ENSEIRB-MATMECA
Université de Liège

2A Électronique

---

# Création, développement et évaluation d'une infrastructure de mesures du réseau basée sur les unikernels

---


***Élèves :***
Maxime LETEMPLE

***Enseignant :***
Benoît DONNET
Gauthier GAIN
Sami Ben Mariem


29 septembre 2023

# Table des matières



# Table des figures

# 1 Contexte

## 1.1 Présentation de l'université de Liège

L'institut Montefiore est le département d'électronique et d'informatique de l'Université de Liège. il a été fondé 1883 par Georges Montefiore-Levi. Cet institut est situé en périphérie de Liège, au campus du Sart Tilman. Cet institut couvre une grande partie de la recherche actuelle en informatique et en électronique, notament l'intelligence artificielle, le machine learning, la cybersécurité, les réseaux, les systèmes d'exploitation.

FIGURE 1 – Lieu du stage à l'université de Liège

## 1.2 Organisation du stage

# 2 État de l'art

## 2.1 Scamper

Scamper est une suite d'outils destinés à l'exploration et à l'analyse des réseaux informatiques, principalement axés sur les mesures et la collecte d'informations à grande échelle. Il a été développé pour fournir une compréhension approfondie de la topologie du réseau, de la latence, de la perte de paquets et d'autres paramètres de performance. Les fonctionnalités de Scamper permettent aux chercheurs, aux ingénieurs réseau et aux administrateurs système de déterminer la qualité et la stabilité des connexions réseau.

## 2.2 Unikraft

Un unikernel est une unité logicielle auto-suffisante qui combine à la fois le code de l'application et les ressources minimales du système d'exploitation nécessaires pour exécuter cette application. Contrairement aux systèmes d'exploitation traditionnels qui exécutent plusieurs processus et services en parallèle, les unikernels sont conçus pour être très spécifiques à une application unique. Cette approche vise à réduire la taille, la complexité et les vulnérabilités potentielles associées à l'exécution de systèmes d'exploitation complets. L'unikernel utilisé ici est Unikraft. C'est un projet d'unikernel open source qui aide à construire des unikernels plus rapidement et plus efficacement. Unikraft fonctionne avec un système d'applications et de bibliothèques.

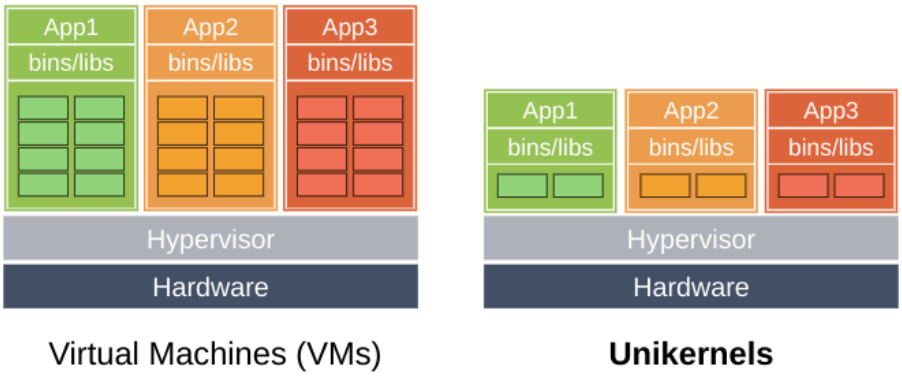


FIGURE 2 – Comparaison entre une machine virtuelle et un unikernel

Un unikernel ne va compiler que les bibliothèques qui lui sont nécessaires, alors qu'une machine virtuelle contient un système d'exploitation entier. L'unikernel consommera donc moins de stockage et moins de mémoire vive.

