Developpement réseau

Léo L'huillier

Stage de deuxième année

De septembre à décembre 2022

Promotion 2026

I. Remerciements

Je tiens a remercier mon tuteur de stage Baptiste Povlé qui m'a offert le poste de stagiaire et qui m'a suivi durant toute la durée de mon stage.

Je remercie particulierement Hai Dang Nguyen et Hugo Martineau qui m'ont accordés beaucoup de temps en aide et expliquations et dont les conseils ont été précieux.

Je remercie egalement léequipe de Snowpack pour leur bienveillance, leur soutient et leur confiance. Je suis heureux d'avoir passer mon stage à leurs côté.

Sommaire

II. Introduction

Dans le cadre de mes études à EPITECH, j'ai eu la chance de réaliser un stage d'une durée de 4 mois chez Snowpack. Snowpack est une start up issue du Commissariat à l'énergie atomique et aux énergies alternatives (CEA) et travaillant dans le secteur de la cybersécurité. Rattachée à l'équipe de développement, j'ai pu participer à la réécriture dans un nouveau language de programation de la solution de Snowpack.

Snowpack évolua pendant plus de 4 ans de recherches et développement ainsi que 3 brevets au sein du CEA.

Finalement, Snowpack prend son indépendance en mai 2021 grâce à une équipe de 4 personnes aux compétences complémentaires : Fréderic Laurent, qui a apporté l'idée du projet et CEO et Baptiste Povlé, CTO, forment tous deux l'équipe technique et de développement. Rejoint par Sebastien Groyer, COO, pour la partie management et finances et David Gonzalez, CRO , pour la partie vente et commerciale bien que celui-ci ait quitté l'aventure en juillet 2022.

J'ai pu comprendre lors de mon entretien avec Baptiste Povlé, mon matre de stage, la rason de ma présence dans l'entreprise.

Le produit de Snowpack etait trop lent et dans le but de le rendre plus rapide et l'optimisé, il avait été decidé de changer de langage de programation.

Afin de rendre compte des quatres mois passés au sein de Snowpack, il convient de présenter l'entreprise et l'environnement dans lequel j'ai passé ce dernier semestre. Je donnerai ensuite des explications dans le but que chacun puisse comprendre ma place au sein de l'entreprise. Je finirai ensuite par expliquer les missions qui m'ont été confiées et les apports que j'ai pu en tirer.

III. L'entreprise

III.1. SNOWPACK become invisible

L'objectif de Snowpack est de rendre ses utilisateurs complètement invisibles sur internet. C'est-à-dire permettre l'échange de fichiers et d'information de façon complètement anonyme et sécurisé, rendre donc impossible pour un individu malveillant d'accéder à vos informations ou votre identité.

III.2. Quelques chiffres

Snowpack en quelques chiffres c'est :

— plus de deux millions d'euros levés pour ses recherches
— des serveurs dans plusieurs pays d'europe
— un site internet
— cimq offres differentes
— une application mobile sur google play
— un produit pouvant être utiliser sur Linux, Windows, Android et prochainement Apple

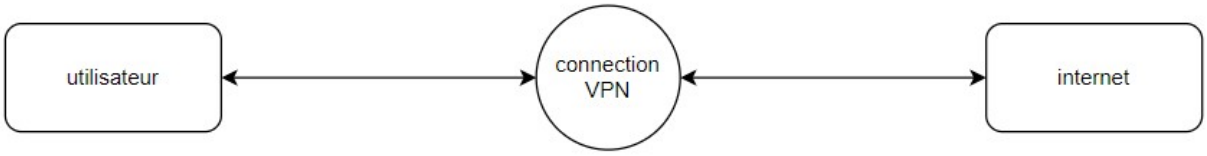
III.3. Présentation générale

Je vais ici expliquer ce qu'est Snowpack et ce qui le differencie de ses concurents, je tiens à présiser que chaque terme technique utilisé ici sera expliqué par la suite de façon à ce que n'importe qui puisse comprendre pleinement ce rapport.

Pour revenir donc sur ce qu'est Snowpack, son objectif est de rendre ses utilisateurs complètement invisible sur internet. C'est-à-dire permettre l'échange de fichiers et d'informations de façon complètement anonyme et sécurisé, Rendre donc impossible pour un individu malveillant d'accéder à vos informations ou votre identité.

Bien que ce principe puisse sembler similaire à certains outils déjà existants, tel que TOR (the onion routing) ou un VPN (réseau privé virtuel), son approche rend Sonwpack bien meilleur.

Par exemple, les VPN dont la popularité va grandissante, vont assurer l'échange d'information sécurisé grâce à un essemble de technologie de chiffrement. Mais la sécurité des informations n'est toutefois pas infaillible.



