Déploiement et Utilisation Transparente de mécanismes protocolaires légers

El-Fadel Bonfoh, Djolo Cédric Tape, Christophe Chassot, Samir Medjiah

Déploiement et Utilisation Transparente de mécanismes protocolaires légers

El-Fadel Bonfoh, D. Cédric Tape, C. Chassot, et S. Medjiah

Université de Toulouse, LAAS-CNRS, F-31031 Toulouse, France

L'adoption de nouveaux mécanismes protocolaires, en particulier de niveau L4, sur Internet reste critique et leur déploiement effectif dans les systèmes d'exploitation (OS) reste très lent. Jusqu'à présent, malgré ses limites avérées et la pléthore d'alternatives existantes telles que QUIC ou DCTCP, près de 90% des applications continuent à tourner sur TCP. Dans le même temps, eBPF, technologie Linux récente, nous ouvre la possibilité de déploiement dynamique et robuste de fonctionnalités dans le noyau des OS. Partant de ces observations, nous introduisons la conception d'un cadriciel basé sur eBPF et capable *(1)* de déployer à la volée des mécanismes protocolaires légers et *(2)* de rediriger de manière transparente les connexions de niveau L4 des applications basées sur TCP vers ces mécanismes dynamiquement déployés. Ce papier présente l'implantation sous Linux 5.0 du premier prototype dudit cadriciel et évalue ses performances et impacts sur la transmission des données de bout en bout.

Mots-clefs : Protocoles de Transport, TCP/UDP, UDP-Lite, eBPF, Déploiement dynamique, Linux OS.

1 Introduction

L'Internet a connu une croissance rapide et continue des mécanismes protocolaires de niveau L4 non seulement pour satisfaire les besoins fonctionnels des applications, mais également pour répondre à leurs besoins en QoS et, plus récemment, en QoE. Malheureusement, le creusé entre la vitesse de développement et celles du déploiement et de l'adoption de ces nouveaux mécanismes est large. Internet ne repose dans les faits que sur TCP et UDP ; près de 90% du trafic se base sur TCP [MKZ+14]. Le déploiement et l'adoption des nouvelles solutions de Transport restent lents et dans bien de cas quasi impossible.

En effet, pour supporter un nouveau protocole, les *développeurs d'OS* doivent l'ajouter dans le noyau de l'OS ; ceci étant fastidieux et propice à erreur et bogue, il ne saurait être motivé que par une forte demande des *programmeurs d'application*. Dans le même temps, pour les *programmeurs d'application*, modifier régulièrement leur application pour supporter un nouveau protocole est chronophage et source d'instabilité, ce qui les conduit à utiliser massivement TCP (jugé stable) malgré ses limites bien connues dans plusieurs contextes [DFDC10]. Cette situation globale conduit au paradoxe de la poule et de l'oeuf et explique le manque d'innovation effective à la couche Transport d'Internet. Vu le nombre élevé d'applications basées sur TCP, nous gageons que la capacité de rediriger les connexions TCP vers d'autres mécanismes protocolaires devrait définitivement accélérer le déploiement et encourager l'adoption de nouveaux protocoles sur Internet. Le protocole vers lequel la redirection est adressée pouvant déjà exister dans l'OS ou faire l'objet d'un déploiement à la volée, en particulier quand le protocole est léger. Ce *déploiement* et cette *redirection* devront être réalisés de manière transparente pour les applications et sans risque de sécurité pour l'OS.

Récemment introduite comme sous-système de l'OS Linux, la technologie eBPF nous ouvre la possibilité d'insérer dynamiquement et depuis l'espace utilisateur des fonctionnalités nouvelles dans le noyau de l'OS. eBPF se repose sur trois composants principaux : *(1)* les *maps*, structure de données index/valeur associée à un fichier système et utilisée pour partager des données entre les programmes tournant dans l'espace utilisateur et ceux s'exécutant dans le noyau de l'OS ; *(2)* les *helpers*, ensemble de fonctions précompilées utilisées par les programmes eBPF pour interagir avec le noyau ; et *(3)* les *tail calls*, primitives mises à usage pour construire une chaîne de programmes eBPF et introduire de la modularité dans l'infrastructure eBPF. De plus, eBPF intègre un vérificateur qui s'assure que le code déployé ne présente aucun risque pour l'OS, ce qui rend robustes les déploiements de mécanismes. Tirant parti des possibilités de la technologie