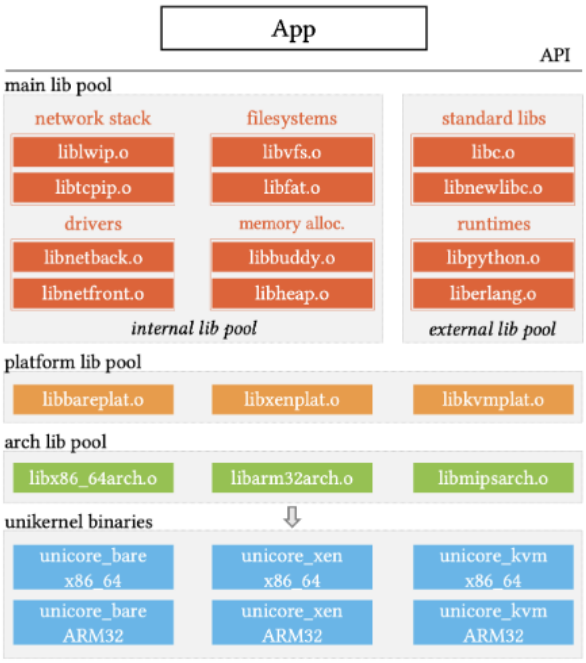


FIGURE 3 – Fonctionnalités des unikernels

Unikraft permet de créer des executables pour les principales architectures sur le marché. Sa structure très modulable permet également l'ajout et la modification de microlibraries système. Enfin, les binaires produits peuvent fonctionner sur bare metal ou avec des hyperviseurs de type 1, tels que xen ou kvm. Ceux-ci permettent l'accès direct aux ressources du processeur, sans devoir passer par la couche système.

## 2.3 Objectifs du stage

L'objectif principal est ici de porter Scamper sur Unikraft. Plus spécifiquement, c'est le module TNT de Scamper qui va être porté. Un tunnel MPLS, ou Multiprotocol Label Switching tunnel, est un mécanisme de réseau utilisé pour acheminer efficacement le trafic

de données à travers un réseau, en particulier dans le contexte des réseaux de télécommunication et des réseaux d'entreprise. Le fonctionnement d'un tunnel MPLS implique l'attribution de libellés (labels) aux paquets de données entrants à leur point d'entrée dans le réseau, puis le routage de ces paquets en fonction de ces libellés à travers le réseau. TNT est donc un traceroute augmenté qui est capable de trouver ces tunnels MPLS cachés.
Scamper est actuellement utilisé sur des machines peu performantes et est assez difficile à déployer. Il faut à chaque fois installer linux sur une machine puis installer scamper depuis les dépôts disponibles sur le site web de CAIDA. C'est pourquoi ici l'objectif a été de démontré la facilité de déploiement, en plus de montrer l'augmentation de performances.

Les sources du projet peuvent être trouvées ici (https ://github.com/maxletemple/uTNT).

# 3 Tâches Réalisées

## 3.1 Portage de l'application

L'application portée ici est TNT. Il s'agit d'un driver de Scamper qui consiste en un traceroute modifié permettant de détecter les tunnels MPLS. le code de Scamper sera dans une bibliothèque Unikraft, tandis que le code de TNT sera dans l'application à proprement parler.
Le traceroute effectué est globalement similaire à un traceroute classique. Pour trouver le chemin par lequel passent les paquets pour aller de la machine à un autre point, le traceroute effectue plusieurs pings. Autrement dit, il envoie des paquets ICMP dans lequels il modifie le TTL. Il commence par un TTL de 1, c'est donc le routeur qui lui répond une erreur (code 11, temps dépassé). Il poursuit avec un TTL de 2, puis 3, jusqu'à recevoir une réponse (code 0, réponse d'écho) du routeur. C'est ce programme qui va être implémenté ici.
La bibliothèque de fonctions C utilisée est musl, ainsi que LwIP pour la couche réseau. Ces deux bibliothèques permettent la compatibilité de la plupart des appels de primitives de Scamper. Ces bibliothèques sont très utilisées dans le monde des systèmes embarqués car elles sont très économes en ressources. les versions utilisées ici sont des versions légèrement modifiées et adaptées à Unikraft. Un certain nombre de fonctionnalités y sont manquantes.

## 3.2 Compilation de TNT et Scamper

Sur Linux, Scamper s'installe de manière très classique, c'est à dire qu'on compile les sources avec la commande `make` puis on les installe avec `make install`. Pour utiliser Scamper, on lance dans un premier temps la commande `scamper`, puis, dans un autre terminal, on lance la commande liée à la mesure que l'on veut effectuer. On peut effectuer un traceroute classique avec `sc_tracediff`, ou ici le traceroute modifié (TNT) avec `sc_tnt`.
Pour tester le programme, on utilise Qemu. Il s'agit d'un logiciel de machine virtuelle, permettant d'émuler un processeur, et est capable d'utiliser KVM. KVM est un hyperviseur de type 1 directement intégré sur la plupart des noyaux Linux, et qui permet d'accéder directement aux ressources du processeur. De cette manière, les machines virtuelles sont beaucoup plus rapides car Qemu execute directement les instructions sur le processeur.
Une fois les tests effectués, l'objectif est de déployer uTNT sur un environnement Xen, qui permet l'execution en bare metal de plusieurs unikernels à la fois.

