

# **Communication D2D (device-to-device) basée sur la technologie blockchain**

Encadrants : **Maria Potop-Butucaru, Billel Zeghoudi**

Étudiants : **FAYE El hadji Abdou 21318704, NGUYEN Huu Truc  
21310174, ROSSIGNOL Paul 21311838, YAHY Yanis 21312748**

## Table des matières

<b>I. Cahier des charges.....</b>	<b>3</b>
1.1 Introduction.....	3
1.2 Objectifs.....	3
- 1.2.1 Principaux.....	3
- 1.2.2 Secondaires.....	3
<b>II. Plan de développement.....</b>	<b>3</b>
<b>III. Analyse.....</b>	<b>5</b>
3.1 La localisation.....	5
3.2 La protection.....	6
3.3 Conclusion.....	7
<b>IV. Conception envisagée.....</b>	<b>7</b>
4.1 Localisation de l'appareil et preuve de localisation.....	7
4.2 La Blockchain et l'ajout du device to device.....	8
4.3 Gestion des appareils malfaisants.....	9
4.4 Conclusion sur la conception actuelle.....	9
<b>V. Compte rendu du Projet.....</b>	<b>10</b>
5.1 Déroulement du projet.....	10
5.2 Réalisation du projet.....	10
5.3 CODE DU PROJET.....	10
5.3.1 Code Blockchain.....	10
5.3.1.1 Solidity.....	10
5.3.1.2 JavaScript.....	12
5.3.2 Code et Simulation.....	18
5.3.2.1 Code.....	18
5.3.2.1.1 Device.....	19
5.3.2.1.2 - Sensor.....	22
5.3.2.1.3 - Access Point.....	24
5.3.2.2 Simulation.....	26
5.3.2.3 NS3.....	27
<b>VI. Conclusion du projet.....</b>	<b>30</b>
<b>VII. Perspective future du projet.....</b>	<b>30</b>
<b>VIII. Bibliographie.....</b>	<b>32</b>

# I. Cahier des charges

## 1.1 Introduction

La technologie du GPS est aujourd'hui utilisée de manière assez excessive, ce qui est très coûteux d'un côté économique et d'un côté énergétique. Nous allons donc proposer une manière de communication entre deux appareils sans utiliser de manière directe le GPS.

## 1.2 Objectifs

### - 1.2.1 Principaux

Trouver un moyen de communication entre deux appareils sans utilisation directe du GPS et en essayant d'implémenter un système de blockchain. Aussi trouver le moyen de le faire avec le moindre coût possible tout en assurant une décentralisation du réseau.

### - 1.2.2 Secondaires

Sécuriser les communications de manière à ce qu'aucune donnée ne soit volée, optimiser l'utilisation de la blockchain de façon à ce qu'il n'y ait pas trop d'informations à stocker.

# II. Plan de développement

Nous avons commencé par nous concentrer sur trois documents[1][2][3] fournis par nos encadrants afin de nous familiariser avec les notions de la blockchain et les différentes méthodes de localisation sans utiliser directement un GPS. Après avoir acquis les bases, nous avons tenté de trouver une structure envisageable pour réaliser ce projet. Cependant, nous n'avons pas approfondi suffisamment les notions mentionnées dans les documents de recherche. Par conséquent, nous avons cherché d'autres ressources pour approfondir notre compréhension et trouver d'autres solutions potentielles pour résoudre notre problématique ou même améliorer les solutions déjà proposées. Après discussions avec nos encadrants, nous avons été orientés vers une blockchain spécifique, "Helium"[6], qui repose fortement sur l'internet des objets (IoT). Cela nous a permis de voir le projet sous un angle différent. Nous sommes passés d'un système utilisant des outils spécifiques tels que des capteurs et d'autres appareils, principalement pour la réception et l'envoi de signaux, à un système utilisant déjà des appareils connectés tels que des téléphones et des parcmètres, ainsi que toutes sortes d'objets pouvant être connectés. Cela forme donc un vaste réseau pouvant être utilisé pour

**Projet 11**  
**Com. D2D Blockchain**

mener à bien notre projet en localisant un appareil de manière relativement sûre et en étant bien plus économe en termes d'énergie et de coût de mise en place de la structure du réseau.

En ce qui concerne la répartition des tâches, nous avons commencé le projet de manière assez individuelle. Chacun avait un ou plusieurs documents à analyser et à comprendre. Cependant, au fur et à mesure de nos entretiens avec nos encadrants et de notre avancement sur le projet, nous avons commencé à réfléchir tous ensemble sur les mêmes documents. Nous avons discuté des différentes façons d'implémenter ce qui était décrit dans les ressources utilisées, ainsi que des solutions que nous pouvions proposer pour résoudre la problématique du projet. En effet, les notions à étudier devenaient de plus en plus techniques.

Nous avons décidé de garder les objectifs possibles en vue et avons donc décidé de créer la partie de stockage des données dans la blockchain ainsi que de créer une simulation du comportements des capteurs et de l'appareil.

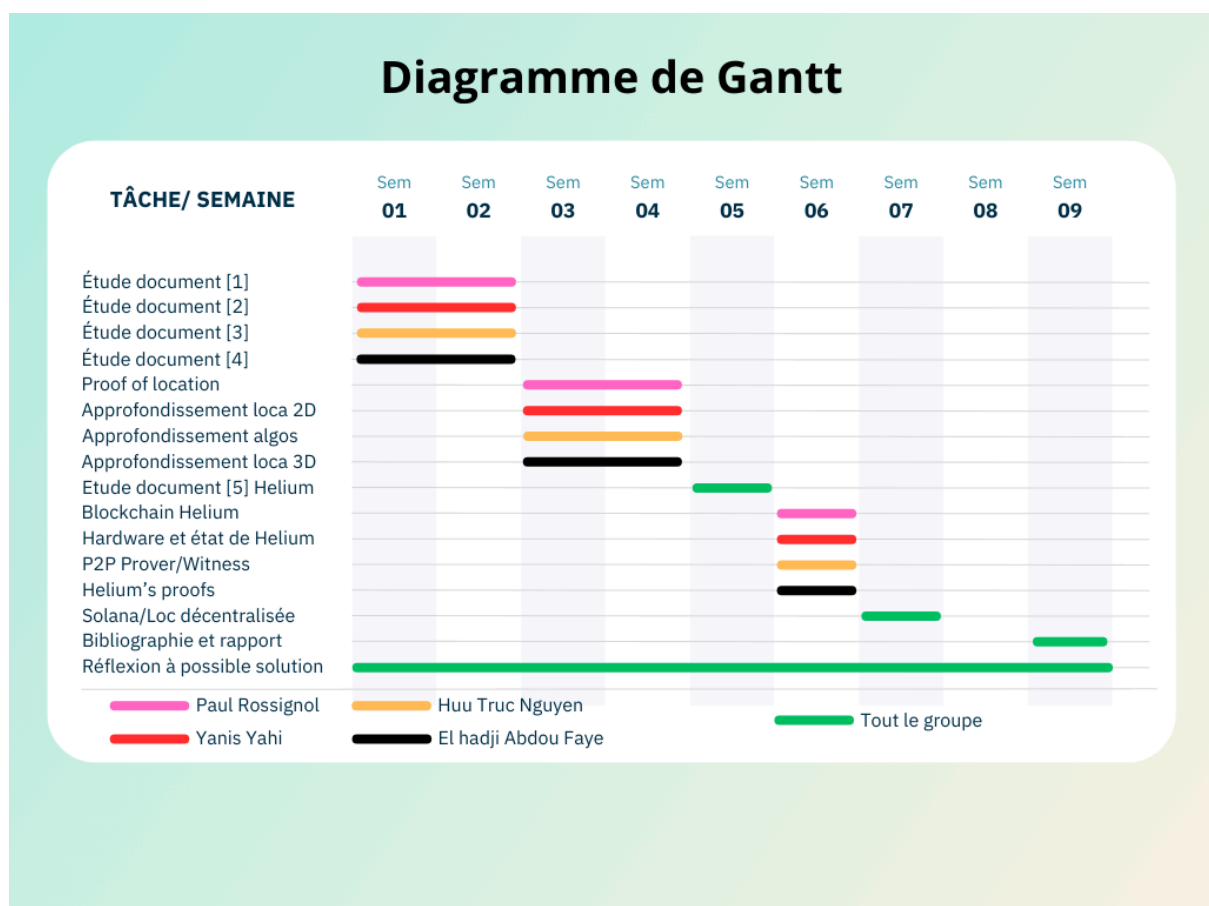


Figure 1. Diagramme de Gantt

[illegible]

### III. Analyse

### 3.1 La localisation

Selon les spécifications (si nous sommes seulement à l'intérieur), nous pourrions même utiliser FILA [7] pour obtenir une distance plus précise. Avec les distances connues, pour avoir les coordonnées, il faut utiliser la méthode de multilatéralisation. Cependant, un problème persiste, c'est de savoir qui va faire et envoyer les résultats des calculs aux prochains membres du système : en effet, avec plusieurs membres qui doivent d'abord faire leurs propres calculs de distance, il est nécessaire de décider vers qui ils devront envoyer les données calculées, ainsi que d'envoyer le résultat de l'appareil en question. Cette problématique sur la décision restera constamment importante tout au long du projet.