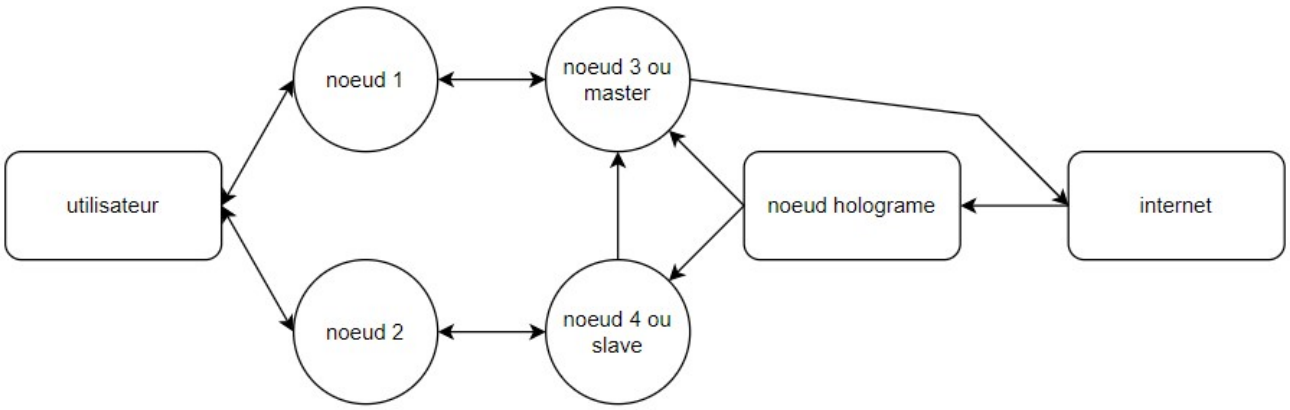
Dans cet exemple d'un utilisateur connecté au net via un VPN, chaque noeud du VPN a accès aux informations entre le VPN et le net. Ainsi, quelqu'un qui parvient à casser le chiffrement effectué par le VPN a alors accès à toutes les informations de l'utilisateur.

Quant à TOR, voici la définition qu'on peut trouver sur wikipedia : "Tor est un réseau informatique superposé mondial et décentralisé. Il se compose de serveurs, appelés nœuds du réseau et dont la liste est publique. Ce réseau permet d'anonymiser l'origine de connexions TCP". Mais le probleme de TOR est que la grande majorité de ses serveurs apprtiennent à de grandes agences americaines et l'état americain y a donc accès. Ces données ne sont en definitive, pas vraiment protégées.

Snowpack est aussi meilleur puisque les VPN sécurisent les données mais ne donne pas d'anonymité. TOR, offre de l'anonymité mais pas de sécurité, Snowpack au contraire offre les deux.

III.4. La technologie

Snowpack, pour rendre ses utilisateurs invisibles, va fractionner les données envoyées en deux parties prenant chacune une route differente. Les données fractionnées vont alors être reconstituées par un noeud hologramme qui va interagir avec le net. De cette façon, quelqu'un qui aurait accès à un noeud du réseau ne peut retrouver aucune des données de l'utilisateur.



Aucun "flocon" (message fragmenté) ne contient d'information sur l'expéditeur, on ne peut donc pas reconstituer le message sans les deux "flocons" complémentaires. Chaque noeud va servir de relais pour les flocons et crée un manteau neigeux dans lequel les données sont protégées.

IV. informations nécéssaires

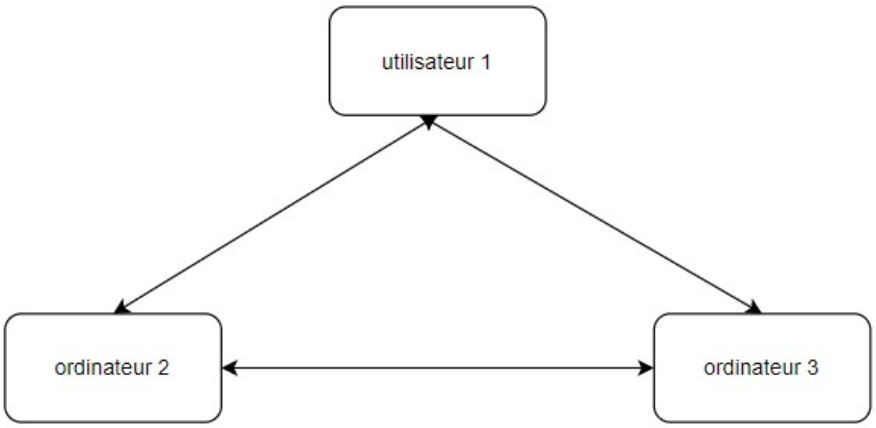
IV.1. Le code, qu'est ce que c'est

Je définirai le code comme étant une suite de fonctions écrites dans un langage de programmation. Une fonction est une partie du code à qui l'on définit une action particulière. Par exemple, on pourrait dire que la fonction d'un vendeur est de vendre des choses et que la société dans laquelle il travaille, représente le code dans lequel il évolue. L'architecture du code va définir l'ordre dans lequel les fonctions vont être utilisées pour permettre leur bon fonctionnement.

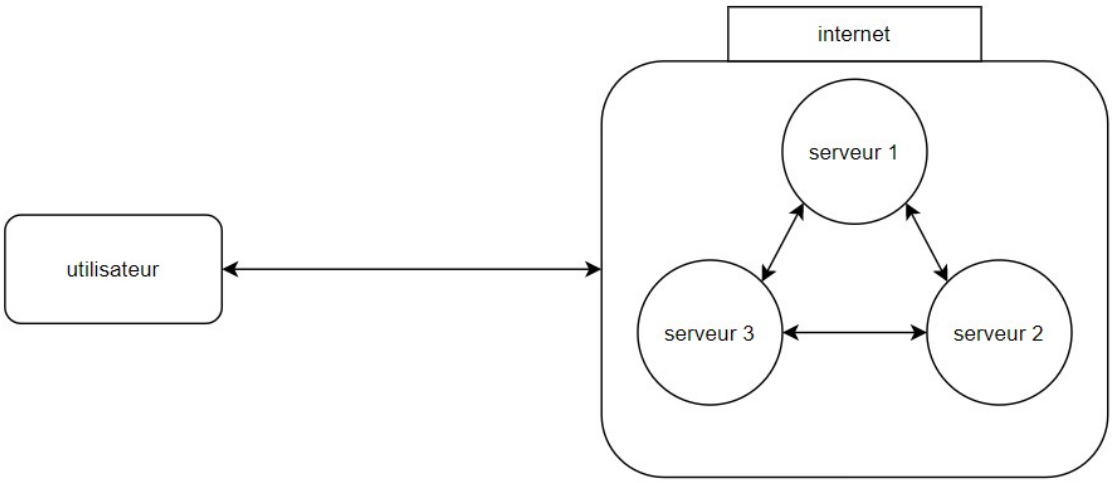
Un langage de programmation est donc ce qui va permettre d'écrire du code qu'un ordinateur va pouvoir comprendre. On peut aussi définir un langage de programmation comme étant de haut ou de bas niveau, un langage de haut niveau étant plus simple à écrire et comprendre mais moins puissant. En opposition, un langage de bas niveau est généralement plus compliqué à écrire mais plus puissant et plus optimisé pour certaines taches. Le code précédemment écrit par SNOWPACK était écrit en python, un langage de haut niveau, et ma mission était de le réécrire en Cplusplus (pouvant aussi s'écrire Cpp ou C++) car ce langage est plus rapide et plus optimisé pour faire du réseau. Je n'avais personnelemt coder qu'en C, un autre langage de bas niveau assez proche du C++.