## 3.3 Adaptation de la structure du code de Scamper

Étant donné qu'en environnement Linux il faut lancer deux commandes, ce sont donc deux fonctions `main()` qui sont appelées : une pour lancer Scamper et une pour lancer sc_tnt. Pour les faire fonctionner sur Unikraft, on utilise le code de sc_tnt dans l'application et Scamper va être une bibliothèque qui sera appelée. Cette méthode est la plus efficace car sc_tnt est contenu dans un seul fichier C. Il suffit donc de copier le code et de le coller dans le fichier `main.c` du projet. Scamper est donc une bibliothèque qu'il faut correctement configurer. On appelle cette bibliothèque lib-scamper. Dans ce dossier, on ajoute un dossier `include`, dans lequel on ajoute tous les fichiers headers correctement hiérarchisés, et un dossier `src`, dans lequel on ajoute les fichiers sources. Il faut ensuite créer un fichier Makefile.uk, que Unikraft va interpréter et qui va rendre les fichiers disponibles au compilateur.

## 3.4 Rendre Scamper monoprocessus

Scamper utilise donc deux processus pour fonctionner : un serveur d'écoute, qui va attendre un ordre, faire toute les mesures et renvoyer les résultats au deuxième processus, qui correspond à la deuxième commande écoutée.

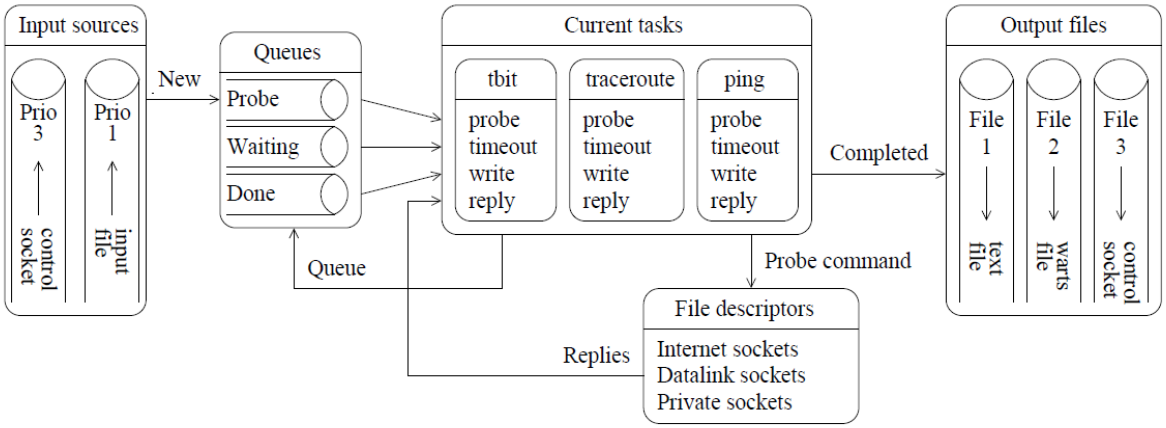


FIGURE 4 – Architecture de Scamper

Unikraft est monoprocessus : il se contente de démarrer, d'executer son programme puis s'interrompt. Il a donc fallu trouver un moyen de fusionner ces processus.
Le meilleur moyen est de lancer chacune des fonctions `main()` dans deux threads différents, qui seront donc dans le même processus.

## 3.5 Communication entre les threads

Un des problèmes majeurs lors du portage d'une application sur un unikernel est la compatibilité lors de l'appel des primitives. En effet, elles ne sont pas forcément disponibles, et pour celles qui le sont elles ne font pas forcément la même chose. Sur cette application, il a parfois suffit de retrouver les bon noms de primitive et de simplement modifier les includes dans chaque fichier. Mais ici, le problème a été plus important. En effet, pour assurer la communication entre ses processus, Scamper utilise des sockets, et

plus précisément la famille de sockets AF_UNIX. Comme son nom l'indique, ces sockets sont disponibles sur les systèmes d'exploitation de type UNIX, ce qui n'est pas le cas ici. L'avantage de ces sockets est qu'il suffit d'appeler la fonction `socketpair()` pour recevoir un tableau de deux sockets, ainsi en fournissant une de ces sockets à l'autre processus ceux-ci peuvent communiquer. On ne peut pas procéder de la sorte ici, seule la famille de sockets AF_INET est disponible.
Pour lier deux sockets AF_INET ensemble, il faut qu'un contact entre les deux se fasse. À la place de la fonction `socketpair()`, on crée donc temporairement un thread (fichier `main.c` dans la fonction `do_files`).

