Rapport de Projet IN407 (2023 /2024)

Projet réalisé en binôme :

- **MACHTER Massinissa 22304222**
- **KHAZEM Lynda 22304263**
- **Groupe TD04**

Préambules :

Pour exécuter le projet, **faut d'abord exécuter le script « mecanisme_installation.py »**, qui permet d'installer les packages python qu'on utilise comme (Numpy, matplotlib…).A cet instant, **il vous reste qu'a exécuter le script « main.py »** dans le dossier modules.

Nos modules sont tous implémenter dans le dossier modules, avec des codes et techniques qui sont tous commentés.

Conceptualisation :

L'application est conçue pour comparer les différentes stratégies de gestion de flux à l'entrée d'un un réseau de communication. L'architecture de l'application est organisée autour de plusieurs modules.

Structures abstraites de données (structure.py)

Ce module définit les structures de données fondamentales nécessaires au fonctionnement de l'application. Il comprend les classes suivantes :

Buffer: Représente la file d'attente où les paquets sont stockés avant d'être transmis. Le buffer a une capacité maximale et un taux de transmission défini. Les méthodes principales incluent l'ajout de paquets, le retrait de paquets, leur transmission et la gestion de nombre de paquets perdus.

Source: Représente une source de paquets. Cette classe génère des paquets et les envoie vers un buffer spécifié (buffer passée en paramètre dans __init__()).

NB : Dans la partie 1 , les sources partagent le même buffer , par contre dans la seconde partie de projet on considère a chaque source Si un buffer Bi de capacité Ci.

Paquet : Chaque paquet est caractérisé par un identifiant et son temps d'attente dans sa file qui sera calculer après sa transmission par le buffer en considérant son temps d'arrivée dans le buffer (la file).

Interactions entre les Classes :

Les instances de la classe Source envoient des paquets qu'elle crée a partir de la classe Paquet() vers le Buffer (partie1) ou vers leur buffer bi (partie 2) en utilisant la méthode envoie_paquets(n) , n le nb de paquet a envoyés .

Les paquets envoyés à un buffer sont ajoutés à sa file d'attente en utilisant la méthode __iadd__ de la classe Buffer.

La transmission des paquets depuis le buffer vers leur destination se fait en utilisant la méthode transmission de la classe Buffer.

Processus poisson de paramètre lambda :

Dans tout notre projet les paquets arrivent selon un processus de Poisson(Lambda), ici générer a l'aide de la fonction random.poisson(lambda,duree_en_sec) du module numpy,

Où duree_en_sec est le nombre d'arrivée (1sec <-> 1 arrivée) qui varient selon Poisson(lambda)

On a défini également un délai d'attente entre 2 arrivées, en tirant aléatoirement un nombre u entre 0 et 1 et définir le délai = (-1/lambda) *log(u)

Programme de test :

Le module Test.py permet de tester tous nos fonctionnalités de l'application indiqué dans la partie 1, en fonction des paramètres que vous fournissez, comme le taux de transmission, la capacité du buffer, le paramètre lambda et le nombre de sources.

Stratégies de gestion (strategies.py)

Dans cette partie de notre projet, nous examinons l'implémentation de différentes stratégies de sélection de file d'attente qui pour but de limiter le nombre de paquets perdus.

Chaque stratégie est implémentée comme des fonctions distinctes :

choisirFileAvecPlusDePaquets : La file d'attente choisie est celle contenant le plus grand nombre de paquets.

choisirChaqueFileTourDeRole : Un paquet est pris de chaque file d'attente, à tour de rôle.

choisirFileAleatoirement : La file d'attente est choisie de manière aléatoire

Interactions entre les Stratégies et le Système : Le paquet retiré du buffer Bi choisit de la source Si est ensuite ajouté à la file principale du système B (Voir la figure sur sujet).

L'interface graphique :

En utilisant tkinter, nous avons pu créer une interface graphique intuitive et réactive, permettant à l'utilisateur de choisir entre les deux parties du projet, toutes les simulations et l'analyse des performances faite en respectant bien l'arrivée des paquets selon la loi poisson.

Partie 1 :

Dans cette partie, l'utilisateur a deux options :

Visualiser les Performances du Réseau en Fonction de λ :

- L'utilisateur peut visualiser graphiquement le taux de pertes de paquets en fonction du paramètre λ (moyenne d'arrivée de paquets par seconde) qui varient de 1 a 30.
- Faite dans le module « analysePerformances.py » avec la bibliothèque matplotlib .

Lancer la Simulation pour Voir la Dynamique du Système :

- Cette simulation est faite avec deux sources, les liens d'arrivée, le buffer et les liens de transmission, simule l'arrivée d'un paquet dans le buffer, son retrait du buffer et sa transmission.
- Faite dans le module « Visual1.py », en mettant a jour a chaque fois le taux d'arrivée des paquets depuis les sources , les stat(nbr paquets envoyés et perdus) et l'état d'occupation du buffer .

Partie 2 :

Dans cette partie également, l'utilisateur a deux options :

Comparer les Performances des 3 Stratégies :

- Les performances sont mesurées en fonction du temps moyen d'attente d'un paquet dans le système et du taux de pertes de paquets.
- Faite dans le module « analysePerformancesstrategies.py »avec graphisme a barres de matplotlib.

Dynamicité du Système avec Stratégie à Choix :

- Cette simulation est faite avec 2 sources Si chacune avec son buffer bi et son lambda Li, et la file principale B (voir la figure sur le sujet)
- Dans cette partie on laisse le choix à l'utilisateur de choisir la stratégie de gestion des files réalisé dans le module « Visual2.py ».

Les deux modules « analysePerformancesstrategies.py » et « Visual2.py » interagit directement avec le module « strategies.py ».

Conclusion :

Ce projet représente une occasion enrichissante d'explorer en profondeur les fondements de la gestion des files d'attente et de la simulation des réseaux de communication. En accordant une importance particulière à la compréhension théorique et à la mise en pratique, il établit une assise solide pour approfondir les concepts avancés dans le domaine de l'informatique