IV.2. Internet, qu'est ce que c'est

Il me semble ensuite important d'expliquer ce qu'est un réseau et comment marche internet de façon général. Je définirai un réseau comme étant tout simplement plusieurs ordinateurs communicants entre eux.

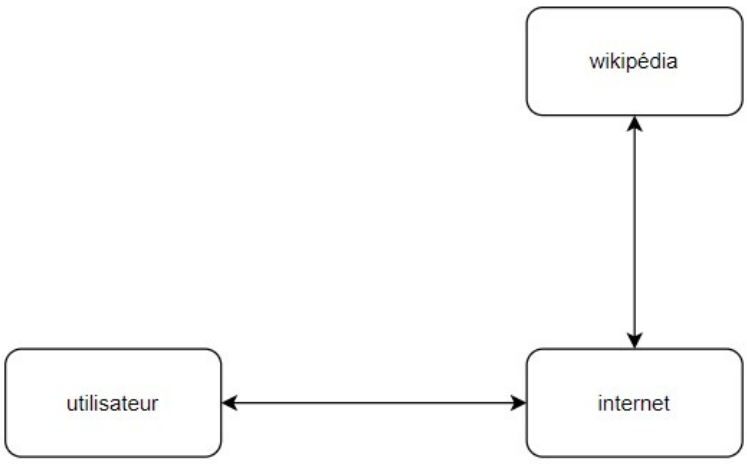


Un serveur est alors une sorte de gros ordinateur possédant beaucoup de données et communicant dans un réseau, qu'on appelle de façon générale internet. Un ordinateur va pouvoir interagir sur ce réseau à l'aide d'une carte réseau.



IV.3. La place de Snowpack sur internet

Pour expliquer la place de SNOWPACK sur internet, imaginons que Wikipédia a un serveur avec toutes ses informations dessus, quand un utilisateur va demander à Google d'ouvrir une page Wikipédia, google va demander les informations à la carte réseau. La carte réseau va alors demander les informations au serveur de Wikipédia en passant par le réseau internet. Wikipédia en recevant la demande, va répondre en envoyant les informations à la carte réseau qui va enfin les relayer à Google pour les afficher sur l'ordinateur de l'utilisateur.

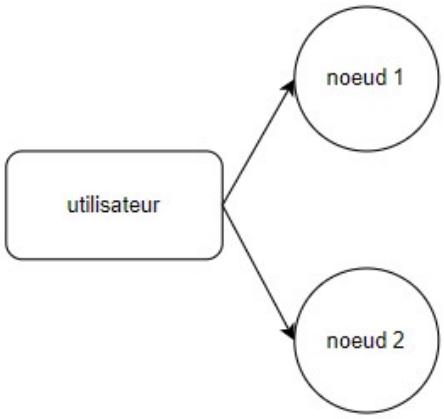


Le problème, c'est que si un individu malveillant surveille le serveur de Wikipédia, il est capable de déterminer quel utilisateur est allé sur quelle page Wikipédia. Le rôle de snowpark est alors de faire en sorte grâce à son propre réseau, de rendre cette demande d'information anonyme et sécurisé de telle façon que l'individu malveillant ne puisse accéder ni à l'identité de l'utilisateur ni à la page Wikipédia

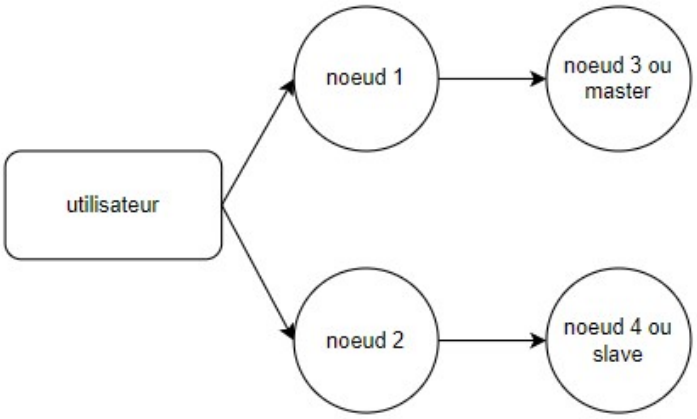
à laquelle il souhaite accéder.

Le rôle de Snowpack est donc de s'assurer que les données des utilisateurs soient protégées. Je vais donc maintenant expliquer de façon plus détaillée la façon dont SNOWPACK s'y prend pour réussir sa mission et créer son propre réseau sécurisé.

