Rapport

Nom du projet : NetSmooth
Zorzi,Lucas,Jicquel

Résumé : Nous souhaitons créer un simulateur de réseaux palliant les problèmes de Marionnet (simulateur actuellement utilisé en ASR).

24 mars 2017

Table des matières

1 Problématique **3**
1.1 Marionnet 3

2 Fonctionnalités attendues **4**

3 Solutions **4**
3.1 LXC 4
3.2 Le choix du langage 5
3.3 Le choix de la bibliothèque graphique 5

4 Fonctionnement de LXC **5**
4.1 Les containers, leur fonctionnement 5
4.2 Fichier de configuration 6

5 Architecture du programme **7**
5.1 Modèle Vue Contrôleur 7
5.2 Interactions entre le MVC et les containers 11

6 La communication entre les machines **12**
6.1 Les bridges 12
6.2 L'algorithme 12

7 Conclusion **16**
7.1 Objectifs atteint ? 16
7.2 Difficultés rencontrées 16
7.3 Critiques 16
7.4 Avis personnel 16
7.5 Statistiques de développement 16

1 Problématique

La virtualisation est le fait de créer une version virtuelle d'une entité physique, ces versions virtuelles sont alors appelées Machines virtuelles ou VM (Virtual Machine). Nous pouvons prendre comme exemple Virtual Box, qui est un outil de virtualisation permettant d'émuler un système d'exploitation (linux ou autre). Les différentes ressources de la machine hôte sont alors partagées et allouées dynamiquement aux différentes machines virtuelles par des logiciels appelés hyperviseur.

La virtualisation peut être utile dans diverses situations. En effet, virtualiser permet de tester un environnement afin de trouver ses meilleurs caractéristiques avant de le créer. De plus, la virtualisation peut être utilisée à des fins pédagogiques. Dans le cadre d'études informatiques, il peut être pratique de simuler des réseaux dans lesquels des machines dialoguent. Marionnet est un exemple de logiciel de simulation de réseaux.

1.1 Marionnet

Marionnet est un simulateur de réseau. Voici un exemple d'utilisation : On souhaite simuler un réseau constitué de 2 sous-réseaux. Chaque sous-réseau contient 3 machines. On relie donc chaque groupe de 3 machines avec un hub, puis les deux sous-réseaux avec une passerelle (gateway).

On peut ensuite configurer les tables de routage, les adresses ipv4, ipv6 etc. pour simuler des échanges entre les machines.

Cependant, après avoir utiliser un tant soit peu Marionnet, il s'avère que plusieurs défauts gênent son utilisation :

— crashs assez fréquents ;
— impossible de changer les configurations des machines en marche ;
— processus d'arrêt des machines trop fastidieux ;
— interface peu ergonomique ;
— logiciel trop gourmand en ressources.

Face à ces problèmes, nous avons donc décidé de **créer un nouveau simulateur de réseaux**.

2 Fonctionnalités attendues

Avant le codage, nous désirions assurer plusieurs fonctionnalités :
— Créer/supprimer des **entités** (ordinateurs,passerelles ou des hubs)
— Relier un ordinateur ou une passerelle avec un hub avec un cable
— Le fait de supprimer une entité supprime ses cables
— Supprimer un cable
— Déplacer une entité
— Afficher la configuration d'une entitée
— Modifier la configuration d'une entitée
— Eteindre/allumer une entitée
— Lancer des terminaux
— Sauvegarder/charger une session

3 Solutions

Pour mener à bien le projet, nous devions trouver le moyen de simuler un ordinateur ou une passerelle. Notre tuteur nous a donc conseillé **LXC** (**L**inu**X** **C**ontainer).

3.1 LXC

LXC est un outil de virtualisation permettant de créer des environnements virtuels, différents systemes d'exploitations sont mis à disposition (Ubuntu, Debian...). Ces Environnements sont appelés containers. Le partage des ressources est assuré par l'outil Cgroups du noyau, qui permet de limiter, compter et isoler l'utilisation des ressources.

Chaque environnement virtuel est isolé, de la même manière que l'isolement d'un programme avec "*chroot*" : chaque environnement est créé de manière à ce qu'il n'ait pas accès au système d'exploitation de la machine hôte. En revanche, la machine hôte, elle, a accès aux machines virtuelles. Cette isolation entre les machines virtuelles et la machine hôte, permet de garantir une certaine sécurité.

