence de ces paramètres sur le choix des sous-flots par l'ordonnanceur.

4.3 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.

5 État d'avancement

5.1 Outil de coordination : git

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

5.2 Mise en place : mininet et MPTCP [1]

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

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

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

5.3 Topologies virtualisées

J'ai reproduit la topologie où MPTCP est en concurrence avec un flux TCP [5]. Il reste à établir les tables de routage de chaque hôte pour pouvoir tester les performances de MPTCP.

5.4 Code [2]

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.

1. par M. Ly
2. par M. Lam et M. Dubois

Références

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

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

[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] 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.

[5] 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.