eBPF, nous démontrons la faisabilité de notre approche en concevant et en implémentant un cadriciel, nommé *Hooker*, capable d'interrompre le flux d'exécution du protocole TCP et de rediriger les données de l'application vers un autre mécanisme protocolaire. Dans ce premier prototype, *Hooker* redirige les données soit vers le protocole natif UDP de l'OS, soit vers le protocole UDP-Lite qu'il déploie à la volée.

Dans le reste de ce papier, nous présentons la conception et l'implantation du cadriciel *Hooker*. Ensuite, nous procédons à ses tests fonctionnels et évaluons ses performances. Et enfin, nous discutons des perspectives de nos travaux.

2 Conception et Implémentation du Cadriciel *Hooker*

Cette section présente la structure interne du cadriciel *Hooker* et ses interactions avec la pile protocolaire de l'OS et les applications. Comme illustré à la Fig. 1, Le *Hooker* est constitué de trois principaux éléments logiciels : *hooker_userspace* s'exécutant dans l'espace utilisateur de l'OS, et *msg_redirector* et *hooker_ingress* qui sont des programmes eBPF tournant dans l'espace noyau de l'OS. Pour rappel, l'objet du *Hooker* est *(1)* de rediriger de manière transparente les connexions TCP vers un autre mécanisme protocolaire et *(2)* au besoin, de déployer à la volée ce mécanisme s'il n'est pas déjà sur l'OS. À cette fin, le *Hooker* va se comporter de manière différenciée sur les paquets entrants et sortants en plaçant à différents niveaux du système plusieurs *hooks points* et en faisant usage de plusieurs types de programmes eBPF. En effet, chaque programme eBPF a un type spécifique et se doit être attaché à un *hook point* aussi connu comme événement noyau (par exemple la réception d'un paquet, l'ouverture d'un socket, l'invocation d'un appel système, etc.). Le type du programme eBPF est particulièrement important pour le vérificateur eBPF enfin de valider l'ensemble des *helpers* que le programme utilise.

Sur un hôte exécutant l'OS Linux, le *Hooker* s'attache au `cgroupv2 root` [CGR]; ainsi, en prenant avantage du modèle hiérarchique des cgroups, il est en mesure de traiter tous les paquets entrants et sortants de tous les processus sur l'hôte. *Hooker* crée et maintient une *map* de type `SOCKMAP` dont la clef est un quadruplet d'informations d'adressage (les adresses IP et les numéros de port source et destination). Cette clef est utilisée par le composant *msg_redirector* pour identifier le socket précis sur lequel les données doivent être transmises ou redirigées. À chaque établissement ou fermeture de connexion, `SOCKMAP` est mise à jour par le composant *msg_redirector* grâce à un programme eBPF de type `SOCK_OPS`.

Le programme eBPF `SK_MSG` du composant *msg_redirector* est exécuté chaque fois que l'application TCP invoque l'appel système `sendmsg()` pour envoyer une donnée. Le composant *hooker_userspace* lorsqu'il invoque la primitive `recvmsg()`, ne reçoit que le payload du paquet sans aucune information de Transport. Ainsi pour permettre à *hooker_userspace* d'identifier le processus à qui appartient le paquet, important pour correctement adresser la réponse du destinataire au processus approprié, *msg_redirector* va faire usage du helper `bpf_msg_push_data()` pour rajouter un tag au payload avant de mettre à disposition le paquet sur le socket de redirection. Enfin, pour rediriger les données entre le socket TCP de l'application et le socket de redirection, dans un sens comme dans l'autre, *msg_redirector* va utiliser le helper `bpf_msg_redirect_map()`. Le socket de redirection est créé et maintenu par le composant *hooker_userspace* qui récupérera les données par invocation de l'appel système `recvmsg()`, avant de les envoyer sur un socket UDP ou UDP-Lite. Le protocole UDP-Lite est déployé au même moment que les composants eBPF du Hooker (*msg_redirector* et *hooker_ingress*). Sur le chemin retour, le programme eBPF de type `XDP` maintenu par le composant *hooker_ingress* intercepte tous les paquets dès qu'ils arrivent sur la carte réseau (NIC). Le paquet intercepté peut ensuite être *(i)* supprimé (verdict `XDP_DROP`), *(ii)* retransmis à la même carte réseau (verdict `XDP_TX`), ou, comme c'est le cas dans le prototype actuel, *(iii)* passé à la pile protocolaire appropriée (verdict `XDP_PASS`) pour des traitements additionnels.