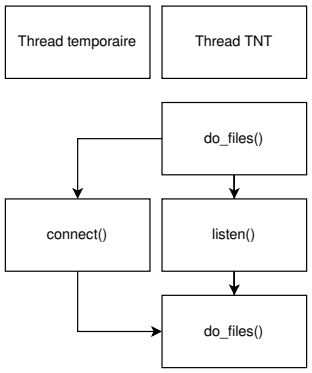


FIGURE 5 – Création d'une paire de sockets

## 3.6 Virtualisation et Création de l'interface

Une fois le programme compilant et fonctionnel, il faut s'assurer de sa connectivité à Internet. Pour ce faire, on crée un pont réseau. Un pont réseau est un composant qui permet de segmenter, de filtrer et de connecter des segments de réseau pour améliorer les performances, la sécurité et la gestion du trafic. Ils sont couramment utilisés dans les réseaux locaux (LAN) pour organiser et optimiser le flux de données entre les dispositifs connectés. Ici, on crée un pont réseau virtuel afin de connecter les différents unikernels entre eux. Ce test a été réalisé sur Linux, mais Xen supporte aussi la créations de ponts et d'interfaces réseau.

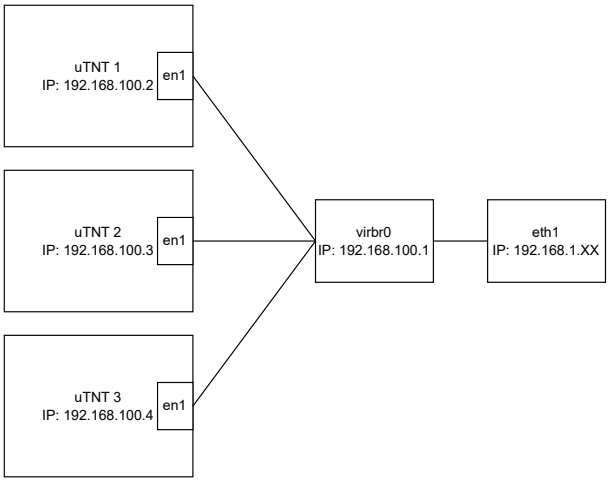


FIGURE 6 – Schéma des interfaces réseau

Cette partie a pris beaucoup de temps de débogage, car pour effectuer un traceroute, scamper utilise la commande ping, qui envoie des paquets ICMP (voir chapitre 3.1 : Portage de l'application). Or Scamper, pour correctement fonctionner, a besoin de forger lui-même le paquet, et ce bit par bit et en incluant la couche TCP/IP. Cette fonctionnalité n'a pas été implémentée correctement dans LwIP. Il a donc fallu réécrire une partie du code de LwIP pour que, lors d'une création de socket, on puisse ajouter l'option `IP_HDRINCL` dans la fonction `scamper_icmp4_open_fd` du fichier `scamper_icmp4.c`.

## 3.7 Mesures

L'ensemble des mesures a été fait sous trois configurations différentes :

— Scamper installé sur une machine virtuelle (Ubuntu 16.04 installé avec l'utilitaire Vagrant)

— Scamper lancé avec Docker

— uTNT lancé avec Qemu

### 3.7.1 Consommation des ressources

Les premières mesures se sont portées sur la consommation de mémoire, de temps et de charge CPU, ainsi que de stockage. Dans tous les cas, uTNT surpasse Docker et la VM. La figure 7 montre que l'utilisation de temps CPU est bien plus faible et bien plus rapide du côté de uTNT. En effet, le temps de boot de uTNT est bien plus rapide du fait de la taille de l'unikernel, de plus, seules les biblothèques nécessaires ont été compilées et incluses dans l'image, celle-ci prend donc beaucoup moins de place dans le stockage (figure 8), et l'appel à moins de bibliothèques partagées implique une consommation de mémoire vive plus faible (figure 9).

