Rapport Projet Réseaux Avancés

Berkennou Brahim
Boukari Idir
NAIT ALI Idir
Ouksili Samy

SOMMAIRE

➢ ***Objectif :***

Développer une application qui permettra de mettre en œuvre un réseau ad-hoc qui contient 4 machines dont l'une d'entre elles est un serveur DHCP pour ce réseau. Ce dernier sera connecté avec un poste de contrôle (serveur) en utilisant un point d'accès Wifi/Bluetooth.

Partie 1 : Conception du projet

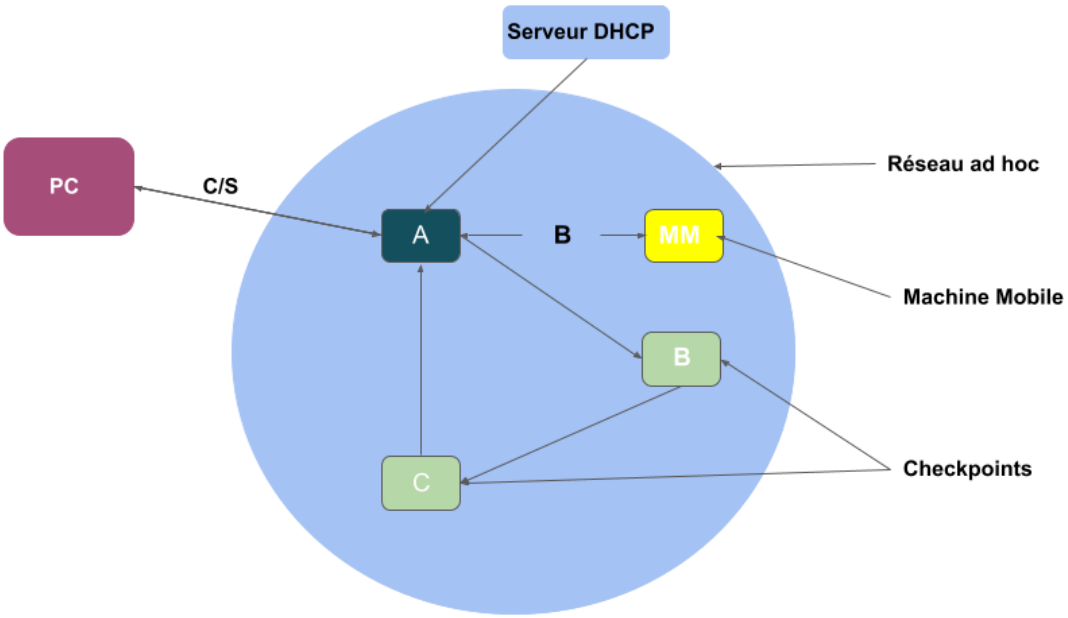
➢ ***Composition du réseau :***

1) Réseau ad-hoc :

- Contient :
  - 3 machines (nœuds) ***A***, ***B*** et ***C*** où ***A*** est un nœud principal (Serveur DHCP).
  - Une machine mobile (laptop/smartphone) ***MM***.
- Les nœuds feront office de checkpoint (point de contrôle) dans notre application.
- En calculant l'estimation de la distance entre la machine mobile ***MM*** et le nœud attendu (***A***, ***B*** ou ***C***), on décidera si le checkpoint est validé et puis demander à ***MM*** d'envoyer le rapport.
- La machine mobile aura pour obligation :
  I. Passer par tous les nœuds dans l'ordre prédéfinis.
  II. Envoyer un rapport au nœud principal ***A*** lors du passage prés de chaque checkpoint(client/serveur) qui sera transmis par la suite au poste de contrôle ***PC***.

2) Poste de contrôle :

- Connecté au nœud principal du réseau ad-hoc via Wifi/Bluetooth.
- Fera office de donneur d'ordre pour lancer la procédure de passage de la machine mobile sur les nœuds en indiquant l'heure et le parcours à suivre.
- Recevra et validera les rapports envoyés par la machine mobile.



➢ ***Utilité :***

Cette application sera destinée pour améliorer l'organisation et l'encadrement du travail du rondier qui effectuera assez souvent des rondes dans un site de surveillance.

Cette dernière permettra aux rondiers de :

- Bien maîtriser leurs rondes et leurs intervalles de temps.
- Pouvoir envoyer les rapports de ronde à propos des checkpoints en temps réel.

Ce système permettra aux responsables de poste de contrôle de :

- S'assurer que le rondier effectue bien les rondes demandées.
- Connaître la position du rondier/agent de sécurité dans le site à n'importe quel moment.
- Intervenir le plus rapidement possible en cas de danger ou de besoin, car le rondier/agent de sécurité sera en mesure d'envoyer un rapport ou un signal de danger en temps réel.

➢ ***Autres domaines :***

Cette application pourra être utilisée dans d'autre domaines, on peut citer les exemples suivants :

I. La livraison : que ce soit livraison de colis ou bien de marchandises où le livreur sera amené à passer par toutes les adresses des clients qu'il a reçues du poste de contrôle et envoyer un rapport de livraison pour chaque adresse → bien livré / pas livré / client absent / déposé dans la boîte aux lettres sans signature/...