LXC possède sa propre API, notamment en C/C++, ce qui permet de l'intégrer à un projet efficacement. Il est plus léger que des technologies comme Docker (qui hérite de LXC) et convenait donc à notre utilisation.

3.2 Le choix du langage

Nous avons choisi de coder notre projet en **C++**. En effet, LXC possède une API dans ce langage, et nous souhaitions utiliser un langage orienté objet, langage plus adapté à la création d'interfaces graphiques. De plus, c'était une occasion d'apprendre un nouveau langage.
Nous avons aussi utilisé des scripts en **Shell** afin de modifier la configuration de la machine hôte [1].

3.3 Le choix de la bibliothèque graphique

Nous ne connaissions pas de bibliothèques graphique en C++. Nous avons donc fait quelques recherches et nous nous sommes tournés vers Qt de façon assez arbitraire.

4 Fonctionnement de LXC

4.1 Les containers, leur fonctionnement

Les environnements virtuels, ou, containers (conteneurs en francais) doivent, en premier temps, être créés à l'aide de la commande "*lxc-create ...*" (voir notices pour plus de précisions sur les commandes). De nombreux systèmes d'exploitation seront alors proposés, (nous avons choisi de prendre Debian, Jessie, i386). Par défaut, les machines créées ne sont pas configurées : elles n'ont pas d'interfaces, elles n'ont pas de compilateur, et les utilitaires préinstallés sont très rudimentaires (pas de ping, ifconfig, tcpdump...).

Chaque container possède un fichier de configuration situé à l'emplacement suivant : "*/var/lib/lxc/<nom du container>/config*". Il est possible de configurer de nombreux aspect du container dans ce fichier (par exemple : modifier les variables d'environnement, ou changer le nom d'hôte ("*hostname*") du container. Voir "*man lxc*" pour en savoir plus). Ce fichier va notamment permettre de paramétrer la configuration réseau du container à son démarrage (c'est ce qui va nous intéresser en priorité avec ce fichier).

Les containers se lancent avec la commande "*lxc-start ...*" ; il est préférable de les lancer en démons (en arrière-plan), puis d'y "*attacher*" un terminal avec "*lxc-attach*", afin d'être connecté en super utilisateur (ou root) car, premièrement, par défaut, aucun profil

1. Machine sur laquelle est lancée le logiciel

d'utilisateur n'est créé sur le container, et, de plus, cela permet d'avoir l'entier contrôle du container afin de, par exemple, modifier son adresse ipv4.

4.2 Fichier de configuration

Comme expliqué plus haut, les paramètres réseaux du container peuvent être modifiés via le fichier config. Dans ce fichier, chaque "*block*" permet de définir une interface. Un block commence toujours par la définition du type de réseau, nous allons donc nous intéresser tout d'abord au type de réseau ("*lxc.network.type*"). Plusieurs types de réseaux sont disponibles, mais, nous allons choisir le type veth, qui signifie Virtual Ethernet, ce type permet d'établir un lien entre l'interface virtuelle du container et un pont, préalablement créé sur la machine host (nous verrons cette liaison plus en détail par la suite).

Ce fichier permet aussi d'établir des adresses ipv4 ("*lxc.network.ipv4*"), ipv6 ("*lxc.network.ipv6*"), et MAC ("*lxc.network.hwaddr*"), ainsi que leur Broadcast. Cela permet de mettre en place le réseau virtuel avant de lancer les machines ; bien que ces paramètres puissent être modifiés une fois la machine lancée à l'aide de l'outil "*ifconfig*". Attention, les tables de routage ne peuvent pas être configurées d'avance, il faut les configurer une fois le VE lancé.

Le dernier paramètre que nous verrons pour ce fichier est le lien ("*lxc.network.link*") ; ce paramètre va permettre d'indiquer à quel bridge [2] nous voulons connecter notre interface.

