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

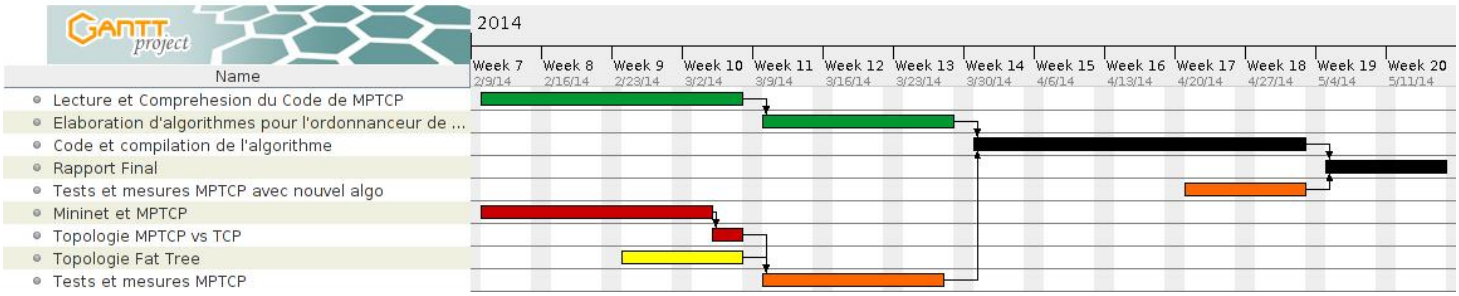

Figure 1 – **Diagramme de Gantt général**. Les couleurs correspondent à la répartition grossière entre les membres de l'équipe : en *rouge* M. Ly, en *jaune* M. Ravier et en *vert* M. Dubois et M. Lam.

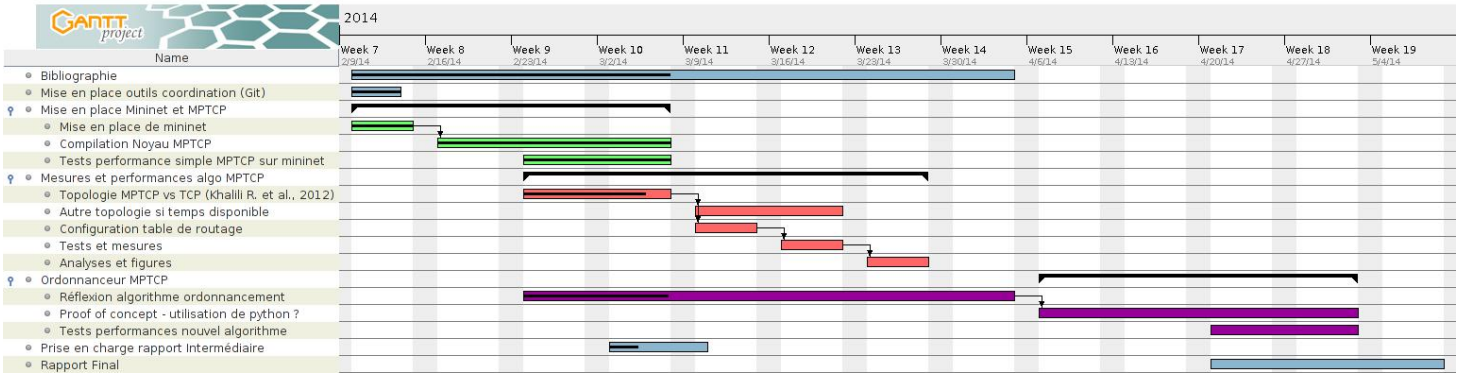

Figure 2 – **Diagramme de Gantt Romain Ly.**

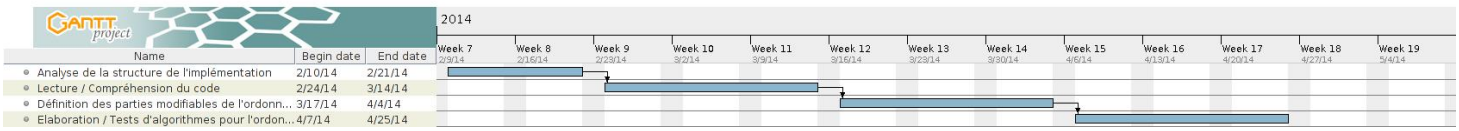

Figure 3 – **Diagramme de Gantt Kevin Lam et Quentin Dubois.**

2 Contexte technologique

3 Analyse

4 Conception

4.1 Topologies virtualisées

L'expérimentation de MPTCP *in situ* peut s'avérer difficile lorsqu'on ne connaît pas ou on n'a pas la main sur les chemins utilisés par les sous-flots.

Mininet permet de créer un réseau virtuel et d'utiliser un vrai noyau (ici celui de MPTCP) dans une seule machine. Le contrôleur du réseau créé permettra d'utiliser au plein potentiel MPTCP.

Le noyau de MPTCP sera compilé et installé dans une machine virtuelle contenant mininet et tournant sur ubuntu. Nous créerons les topologies virtuelles grâce à l'API python et nous effectuerons les tests et les mesures de performance de la même manière.

4.2 Performances MPTCP

Pour pouvoir mesurer les performances, nous allons faire varier les propriétés des chemins empruntés pas les sous-flots de manière asymétrique pour déterminer les performances de MPTCP. Les contraintes appliquées auront comme critères la latence (critère actuellement priviliégé par l'ordonnanceur), la capacité, le taux d'erreur, etc.

4.3 Algorithme d'ordonnancement

L'écriture et le test de l'algorithme d'ordonnancement dans le noyau linux peut s'avérer une tâche difficile en 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 où on qui utilisera des fonctions de *callback* pour lancer les 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 [4]. 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] 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.