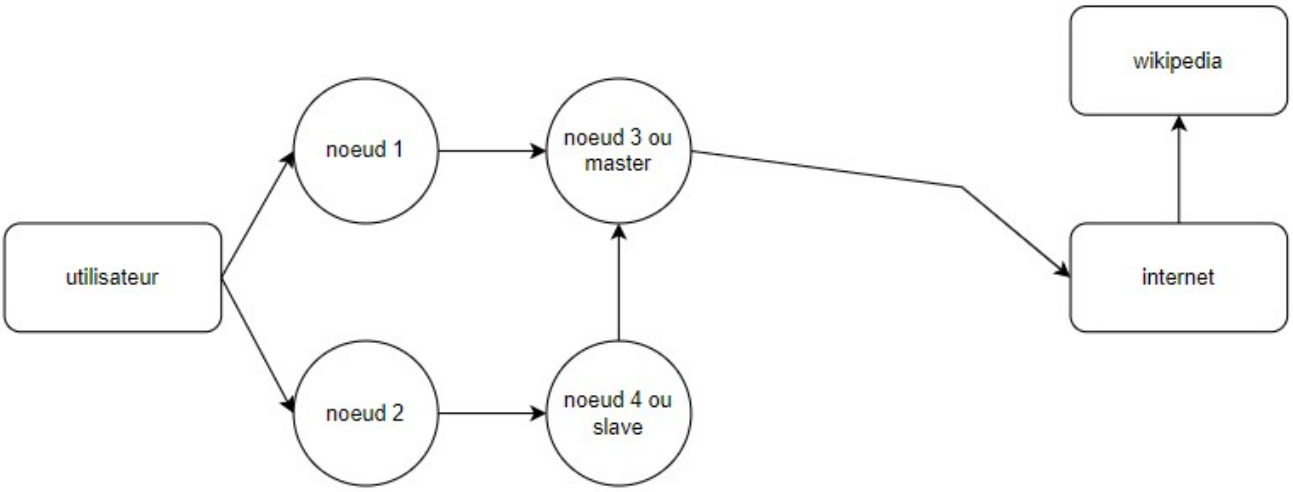
Pour rependre l'exemple de la page Wikipédia, quand google va envoyer la demande d'information au serveur Wikipédia à la carte réseau, Snowpack va rediriger cette demande pour faire en sorte qu'elle passe par le réseau Snowpack et non directement au serveur Wikipédia. La demande va donc être séparée en deux et encrypter puis envoyer à deux différents serveurs appartenants à Snowpack où un de leurs partenaires. Ces deux premiers serveurs sont appelés noeud 1 et noeud 2.



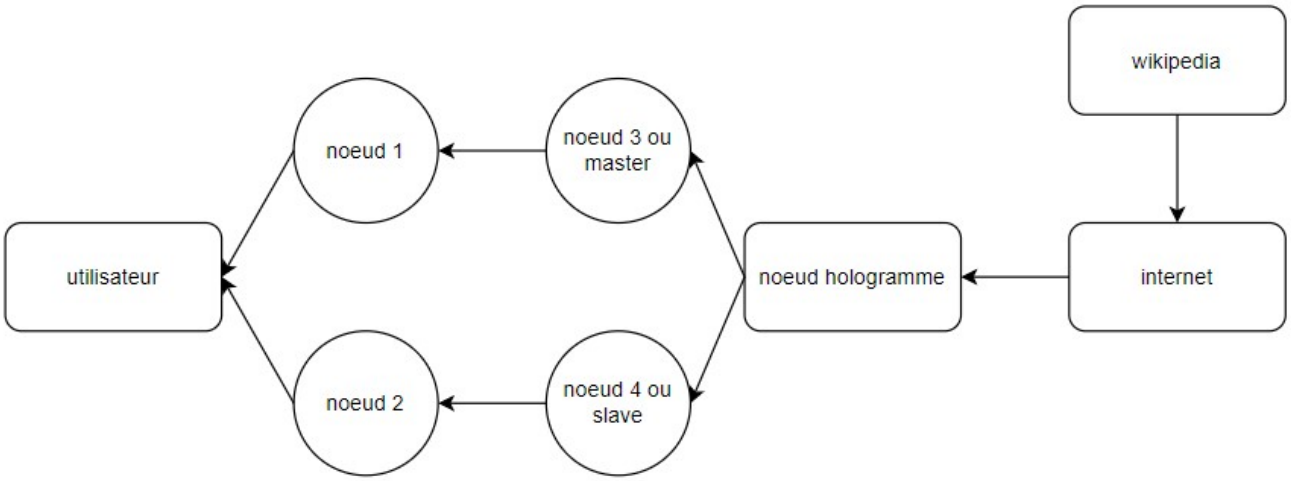
Chacun de ces serveurs va renvoyer la demande aux noeuds suivants respectivement noeud 3 (ou master) et noeud 4 (ou slave).



Le slave va ensuite envoyer sa partie du message au master possédant l'autre partie. Le master va alors reconstituer la demande de google et l'envoyer au serveur Wikipédia.



Le serveur Wikipédia, va répondre en envoyant les informations au noeud hologramme. Celui-ci va découper le message en deux parties de façon similaire à l'utilisateur et va envoyer chacun des deux messages au master et slave qui vont relayer le message jusqu'à l'utilisateur. Celui-ci va alors reconstruire le message contenant les informations et les renvoyer à google pour qu'il les affiche.



De cette façon, aucun individu malveillant ne peut accéder aux données de l'utilisateur puisqu'elles ne sont spécifiées à aucun endroit dans le réseau. En effet, même les noeuds 1 et 2 ne savent pas qui est l'utilisateur, mais seulement qu'ils doivent envoyer leurs messages respectifs à un certain endroit sans savoir de qui il s'agit.