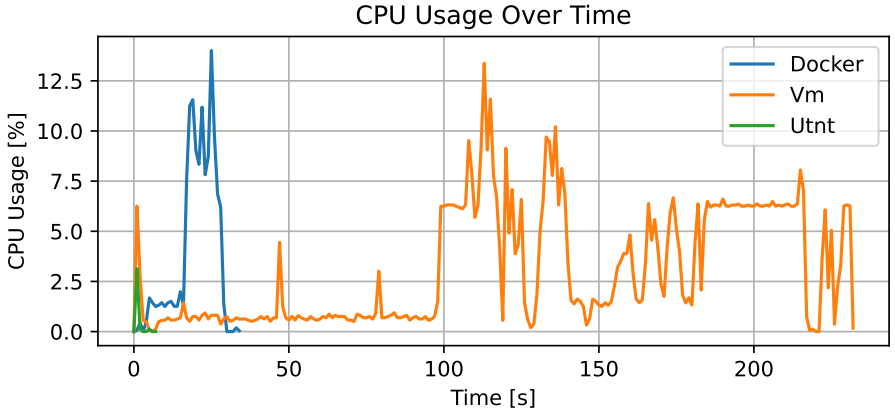


FIGURE 7 – Sollicitation du CPU au cours du temps par Docker, la VM et uTNT

La figure 7 a été réalisée sur un serveur de 16 coeurs. On remarque que uTNT ne consomme que 2.5% de la charge CPU à son maximum, et ce sur une durée beaucoup plus courte que Docker et la VM. En effet, celui-ci est peu multithreadé, alors que Docker et la VM le sont. C'est un avantage, car cela signifie que plusieurs unikernels peuvent être lancés sur un même machine avec beaucoup de coeurs, sans qu'ils interfèrent trop entre eux.

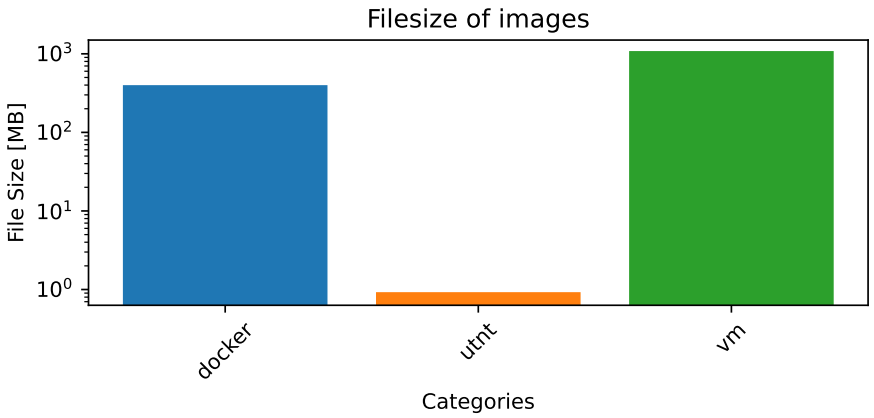


FIGURE 8 – Taille des images de Docker, uTNT et de la VM

La taille des images (figure 8) est naturellement beaucoup plus faible sur uTNT (897 ko), car le kernel n'embarque que ce qui est nécessaire pour le fonctionnement de Scamper. Autrement dit quelques fonctions systèmes et la stack réseau de LwIP. Docker et la VM contiennent quant à eux l'ensemble des fonctions système de Linux.