2. expliqué plus tard

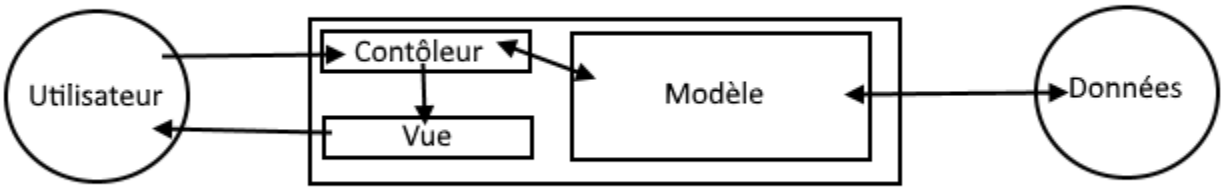
5 Architecture du programme

5.1 Modèle Vue Contrôleur

Pour notre projet nous avons décidé d'utiliser le modèle d'architecture MVC (Modèle Vue Contrôleur). Nous avons choisi le modèle MVC car il permet au projet d'être modulable, d'ajouter plus facilement de nouvelles fonctionnalités et de modifier une partie du logiciel sans affecter les autres. Pour nous il simplifie le développement et facilite la compréhension du code.

Ce modèle découpe l'architecture d'un projet en trois parties :

— -Le Modèle correspond aux données et à leur accès, il communique avec le contrôleur.
— -Le Contrôleur reçoit et traite les données avant de les envoyer à la vue, pour les afficher, ou au modèle, pour les stocker.
— -La Vue présente les données reçues par le contrôleur et permet à l'utilisateur d'interagir avec.