II. La santé : plus exactement dans l'organisation des visites de chambres de malades faites par un médecin. Ce dernier doit envoyer un rapport sur l'état de santé de chaque malade visité, ainsi, valider la tournée envoyée par le poste de contrôle et permettre à ce dernier d'être averti de la complétude de la tournée. Par conséquent, il pourra traiter les rapports reçus en temps réel et intervenir à l'égard des malades.

➢ ***Amélioration (Si on aura le temps) :***

- Améliorer l'application en ajoutant des connexions à partir des smartphones via Bluetooth/Wifi
- Associer une page web à l'application pour permettre son utilisabilité sur tous les appareils.

Partie 2 : Étapes de réalisation du projet

- Étape 1 : On a mis en place un réseau ad hoc qui est nécessaire pour le fonctionnement de notre application en appliquant les commandes suivantes sur l'invité de commande du raspberry :

on se place dans le dossier network :

```
cd /etc/network
```

Ensuite, pour conserver le fichier d'interface d'origine pour l'interface wifi, on a fait une sauvegarde sous la forme :

```
 cp interfaces wifi-interface
```

La prochaine étape consiste à créer un nouveau fichier pour notre interface ad hoc. On procède comme suit pour créer un fichier et le modifier :

```
 nano adhoc-interface
```

On a collé ce qui suit dans le fichier *adhoc-interface* :

```
auto lo
  iface lo inet loopback
  iface eth0 inet dhcp

  auto wlan0
  iface wlan0 inet static
  address 192.168.1.1
  netmask 255.255.255.0
  wireless-channel 4
  wireless-essid RPitest
  wireless-mode ad-hoc
```

Cette configuration d'adhoc-interface laissera l'adresse IP de notre RPi être 192.168.1.1,et le masque de réseau sera 255.255.255.0

L'étape suivante est de configurer un serveur DHCP qui attribue les adresses IP aux l'appareil qui se connecte à notre réseau ADHOC:

```
sudo apt-get install isc-dhcp-server
```

Après cela, nous devons éditer ce *dhcpd.conf* fichier en utilisant :

```
sudo nano /etc/dhcp/dhcpd.conf
```

On ajoute ça au fichier :

```
ddns-update-style interim;
  default-lease-time 600;
  max-lease-time 7200;
  authoritative;
  log-facility local7;
  subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.5 192.168.1.150;
 }
```

Dans les lignes ci-dessus, tout nouvel appareil connecté à RPi se verra attribuer une adresse IP dans la plage `192.168.1.5 192.168.1.150`.

- Étape 2 : On a commencé à coder client - serveur entre deux machines et passer des informations.

- Étape 3 : On a travaillé sur le module bluetooth, on a trouvé un code sur github qu'on a modifié en se documentant sur le module *pydbus*. On a vérifié s'il récupérait toutes les adresses MAC. Ensuite, on lui a fixé une adresse MAC précise pour voir si le *rssi* change en fonction de la distance avec le checkpoint.

- Étape 4 : On s'est mis d'accord sur un protocole réseau et on a continué d'écrire le code serveur-client où chaque membre de l'équipe a implémenté un code de son côté.

- Étape 5 : On a fait les tests fonctionnels nécessaires pour notre application.

- Étape 6 : Débogage du code implémenté.

Partie 3 : Fonctionnement du réseau

Voilà le déroulement en détail des rondes dans le site de surveillance en utilisant l'application qu'on a conçue :

- Le responsable du poste de contrôle commence par allumer le serveur.
- Au moment de commencer la ronde, le rondier connecte la machine mobile ***MM*** au serveur.
- Les checkpoints ***A***, ***B*** et ***C*** se connectent également au serveur.
- Le rondier commence à faire sa ronde sur les checkpoints selon l'ordre établi par le serveur.
- Dans l'ordre, chaque checkpoint à son tour essaye de détecter la ***MM*** en utilisant son adresse MAC bluetooth.
- Dès que le checkpoint à qui est le tour arrive à trouver la ***MM*** du rondier, il informe le serveur de l'arrivée du rondier, qui à son tour va demander un rapport au rondier.
- Le rondier ne peut pas continuer sa ronde sans envoyer son rapport du checkpoint qu'il vient de visiter. Une fois que c'est fait, il continue de faire sa ronde dans l'ordre préétabli jusqu'à finir sa ronde.

Partie 4 : Explication du code

Pour implémenter ce code, on a utilisé deux différents langages de programmation :

- **python** pour les 2 fichiers *rondier.py, checkpoint.py.*
- Le langage **C** pour les 3 fichiers du serveur *serveur.c, tcp.c* et *utils.c.*

Explication :

- ***rondier.py*** : Ce code est implémenté sur la machine mobile ***MM*** qui sera utilisée par le rondier. Il lui permet de se connecter au serveur et d'échanger des informations avec lui.