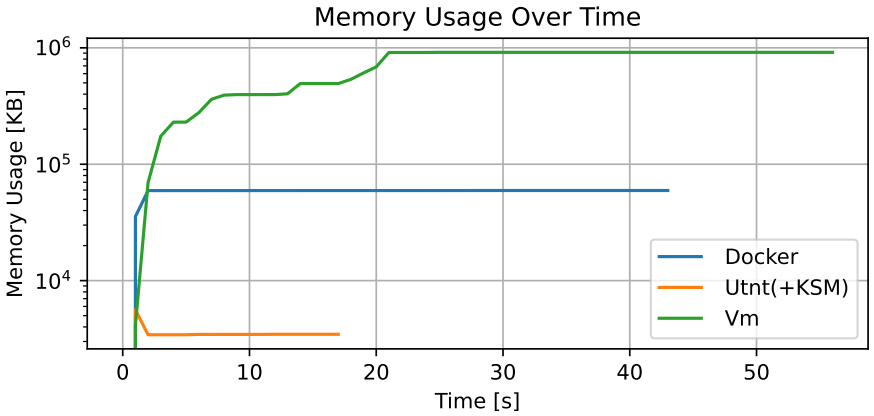


FIGURE 9 – Utilisation de la RAM au cours du temps de Docker, uTNT et de la VM

La quantité de RAM utilisée par uTNT est également très faible. Cette RAM a encore été réduite grâce à KSM, un utilitaire qui permet de fusionner les pages de mémoire vides lors de l'execution d'un programme. Le léger pic que l'on voit au début correspond au démarrage de l'unikernel, puis Scamper utilise par la suite encore moins de mémoire vive. Docker, quant à lui, utilise environ 59 Mo de RAM, et la machine virtuelle environ 914 Mo. C'est de l'ordre de grandeur d'un système d'exploitation au repos sans interface graphique associée.

### 3.7.2 Étude d'un cas de déploiement concret

L'objectif de ce stage a été de démontrer la rapidité de déploiement d'un Unikernel au moment d'un besoin de mesure réseau, n'importe où sur Internet. Il a donc fallu démontrer la rapidité de déploiement dans un cas concret. Ici 5 serveurs OVH ont été loués à Singapour, au Canada, en Australie, en France et en Pologne. Un script Python reçoit une

liste de mesures à effectuer. Il se charge ensuite de distribuer les tests automatiquement en fonction des noeuds (serveurs) disponibles, et ce aux bon instants demandés. On peut également choisir l'attitude du script face à un trop grand nombre de requêtes par rapport aux noeuds disponibles : soit il attend qu'un noeud soit disponible, soit il abandonne cette requête.

Ce script a ici été chargé de lancer un test par seconde pendant une minute, et réalise un traceroute vers le serveur 1.1.1.1. Ce procédé a été refait 30 fois de manière à obtenir une moyenne et un écart-type.

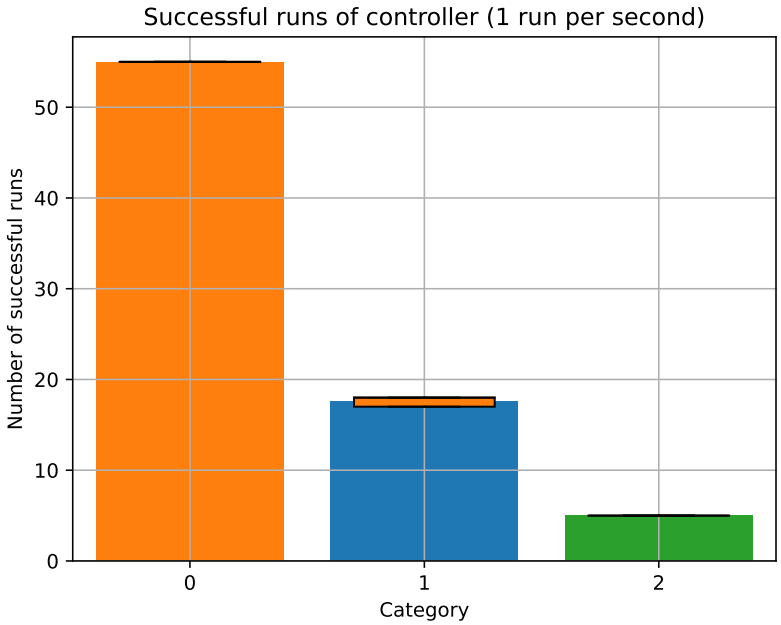


FIGURE 10 – Nombre de test réussis de uTNT (0), Docker (1) et de la VM (2) pour 60 requêtes

La figure 10 montre que dans la configuration de script "annuler si pas de serveur disponible, uTNT parvient à réaliser quasiment tous les tests, alors que Docker n'en réalise pas la moitié et la VM se contente de lancer un seul test sur chaque seveur.