V. Mes missions

V.1. Introduction

Bien que la théorie ne soit pas un problème, la réalisation est plus compliqué à mettre en place et l'architecture et le langage utilisés pour le code peuvent changer grandement les performances d'un systeme. Ces performances sont pourtant un element clef puisqu'elles peuvent influencer l'exeperience utilisateur et Snowpack ayant un service à vendre ne pouvait pas se permetre de limiter ses performances. Par example, s'il fallait dix secondes seulement pour ouvrir une page web, personne n'utiliserait la technologie Snowpack.

Ma mission s'agissais donc de réécrire le code précedemment en python, en c++, un langage plus rapide et adapté pour le réseau.

V.2. Premiere mission, comprendre la technologie

La première mission qui m'a été confiée a été de comprendre ce qu'était le réseau Snowpack (expliquer précédemment) en lisant un des brevets et le code qui avait déjà été produit. Un brevet n'est pas une lecture aisée mais cela m'a permis de comprendre rapidement comment fonctionnait le produit de Snowpack et de me rendre compte de la façon dont était écrit un brevet. Ce travail n'était pourtant pas quelque chose d'évident car il me fallait comprendre le réseau dans les moindres détails afin de pouvoir travailler efficacement.

V.3. Seconde mission, limiter la bande passante

Pour continuer à comprendre comment le produit de Snowpack fonctionnait, et apprendre à utiliser le python, j'ai dû trouver un moyen de limiter la bande passante sur le réseau. En effet, puisque le produit est payant, la version gratuite doit être limitée afin d'augmenter les ventes. J'ai donc dû me plonger dans le code déjà existant et comprendre où et comment réaliser ma tâche.

La bande passante est synonyme du taux de transfert de données pour les réseaux informatiques, c'est-à-dire le volume de données pouvant être transporté d'un point à un autre dans un laps de temps donné (généralement une seconde).
Dans cette acception, la bande passante d'un réseau est généralement exprimée en bits par seconde (bit/s). Sur les réseaux modernes, le débit est mesuré en millions de bits par seconde (mégabits par seconde, ou Mbit/s) ou en milliards de bits par seconde (gigabits par seconde, ou Gbit/s).

Pour ce faire, j'ai donc utilisé la librairie nftables déjà utilisé dans le code. Nftables est une librarie délivrée par Netfilet qui permet d'intercepter les échanges entre un ordinateur et internet et de les traiter différemment. Il s'agit donc d'un élément-clef de la technologie Snowpack. En effet, sans cela, la connexion à internet n'intégrerait pas le réseau Snowpack.

Une option pour limiter la bande passante existe dans la librairie c'est donc ce que j'ai utilisé. Je pouvais limiter par nombre de paquets (un paquet est le message envoyé par l'ordinateur d'un utilisateur sur internet) par seconde, jour, mois ou année et j'ai finalement choisi de limiter la bande passante à 1 MEGA par seconde ou mégabits/secondes. En effet, il n'y a pas vraiment d'intérêt à limiter au mois ou à l'année.

Grâce à cette mission, j'ai pu comprendre comment python fonctionnait ainsi que le code existant et j'ai appris à utiliser la librairie nftables en python.

V.4. Troisième mission, la librairie nftables

Avant de commencer la réécriture en Cpp, il était nécessaire de remplacer ou d'adapter la librairie nftables et permettre le bon fonctionnement de la platforme. Les libraires ou bibliothèque logicielle sont un ensemble de codes déjà écrits. Elles permettent donc de gagner énormément de temps et de moyen. Il me fallait donc comprendre l'utilisation qui en était faite en python, trouver une librairie permettant des actions similaires à nftables en C++ ainsi que documenter ces recherches.

En effet, puisque Snowpack est financé par l'argent public, il est important de documenter les recherches de ce type pour que les investisseurs mesurent l'avancement du produit.

J'ai donc cherché un moyen de remplacer nftables en c++ et j'ai trouvé 3 solutions, deux libraires que sont libnftnl et libnftables. La troisième solution n'est pas une librairie mais puisque l'on peut effectuer des commandes nft par terminal, il est possible d'effectuer ces commandes en les faisant executer par le système d'exploitation.
J'ai ensuite effectué des tests pour comparer la vitesse d'exécution de chacune de ces trois méthodes tout en les documentant en anglais puisqu'un de mes collègues ne parlait pas français (disponible traduit à la fin de ce document).
Et j'ai finalement choisi d'utiliser libnftables qui bien que n'étant pas la plus rapide était un bon compromis entre la vitesse d'exécution et la simplicité d'écriture.

V.5. Quatrième mission, recodage

Une fois que j'ai eu fini avec nftables, j'ai pu commencer la version Cpp du code. Il me semble d'ailleurs important de noté ici qu'en parallèle de mon travail, le reste de l'équipe de développement travaillait à refaire l'architecture de façon à rendre le réseau plus rapide et efficace.

j'ai alors commencé par coder des fonctions de bas niveau assez basique n'étant pas censées dépendre de l'architecture en prenant comme exemple ce qui existait déjà en python. Ces fonctions sont très utiles puisqu'elles sont utilisées de nombreuses fois et doivent donc être optimisées le plus possible. La première de ces fonctions est une des plus importantes puisqu'elle permet de crypter chaque "flocon" et créer ainsi une couche de chiffrement rendant impossible l'accès aux données.

Pour expliquer rapidement le principe, selon wikipédia, un message A va être séparé en deux parties B et C grâce à une opération de chiffrement. Pour reconstituer le message A, il est nécessaire de posséder les deux messages B et C. Bien que le principe soit relativement simple, les données sont protégées avec un haut niveau de sécurité.

J'ai personnelement beaucoup aimé cette première expérience de C++ puisqu'elle m'a permis de mettre en oeuvre et de me renseigner sur un principe cryptographique que je trouve très intéressant.