Au début du code, on a récupéré l'adresse IP et le port du serveur. puis on a connecté le rondier avec le serveur en appelant la fonction `client.connect((hote, port))`
Une fois connecté, le rondier envoie le chiffre "2" au serveur pour s'identifier et l'informer qu'il est prêt pour commencer sa ronde en appelant la fonction `client.send("2".encode(format))`

Le rondier sera amené à patienter tranquillement avant de recevoir le symbole "D" de la part du serveur qui signifie qu'il est temps de commencer sa ronde. Afin de recevoir ce message on a fait appel à la fonction `client.recv(size).decode(format)`. Une fois le message reçu, le rondier confirme au serveur qu'il va exécuter sa ronde en lui envoyant un accusé de réception (ack) .

Ensuite, le rondier fait sa ronde et passe par les checkpoints pour les valider et reste toujours à l'attente de recevoir la demande de rapport de la part du serveur dès que la validation du checkpoint concerné est faite. Le rondier reçoit l'information du serveur et rédige son rapport sur l'endroit ou le checkpoint est placé et qu'il vient de visiter, et puis il l'envoie au serveur.

Enfin, le serveur informe le rondier de la fin de ronde en lui faisant un signe "E". Le rondier se ferme la connection en appelant la fonction `client.close()`

- ***checkpoint.py*** : Ce code est implémenté sur les raspberry ***A***, ***B*** et ***C*** qui servent de checkpoint pour les rondes. Il permet de détecter la machine mobile ***MM*** du rondier et d'informer le serveur de son arrivée.

Le checkpoint commence d'abord par se connecter au serveur avec la fonction `client.connect(ADDR)`. Ensuite, il s'identifie au serveur en envoyant le chiffre "1" d'abord pour faire savoir que c'est un checkpoint, puis il envoie son numéro qui représente son ordre de passage dans la ronde que va faire le rondier.

Le checkpoint attend attentivement son tour. Dès que le serveur le prévient que c'est le cas, il commence à chercher la machine mobile ***MM*** via le bluetooth en utilisant des fonctionnalités du module *pydbus* sur python qui permettent de récupérer les périphériques à proximité avec la force du signal *rssi*.

La recherche se poursuit jusqu'à ce que le checkpoint réussisse à détecter la ***MM*** du rondier. Chaque checkpoint fait ça lorsque le tour lui est attribué dans un ordre prédéfini.
Afin de trouver la ***MM***, le checkpoint utilise son adresse MAC bluetooth. Et elle ne peut être détectée que lorsque le signal est supérieur à -50 dBm (toutefois cela reste une approximation).

Dès que la ***MM*** est détectée, le checkpoint alerte le serveur de l'arrivée du rondier en lui envoyant "P" ce qui signifie que le checkpoint est validé.

- ***Serveur.c*** : Ce code est implémenté sur la machine du serveur. Il permet aux checkpoints ***A***, ***B*** et ***C*** et à la ***MM*** de se connecter en utilisant son adresse IP. Il est accompagné de 2 fichiers *tcp.c* et *utils.c* qui contiennent des fonctions utiles pour le code *serveur.c*.

Le serveur commence par ouvrir une socket tcp, ensuite, il reste à l'écoute pour d'éventuelles connexions.

A chaque connexion, il attend une identification, le caractère ‘2’ pour le rondier et ‘1’ pour un checkpoint. Dans le cas où ce c’est un checkpoint qui s’est connecté, ce dernier doit suivre son identification par un numéro qui définit son ordre de passage dans la ronde que va faire le rondier.

Le serveur lance un thread pour chaque connexion, ensuite il les bloque avec des mutex en appelant la fonction `pthread_mutex_lock;`

Le serveur utilise une variable *NB_STATION* qui détermine le nombre de machines en plus de celle du rondier qu’il doit attendre pour se connecter avant qu’il arrête l’écoute.

Dans notre cas, on a 4 machines donc `#define NB_STATION 4`. Une fois qu’ils sont connectés, le serveur n’accepte plus de nouvelles connections et libère le thread du rondier en appelant la fonction `pthread_mutex_unlock`, ce thread a été créé lors de la connexion du rondier avec l’appel `pthread_create`

Dès que le thread du rondier est libéré, la fonction *rondier* alerte le rondier de commencer sa ronde et elle attend de recevoir un accusé de réception de sa part. Ensuite, ce thread s’occupe de débloquer les threads des checkpoints. On note que le checked mutex reste bloqué jusqu'à ce que le rondier valide son checkpoint.

Le choix d’utiliser des threads est volontaire, ils ne sont pas obligatoires pour assurer le bon fonctionnement de la version actuelle de notre application. En revanche, ils seront très utiles pour une prochaine version où on sera peut-être amené à gérer plusieurs connexions en parallèle au serveur.

- ***tcp.c*** : qui contient la fonction *install_server()* qui crée une socket, et lui modifie les paramètres pour la rendre polymorphe, c à d elle accepte les connexion IPV4 mais également IPV6. Ensuite, on reste à l’écoute sur le port donné en entrée.

- ***utils.c*** : Ce fichier est une bibliothèque qui contient des fonctions qui permettent de gérer les erreurs.