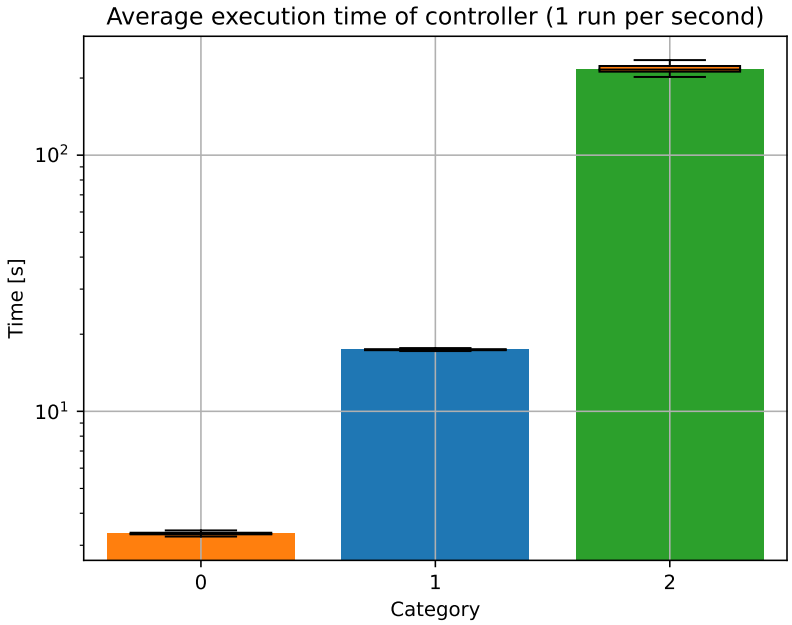


FIGURE 11 – Temps d'execution moyen sur un serveur de uTNT (0), Docker (1) et de la VM (2)

En figure 11, on peut voir que uTNT est encore une fois le plus rapide, avec très peu d'écart type. L'écart type plus élevé sur la VM provient probablement du téléchargement du système d'exploitation qui est assez important, et tous les serveurs ne sont pas à la même distance du serveur hébergant Ubuntu. De ce point de vue, uTNT est encore une fois peu impacté car le transfert ne comprend qu'une image de moins d'un mégaoctet, ce qui assure sa rapidité de téléchargement sur n'importe quelle machine.

# 4 Conclusion

Pour conclure, l'ensemble des objectifs du stage ont été réalisés. L'utilisation d'unikernels s'est avérée extrêmement efficace en termes de consommation de ressources (mémoire, CPU, stockage). Comparé à des solutions traditionnelles comme les machines virtuelles et les conteneurs Docker, uTNT a montré des performances nettement meilleures. Ce stage a démontré le potentiel des unikernels en tant que plateforme pour des applications de mesure réseau performantes et légères. Il ouvre également la voie à des développements futurs pour étendre les fonctionnalités de uTNT et explorer d'autres domaines d'application des unikernels dans le domaine des réseaux informatiques.

# 5 Conclusion personnelle

À titre personnel, je suis très fier d'avoir pu travailler sur ce stage. En travaillant sur ce projet, j'ai renforcé mes compétences en développement logiciel. Le portage de l'application Scamper sur l'environnement Unikraft a nécessité une compréhension approfondie du code source, ainsi que des compétences en débogage et en résolution de problèmes. Cette expérience a été particulièrement enrichissante pour mon développement professionnel.

J'ai beaucoup appris sur les systèmes d'exploitation et les réseaux, et c'est ce que je recherchais en acceptant ce stage. En plus de cela, un article scientifique va être rédigé sur mon travail. Il s'agit d'un premier pas pour moi dans la recherche, et c'est très gratifiant. Enfin, je tiens à remercier Benoît Donnet pour l'opportunité de stage, ainsi que Gauthier Gain et Sami Ben Mariem pour leur précieuse aide, leur expérience m'a été très utile tout au long de ce projet.

# Références

[1] Y. VANAUBEL, P. MERINDOL, J.-J. PANSIOT, and B. DONNET, Through the Wormhole : Tracking Invisible MPLS Tunnels, Proceedings of the 2017 Conference on Internet Measurement Conference

[2] Y. VANAUBEL, P. MERINDOL, J.-J. PANSIOT, and B. DONNET, TNT, Watch me Explode : A Light in the Dark for Revealing All MPLS Tunnels

[3] M. LUCKIE, Scamper : a Scalable and Extensible Packet Prober for Active Measurement of the Internet, ACM SIGCOMM Internet Measurement Conference, November 2010. See : https://www.caida.org/tools/measurement/scamper/

[4] Unikernels : Rethinking Cloud Infrastructure. http://unikernel.org/.

[5] Unikraft : https://unikraft.org/

[6] Kuenzer, S., Bădoiu, V., Lefeuvre, H., Santhanam, S., Jung, A., Gain, G., Soldani, C., Lupu, C., Teodorescu, S. L., Răducanu, C., Banu, C., Mathy, L., Deaconescu, R., Raiciu, C., & Huici, F. (2021). Unikraft : Fast, specialized unikernels the easy way.