3 Evaluations du cadriciel *Hooker*

Le cadriciel proposé est évalué au travers d'expérimentations basées sur un transfert de trois fichiers de types variés (simple texte, image et vidéo) possédant naturellement des tailles différentes, 3KB, 510KB et 26MB respectivement. Le temps de téléchargement des fichiers et le délai de déploiement des composants eBPF du cadriciel *Hooker* sont évalués.

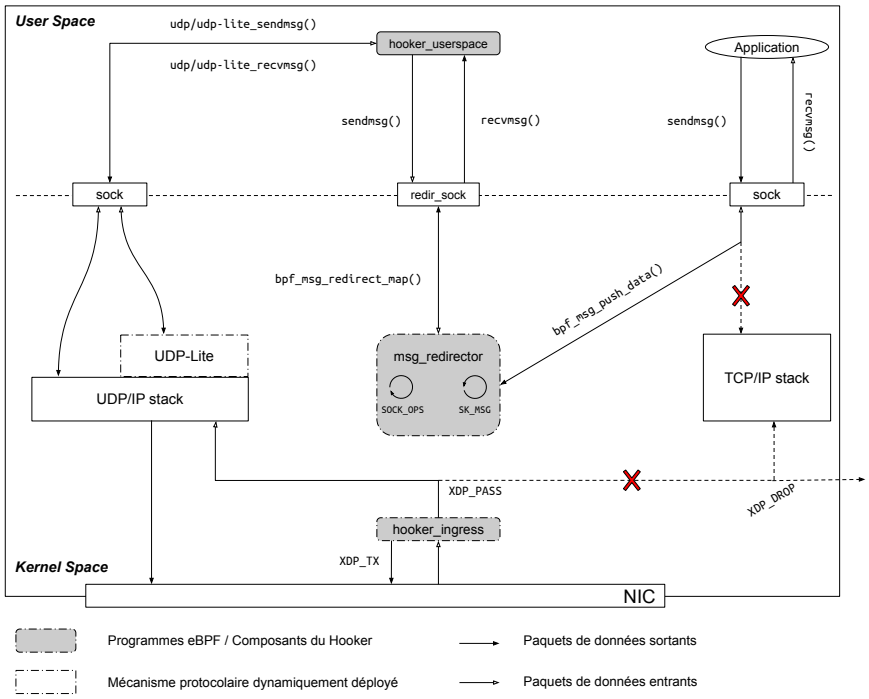

FIGURE 1: Cadriciel Hooker

3.1 Configuration du banc d'essai

Le banc d'essai utilisé pour les expérimentations est constitué de deux machines virtuelles. L'une des machines est utilisée en tant que serveur de streaming de données tandis que l'autre abrite l'application jouant le rôle de client. Chaque machine tourne sous la version 5.0 de l'OS Linux et est équipée d'un processeur Intel Core i7-7700 de 6.3GiB de RAM. Le lien réseau liant les deux machines virtuelles est émulé par l'outil `netem`. Le RTT moyen est fixé à 50 ms et le taux de perte est configuré sur zéro. L'objet étant d'évaluer les capacités de déploiement et de redirection du *Hooker*, il faut assurer que les conditions réseau ne pénalisent aucun des mécanismes protocolaires utilisés, d'où la valeur du taux de perte qui permet d'un côté à UDP et UDP-Lite d'être aussi fiable que TCP, et de l'autre côté à TCP d'être aussi rapide que UDP et UDP-Lite.