### 3.2 La protection

Maintenant que nous avons les coordonnées du client, il est important de vérifier la véracité de ces coordonnées. En effet, une personne malveillante pourrait falsifier son emplacement pour tirer avantage des bénéfices de l'endroit sans être physiquement présente. Pour vérifier la véracité d'un client tout en restant décentralisé, il faut faire intervenir des acteurs agissant comme témoins. Un témoin ici va confirmer si une personne est vraiment là en observant ses alentours et en communiquant avec les voisins. Ce témoin peut être un autre client dont l'identité est connue par le système, mais cela introduit une possibilité de coopération entre plusieurs acteurs malveillants, autrement dit, la possibilité d'une attaque Sybil. Il serait également impossible de vérifier s'il n'y a que lui dans le périmètre ou si le nombre de personnes est trop faible pour que les appareils puissent témoigner de la présence de la personne.

Nous pourrions aussi utiliser ces mêmes capteurs comme témoins : avec l'emplacement de chaque capteur connu à l'avance, il est possible de vérifier la présence de la personne directement en communiquant avec les appareils présents aux coordonnées pour vérifier la présence. Cela résoudrait le problème de fluctuation du nombre de personnes dans le périmètre, mais les capteurs ne sont pas complètement infallibles non plus ; ils peuvent tomber en panne à tout moment ou si la personne malveillante arrive à trouver l'emplacement d'un des capteurs, elle pourrait le changer pour que celui-ci coopère avec elle. Nous devons donc réfléchir à la manière d'éviter cela.

Que le témoin soit une autre personne ou un capteur, il faut toujours communiquer entre les membres du système, et par conséquent, nous sommes obligés d'assurer que la communication ne puisse pas être un maillon faible du système. Il faut donc avoir un moyen pour garantir l'unicité de l'identité du client. Il est également nécessaire de garder une preuve temporelle de la communication pour éviter l'utilisation de messages obsolètes.

Ayant garanti l'intégrité des communications, il est alors possible d'avoir une preuve de présence. Mais pour permettre à d'autres personnes de vérifier la véracité de la présence d'un autre, comme par exemple lorsqu'un client essaie de contacter un autre client, il faut un moyen pour ce client de vérifier que la personne qui l'a contacté est bien là. Nous devons mettre en place une solution qui permettrait à une personne de regarder et de voir s'il y a eu des preuves de présence récentes concernant l'autre personne. Cette historique doit donc non seulement être sécurisée mais aussi suffisamment accessible à tout moment. Il faut donc trouver un moyen de stocker cette historique de manière décentralisée pour éviter les problèmes de point de défaillance unique.

### 3.3 Conclusion

En conclusion, pour que le projet puisse réussir, il est essentiel de résoudre les problèmes mentionnés ci-dessus et de garantir la collaboration harmonieuse de tous les éléments impliqués.

## IV. Conception envisagée

### 4.1 Localisation de l'appareil et preuve de localisation

La première étape est de localiser l'appareil, pour cela plusieurs systèmes sont possibles. Nous avons choisi de combiner deux techniques : le fingerprinting et la trilatération. Tout d'abord, il faut répartir dans la zone à couvrir des "anchors" qui sont tous connectés à un point d'accès. Les anchors seront placés de manière à maximiser la zone de couverture, en utilisant une stratégie d'échantillonnage intelligent qui prend en compte la géométrie de la zone afin de les placer de manière optimale. Ainsi, le point d'accès connaîtra l'ensemble des localisations des différents anchors avec leurs identifiants uniques. Ensuite, il y aura une phase de préparation, pendant laquelle les anchors vont effectuer la méthode de "fingerprinting". L'ensemble des anchors va quadriller leur zone de couverture pour former des grilles, estimant la puissance moyenne du signal si un appareil se trouve dans cette zone. Ensuite, ils vont associer les identifiants des grilles avec la puissance moyenne. Suite à cela, l'ensemble des anchors va passer en phase d'écoute et attendre la réception d'un signal, étant donné que c'est l'utilisateur qui veut connaître sa position. Lorsque l'utilisateur se connecte au point d'accès, un identifiant unique lui sera attribué en tant qu'information de session que le point d'accès pourra utiliser pour le contacter. Il sera accompagné d'une paire de clés publique et privée déterminées grâce à l'algorithme RSA afin de sécuriser les échanges et

surtout de pouvoir les signer. Suite à cela, lorsque l'utilisateur veut connaître et prouver sa localisation, il envoie aux anchors autour de lui un signal composé de son id et du temps auquel l'appareil a émis le signal, le tout chiffré par sa clé privée grâce à l'algorithme RSA. Ainsi, les anchors qui reçoivent le message peuvent déchiffrer le message grâce à la clé publique de l'appareil. Ils vont alors analyser la puissance du signal en utilisant le RSSI (Received Signal Strength Indicator) pour trouver les grilles correspondant à sa zone de couverture et le ToA (Time of Arrival) pour déterminer une distance entre l'anchor et l'appareil. Cet ensemble d'informations de position va alors être signé avec la clé privée de l'anchor et envoyé avec l'identifiant du device, l'identifiant de l'anchor et le temps où le signal original a été émis au point d'accès. Tout cela sera chiffré avec la clé publique du point d'accès afin d'obtenir une plus grande sécurité. Enfin, le point d'accès pourra, s'il a obtenu au moins trois ensembles d'informations de localisation de trois anchors différents ayant le même timestamp provenant de l'appareil voulant savoir sa localisation, effectuer une trilatération en plus du fingerprinting pour obtenir une localisation plus précise et ajouter toutes ces informations à la blockchain.

## 4.2 La Blockchain et l'ajout du device to device

Pour ce qui est de la blockchain, elle sera stockée dans l'espace de stockage des points d'accès. Le point d'accès enverra alors à l'appareil sa localisation avec la preuve que celle-ci a été vérifiée par les anchors, étant donné qu'il a réussi à entrer en communication avec lui dans une "short range communication". Ce message sera chiffré avec la clé publique de l'appareil afin que seule l'appareil puisse déchiffrer ce message et obtenir sa localisation. Ensuite, si l'appareil le souhaite, il pourra devenir lui-même une partie du réseau en remplaçant les anchors. Il émettra alors un signal aux anchors proches pour leur indiquer qu'il prend le relais sur la phase de calcul de distance et de "fingerprinting", ainsi que dans la proof of history permettant la validation des blocs de la blockchain. En participant à ces activités, l'appareil obtient un score représentant la confiance que le système lui accorde. Ce score augmentera en fonction du nombre d'appareils avec lesquels il collabore pour trouver la localisation, ainsi que sa participation au minage de blocs.

## 4.3 Gestion des appareils malfaisants

Certains appareils peuvent être malveillants. Pour éviter cela, plusieurs méthodes sont possibles. Les anchors qui ont été désactivés peuvent se réactiver en fonction du score de l'appareil (plus le score est faible, plus les réactivations sont fréquentes) et vérifier les informations de position partagées par l'appareil en les comparant aux signaux qu'ils reçoivent. Ainsi, si l'appareil envoie un signal que les anchors n'ont jamais reçu ou qui est incorrect, le score de celui-ci diminue voire il sera isolé du système.



Dans le but d'encourager la participation des appareils, l'ajout d'un "token" pourrait être envisagé pour récompenser les participants du système pour l'ensemble des transactions et la gestion de la blockchain. De plus, l'ajout de Smart Contracts sur la blockchain pourrait être envisagé. Ce sont des contrats intelligents qui s'exécutent de manière automatique, mais ils nécessitent une gestion parfaite de leur code, car une fois implémentés, ils deviennent immuables. Ainsi, la présence d'erreurs dans le code pourrait entraîner plus de vulnérabilités que de sécurité.

