Manuel d'utilisateur : Topologie d'un réseau virtualisé en anneau en utilisant des sockets ethernet

Le projet est constitué de plusieurs modules Python :

- Tokenmain.py : Programme de lancement
  Configuration et lancement d'un seul ou de plusieurs ordinateurs virtuels
- Computer.py : Classe
  Un ordinateur virtuel qui instancie une pile de réseau et qui lance une ou plusieurs applications qui envoient et reçoivent des messages / fichiers / … Il s'agit des « utilisateurs » du réseau en anneau.
- NetworkStack.py : Classe
  Pile de réseau avec ces couches. Une séparation des couches par création des classes indépendantes peut être préférable. L'implémentation de base contient une instance de la classe de couche physique (réseau en anneau virtualisé sur l'ethernet). Plusieurs couches supplémentaires sont actuellement implémentées avec deux fonctions, une pour la pile entrante et une pour la pile sortante (incoming et outgoing).

La configuration initiale contient un ou plusieurs ordinateurs virtuels qui lancent une application d'échange de 1000 messages sur le réseau à plusieurs machines d'une manière systématique. Par exemple,

computer1=Computer(ownIdentifier="A", sendDestinations=["B","C"], masterHost='127.0.0.1', baseport=10000)

va créer une instance d'un ordinateur virtuel qui s'appelle « A » et qui envoie les messages à « B » et « C ». Si vous avez choisi d'utiliser d'autres identifiants, le changement devrait être facile à faire dans le code.

Le fichier à adapter à votre protocole est NetworkStack.py. Il s'agit d'une classe qui implémente l'ensemble de la pile réseau.

Comme dans le cas d'une carte réseau réelle, où une interruption (IRQ) appelle une fonction dédiée pour lire un paquet après sa réception, la couche physique virtualisée appelle la couche réseau 2, en occurrence la méthode « layer2_incomingPDU ». Cette fonction décapsule sa partie et appelle layer3_incomingPDU avec les informations nécessaires, etc. jusqu'à l'arrivée du message à application_layer_incomingPDU, le cas échéant (c'est-à-dire, si le paquet était adressé à une application sur cet ordinateur).

Une application qui souhaite envoyer un message appelle la fonction application_layer_outgoingPDU. En occurrence sur notre configuration initiale, il y a l'application d'envoi de message qui envoie en continu 1000 messages aux ordinateurs spécifiés.

C'est le travail des fonctions « layerx_outgoingPDU »des différentes couches x de réseau d'encapsuler les informations (SDU) progressivement dans les paquets (PDU) et de synchroniser

l'envoi de la machine actuelle avec la retransmission des paquets pour les ordinateurs suivants dans l'anneau dans chaque couche.

Il est important de mettre en évidence la forme de chaque conversion SDU/PDU avant son implémentation. Notamment la couche de démultiplexage / multiplexage doit contenir un tampon qui est capable d'envoyer chaque slot à la couche plus haute (si incoming) ou de rassembler plusieurs arrivées de la couche plus haute pour former une seule PDU sortant (si outgoing).

Vu qu'il y a deux threads qui tournent potentiellement, un qui lance layer2_incomingPDU et l'autre qui lance application_layer_outgoingPDU (et layer4_outgoingPDU en conséquence), vous avez besoin de synchroniser les deux flux à un moment précis. Dans le code initial, cette synchronisation est déjà implémentée en vrac dans le layer3, qui détermine d'une manière aléatoire si un paquet est retransmis ou échangé par un paquet de cet ordinateur.

Maintenant, c'est à vous de jouer : Changez l'identifiant de la machine, ajoutez ou diminuez le nombre de couches, implémentez votre protocole, assurez que les paquets arrivent aux bons ordinateurs !

Si vous arrivez à la question de routage et/ou à la question d'ajout ou de rétraction d'un ordinateur du réseau, vous avez probablement besoin de lire la suite.

Documentation de la couche physique virtualisée

L'anneau virtualisé s'appuie sur des sockets du protocole TCP/IP d'un réseau ethernet. L'anneau est construit en utilisant des ports TCP à partir de la valeur fournie dans le paramètre baseport. Chaque nœud virtualisé (ordinateur virtuel / instance de la classe couche physique) possède deux interfaces avec une bouche et une oreille chacun.

Le premier nœud virtualisé (ordinateur virtuel) écoute (oreille) avec son interface 0 sur le port TCP « baseport ». Le deuxième sur le port TCP « baseport+1 » etc. Si un port n'est pas disponible (ex. utilisé par une autre application), les ports sont testés en ordre croissant jusqu'à l'arrivée à un port ouvert.

Le premier nœud virtualisé parle (bouche) avec son interface 0 au port TCP de l'ordinateur suivant « baseport+1 », le deuxième a son successeur « baseport+2 » et suite jusqu'à l'arrive au dernier nœud qui ferme l'anneau en parlant au « baseport ».