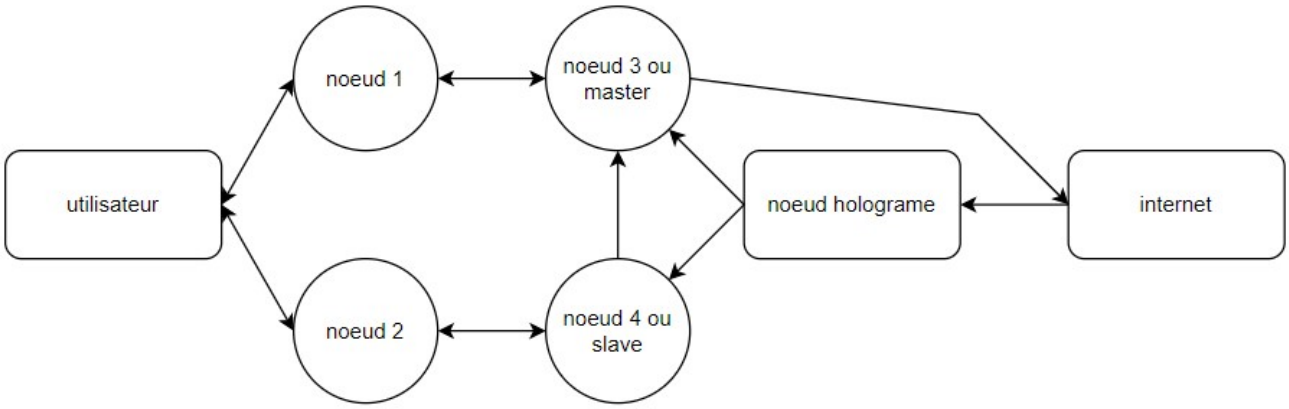
J'ai ensuite continué à coder des fonctions de bas niveau et bien que ces fonctions soit très utiles pour le projet dans sa globalité, il était dur de concevoir totalement la place de ces fonctions dans un projet encore inexistant.
Je n'étais à ce moment là pas vraiment efficace car encore peu à l'aise avec le C++ et j'ai donc dû recommencer certaines choses plusieurs fois. En effet comme je travaillais avec un langage que je ne connaissais pas bien, il m'est arrivé de traduire certaines fonctions d'un langage à l'autre et c'est un problème car je n'exploitais pas pleinement le Cpp.

Je regrette donc certaines décisions prises durant cette mission mais je pense avoir beaucoup appris de mes erreurs et je sais que je ne les reproduirais pas. Je regrette aussi de ne pas y avoir participé à la refonte de l'architecture car ça me semblait très intéressant et j'ai par la suite dû passer du temps à comprendre la nouvelle architecture.

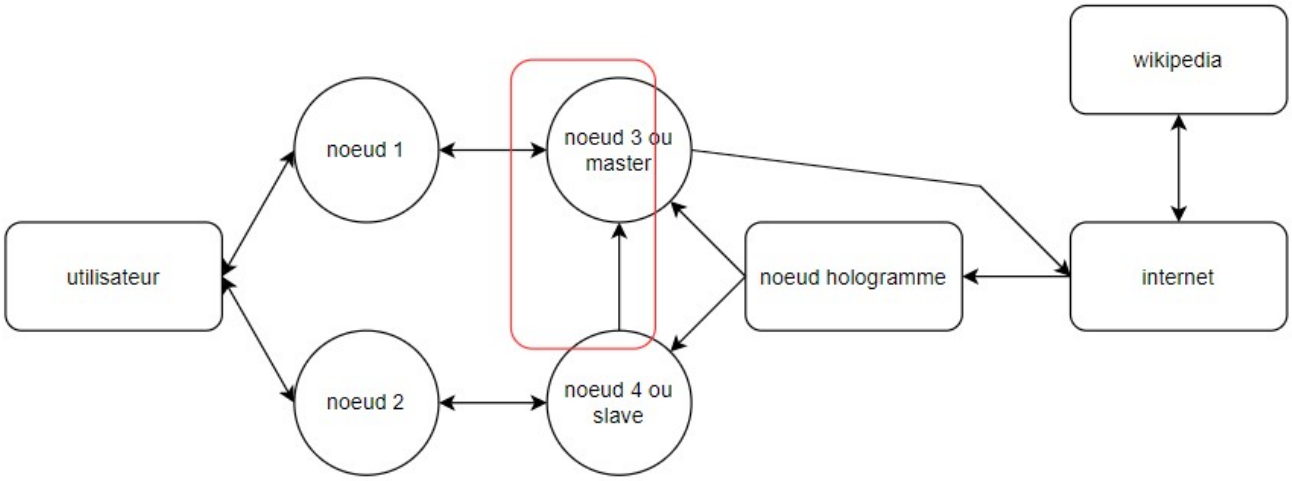
V.6. Cinquième mission, connection entre le noeud master et le noeud slave

Une fois la nouvelle architecture au point, j'ai pu rentrer dans le vif du sujet et pendant que mes collègues plus à l'aise avec l'architecture avançaient de leurs côtés, ils me confièrent la tâche de connecter entre eux les noeuds master et slave, un peu d'explications :

Avant de pouvoir communiquer avec internet, il faut construire le réseau. c'est une tache assez délicate puisqu'il faut connecter chaque noeud entre eux en suivant un ordre donné par l'utilisateur. Le programme côté utilisateur doit se connecter en premier aux deux premiers noeuds, puis va demander à ceux-ci de se connecter aux deux noeuds suivants.



Mais dans un système de plusieurs utilisateurs, pour être sûr que la route de deux utilisateurs ne se mélange pas, il faut établir une connexion entre les noeuds 3 et 4 avant d'envoyer plus de messages, ce dont je me suis occupé.