En outre, l'ajout d'algorithmes IDS (système de détection d'intrusions) permettant la détection des anomalies basées sur le comportement des appareils pourrait être déployé, soit sur le réseau soit au niveau de l'application.

## 4.4 Conclusion sur la conception actuelle

Ainsi, il s'agira d'un système Device to Device (D2D) permettant la localisation d'un appareil. Dans un premier temps, un ensemble d'anchors permettra, grâce à la réception du signal, la création d'une preuve de localisation qui sera ensuite envoyée de manière sécurisée à l'utilisateur à l'aide d'un ensemble de clés et d'identifiants. Il est important de noter que l'ensemble du système pourra encore évoluer au cours du projet en fonction des défis techniques et matériels auxquels nous serons confrontés, mais nous disposons d'une base solide pour entamer la mise en place du système.

# V. Compte rendu du Projet

## 5.1 Déroulement du projet

Nous nous sommes mis ensemble pour voir ce qui faut faire pour atteindre le but et avons donc décidé qu'un groupe va faire la partie blockchain et l'autre partie va faire la partie simulation. Puisque les deux parties font appel à des notions qui nous sont étrangères, il a fallu un bon moment pour comprendre comment utiliser non seulement les applications (RemixIDE et Omnet++) et le langage utilisé (Solidity et C++ de Omne). La réalisation de la partie blockchain a été plus facile que la réalisation de la partie simulation a cause d'un manque de tutoriel et guide sur Internet concernant Omnet.

## 5.2 Réalisation du projet

Pour la partie blockchain, deux choses devaient être faites: le code solidity et le code la trilatération. Le code solidity est là pour créer un smart contrat et mettre les informations nécessaire dans la blockchain alors que le code de la trilatération devait être fait en JavaScript pour que nous puissions le faire marcher avec le smart contract. En effet, Solidity comme langage n'a pas la capacité de faire de la trilatération, il était nécessaire de faire ce code dans un autre langage, ici Javascript, et d'utiliser l'API web3.js pour pouvoir faire appelle au smart contract.

## 5.3 Code du projet

### 5.3.1 Code Blockchain

#### 5.3.1.1 Solidity

```
contract Storage {

    int256 constant LATITUDE_FACTOR = 1e8; // 10^8
    int256 constant LONGITUDE_FACTOR = 1e8; // 10^8

    struct Coordinates {
        int256 latitude;
        int256 longitude;
        uint256 time;
        string ID1;
        string ID2;
        string ID3;
        uint256 speed;
    }

    Coordinates private coordinates;

    /**
     * @dev Store latitude, longitude, time, ID, and speed coordinates
     * @param lat Latitude coordinate
     * @param lon Longitude coordinate
     * @param _time Time of recording
     * @param _ID1 Identifier
     * @param _ID2 Identifier
     * @param _ID3 Identifier
     * @param _speed Speed in km/h
     */
    function storeCoordinates(int256 lat, int256 lon, uint256 _time, string memory _ID1, string memory _ID2, string memory _ID3, uint256 _speed) public {
        coordinates.latitude = lat * LATITUDE_FACTOR;
        coordinates.longitude = lon * LONGITUDE_FACTOR;
        coordinates.time = _time;
        coordinates.ID1 = _ID1;
        coordinates.ID2 = _ID2;
        coordinates.ID3 = _ID3;
        coordinates.speed = _speed;
    }

    /**
     * @dev Return latitude, longitude, time, ID, and speed coordinates
     * @return Latitude, longitude, time, ID, and speed coordinates
     */
    function getCoordinates() public view returns (int256, int256, uint256, string memory, string memory, string memory, uint256) {
        int256 nlat = coordinates.latitude / LATITUDE_FACTOR;
        int256 nlong = coordinates.longitude / LONGITUDE_FACTOR;
        return (nlat, nlong, coordinates.time, coordinates.ID1, coordinates.ID2, coordinates.ID3, coordinates.speed);
    }
}
```

Cette partie du code représente le smart contrat qui permet de stocker les données dans la blockchain grâce à la méthode store Coordinates. Nous avons la latitude, la longitude que l'AP aura calculer grâce à la trilatération, le temps, les ID des capteurs ainsi que la

vitesse. Nous pouvons aussi reprendre les données les plus récentes avec la méthode `getCoordinates`.

### 5.3.1.2 JavaScript

```
{
  "inputs": [],
  "name": "getCoordinates",
  "outputs": [
    {
      "internalType": "int256",
      "name": "",
      "type": "int256"
    },
    {
      "internalType": "int256",
      "name": "",
      "type": "int256"
    },
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    },
    {
      "internalType": "string",
      "name": "",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "",
      "type": "string"
    },
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "int256",
      "name": "lat",
      "type": "int256"
    },
    {
      "internalType": "int256",
      "name": "lon",
      "type": "int256"
    },
    {
      "internalType": "uint256",
      "name": "_time",
      "type": "uint256"
    },
    {
      "internalType": "string",
      "name": "_ID1",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "_ID2",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "_ID3",
      "type": "string"
    },
    {
      "internalType": "uint256",
      "name": "_speed",
      "type": "uint256"
    }
  ],
  "name": "storeCoordinates",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
}
```

Ce long morceau d'information représente l'ABI (Application Binary Interface). L'ABI est nécessaire pour que les codes dans un langage différents puissent interagir avec le smart contract.

```
function Compute(p1,p2,p3){
  var a;
  var b;
  var c;
  var d;
  var f;
  var g;
  var h;
  var i;
  var j=3.14159265359;
  var k;
  c=p2[0]-p1[0];
  d=p2[1]-p1[1];
  f=(180/j)*Math.acos(Math.abs(c)/Math.abs(Math.sqrt(Math.pow(c,2)+Math.pow(d,2))));
  if((c>0&&d>0)){
    f=360-f;
  }else if((c<0&&d>0)){
    f=180+f;
  }else if((c<0&&d<0)){
    f=180-f;
  }
  a=C(c,d,B(A(D(p2[2])),f);
  b=C(p3[0]-p1[0],p3[1]-p1[1],B(A(D(p3[2])),f);
  g=(Math.pow(B(A(D(p1[2])),2)-Math.pow(a[2],2)+Math.pow(a[0],2))/(2*a[0]);
  h=(Math.pow(B(A(D(p1[2])),2)-Math.pow(b[2],2)-Math.pow(g,2)+Math.pow(g-b[0],2)+Math.pow(b[1],2))/(2*b[1]);
  i=C(g,h,0,-f);
  i[0]=(i[0]+p1[0])-0.086;
  i[1]=(i[1]+p1[1])-0.004;
  k=E(i[0],i[1],p1[0],p1[1]);
  if(k>p1[2]*2){
    i=null;
  }else{
    if(i[0]<-90||i[0]>90||i[1]<-180||i[1]>180){
      i=null;
    }
  }
  //i[0] *= Math.pow(10, 15);
  //i[1] *= Math.pow(10, 15);
  return [i[0],i[1]];
}
```

```
function A(a){
    return a*7.2;
}

function B(a){
    return a/900000;
}

function C(a,b,c,d){
    e=3.14159265359;
    return [a*Math.cos((e/180)*d)-b*Math.sin((e/180)*d),a*Math.sin((e/180)*d)+b*Math.cos((e/180)*d),c];
}

function D(a){
    return 730.24198315+52.33325511*a+1.35152407*Math.pow(a,2)+0.01481265*Math.pow(a,3)+0.00005900*Math.pow(a,4)+0.00541703*180;
}

function E(a,b,c,d){
    var e=3.14159265359;
    var f=e*a/180;
    var g=e*c/180;
    var h=b-d;
    var i=e*h/180;
    var j=Math.sin(f)*Math.sin(g)+Math.cos(f)*Math.cos(g)*Math.cos(i);
    if(j>1){
        j=1;
    }
    j=Math.acos(j);
    j=j*180/e;
    j=j*60*1.1515;
    j=j*1.609344;
    return j;
}
```