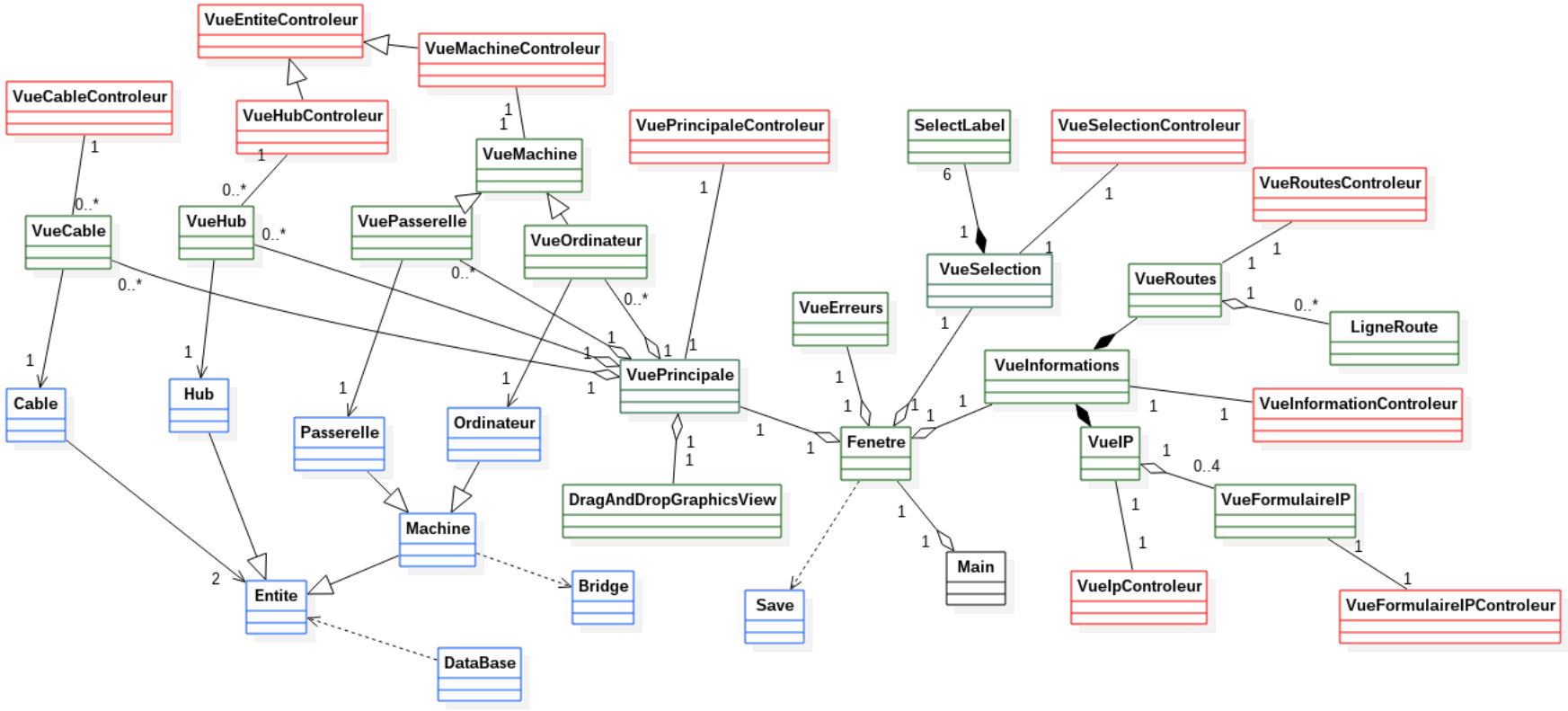


Le système fonctionne de la façon suivante :

1. L'utilisateur, via ses actions sur l'interface, va faire une demande au contrôleur. Le contrôleur va traiter toutes les demandes de l'utilisateur.
2. Pour les traiter, le contrôleur va si besoin envoyer une requête au modèle.
3. Le modèle va recevoir la requête et envoyer une réponse au contrôleur.
4. Le contrôleur reçoit la réponse et va mettre à jour la vue.
5. La vue modifiée est envoyée à l'utilisateur.

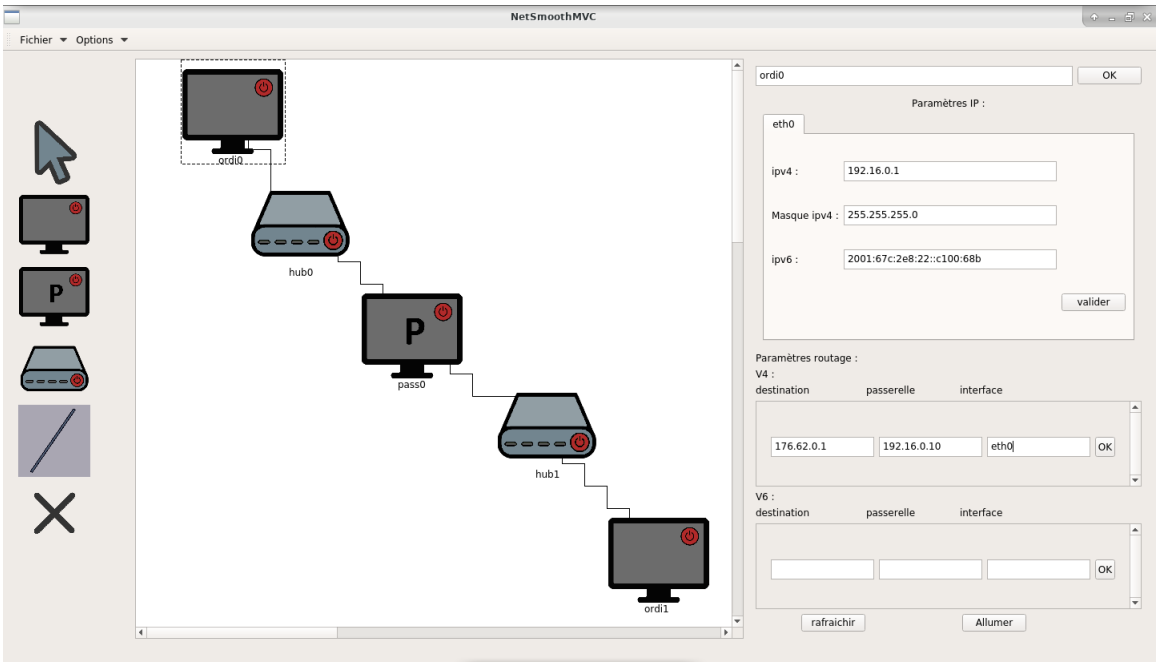
Dans notre projet, le modèle correspond aux données liées aux environnements virtuels (les adresses IP, les routes, ...), le contrôleur correspond aux classes qui vont écouter et réagir aux évènements de la vue, la vue correspond à l'interface graphique.