Pour ce faire, les noeuds 3 et 4 vont se voir confier un message d'auto découverte par les noeuds 1 et 2. Ces deux messages sont complémentaires et lemessages original ne peut être retrouvé qu'avec chacun des deux messages. Le noeud 3 aussi appeler master vont donc chercher à se connecter avec le noeud 4 ou slave. Pour ce faire, le noeud 3 va se connecter à chaque noeud du réseau.

Mais, pour pouvoir se connecter à chacun des noeuds du réseau, il faut avoir les adresses Ip de chacun de ceux-ci. Pour ce faire, il est possible de récupérer la liste de ceux-ci sur l'api de Snowpack. Selon redhat, Une API, ou interface de programmation d'application, est un ensemble de définitions et de protocoles qui facilite la création et l'intégration des applications. Les API permettent à votre produit ou service de communiquer avec d'autres produits et services sans connaître les détails de leur mise en œuvre.

Je devais donc d'abord récupérer la liste des noeuds afin d'avoir accès à toutes les adresses Ip, mais aussi être capable de modifier les données de certains d'entre eux. Par exemple, quand un noeud est en activité, c'est-à-dire qu'il est utilisé par quelqu'un sur le réseau, cela doit se voir sur la plateforme. Et au contraire, si un noeud se déconnecte (soit dû à une action de l'utilisateur, soit dû à une cause externe), il faut également actualiser ses informations sur la plateforme.

Le master va ensuite envoyer son message d'auto-decouverte à chacun d'entre eux puis fermer toutes les connections.

le Slave, quant à lui, se contente d'attendre le message d'auto-decouverte de son master. Une fois celui ci reçu, le slave saura quel neoud du reseau est son master et va se connecter avec lui. Ils ne leur restent plus qu'à se conencter avec le node holograme.

V.7. Sixième mission, SNOWPACK côté utilisateur

Ma dernière mission fût celle qui m'a la plus mise au défit car je devais m'occuper de toute la partie utilisateur du code. Mes missions précédentes avaient été rendues plus simple par mes collègues avec qui je travaillais mais cette fois ayant travaillé de façon beaucoup plus autonome la tâche fût plus ardue.

Quand l'utilisateur construit son réseau, il choisit chacun de ses noeuds ainsi que leur place dans le réseau. La première chose à faire est donc de connecter l'utilisateur avec les deux premiers noeuds. Il

faut ensuite garder en mémoire quels noeuds sont directement connectés avec l'utilisateur puisqu'il faut envoyer un message simultanément à chacun des deux noeuds. La seconde étape est alors de donner un rôle à chacun de ces deux noeuds. Le message alors envoyé contient toutes les informations nécessaires aux noeuds 1 et 2 pour finir la construction du réseau. Le noeud 1 sait alors qu'il doit se connecter avec le noeud3 et lui donner le rôle de master et la démarche est similaire pour les noeuds 2 et 4.

Après avoir donné leurs rôles aux noeuds, l'utilisateur ne fait qu'attendre un message du noeud 1 lui signifiant que le réseau est construit. Et bien que cela peut sembler facile au premier abord, c'est en fait beaucoup plus compliqué car il faut s'assurer que chaque étape a été réussie et réagir en conséquence dans le cas contraire.

Cette mission a été pour moi très instructive car elle m'a permise de voir en action de nombreux détails impliqués dans la construction du réseau et de comprendre celui-ci de façon plus globale. En effet, comprendre le réseau de façon générale et voir tous les détails qui permettent sa construction sont deux choses différentes.

VI. Conclusion

Je pense avoir énormément appris au sein de Snowpack. Autant d'un point de vue professionnel et technique que social. En effet j'ai pu découvrir pour la première fois le monde du travail. J'ai aussi pu discuter avec des collègues de cultures différentes passionnés par leur travail.

J'ai appris que la communication était très importante au sein d'une équipe. J'ai donc appris à développer ces compétences autant en français qu'en anglais. J'ai aussi appris qu'il était important de reconnaitre ses erreurs et de demander de l'aide quand on en sens la nécessiter.

J'ai aussi appris à utiliser deux nouveaux langages de programmation et à documenter chacune de mes actions pour que mon travail puisse être facilement utilisé par mes collègues.

Je ne suis pas sûr de vouloir m'orienter vers ce milieu dans le futur, mais je suis tout de même très content d'avoir eu une bonne expérience dans celui-ci et je tiens à remercier encore une fois mes collègues qui ont rendu cela possible.

VII. Réference

VII.1. Documentation technique réaliser durant mon stage

VII.1.a. Pourquoi passer du python au Cpp

Pour augmenter les performances du réseau.

VII.1.b. Comment faire

La première chose ici est de définir si nous voulons utiliser une bibliothèque dynamique ou une bibliothèque statique. bien que réutilisables dans plusieurs programmes, les biblothèques statiques sont verrouillées dans un programme au moment de la compilation. Les bibliothèques dynamiques ou partagées, en revanche, existent sous forme de fichiers séparés en dehors du fichier exécutable. L'inconvénient d'utiliser une bibliothèque statique est que son code est verrouillé dans le fichier exécutable final et ne peut pas être modifié sans une recompilation. En revanche, une bibliothèque dynamique peut être modifiée sans qu'il soit nécessaire de la recompiler. Parce que les bibliothèques dynamiques vivent en dehors du fichier exécutable, le programme n'a besoin de faire qu'une seule copie des fichiers de la bibliothèque au moment de la compilation. Alors que l'utilisation d'une bibliothèque statique signifie que chaque fichier de votre programme doit avoir sa propre copie des fichiers de la bibliothèque au moment de la compilation.