Les deux images ci-dessus représentent le code de la trilatération et permet à l'AP de calculer la latitude et longitude de l'appareil mobile.

```
function floatToInt256(value) {  
  // Scale the floating-point value to keep up to 5 decimal places  
  var scaledValue = value * 1e8;  
  
  // Round the scaled value to the nearest integer  
  var intValue = Math.round(scaledValue);  
  
  // Convert the rounded integer value to a BigInt  
  intValue = BigInt(intValue);  
  
  return intValue;  
}  
  
async function storeCoordinates(p1, p2, p3, time, ID, speed, contractInstance) {  
  const computedCoordinates = Compute(p1, p2, p3);  
  
  var int1 = floatToInt256(computedCoordinates[0]);  
  var int2 = floatToInt256(computedCoordinates[1]);  
  
  const accounts = await web3.eth.getAccounts();  
  
  contractInstance.methods.storeCoordinates(int1, int2, time, ID, speed)  
    .send({ from: accounts[0] })  
    .on('transactionHash', function(hash){  
      console.log('Transaction Hash: ' + hash);  
    })  
    .on('receipt', function(receipt){  
      console.log('Transaction Receipt:', receipt);  
    })  
    .on('error', function(error) {  
      console.error('Transaction Error:', error);  
    });  
}
```

Ici, nous avons le code coté Javascript pour stocker les informations: après avoir calculer les coordonnées de l'appareil nous allons d'abord transformer ces coordonnées en valeurs non décimales: en effet, puisque solidity n'est pas capable de gérer les valeurs flottantes, il faut le transformer en integer avant de le stocker. Ensuite, nous allons faire appel au compte avant de pouvoir invoquer la méthode.

```
async function getCoordinates(contractInstance) {
  try {
    const result = await contractInstance.methods.getCoordinates().call();

    const lat = result[0] / 1e8;
    const lon = result[1] / 1e8;
    const time = result[2];
    const ID = result[3];
    const speed = result[4];

    console.log("Retrieved coordinates:", lat, lon);
    console.log("Time:", time);
    console.log("ID:", ID);
    console.log("Speed:", speed);

    return [lat, lon, time, ID, speed];
  } catch (error) {
    console.error("Error retrieving coordinates:", error);
    return null;
  }
}

async function main() {
  const contractAddress = '0x0fC5025C764cE34df352757e82f7B5c4Df39A836';

  // Create contract instance with the deployed address
  var contractInstance = new web3.eth.Contract(contractABI, contractAddress);

  const point1 = [-19.6685, -69.1942, 84];
  const point2 = [-20.2705, -70.1311, 114];
  const point3 = [-20.5656, -70.1807, 120];

  const time = 1620352976;
  const ID = "ABC123";
  const speed = 50;

  // Store coordinates in the deployed contract
  await storeCoordinates(point1, point2, point3, time, ID, speed, contractInstance);

  // Retrieve coordinates from the deployed contract
  await getCoordinates(contractInstance);
}

main();
```

Finalement, nous avons ici la méthode pour prendre les coordonnées les plus récentes ainsi que la méthode main. Les valeurs sont exemplaires et si dans le futur devront être mises automatiquement. Cela s'applique aussi à l'adresse du contrat, nous devrons faire le déploiement du contrat directement par javascript et ainsi obtenir son adresse.

### 5.3.2 Code et Simulation

#### 5.3.2.1 Code

La simulation joue un rôle fondamental dans l'évaluation et la validation des systèmes complexes, offrant un environnement contrôlé pour étudier les comportements et les performances dans des conditions diverses. Dans le cadre de notre projet D2D, notre équipe a utilisé OMNeT++ pour simuler et analyser les interactions entre les appareils.

Cette introduction se concentre sur la méthodologie utilisée pour concevoir et exécuter notre simulation OMNeT++, ainsi que le code que chaque appareil utilise pour communiquer, ainsi que sur les résultats préliminaires de notre étude. Nous discuterons alors des conclusions initiales tirées de notre analyse.

Il est noté que plusieurs simulations seront analysées pour couvrir des scénarios importants. La première afin de présenter la simulation implique un seul appareil sans pertes de données qui communiquent aux sensors. La deuxième introduira des pertes ou des obstacles ou encore des mouvements afin d'évaluer les problèmes que cela posera. Et enfin la troisième simulation explorera le cas de deux appareils, l'un agissant de manière coopérative et l'autre malveillant tentant de se faire passer pour le premier.

Pour commencer ci-dessous vous trouverez une simulation vous montrant comment une simulation fonctionne en wireless. La simulation nous offre le début de la communication où l'hostA correspond au device qui broadcast dans sa zone son message et comment il est reçu par les sensors :

 Recording\_2024-05-08\_200945.mp4



Ainsi pour commencer nous allons étudier les différents codes qui seront dérivés pour chaque simulation.

#### 5.3.2.1.1 Device

```
#include <omnetpp.h>
#include <cstring>
#include <ctime>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/err.h>
#include <openssl/rand.h>

using namespace omnetpp;

class Device : public cSimpleModule {
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void broadcastMessage();
    virtual std::string signMessage(const std::string& message);
    virtual std::string getPublicKey();
    virtual void processReceivedMessage(cMessage *msg);

private:
    RSA *rsa;
    std::string privateKey;
    std::string publicKey;
    int deviceId;
};

Define_Module(Device);

void Device::initialize() {
    deviceId = getParentModule()->getIndex();

    // Generate RSA key pair
    rsa = RSA_generate_key(2048, RSA_F4, nullptr, nullptr);

    // Convert RSA keys to strings
    BIO *bioPrivate = BIO_new(BIO_s_mem());
    PEM_write_bio_RSAPrivateKey(bioPrivate, rsa, nullptr, nullptr, 0, nullptr, nullptr);
    char *privateKeyBuffer;
    size_t privateKeyLen = BIO_get_mem_data(bioPrivate, &privateKeyBuffer);
    privateKey = std::string(privateKeyBuffer, privateKeyLen);
    BIO_free(bioPrivate);

    BIO *bioPublic = BIO_new(BIO_s_mem());
    PEM_write_bio_RSAPublicKey(bioPublic, rsa);
    char *publicKeyBuffer;
    size_t publicKeyLen = BIO_get_mem_data(bioPublic, &publicKeyBuffer);
    publicKey = std::string(publicKeyBuffer, publicKeyLen);
    BIO_free(bioPublic);

    // Broadcast message periodically
    scheduleAt(simTime() + exponential(1), new cMessage("BroadcastMessage"));
```

```
}

void Device::handleMessage(cMessage *msg) {
    if (strcmp(msg->getName(), "BroadcastMessage") == 0) {
        broadcastMessage();
        scheduleAt(simTime() + exponential(1), new cMessage("BroadcastMessage"));
    }
    else {
        processReceivedMessage(msg);
    }
    delete msg;
}

void Device::broadcastMessage() {
    // Get current time
    std::time_t currentTime = std::time(nullptr);
    std::string message = std::to_string(deviceId) + "-" + std::to_string(currentTime);

    // Sign the message with private key
    std::string signedMessage = signMessage(message);

    // Create a new message
    cMessage *broadcastMsg = new cMessage("BroadcastedMessage");
    broadcastMsg->addPar("signedMessage");
    broadcastMsg->par("signedMessage").setStringValue(signedMessage);
    broadcastMsg->addPar("publicKey");
    broadcastMsg->par("publicKey").setStringValue(publicKey);

    // Send the message to all connected gates (broadcast)
    send(broadcastMsg, "out");
}

std::string Device::signMessage(const std::string& message) {
    // Hash the message using SHA-256
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256_CTX sha256;
    SHA256_Init(&sha256);
    SHA256_Update(&sha256, message.c_str(), message.length());
    SHA256_Final(hash, &sha256);

    // Convert hash to hexadecimal string
    std::stringstream ss;
    for(int i = 0; i < SHA256_DIGEST_LENGTH; i++) {
        ss << std::hex << std::setw(2) << std::setfill('0') << (int)hash[i];
    }

    // Encrypt the hash with private key
    unsigned char* encryptedHash = new unsigned char[RSA_size(rsa)];
    RSA_private_encrypt(ss.str().length(), (const unsigned char*)ss.str().c_str(), encryptedHash, rsa, RSA_PKCS1_PADDING);
}
```