L'anneau le plus petit ne contient qu'un seul nœud virtualisé : il écoute sur le port « baseport » et il parle à « baseport ».

À partir d'un certain nombre de nœuds virtualisés, configurables par « numberOfNodesPerRing » (4 par défaut), un deuxième anneau est créé. Ce deuxième anneau forme une boucle (un anneau) des nœuds virtualisés indépendants du premier anneau. Pour permettre le routage entre les deux anneaux, l'interface 1 du dernier nœud du premier anneau est branchée sur le deuxième anneau. Si on nomme chaque interface sur chaque nœud par un triple de chiffre xyz tel que x signifie l'anneau, y

le nœud dans l'anneau et z l'interface (ex. 530 serait l'interface 0 du quatrième nœud de 6^{e} anneau), la situation par défaut avec 6 nœuds serait :

Anneau 1 : 000 -> 010 -> 020 -> 030 -> boucle à 000
Anneau2 : 031 -> 100 -> 110 -> boucle à 031

Dans cet exemple, le nœud 03 recevrait les paquets d'anneau1 sur l'interface 0 et les paquets d'anneau 2 sur l'interface 1.

L'offset de numéro de port TCP pour l'interface 1 est la valeur de la variable « numberOfNodesPerRing », l'offset par anneau est fixé à 100.

L'implémentation de la couche physique virtualisée se base sur un superviseur qui écoute sur le port « baseport-1 » et qui est démarré dans le constructeur de la classe PhyNetwork.

Chaque nœud virtualisé se connecte au superviseur (qui peut tourner sur l'ordinateur physique local ou distant) lors de l'instanciation de la classe LayerPhy. Le constructeur demande également au master l'insertion dans l'anneau. Un nœud peut sortir de l'anneau avec un appel à la méthode API_leave et rentrer avec un appel à API_enter. Les données sont envoyées grâce à la méthode API_sendData. La réception d'un paquet lance l'appel à la fonction de callback donné au constructeur de la classe.

Plus de détail sur le protocole peut être trouvé dans le code source de PhyLayer.

Le débogage de la couche physique virtualisée est facilité grâce à la classe DebugOut, il suffit d'activer le print dans la méthode « out ».

Les modules / classes :

1. PhyNetwork : La gestion logique du réseau en anneau
2. PhyMaster : La gestion du master qui répond aux nœuds qui souhaitent entrer/sortir de la topologie en anneau et qui demande aux autres nœuds les reconfigurations en conséquence.
3. LayerPhy : La couche physique de réseau en anneau, contient notamment la partie client qui se connecte au master et qui ouvre/ferme les oreilles et qui se connecte/déconnecte aux autres nœuds.
4. TCPServer : Classe de base utilisée par PhyMaster et LayerPhy pour lancer un serveur sur un port TCP
5. TCPClient : Classe de base utilisée par LayerPhy pour les bouches qui se connecte à un serveur sur un port TCP
6. DebugOut : Classe pour faciliter le débogage en centralisant les sorties des messages

Happy hacking !

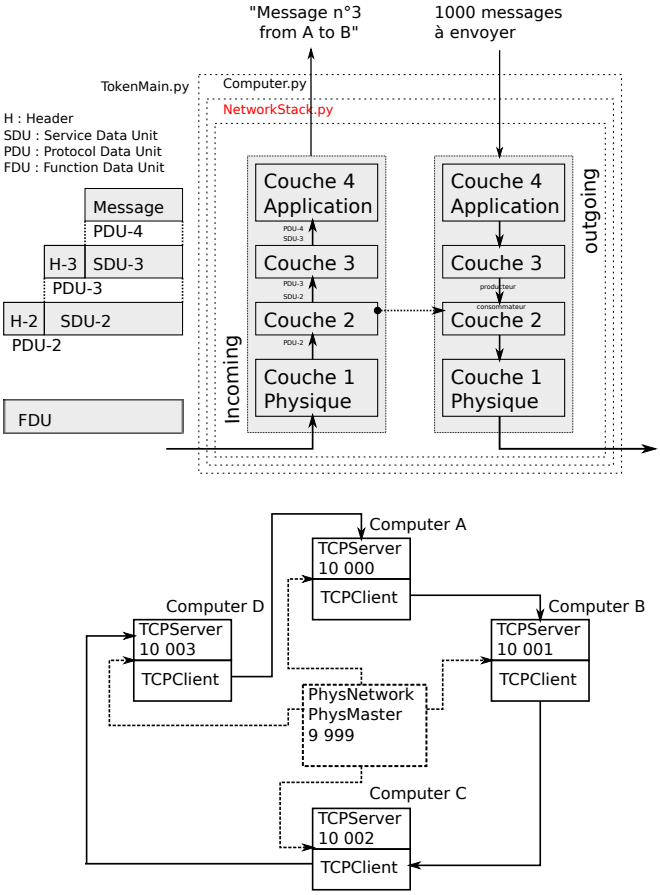

Figure 1 : Principes