Diagramme d'analyse



Rouge : Contrôleurs
Vert : Vues
Bleu : Modèles

Le logiciel a une seule interface :

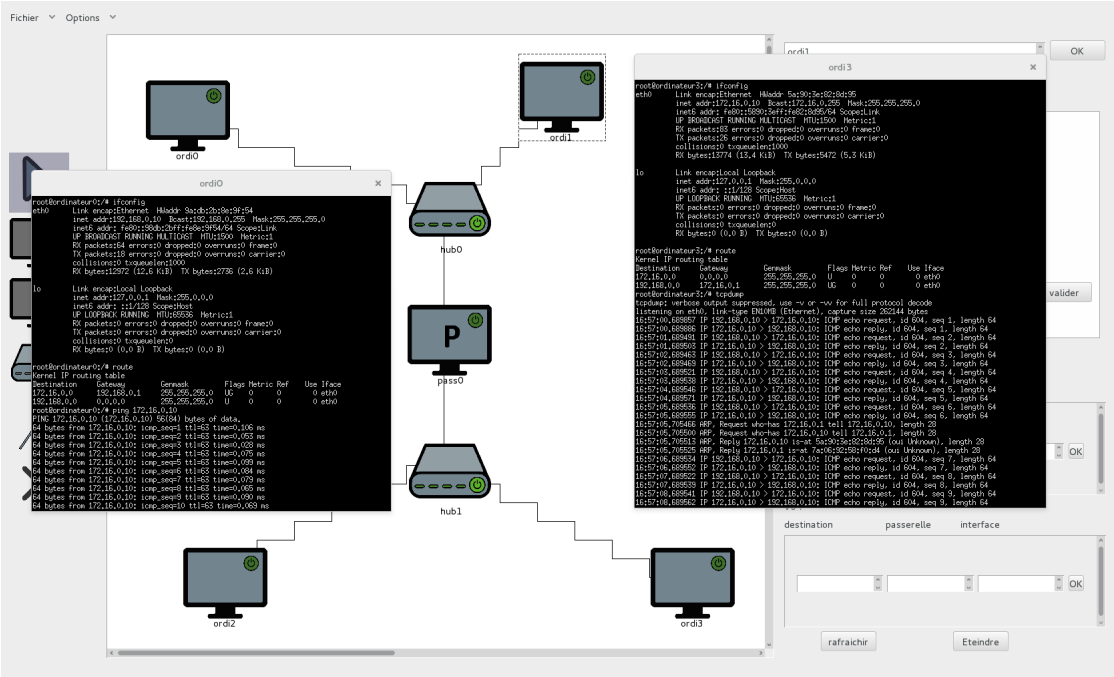


Interface de NetSmooth

L'interface graphique de notre logiciel est décomposée en 4 parties distinctes :

— La zone de sélection (à gauche), qui permet à l'utilisateur de choisir l'élément ou l'outil qu'il veut utiliser.
— La zone principale (au milieu), qui représente le réseau virtuel que l'utilisateur va créer.
— La zone d'information (à droite) , qui permet de récupérer et de modifier les paramètres d'un ordinateur ou d'une passerelle présente dans la zone principale.
— La barre d'outils (en haut), qui permet à l'utilisateur d'effectuer certaines actions comme la sauvegarde de son travail ou le chargement d'un réseau enregistré.

Voici un exemple avec un ping entre deux machines capturé avec tcpdump :



Ping entre deux machines

5.2 Interactions entre le MVC et les containers

Les containers sont installés sur la machine hôte. Il faut donc maintenant les relier avec le Modèle Vue Contrôleur.
Pour ce faire, nous avons contenu toutes les intéractions entre les containers et le MVC dans le Modèle. En effet, chaque modèle de machine (ordinateur/passerelle) possède une référence vers un container. Ainsi, le modèle de chaque machine permet de modifier l'état ou la configuration de son container.

6 La communication entre les machines

Pour que les machines virtuelles puissent communiquer entre elles, il est nécessaire de les connecter entre elles par un **Bridge** (ou Pont en français).

6.1 Les bridges

Les bridges (ou pont en francais) sont des équipements réseaux qui permettent de relier deux (ou plus) interfaces de manière complètement transparentes : en observant les paquets qui transitent, le pont peut connaitre les adresses mac des interfaces, et ainsi, rediriger les paquets. Les ponts peuvent par exemple, être utilisés pour rediriger une connexion Ethernet : une machine se connecte en Ethernet, une seconde machine se connecte à la première, elles établissent un pont entre deux de leurs interfaces (une interface de la première machine, et une interface de la deuxième machine), et, ainsi, la deuxième machine peut avoir accès à internet.