**Projet 11**  
**Com. D2D Blockchain**

```
// Convert encrypted hash to hexadecimal string
std::string signedMessage((char*)encryptedHash, RSA_size(rsa));
delete[] encryptedHash;
return signedMessage;
}

std::string Device::getPublicKey() {
    // Return the public key
    return publicKey;
}

void Device::processReceivedMessage(cMessage *msg) {
    std::string signedMessage = msg->par("signedMessage").stringValue();

    // Decrypt the message with private key
    unsigned char* decryptedHash = new unsigned char[RSA_size(rsa)];
    RSA_private_decrypt(signedMessage.length(), (const unsigned char*)signedMessage.c_str(), decryptedHash, rsa, RSA_PKCS1_PADDING);

    // Convert decrypted hash to string
    std::string decryptedMessage((char*)decryptedHash, RSA_size(rsa));
    delete[] decryptedHash;

    // Extract position from decrypted message
    std::size_t pos = decryptedMessage.find_last_of("-");
    if (pos != std::string::npos) {
        std::string posXStr = decryptedMessage.substr(0, pos);
        std::string posYStr = decryptedMessage.substr(pos + 1);
        double posX = std::stod(posXStr);
        double posY = std::stod(posYStr);

        EV_INFO << "Device received position: (" << posX << ", " << posY << ")" << endl;
    }
    else {
        EV_WARN << "Invalid message format received." << endl;
    }
}

Ln 137, Col 2 | 4611 caractères
```

Le module "Device" est un élément essentiel de notre système de localisation sans fil, jouant un rôle central dans l'initiation et la coordination des communications avec les capteurs ainsi que dans la réception des données de localisation de l'Access Point. Lors de son initialisation, le dispositif génère une paire de clés RSA, comprenant une clé privée et une clé publique. Ces clés assurent la sécurité et l'intégrité des communications en permettant la signature et le chiffrement des messages échangés. Une fois initialisé, le dispositif crée un message contenant son identifiant unique (ID) et le temps de transmission actuel. Cette information temporelle est cruciale pour permettre aux capteurs d'estimer la distance entre eux et le dispositif, un élément clé du processus de localisation. Le message ainsi créé est alors signé à l'aide de la clé privée RSA du dispositif. Cette signature garantit l'authenticité et l'intégrité du message, permettant aux capteurs de vérifier son origine et son intégrité. Une fois signé, le message est diffusé en broadcast à tous les capteurs présents dans la portée du dispositif. Cette diffusion permet à chaque capteur de recevoir le message et de calculer la distance entre lui-même et le dispositif à partir du temps de transmission. En plus d'envoyer des messages aux capteurs, le dispositif peut également recevoir des informations en provenance des capteurs. Bien que ces données ne soient pas directement utilisées pour déterminer sa propre position, elles peuvent être exploitées localement pour diverses fins. En résumé, le module "Device" constitue le point de départ du processus de localisation sans fil, assurant l'authenticité des communications et facilitant l'estimation de la position du dispositif par les capteurs. Son fonctionnement sécurisé et fiable contribue à la précision et à la robustesse globales du système de localisation.

### 5.3.2.1.2 - Sensor

```
#include <omnetpp.h>
#include <cstring>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/err.h>

using namespace omnetpp;

class Sensor : public cSimpleModule {
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void sendToAccessPoint(int deviceId, double distance, double posX, double posY, const std::string& publicKey);
    virtual std::string decryptMessage(const std::string& encryptedMessage, const std::string& publicKey);

private:
    double posX;
    double posY;
    double speedOfSignal;
    int sensorId;
};

Define_Module(Sensor);

void Sensor::initialize() {
    posX = par("posX");
    posY = par("posY");
    speedOfSignal = par("speedOfSignal");
    sensorId = getParentModule()->getIndex();
}

void Sensor::handleMessage(cMessage *msg) {
    if (msg->hasPar("signedMessage") && msg->hasPar("publicKey")) {
        std::string signedMessage = msg->par("signedMessage").stringValue();
        std::string publicKey = msg->par("publicKey").stringValue();

        // Decrypt the message with public key
        std::string decryptedMessage = decryptMessage(signedMessage, publicKey);

        // Extract device ID and time from decrypted message
        std::size_t pos = decryptedMessage.find_last_of("-");
        if (pos != std::string::npos) {
            std::string deviceIdStr = decryptedMessage.substr(0, pos);
            std::string timeStr = decryptedMessage.substr(pos + 1);
            int deviceId = std::stoi(deviceIdStr);
            double messageTime = std::stod(timeStr);

            // Calculate elapsed time since message was sent
            double currentTime = simTime().dbl();
            double elapsedTime = currentTime - messageTime;
            // Calculate distance between sensor and device
        }
    }
}
```

```
double distance = elapsedTime * speedOfSignal;
EV_INFO << "Sensor " << sensorId << " calculated distance to Device " << deviceId << ": " << distance << endl;

// Send message to Access Point
sendToAccessPoint(deviceId, distance, posX, posY, publicKey);
}
else {
    EV_WARN << "Invalid message format received." << endl;
}
}
else {
    EV_WARN << "Invalid message received." << endl;
}
delete msg;
}

std::string Sensor::decryptMessage(const std::string& encryptedMessage, const std::string& publicKey) {
    std::string decryptedMessage;

    // Convert public key string to RSA structure
    RSA *rsa = createRSA((unsigned char *)publicKey.c_str(), false);
    if (!rsa) {
        EV_ERROR << "Failed to create RSA key from public key string." << endl;
        return decryptedMessage;
    }

    // Decrypt the message
    unsigned char *decrypted = (unsigned char *)malloc(RSA_size(rsa));
    int decryptedLength = RSA_public_decrypt(encryptedMessage.length(), (const unsigned char *)encryptedMessage.c_str(), decrypted, rsa, RSA_PKCS1_PADDING);
    if (decryptedLength == -1) {
        EV_ERROR << "Failed to decrypt message with RSA." << endl;
        RSA_free(rsa);
        free(decrypted);
        return decryptedMessage;
    }

    decryptedMessage.assign((char *)decrypted, decryptedLength);

    RSA_free(rsa);
    free(decrypted);
    return decryptedMessage;
}

void Sensor::sendToAccessPoint(int deviceId, double distance, double posX, double posY, const std::string& publicKey) {
    cMessage *message = new cMessage("SensorToAPMessage");
    message->addPar("deviceId");
    message->par("deviceId").setIntValue(deviceId);
    message->addPar("distance");
    message->par("distance").setDoubleValue(distance);
    message->addPar("posX");
    message->par("posX").setDoubleValue(posX);
    message->par("posY").setDoubleValue(posY);
    message->addPar("publicKey");
    message->par("publicKey").setStringValue(publicKey);

    send(message, "out");
}
```

```
message->par("posY").setDoubleValue(posY);
message->addPar("publicKey");
message->par("publicKey").setStringValue(publicKey);

send(message, "out");
}
```

Continuons avec le "Sensor", ce module joue un rôle crucial dans notre système de localisation sans fil, en fournissant des données essentielles sur la position et la distance par rapport au dispositif. À son initialisation, le capteur récupère sa propre position dans l'environnement, fournie en tant que paramètres. Cette position est utilisée ultérieurement pour calculer la distance entre le capteur et le dispositif. Lorsqu'un message est reçu en provenance du dispositif, le capteur décode le message à l'aide de la clé publique du dispositif. Ce message contient l'identifiant du dispositif et le temps de transmission, permettant au capteur de calculer la distance entre lui-même et le dispositif, en tenant compte de la vitesse de propagation du signal. Une fois la distance calculée, le capteur prépare un nouveau message contenant son identifiant, l'identifiant du dispositif, sa position, la distance calculée et la clé publique du dispositif. Ce message est ensuite envoyé à l'Access Point pour traitement ultérieur. En plus de son rôle dans la réception et le traitement des messages du dispositif, le capteur peut également recevoir des données d'autres capteurs dans le réseau.

Bien que ces données ne soient pas directement utilisées pour déterminer la position du dispositif, elles peuvent contribuer à améliorer la robustesse et la fiabilité du système de localisation. En résumé, le module “Sensor” constitue un maillon essentiel dans la chaîne de localisation sans fil, en fournissant des données précieuses sur la position et la distance par rapport au dispositif. Son fonctionnement fiable et précis contribue à garantir la précision globale du système de localisation.