3.2 Scénarii et Résultats

Pour chaque fichier transmis, le scénario est identique et est constitué de deux étapes : le premier transfert est effectué sans activer le Hooker et le second se fait avec activation du Hooker sur les deux machines (client et serveur). À chaque étape, nous vérifions que les données sont correctement reçues et grâce à Wireshark, nous validons le succès de la redirection en observant le protocole utilisé pour sortir sur la carte réseau (NIC). Le temps requis pour le transfert des données est illustré sur la Fig. 2. On y observe l'impact du Hooker sur le délai total requis pour achever le téléchargement des données. Comme indiqué à la table 1, le *Hooker* introduit un overhead dû aux opérations de redirection (copies de buffers, etc.) et aux opérations de déploiement des programmes eBPF (compilation et chargement).

Programme eBPF	***Compilation***	***Chargement***	***Redirection ops***
Composants du *Hooker*	6s	20 ms	400 ns
UDP-Lite	200 ms	7 ms	N/A

TABLE 1: Coûts du déploiement et de la redirection

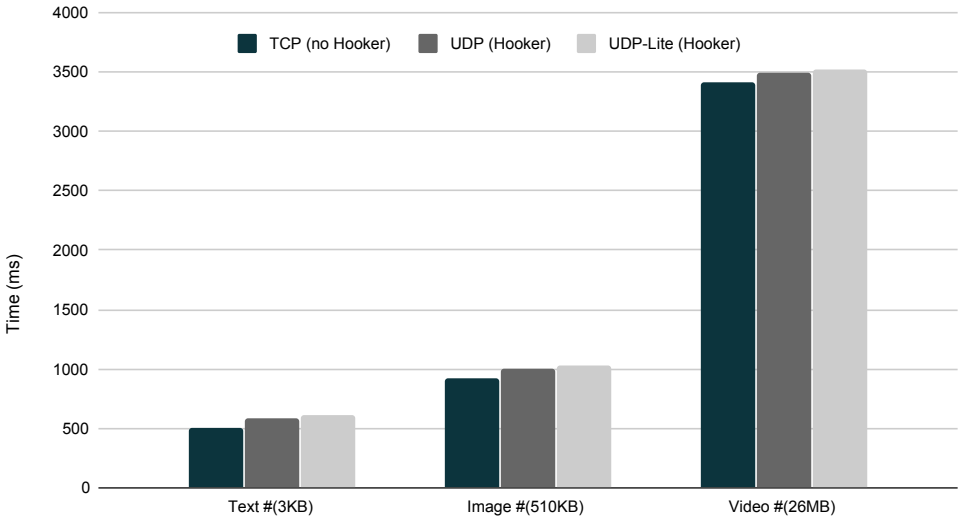

FIGURE 2: Délai de téléchargement des fichiers avec/sans Hooker

4 Conclusion et perspectives

Nous proposons une redirection transparente et un déploiement dynamique de mécanismes protocolaires pour adresser les problématiques d'évolution de la couche Transport d'Internet. Nous avons conçu et implémenté un cadriciel basé sur eBPF capable de rediriger les connexions de niveau L4 des applications basées sur TCP vers le protocole UDP natif ou le protocole UDP-Lite dynamiquement déployé dans le noyau de l'OS. Dans son implémentation actuelle, le cadriciel déploie à la volée UDP-Lite dès son démarrage. Le protocole adapté et le moment propice à son déploiement feront l'objet des perspectives de nos travaux.

Références

[CGR] "Linux Manual Page CGROUPS". accesible sur https ://bit.ly/2SIcn5n.

[DFDC10] E. Dubois, J. Fasson, C. Donny, and E. Chaput. *Enhancing TCP based communications in mobile satellite scenarios : TCP PEPs issues and solutions*, 2010. Proc. of the 5th ASMS and 11th SPSC.

[MKZ+14] D. Murray, T. Koziniec, S. Zander, M. Dixon, and P. Koutsakis. *An analysis of changing enterprise network traffic characteristsics*, 2014. IEEE Asia-Pacific Conference on Communication (APCC).