L'inconvénient d'utiliser une bibliothèque dynamique est qu'un programme est beaucoup plus susceptible de se casser. Si une bibliothèque dynamique par exemple devient corrompue, le fichier exécutable peut ne plus fonctionner. Une bibliothèque statique, cependant, est intouchable car elle vit à l'intérieur du fichier exécutable. Un autre avantage de l'utilisation de bibliothèques statiques est la vitesse au moment de l'exécution. Parce que son code objet (binaire) est déjà inclus dans le fichier exécutable, plusieurs appels à des fonctions peuvent être traités beaucoup plus rapidement que ceux d'une bibliothèque dynamique.

https ://medium.com/@StueyGK/static-libraries-vsdynamic-libraries-af78f0b5f1e4

Puisque nous voulons gagner en performances, il est préférable d'utiliser une bibliothèque statique.

VII.1.c. Pourquoi utiliser nftables

Nftables est un sous-système du noyau Linux assurant le filtrage et classification des paquets réseau/datagrammes/trames. Il est disponible depuis le noyau Linux 3.13 publié le 19 janvier 2014. Nftables remplace les anciennes parties iptables de Netfilter. Parmi les avantages de nftables par rapport à iptables, il y a moins de duplication de code et une extension plus aisée à de nouveaux protocoles. Nftables est configuré via l'utilitaire d'espace utilisateur nft, tandis que les outils hérités sont configuré via les utilitaires iptables, ip6tables, arptables et ebtables frameworks. Nftables utilise les blocs de construction de l'infrastructure Netfilter, telle que les crochets existants dans la pile réseau, le système de suivi des connexions, la mise en file d'attente de l'espace utilisateur composant et sous-système de journalisation.

https ://en.wikipedia.org/wiki/Nftables

Chez un utilisateur normal, un paquet passe par la route préroutage → entrée → sortie, puis est mis en file d'attente pour être traité dans l'application Snowpack.

VII.1.d. Quel solution choisir

- Utiliser les commandes nft directement dans le système d'exploitation.

En effet, puisque nft existe en ligne de commande, il peut etre executé en C++ avec la fonction system().

Positif : très simple a utiliser.
Négatif : très lent.

- Utiliser la librairie Cpp libnftnl.

Libnftnl est une bibliothèque d'espace utilisateur fournissant une interface de programmation netlink de bas niveau au sous-système nftables dans le noyau. La bibliothèque libnftnl était auparavant connue sous le nom de libnftables. Cette bibliothèque est actuellement utilisée par nftables et est basée sur la librairie libmnl.

Positif : très rapide.
Négatif : relativement compliqué a utiliser.

Si nous voulons vraiment utiliser cette bibliothèque, nous devons comprendre et trouver où et quelles valeurs doivent être modifiées pour créer exactement les mêmes règles que le code python.

- Utiliser la librairie Cpp libnftables.

Libnftables est une bibliothèque nftables de haut niveau qui est censée être utile pour l'interface avec nftables. cette bibliothèque est basée sur libnftnl.

Positif, rapide et simple d'utilisation.

VII.1.e. Tests

VII.1.f. Premier test

Pour le premier test, j'ai créé une table, une chaine et 15 règles (nft add rule OUTPUT-SNOWPACK FILTRE tcp dport 22 counter) puis j'ai supprimé la table. J'ai ensuite répété cette action 1000 fois pour avoir une rapide comparaison entre toutes les possibilités de nftables. Voici les résultats.

- système, 53 ms.
- libnftnl, 0.2 ms.
- libnftables, 4 ms.
- ancienne version python, 137 ms.
- version python actuelle, 25 ms.

Un autre test est fait ensuite pour voir le temps qu'il faut pour supprimer ces règles. Ce test créé les mêmes règles, puis les supprime mais le temps est calculé uniquement pour la partie qui supprime les règles. Voici les résultats.

- système, 30 ms.
- libnftnl, 0.09 ms.
- libnftables, 3 ms.

VII.1.g. Second test

Pour le second test, j'ai créé une table, une chaine et 15 règles (nft add rule OUTPUT-SNOWPACK FILTRE tcp dport 22 counter). J'ai ensuite répété cette action 1000 fois avant de supprimé la table. Ce test est fait pour voir si créer la table chaque fois que nous créons les règles est plus rapide que de créer la table et d'ajouter 15x1000 règles. Voici les résultats.

- système, 39 ms.
- libnftnl, 2,5 ms.
- libnftables, 7 ms.
- version python actuelle, 25 ms.

VII.1.h. Troisième test

Pour le troisième test, j'ai créé une table, une chaine et 15 règles (nft add rule OUTPUT-SNOWPACK FILTRE tcp dport i counter). J'ai ensuite répété cette action en incrémentant 1000 fois avant de supprimé la table. Ce test est fait pour s'assurer que créer plusieurs fois la même règle n'est pas plus lent que de créer plusieurs règles. Voici les résultats.

- système, 44 ms.
- libnftnl, 5 ms.
- libnftables, 10 ms.
- version python actuelle, 27 ms.

VII.1.i. Résultats des tests

Ces tests demontrent qu'utiliser le système est bien trop lent et qu'a l'inverse, libnftnl est bien plus rapide.

VII.1.j. Conclusion

Libnftnl est plus rapide mais vraiment plus compliqué à utiliser. Et libnftables est plus rapide que la version python de nftables donc nous devons décider laquel nous voulons utiliser. Pour le moment, nous pouvons déjà utiliser libnftables mais libnftnl exige que l'on passe du temps sur la bibliothèque en elle meme. Libnftables me semble donc etre un bon compromis entre la vitesse et la simplicité.