### 5.3.2.1.3 - Access Point

```
#include <omnetpp.h>
#include <cstring>
#include <cmath>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/err.h>

using namespace omnetpp;

class AccessPoint : public cSimpleModule {
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void processSensorMessage(cMessage *msg);
    virtual void calculateDevicePosition();
    virtual void sendPositionToDevice(double posX, double posY, const std::string& publicKey);
    virtual std::string encryptPosition(double posX, double posY, const std::string& publicKey);

private:
    int receivedSensorMessages;
    std::map<int, std::pair<double, double>> sensorPositions;
    std::map<int, double> sensorDistances;
    std::string devicePublicKey;
};

Define_Module(AccessPoint);

void AccessPoint::initialize() {
    receivedSensorMessages = 0;
}

void AccessPoint::handleMessage(cMessage *msg) {
    processSensorMessage(msg);
    delete msg;
}

void AccessPoint::processSensorMessage(cMessage *msg) {
    int deviceId = msg->par("deviceId").intValue();
    double distance = msg->par("distance").doubleValue();
    double posX = msg->par("posX").doubleValue();
    double posY = msg->par("posY").doubleValue();
    std::string publicKey = msg->par("publicKey").stringValue();

    sensorPositions[deviceId] = std::make_pair(posX, posY);
    sensorDistances[deviceId] = distance;

    receivedSensorMessages++;

    if (receivedSensorMessages == 3) {
        calculateDevicePosition();
    }
}
```

```
}

void AccessPoint::calculateDevicePosition() {
    // Check if we have received messages from at least 3 sensors
    if (receivedSensorMessages < 3) {
        EV_WARN << "Insufficient sensor messages to perform trilateration." << endl;
        return;
    }

    // Perform trilateration to estimate device position
    double sumX = 0, sumY = 0;
    double sumWeights = 0;
    for (auto& entry : sensorPositions) {
        int deviceId = entry.first;
        double posX = entry.second.first;
        double posY = entry.second.second;
        double distance = sensorDistances[deviceId];

        // Calculate weight (inverse of distance) for weighted average
        double weight = 1 / distance;
        sumWeights += weight;

        sumX += posX * weight;
        sumY += posY * weight;
    }
    double devicePosX = sumX / sumWeights;
    double devicePosY = sumY / sumWeights;

    // Encrypt device position with one of the sensor's public key
    std::string encryptedPosition = encryptPosition(devicePosX, devicePosY, devicePublicKey);

    // Send encrypted position to device
    sendPositionToDevice(devicePosX, devicePosY, encryptedPosition);
}

std::string AccessPoint::encryptPosition(double posX, double posY, const std::string& publicKey) {
    // Convert position to string
    std::string positionStr = std::to_string(posX) + "-" + std::to_string(posY);

    // Convert public key string to RSA structure
    RSA *rsa = createRSA((unsigned char *)publicKey.c_str(), false);
    if (!rsa) {
        EV_ERROR << "Failed to create RSA key from public key string." << endl;
        return "";
    }

    // Encrypt the position
    unsigned char *encrypted = (unsigned char *)malloc(RSA_size(rsa));
    int encryptedLength = RSA_public_encrypt(positionStr.length(), (const unsigned char *)positionStr.c_str(), encrypted, rsa, RSA_PKCS1_PADDING);
    if (encryptedLength == -1) {
        EV_ERROR << "Failed to encrypt position with RSA." << endl;
        RSA_free(rsa);
        RSA_free(encrypted);
        return "";
    }

    std::string encryptedPosition((char *)encrypted, encryptedLength);

    RSA_free(rsa);
    free(encrypted);
    return encryptedPosition;
}

void AccessPoint::sendPositionToDevice(double posX, double posY, const std::string& encryptedPosition) {
    // Create message with encrypted position
    cMessage *message = new cMessage("APToDeviceMessage");
    message->addPar("posX");
    message->par("posX").setDoubleValue(posX);
    message->addPar("posY");
    message->par("posY").setDoubleValue(posY);
    message->addPar("encryptedPosition");
    message->par("encryptedPosition").setStringValue(encryptedPosition);

    // Send message to device
    send(message, "out");
}
```

Le module "AccessPoint" joue un rôle central dans notre système de localisation sans fil en collectant et en traitant les données des capteurs pour estimer la position du dispositif. À son initialisation, l'Access Point est prêt à recevoir les messages des capteurs contenant des informations sur leur position, la distance par rapport au dispositif, et la clé publique du dispositif. Lorsque l'Access Point reçoit les données de trois capteurs différents, il utilise la



méthode de trilatération pour estimer la position du dispositif dans l'environnement. Cette méthode combine les positions des capteurs et les distances entre les capteurs et le dispositif pour calculer sa position estimée. Une fois la position estimée du dispositif calculée, l'Access Point chiffre cette position avec l'une des clés publiques des capteurs, garantissant ainsi la confidentialité de l'information. Le message contenant la position chiffrée est ensuite envoyé au dispositif pour utilisation ultérieure. En plus de son rôle dans l'estimation de la position du dispositif, l'Access Point peut également recevoir des informations supplémentaires des capteurs, contribuant ainsi à améliorer la précision et la fiabilité du système de localisation. En résumé, le module "AccessPoint" est un composant central dans le processus de localisation sans fil, en combinant les données des capteurs pour estimer la position du dispositif. Son fonctionnement efficace et sécurisé garantit la précision et la confidentialité des informations de localisation.

Il faudra bien sûr associer cela à la partie blockchain pour stocker les différentes positions, ou encore utiliser le code de la trilatération qui ici est un code basique.

### 5.3.2.2 Simulation

Ainsi avec ce code nous pouvons passer à l'analyse des différentes simulations.

Pour cette première simulation, nous supposons un environnement idéal où aucun élément ne perturbe le fonctionnement de l'application. Cependant, plusieurs problèmes potentiels sont à noter. Tout d'abord, la précision de la localisation dépend du moment où l'utilisateur envoie le message. Ainsi, pour garantir une position actualisée en temps réel, il serait nécessaire d'envoyer régulièrement des messages en broadcast pour mettre à jour la position du dispositif. De plus, il convient de noter qu'un engin se déplaçant à grande vitesse, tel qu'un avion, pourrait sortir de la zone de couverture de l'Access Point et ne pas recevoir sa position.

Maintenant la deuxième situation, nous introduisons des perturbations telles que des ralentissements dans l'envoi du signal ou des obstacles physiques. En plus des problèmes mentionnés précédemment, ces perturbations entraînent une baisse de la précision de la localisation. En effet, des retards dans la réception du signal peuvent induire des estimations de distances erronées, ce qui compromet la précision de la trilatération. De plus, si moins de trois capteurs reçoivent le signal, une localisation précise devient impossible en raison de l'incapacité à effectuer la trilatération.

Pour cette dernière simulation, nous introduisons un dispositif malveillant qui cherche à usurper l'identité d'un autre utilisateur. Plusieurs scénarios sont envisageables. Par exemple, le dispositif malveillant pourrait intercepter un des signaux et tenter de le modifier. Cependant,



le fait que la localisation soit codée avec la clé publique de l'utilisateur cible rend cette tentative inefficace, car seul l'utilisateur concerné peut décoder la localisation précise. De plus, le dispositif malveillant pourrait tenter de remplacer son ID et sa clé publique par ceux de l'utilisateur cible. Pour contrer cette menace, les capteurs peuvent comparer la date d'envoi du signal, un paramètre non modifiable, avec la date d'envoi reçue dans le message codé. Toute différence indiquerait une altération du message, et le score de l'utilisateur correspondant à l'ID pourrait être diminué dans la blockchain, signalant ainsi un comportement malveillant.

Les trois simulations présentées mettent en lumière les défis et les solutions associés à l'implémentation d'un système de localisation sans fil. Dans un environnement idéal, où aucune perturbation n'est présente, le système fonctionne de manière optimale, offrant des localisations précises en temps réel. Cependant, des perturbations telles que des ralentissements dans l'envoi du signal ou des obstacles physiques peuvent entraîner une baisse de la précision de la localisation. De plus, la présence d'un dispositif malveillant peut compromettre la sécurité et l'intégrité du système. Des mesures de sécurité telles que le cryptage des données et la vérification de l'intégrité des messages sont essentielles pour contrer les tentatives d'usurpation d'identité et de manipulation des données. En résumé, la conception et la mise en œuvre d'un système de localisation sans fil nécessitent une approche holistique prenant en compte les aspects techniques, de sécurité et de fiabilité. En anticipant et en adressant ces défis, il est possible de développer un système robuste et efficace, capable de répondre aux besoins variés des utilisateurs dans divers contextes d'utilisation.

### 5.3.2.3 NS3

Nous avons décidé, en parallèle de OMNeT++, d'essayer également de faire des simulations avec NS3, mais les résultats sont assez loin de l'objectif demandé, c'est-à-dire réussir à obtenir les coordonnées d'un appareil grâce à une trilatération. Nous avons donc réussi à faire uniquement une communication entre deux "appareils", avec une connexion sans fil, sans forcément essayer d'obtenir la position de l'appareil grâce à la trilatération.

