ADDA Ahlem Lamia
EL FIKRI Ismail
EL KADOURI Soufiane
SANNI Habil

Générer un identifiant qui comprend une partie immuable

Générer un identifiant qui comprend une partie immuable

Plusieurs problématiques s'offrent à nous :

- La création d'un ID unique qui pourra être retrouvé en cas de litige,
- Les différents problèmes concernant la définition exacte de la proximité concernant les coordonnées gps

Tout d'abord nous avons implémenté les différentes fonctions demandées mais par la suite nous nous somme rendu compte de la complexité du sujet et de toutes les réflexions à avoir en dehors du codage. Nous avons d'abord pensé à fusionner la partie position avec un id aléatoire unique à l'aide d'un UUDI qui est une technique beaucoup utilisée et très efficace qui génère un code unique. Mais le problème de cette fusion poserait des soucis vis-à-vis de la définition que l'on se fait de la proximité par exemple pour un uber deux personnes peuvent être à proximité mais séparé par quelques choses qui les empêcherait de les rejoindres avant un long temps (contourner une autoroute par exemple). On avait donc codé les fonctionnalitées demandées mais une fois implémenté les réflexions au sein du groupe n'ont cessé d'augmenter et nous avons donc essayé de trouver quelques choses de plus approfondi qui permettra une meilleure réponse à la problématique.

Nous avons imaginé d'abord réalisé ce tp en codant des noeuds qui auront un certain ID généré aléatoirement à l'aide de UUDI , un certain x qui sera la position x et un certain y qui sera la position y du noeud (donc de la personne) :

```
class Noeud: # Définition de notre classe de Noeud

        def __init__(self):
                self.id = "" # ID du noeud qui sera associé à une personne
                self.x = 48.5  # Coordonnées x de la position de la personne
                self.y = 50.5  # Coordonnées y de la position de la personne
```

Grâce à cette classe on pouvait donc entamer la fonction **getCurrentLocation(anonymized=False)** avec la possibilité de savoir si on veut la localisation exacte ou non. Si jamais on veut la localisation exacte on renvoie notre x et notre y, mais si jamais on ne souhaite pas avoir la localisation exacte on va approximer cette valeur avec une variable aléatoire (pas trop grande) qui va s'ajouter au x et au y qui modifiera la vraie position et la transformera en position approximative.

La position de l'utilisateur est représenté par le point rouge, et celle-ci va être modifié sans pour autant sortir d'une certaine zone qui est représentée par le cercle vert.

L'image ci-dessous, explique l'approximation que l'on a imaginée :


```
def getCurrentLocation(self,anonymized=False):
    #anonymized: si on veut ou non la localisation exacte.
    #return la localisation actuelle du noeud

    #latitude = y
    #longitude = x

    # On incrémente les deux premiers nombres après la virgule

        if anonymized:
            # Génération d'un nombre aléatoire
            r = randint(1,5) / float(100)

            # Incrémentation du x
            a, b = math.modf(self.x)
            x = round((a + r) + b, 4)

            # Incrémentation du y
            a, b = math.modf(self.y)
            y = round((a + r) + b, 4)

            return (x, y)
        else:
            return (self.x, self.y)
```

Par la suite, il a fallu créer une fonction genMyId() qui génère de manière aléatoire l'identifiant du noeud. Pour cela, on a décidé d'utiliser UUID car on est sûr de ne pas avoir de doublons dans notre liste de noeud. Voici donc notre fonction qui génère un identifiant unique :

```
def genMyId(self):
        self.id = uuid.uuid4()
```

Pour finir, nous avons implémenté la fonction demandée genNetworkID() qui va générer un identifiant unique en prenant en compte la localisation du noeud. Pour cela, nous avons pensé à concaténer les positions du gps avec un UUID mais comme dis au-dessus cela n'est pas la solution la plus optimale :

```
def genNetworkId(self):
        self.id = str(self.x)+str(self.y)+str(uuid.uuid4())
```

Nous avons également pensé à une autre manière de créer l'identifiant en mettant en pratique l'idée des groupes qui ont travaillé sur l'identifiant. Leur idée était de hasher le nom et le prénom suivit d'un numéro aléatoire, on a donc implémenté cette fonction :

```
def hashNomPrenomUuid(self):
        nom = raw_input("Entrez votre nom : ")
        prenom = raw_input("Entrez votre prenom : ")
        final = str(nom)+str(prenom)+str(uuid.uuid4())
        hash_final = hashlib.md5(final)
        print(hash_final.hexdigest())
```

L'avantage de cette technique repose sur le fait qu'on pourra retrouver le nom et le prénom s'il y a un litige judiciaire puisqu'on pourra remonter au nom et au prénom à l'aide d'un gros algorithme qui dé-hash cet identifiant.

De plus, plusieurs questions se posent autour de l'ID et de sa partie immuable comme par exemple le hash de l'adresse MAC afin de trouver un ID unique mais également utiliser un système qui existe déjà comme celui utilisé par Bitcoin qui utilise l'algorithme ECDSA (Elliptic Curve Digital Signature Algorithm) qui est un algorithme de signature numérique à clé publique, variante de DSA. Il fait appel à la cryptographie sur les courbes elliptiques du fait de la difficulté de calculer le logarithme discret d'un grand nombre entier ce qui fait de cet algorithme une sécurité. On pourrait donc utiliser cet algorithme pour résoudre le problème de la clé publique partagé.