Lorsqu'un VE [3] se lance, une interface se créé sur la machine hôte pour chaque interface présente sur le VE (typiquement, les interfaces créées ont un nom qui commence toujours par "*VETH*" suivi de quatre caractères). Ces interfaces sur la machine hôte representent les interfaces de l'environement virtuel, et vont nous permettre de relier les environnements entre eux, avec des **bridges**, créés sur la machine hôte.
Il est donc nécessaire de relier ces interfaces sur la machine hôte à un même bridge pour que les containers puissent communiquer !

6.2 L'algorithme

Dans notre application, les cables sont liés aux bridges, et le fait d'ajouter, ou de supprimer un cable, et surtout d'allumer ou d'éteindre une *Entité*, va modifier toute l'organisation des bridges.

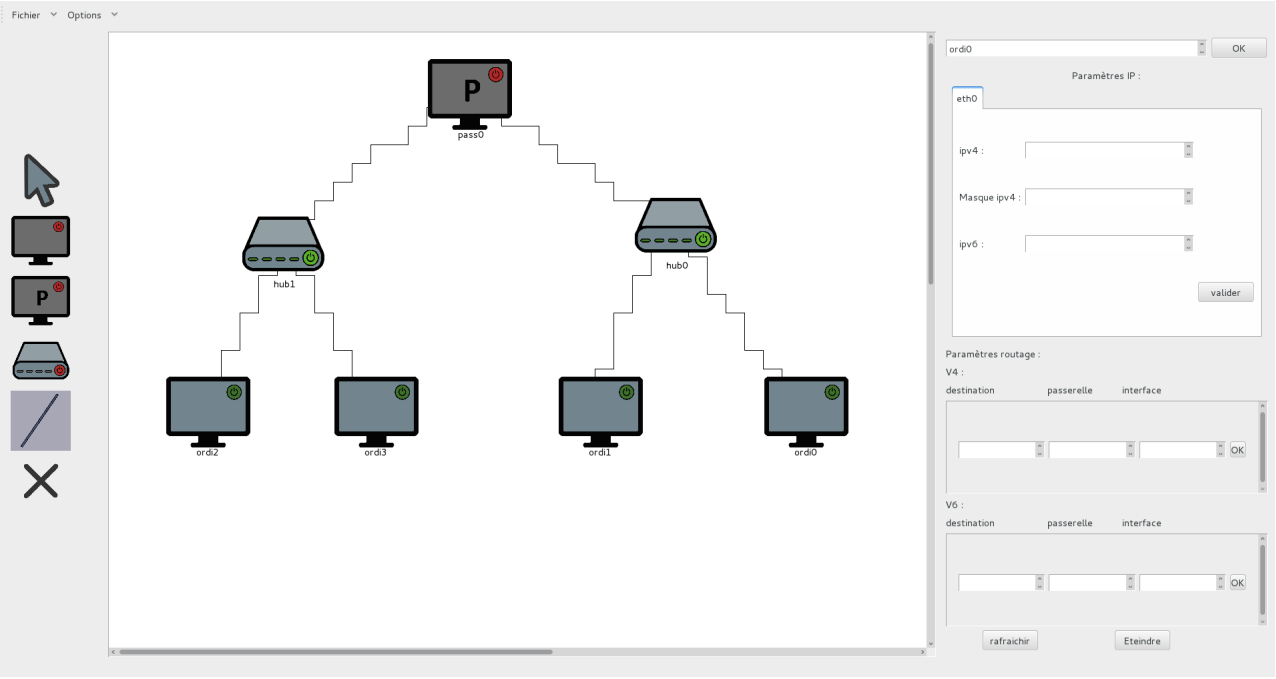
L'algorithme qui permet de modifier l'organisation des **bridges** est assez compliqué, et, nous allons tenter de vous l'expliquer le plus clairement possible dans cette partie.

Avant tout, il faut bien comprendre le mechanisme de NetSmooth : Deux machines connectées entre elles, mais éteintes, ne peuvent pas communiquer. Ainsi, si deux sous-réseaux sont connéctés entre eux par une passerelle, mais que celle-ci est éteinte (comme

3. VE pour Virtual Environnement

sur l'image ci-dessous), ces deux sous-réseaux ne pourront pas communiquer entre eux. Cela signifie qu'il faut **deux bridges** dans ce cas.