Notre topologie est la suivante:

- 2 Noeuds (0,1) appelés "p2pNodes" qui ont une connexion directe entre eux.
- 1 Noeud (0) qui en plus d'être un noeud de la structure peer-to-peer est considéré comme le point d'accès, appelé "wifiApNode", des noeuds Wifi (2,3,4), celui-ci assure donc la communication entre le noeud 1 et les noeuds 2,3,4
- 3 Noeuds (2,3,4) appelés "wifiStaNodes" qui représentent des appareils qui se déplacent et qui sont connectés à l'Access Point (Noeud 0).

Pour rendre la simulation réaliste nous avons décidé de faire bouger les nœuds “wifiStaNodes” pour simuler de vrais appareils portables mais de laisser l’access point statique pour simuler un routeur.

On a donc considéré un scénario où un client envoie un paquet à un serveur, ensuite le serveur envoie à son tour un paquet au client. Nous avons ici attribué le rôle du client au nœud 4 avec l’adresse IP 10.1.3.3 et le rôle du serveur au nœud 1 avec l’adresse IP 10.1.1.2. Nous remarquons donc une flèche vers un peu plus de 2 secondes se déplacer entre le nœud 0 et le nœud 1, celle-ci représente l’envoi du paquet vers le nœud 1 dans un premier temps, ensuite la réponse vers le nœud 0. Cependant nous avons un problème que nous n’avons pas pu résoudre, qui est l’absence de la représentation graphique de l’envoi du paquet entre le nœud 4 et le nœud 0, nous sommes pourtant sûrs qu’un paquet a été envoyé car comme nous le voyons dans la réponse que le terminal nous a donné, après avoir lancé le programme, concorde bien avec le scénario considéré. Nous voyons bien que les adresses IP correspondent à celle du client, c'est-à-dire l’appareil connecté en Wifi avec l’adresse IP 10.1.3.3, et à celle du serveur.

```
2 #include "ns3/point-to-point-module.h"
3 #include "ns3/network-module.h"
4 #include "ns3/applications-module.h"
5 #include "ns3/wifi-module.h"
6 #include "ns3/mobility-module.h"
7 #include "ns3/csmn-module.h"
8 #include "ns3/internet-module.h"
9 #include "ns3/netanim-module.h"
10
11 using namespace ns3;
12
13 int main(int argc, char *argv[])
14 {
15     LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
16     LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
17
18     // P2P
19     NodeContainer p2pNodes;
20     p2pNodes.Create(2);
21
22     PointToPointHelper pointToPoint;
23     pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
24     pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
25
26     NetDeviceContainer p2pDevices;
27     p2pDevices = pointToPoint.Install(p2pNodes);
28
29     // WIFI
30     NodeContainer wifiStaNodes;
31     wifiStaNodes.Create(3);
32     NodeContainer wifiApNode = p2pNodes.Get(0);
33
34     YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
35     YansWifiPhyHelper phy;
36     phy.SetChannel(channel.Create());
37
38     WifiHelper wifi;
39     wifi.SetRemoteStationManager("ns3:MinstrelHtWifiManager");
40
41     WifiMacHelper mac;
42     Ssid ssid = Ssid("ns-3-ssid");
43
44     mac.SetType("ns3:StaWifiMac",
45                 "Ssid", SsidValue(ssid),
46                 "ActiveProbing", BooleanValue(false));
47
48     NetDeviceContainer staDevices;
49     staDevices = wifi.Install(phy, mac, wifiStaNodes);
50
51     int main(int argc, char *argv[])
52     {
53         mac.SetType("ns3:ApWifiMac",
54                     "Ssid", SsidValue(ssid));
55
56         NetDeviceContainer apDevices;
57         apDevices = wifi.Install(phy, mac, wifiApNode);
58
59         MobilityHelper mobility;
60         mobility.SetPositionAllocator("ns3:GridPositionAllocator",
61                                     "MinX", DoubleValue(0.0),
62                                     "MinY", DoubleValue(0.0),
63                                     "DeltaX", DoubleValue(5.0),
64                                     "DeltaY", DoubleValue(10.0),
65                                     "GridWidth", IntegerValue(3),
66                                     "LayoutType", StringValue("RowFirst"));
67
68         mobility.SetMobilityModel("ns3:RandomWalk2DMobilityModel", "Bounds",
69                                 RectangleValue(Rectangle(-50, 50, -50, 50)));
70         mobility.Install(wifiStaNodes);
71
72         mobility.SetMobilityModel("ns3:ConstantPositionMobilityModel");
73         mobility.Install(wifiApNode);
74
75         InternetStackHelper stack;
76         stack.Install(p2pNodes);
77         stack.Install(wifiApNode);
78         stack.Install(wifiStaNodes);
79
80         Ipv4AddressHelper address;
81         address.SetBase("10.1.1.0", "255.255.255.0");
82         Ipv4InterfaceContainer p2pInterfaces;
83         p2pInterfaces = address.Assign(p2pDevices);
84
85         address.SetBase("10.1.3.0", "255.255.255.0");
86         address.Assign(staDevices);
87         address.Assign(apDevices);
88
89         UdpEchoServerHelper echoServer(9);
90
91         ApplicationContainer serverApps = echoServer.Install(p2pNodes.Get(1));
92         serverApps.Start(Seconds(1.0));
93         serverApps.Stop(Seconds(10.0));
94
95         UdpEchoClientHelper echoClient(Ipv4Address("10.1.1.2"), 9);
96         echoClient.SetAttribute("MaxPackets", IntegerValue(1));
```

Image 1/2 du code de la simulation

```
UdpEchoClientHelper echoClient(Ipv4Address("10.1.1.2"), 9);
echoClient.SetAttribute("MaxPackets", UintegerValue(1));
echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
echoClient.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientApps = echoClient.Install(wifiStaNodes.Get(2));
clientApps.Start(Seconds(2.0));
clientApps.Stop(Seconds(10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

pointToPoint.EnablePcapAll("p2p7");
phy.EnablePcapAll("wifi");

AnimationInterface anim("wifi-sim-anim.xml");
anim.SetConstantPosition(p2pNodes.Get(0), 10.0, 10.0);
anim.SetConstantPosition(p2pNodes.Get(1), 20.0, 20.0);

Simulator::Stop(Seconds(10.0));


Simulator::Run();
Simulator::Destroy();

return 0;
```

Image 2/2 du code de la simulation

```
bababoy@fedora:~/ns-allinone-3.41/ns-3.41$ ./ns3 run sim-wifi
[0/2] Re-checking globbed directories...
[2/2] Linking CXX executable /home/bababoy/ns-allinone-3.41/ns-3.41/build/scratch/ns3.41-sim-wifi-default
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00732s server received 1024 bytes from 10.1.3.3 port 49153
At time +2.00732s server sent 1024 bytes to 10.1.3.3 port 49153
At time +2.01252s client received 1024 bytes from 10.1.1.2 port 9
```

Réponse du terminal lors du lancement du programme

Représentation graphique de la simulation :  ns3-wifi-sim.webm

## VI. Conclusion du projet

Le projet de réseau D2D (Device-to-Device) basé sur la technologie de la blockchain offre des perspectives innovantes pour les communications sans fil décentralisées. À travers ce rapport, nous avons exploré en détail les différentes composantes et fonctionnalités de notre système, ainsi que les défis rencontrés et les solutions proposées. Nous avons commencé par décrire l'architecture globale du réseau D2D, mettant en évidence les rôles distincts des dispositifs, des capteurs et de l'Access Point dans le processus de localisation. Nous avons aussi souligné l'importance de la sécurité et de la confidentialité des données, notamment grâce à l'utilisation de techniques de cryptographie asymétrique basée sur RSA. Ensuite, nous avons examiné les simulations de fonctionnement du système dans différentes conditions, de la situation idéale à celle avec des perturbations et la présence d'un dispositif malveillant. Ces simulations ont permis de mettre en évidence les défis potentiels et les stratégies de résolution, soulignant l'importance de la robustesse et de la fiabilité du système. Enfin, nous avons mis à l'avant l'importance de l'approche holistique dans la conception et la mise en œuvre d'un tel système. La combinaison de la technologie blockchain avec les communications sans fil décentralisées ouvre la voie à de nouvelles applications et opportunités dans des domaines tels que la logistique, la gestion des ressources, et bien d'autres. En conclusion, notre projet de réseau D2D blockchain représente une contribution significative à l'avancement de la recherche en matière de communications sans fil décentralisées et sécurisées. En continuant à explorer et à développer ces concepts, nous pouvons envisager un avenir où les communications sans fil sont plus résilientes, sécurisées, décentralisées et accessibles à tous.

## VII. Perspective future du projet

Dans un contexte où la sécurité des données est devenue une préoccupation majeure, notre projet de localisation D2D blockchain revêt une importance capitale. En proposant une alternative sécurisée, transparente et décentralisée, nous répondons à un besoin croissant de protection des informations personnelles et de confidentialité des utilisateurs. En éliminant les risques associés à la centralisation des données, tels que la vulnérabilité aux attaques et aux violations de la vie privée, notre système offre une solution fiable et robuste pour la localisation en temps réel, tout en évitant la concentration de pouvoir et de contrôle entre les mains d'une seule entité. Nous sommes convaincus que notre projet ouvre la voie à de nouvelles perspectives dans le domaine de la localisation et des communications sans fil. En

offrant une plateforme sécurisée et transparente, nous encourageons l'innovation et la créativité dans le développement de services et d'applications décentralisés. De plus, en intégrant des technologies émergentes telles que l'intelligence artificielle et l'IoT, nous envisageons d'augmenter encore davantage la précision et l'efficacité de notre système, ouvrant ainsi de nouvelles opportunités pour l'exploration et l'expansion de notre solution. En conclusion, notre projet représente bien plus qu'une simple alternative aux systèmes de localisation existants. Il incarne une vision audacieuse pour l'avenir, où la sécurité, la confidentialité, la décentralisation des données et la lutte contre les monopoles sont au cœur de notre approche technologique. Nous sommes optimistes quant à son potentiel à façonner un paysage numérique plus sûr, plus équitable et plus résilient pour les générations futures.

## VIII. Bibliographie

- 1 Amoretti M, Brambilla G, Mediolli F, Zanichelli F. Blockchain-Based Proof of Location. In: *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2018, pp 146–153.

**Descriptif :** Ce papier scientifique commence tout d’abord par nous apprendre comment les appareils mobiles sont devenus indispensables dans le monde dans lequel nous vivons. Ces auteurs voient alors l’ajout de la blockchain dans un système de localisation de type "short-range interaction", en définissant une architecture de type "device-to-device" de façon décentralisée qui est la base même de la blockchain. Pour cela, ils se basent sur différentes architectures déjà existantes afin de parvenir à créer un système combinant sécurité, économie d’énergie et décentralisation. Tout cela est accompagné d’une analyse poussée du système avec des tests prouvant leur efficacité et comment y faire face. Mais la blockchain n’est pas gérée correctement, et des explications primordiales sur la manière dont est stocké l’espace de stockage de la blockchain sont nécessaires.

- 2 Mahjri I, Dhraief A, Belghith A, Drira K, Mathkour H. A GPS-less Framework for Localization and Coverage Maintenance in Wireless Sensor Networks. *KSI Transactions on internet and information systems* 2016; **10**: 96.

**Descriptif:** Ce papier de recherche propose une solution de localisation d’un appareil sans utiliser de GPS. Les auteurs de ce papier ont décidé de prendre des senseurs et de les éparpiller sur une grande zone, certains de ces senseurs sont des senseurs ancrés, ceux-ci sont équipés de GPS, mais ce ne sont qu’une petite partie des senseurs. Le but est donc de localiser un appareil qui est dans la zone de localisation d’un senseur, et pour faire ceci, un algorithme fait en sorte de détecter la zone approximative de l’appareil en remontant récursivement aux nœuds ancrés.

- 3 Saad C, Benslimane A, König J-C. AT-Dist: A Distributed Method for Localization with High Accuracy in Sensor Networks. *Studia Informatica Universalis* 2008; **6**: 14.

**Descriptif:** Ce papier présente une méthode appelée AT-Dist permettant aux nœuds ancrés (anchor nodes) de localiser et d’obtenir les coordonnées d’un nœud spécifique, permettant ainsi de limiter le nombre de nœuds nécessitant d’être équipés de fonctionnalités pour s’auto-localiser. Il propose d’utiliser l’inégalité triangulaire, provenant du fait que la distance entre les nœuds est toujours supérieure à la distance

en ligne droite, pour créer des cercles qui contiennent certainement le nœud à localiser. Ainsi, avec l'intersection de trois cercles, nous obtenons une zone contenant avec certitude le nœud recherché.

- 4 Kumari J, Kumar P. Location error analysis of WSN in 3D complex terrain. *Journal of Control and Decision* 2023; : 1–10.

**Descriptif** : Dans ce lien , on découvre que pour les terrains complexes 3D, une localisation en deux étapes combinant trilatération et RSSI, suivie d'une modélisation de terrain via la triangulation de Delaunay, offre une précision accrue pour les réseaux de capteurs, traditionnellement limités aux déploiements 2D.

- 5 Brandes U. A faster algorithm for betweenness centrality\*. *The Journal of Mathematical Sociology* 2001; **25**: 163–177.

**Descriptif**: Ce papier propose un algorithme qui permet d'augmenter l'efficacité de calcul du betweenness d'un noeud (le nombre de chemin qui passe a travers ce noeud spécifiquement et concerne tous les noeux du graphes) grâce à la proposition de nouveaux lemme

- 6 Amir H, Andrew A, Andrew T, Marc N, Rahul G. Helium: A Decentralized Wireless Network. 2018; : 20.

**Descriptif** : Ce document scientifique nous montre comment helium fonctionne avec le rôle des mineurs dans la blockchain, avec la proof of coverage et la proof of serialization et comment ces deux preuves permettent la création d'une preuve de localisation.

- 7 Kaishun Wu, Jiang Xiao, Youwen Yi, Min Gao, and Lionel M. Ni. 2012. FILA: Fine-grained indoor localization. In *2012 Proceedings IEEE INFOCOM*, March 2012. 2210–2218. . <https://doi.org/10.1109/INFOCOM.2012.6195606>

**Descriptif**: Ce papier propose une nouvelle méthode de calcul de distance à l'intérieur entre un appareil et un point d'accès. Il propose d'utiliser les valeurs qu'apportent CSI provenant des sous-couches de la méthode OFDMA (utilisé pour la 4G). Le CSI apportent des informations sur la fréquence et l'amplitude de chaque sous-porteuse et permet donc d'avoir une meilleure précision lorsque nous faisons une moyenne (appelle CSI effectif). Il donne l'équation qui permet de calculer la distance grâce à cette moyenne.

**Projet 11**  
**Com. D2D Blockchain**

- 8 Chitra Javali, Girish Revadigar, Kasper B. Rasmussen, Wen Hu, and Sanjay Jha. 2016. I Am Alice, I Was in Wonderland: Secure Location Proof Generation and Verification Protocol. In 2016 IEEE 41st Conference on Local Computer Networks (LCN), November 2016, Dubai. IEEE, Dubai, 477–485. .  
<https://doi.org/10.1109/LCN.2016.126>

**Descriptif :** Ce papier lié au 1er document explique comment la preuve de localisation est créé en fonction du CS<sub>Ieff</sub> qui est basé sur le “fingerprinting” où la zone de couverture de l’appareil est divisé en grids où il est associé un id et un CS<sub>Ieff</sub> de la zone et de quelle façon cette preuve est codé grâce à un polynôme, qui sera alors chiffré grâce à l’algorithme HIDE pour une plus grande sécurité.

- 9 Yakovenko Anatoly. 2018. Solana: A new architecture for a high performance blockchain (2018), 32. Retrieved from <https://coincode-live.github.io/static/whitepaper/source001/10608577.pdf>

**Descriptif :** Explique le fonctionnement global de la blockchain Solana qui est la suite de HELIUM [6], ce qui nous intéressent ici est surtout la “proof of history”, où les mineurs vérifient la transaction coordonnées par un leader qui est élu tous les 400 millisecondes, pour vérifier que tout le monde devient bien leader à un moment une proof of history est mis en place, ou ils placent à l’entrée d’une fonction de hash les données et les envoies dans la chaîne en série, ainsi il faut connaître le bloc  $h(k)$  pour avoir  $h(k+1)$ , donc on peut vérifier si tous les hash sont correcte de façon parallèle. Ainsi nous avons un mécanisme qui fait confiance à l'ordre des choses.