Mais, si l'utilisateur allume la passerelle, les deux sous réseaux, et la passerelle doivent tous être connectés à **un seul et même bridge**.



Chaque *Entité* possède un *Bridge initial* et un *Bridge actuel*, le bridge initial est fixe, et est assigné a l'*Entité* lors de sa création. Quant à lui, le Bridge Actuel va dépendre des connexions de la machine : il équivaut au bridge auquel est connectée l'*Entité*.

Il y a un ordre de priorité au niveau de l'adoption des bridges : La passerelle est la plus forte, puis, le Hub, et enfin, l'Ordinateur (Passerelle > Hub > Ordinateur). Ainsi, si un *Hub* est connecté à une *Passerelle*. Allumer les deux entités aura pour conséquence extbfl'adoption du bridge Initial de la passerelle par le Hub.

Dans le code, une fonction récursive permet de circuler entre toutes les machines allumées d'un sous-réseau, et de les faire toute adopter un méme bridge Actuel. Nous appellerons cette fonction *TOTO*.

Lorsqu'une Entité s'allume, elle regarde, parmi toutes les Entités dans son voisinage [4] qui est l'Entité de plus forte prioritée ; si c'est elle, elle impose son bridge initial a toutes les machines de son voisinage, via la fontion $TOTO$; ainsi, elles ont toutes un Bridge Actuel identique, qui est le bridge initial de cette Entité :. Au contraire, si elle n'est pas la machine de plus forte priorité, elle adopte le bridge actuel de la première *Entité* supérieure qu'elle voit (toujours dans son voisinage), et s'applique la fonction $TOTO$, ce qui aura pour effet de modifier tous les bridges actuels de toutes les entités dans le voisinage de celle ci.

Lorsqu'une Entité s'éteind, elle regarde autour d'elle, et pour chaque sous-réseau dans son entourage, elle le force à adopter le bridge initial de l'*Entité* qui est directement connéctée à elle.

4. c'est a dire, qui sont connéctées a elle, et qui sont allumées

Voici l'équivalent de ces explications en pseudo-code :

```
TOTO = algorithme permettant de connecter toutes les machines appartenant au même
       sous réseau à un même bridge (et donc, de modifier leur bridge actuel)

allumer Entité()
{
    regarder dans mon voisinage()
    si Je suis l'Entité de plus haute prioritée
    {
        pour chaque Entité 'e' de mon entourage,
            appliquer TOTO a 'e', avec mon Bridge initial
    }
    sinon
    {
        Bridge b+ = bridge Actuel de la premiere Entité de plus
                    haute prioritée que moi, que je vois
        appliquer TOTO sur moi, avec le bridge b+
        /* ce qui touche aussi toutes les machines connectées a moi,
         * qui n'ont pas déja le bridge b+ en bridge Actuel
         */
    }
}

eteindre Entité()
{
    regarder dans mon voisinage()
    pour chaque Entité 'e' de mon entourage,
        appliquer TOTO à 'e', avec le bridge initial de e
}
```

7 Conclusion

7.1 Objectifs atteint ?

Nous avons réussi à remplir les objectifs que nous nous étions fixés.

7.2 Difficultés rencontrées

— Compréhension et prise en main de LXC
— Utilisation bibliothèque Qt
— Compatibilité entre les versions de Qt

7.3 Critiques

Les containers conservent leur modifications (création de fichier, etc.). Cela sera résolu avec les **snapshot**. Des machines connectées à un même hub ne peuvent pas intercepter le trafic avec tcpdump par exemple.

7.4 Avis personnel

Ce projet nous a permis avant tout d'apprendre à utiliser des outils utiles pour développer un logiciel, tels que
— GitHub
— LaTex
— C++
— Qt
Nous nous sommes aussi familiarisé avec un IDE, **Qtcreator**.
Nous avons développé notre **rigueur**, tant dans notre manière de programmer que dans notre gestion du temps.

7.5 Statistiques de développement

Nombres de lignes de code : **5642** (fichiers sources, headers et scripts)

Nombres d'heures : environ **315** heures/personnes

Nombres de commit sur Github [5] : **296** commits

5. lien du dépôt github : https ://github.com/Jicquel/NetSmooth